ICSEA 2012

The Seventh International Conference on Software Engineering Advances

ISBN: 978-1-61208-230-1

November 18-23, 2012

Lisbon, Portugal

ICSEA 2012 Editors

Herwig Mannaert, University of Antwerp, Belgium

Luigi Lavazza, Università dell'Insubria - Varese, Italy

Roy Oberhauser, Aalen University, Germany

Elena Troubitsyna, Åbo Akademi University, Finland

Michael Gebhart, Gebhart Quality Analysis (QA) 82, Germany

Osamu Takaki, Japan Advanced Institute of Science and Technology (JAIST) – Ishikawa, Japan

ICSEA 2012

Forward

The Seventh International Conference on Software Engineering Advances (ICSEA 2012), held on November 18-23, 2012 in Lisbon, Portugal, continued a series of events covering a broad spectrum of software-related topics.

The conference covered fundamentals on designing, implementing, testing, validating and maintaining various kinds of software. The tracks treated the topics from theory to practice, in terms of methodologies, design, implementation, testing, use cases, tools, and lessons learnt. The conference topics covered classical and advanced methodologies, open source, agile software, as well as software deployment and software economics and education.

The conference had the following tracks:

- Advances in fundamentals for software development
- Advanced mechanisms for software development
- Advanced design tools for developing software
- Advanced facilities for accessing software
- Software performance
- Software security, privacy, safeness
- Advances in software testing
- Specialized software advanced applications
- Open source software
- Agile software techniques
- Software deployment and maintenance
- Software engineering techniques, metrics, and formalisms
- Software economics, adoption, and education
- Business technology
- Improving research productivity

Similar to the previous edition, this event continued to be very competitive in its selection process and very well perceived by the international software engineering community. As such, it is attracting excellent contributions and active participation from all over the world. We were very pleased to receive a large amount of top quality contributions.

We take here the opportunity to warmly thank all the members of the ICSEA 2012 technical program committee as well as the numerous reviewers. The creation of such a broad and high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and efforts to contribute to the ICSEA 2012. We truly believe that thanks to all these efforts, the final conference program consists of top quality contributions.

This event could also not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the ICSEA 2012 organizing committee for their help in handling the logistics and for their work that is making this professional meeting a success.

We hope the ICSEA 2012 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in software engineering research.

We hope Lisbon provided a pleasant environment during the conference and everyone saved some time for exploring this beautiful city.

ICSEA 2012

Committee

ICSEA Advisory Chairs

Herwig Mannaert, University of Antwerp, Belgium
Jon G. Hall, The Open University - Milton Keynes, UK
Mira Kajko-Mattsson, Stockholm University & Royal Institute of Technology, Sweden
Luigi Lavazza, Università dell'Insubria - Varese, Italy
Roy Oberhauser, Aalen University, Germany
Elena Troubitsyna, Åbo Akademi University, Finland
Luis Fernandez-Sanz, Universidad de Alcala, Spain
Michael Gebhart, Gebhart Quality Analysis (QA) 82, Germany

ICSEA 2012 Research Institute Liaison Chairs

Oleksandr Panchenko, Hasso Plattner Institute for Software Systems Engineering - Potsdam, Germany
Teemu Kanstrén, VTT Technical Research Centre of Finland - Oulu, Finland
Osamu Takaki, Japan Advanced Institute of Science and Technology (JAIST) – Ishikawa, Japan
Georg Buchgeher, Software Competence Center Hagenberg GmbH, Austria
Simon Tsang, Telcordia - Piscataway, USA

ICSEA 2012 Industry/Research Chairs

Herman Hartmann, University of Groningen, The Netherlands
Hongyu Pei Breivold, ABB Corporate Research, Sweden

ICSEA 2012 Special Area Chairs

Formal Methods
Paul J. Gibson, Telecom & Management SudParis, France

Business and process techniques
Maribel Yasmina Santos, University of Minho, Portugal

Testing and Validation
Florian Barth, University of Mannheim, Germany

ICSEA 2012 Technical Program Committee

Shahliza Abd Halim, Universiti of Technologi Malaysia (UTM) - Skudai, Malaysia
Mohammad Abdallah, Durham University, UK
Adla Abdelkader, University of Oran, Algeria
Moataz A. Ahmed, King Fahd University of Petroleum & Minerals – Dhahran, Saudi Arabia

Table of Contents

Enhancing Contexts for Automated Debugging Techniques

Yan Lei, Chengsong Wang, Xiaoguang Mao and Quanyuan Wu
School of Computer
National University of Defense Technology
Changsha, China
yanlei@nudt.edu.cn, jameschen186@gmail.com, xgmao@nudt.edu.cn, quanyuanwu@nudt.edu.cn

***Abstract*—Most existing automated debugging techniques just focus on selecting a set of suspicious statements that may cause failures and ranking them in terms of suspiciousness. Therefore, these techniques always ignore the contextual information of how suspicious statements behave and propagate in the program. However, the contextual information is useful for discovering and understanding bugs. Hence, this paper proposes a novel approach to enhance contexts for automated debugging techniques. Based on localization results obtained from automated debugging techniques, our approach utilizes program slicing to classify suspicious statements into different contexts, and assigns different suspiciousness to the contexts and their elements. The experimental study shows that our approach can substantially improve debugging effectiveness.**

Keywords-automate debugging; program spectra; program slicing; statistical analysis.

I. INTRODUCTION

Software debugging has been recognized as one of the most time-consuming tasks in the development and maintenance of software [5]. With the aim at reducing the cost of debugging, a great number of research techniques are proposed to support automating or semi-automating the process of debugging and improve its performance [3-25].

However, existing automated debugging techniques just focus on selecting a subset of statements potentially responsible for failures and ranking them according to some criterion. Therefore, they ignore the contextual information of how suspicious statements behave and propagate in the program. The recent research [1] has found that the lack of contextual information may affect the activity of discovering and understanding bugs. For example, the faulty statement is included in the set of suspicious statements, and developers inspect the statements in this set. Due to the lack of contextual information, developers may judge the faulty statement is not responsible for program failures and just ignore this statement when inspecting it. Hence, it is vital to enhance the contexts for automated debugging techniques.

Program slicing technique [14-16] utilizes data and control dependence to identify the set of program statements that may affect or be affected by the values at some statement of a program. The set of statements is referred to as a program slice. A program slice can be essentially regarded as a context that shows a causal chain of how data and control propagates in a program. However, program slicing treats the statements of a program slice with the same suspiciousness to be faulty and thus lacks the guidance as to how the statements in a slice should be examined. That always leads developers to be frustrated and tiresome as the size of a slice can substantially increase with the increasing size and complexity of today's software [1]. One possible way to address this issue is to use the promising ability of some automated debugging techniques to assign different suspiciousness to the statements of a slice. More importantly, it implies that automated debugging techniques can adopt program slicing to enhance contexts for themselves.

Among current research, one promising automated debugging technique exploits the correlations between program entities and program failures via statistically analyzing coverage information [3-13]. This technique is generally referred to as spectrum-based fault localization (SFL). SFL usually collects coverage information and test results from dynamic executions to construct program spectra from passed and failed executions. After gathering spectra information, SFL adopts a ranking metric to evaluate the suspiciousness of program entities to be faulty and gives a ranking list of all entities in terms of suspiciousness. The research [3-13] has shown that SFL has a promising structure of evaluating the suspiciousness of an entity to be faulty. Nevertheless, SFL just outputs a ranking list of isolated entities and fails to provide the contextual information for discovering and understanding these suspicious statements.

Considering the popularity and ability, this paper chooses SFL and enhances contexts for it by using program slicing technique. Hence, we propose a debugging approach which uses program slicing to enhance contexts for SFL. Our approach utilizes SFL to compute the suspiciousness of each statement to be faulty. Then, the approach uses program slicing to identify the most suspicious statement and its relevant statements as a context showing how the most suspicious statement behaves in a program. Finally, except for the statements of all constructed contexts, the most suspicious statement of the remaining statements and its relevant statements constitute a new context, and this step would be iterated until each statement is classified into a particular context. Because a context is constructed from the most suspicious statement in it, our approach assigns the suspiciousness of this statement to the context. For each statement, its suspiciousness keeps unchanged. In addition, our approach offers two modes to developers with different experiences, and utilizes visualization and program dependence to further assist developers. It can be seen that our approach provides useful contexts and recommends examining guidance of all contexts and their elements in terms of suspiciousness.

This paper conducts an experimental study on two standard benchmarks: the *Siemens suite* and *Space* [28], and compares our approach with nine ranking metrics of SFL: Ochiai [3], Jaccard [4], Tarantula [5], Wong2 [6], Wong3 [6], Ample [7], CBI [8], Optimal and Optimal^P [9]. The results demonstrate that the proposed approach outperforms all nine ranking metrics of SFL.

The remainder of this paper is organized as follows. Section II introduces the background of SFL. Section III describes our approach. Section IV conducts an experimental study. Related work is introduced in Section V and conclusions are given in Section VI.

II. Background of SFL

$$\begin{array}{cc} & \begin{array}{cc} N \ statements & errors \end{array} \\ M \ spectra & \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M1} & x_{M2} & \cdots & x_{MN} \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_M \end{bmatrix} \end{array}$$

Figure 1. Input to SFL.

Spectrum-based fault localization (SFL) [3] is a dynamic program analysis technique. It typically uses coverage information of passed and failed runs to rank program entities whose activity correlates most with the failures. Passed runs are executions of a program that output as expected, whereas failed runs are executions of a program that output as unexpected. There are various types of entities, such as blocks, functions, branches, paths, etc. This study adopts statements as the entities.

First, we assume that a program *P* comprises a set of statements *S* and runs against a set of test cases *T* that contains at least one failed test case, with $N=|S|$ and $M=|T|$, respectively (see Fig. 1). The above matrix $M\times(N+1)$ represents the input to SFL. An element x_{ij} is equal to 1 if statement *j* is covered by the execution of test run *i*, and 0 otherwise. The error vector *e* at the rightmost column of the matrix represents the test results. The element e_i is equal to 1 if run *i* failed, and 0 otherwise. Except the error vector, the rest of the matrix is expressed in terms of matrix *A*. The i^{th} row of *A* indicates whether a statement was covered by run *i*. The j^{th} column of *A* indicates statement *j* was covered by which runs.

SFL usually measures the suspiciousness of a statement to be faulty from similarity between its statement spectra and error vector in the above matrix (see Fig. 1), and finally outputs a ranking list of all statements in descending order of suspiciousness. The similarity is quantified by ranking metrics.

TABLE I shows nine ranking metrics of SFL and how the suspiciousness of statement *j* was computed by the corresponding ranking metrics. $a_{pq}(j)=|\{i|x_{ij}=p\wedge e_i=q\}|$, and $p, q\in\{0,1\}$. $a_{0q}(j)$ represents the number of passed (q=0) or failed (q=1) test cases that do not execute statement *j*. $a_{1q}(j)$ denotes the number of passed (q=0) or failed (q=1) test cases that execute statement *j*.

TABLE I. Formulas of SFL

Name	Formula	Name	Formula
Ochiai	$\frac{a_{11}(j)}{\sqrt{(a_{11}(j)+a_{01}(j))*(a_{11}(j)+a_{10}(j))}}$	Jaccard	$\frac{a_{11}(j)}{a_{11}(j)+a_{01}(j)+a_{10}(j)}$
Tarantula	$\frac{\frac{a_{11}(j)}{a_{11}(j)+a_{01}(j)}}{\frac{a_{11}(j)}{a_{11}(j)+a_{01}(j)}+\frac{a_{10}(j)}{a_{10}(j)+a_{00}(j)}}$	Ample	$\left\|\frac{a_{11}(j)}{a_{01}(j)+a_{11}(j)}-\frac{a_{10}(j)}{a_{00}(j)+a_{10}(j)}\right\|$
CBI	$\frac{a_{11}(j)}{a_{11}(j)+a_{10}(j)}-\frac{a_{11}(j)+a_{01}(j)}{a_{11}(j)+a_{01}(j)+a_{00}(j)+a_{10}(j)}$	Wong2	$a_{11}(j)-a_{10}(j)$
Optimal	$\begin{cases}-1 & \text{if } a_{01}(j)>0 \\ a_{00}(j) & \text{if } a_{01}(j)\le 0\end{cases}$	Optimal^P	$a_{11}(j)-\frac{a_{10}(j)}{a_{10}(j)+a_{00}(j)+1}$
Wong3	$a_{11}(j)-h, \text{ where } h=\begin{cases}a_{10}(j) & \text{if } a_{10}(j)\le 2 \\ 2+0.1*(a_{10}(j)-2) & \text{if } 2<a_{10}(j)\le 10 \\ 2.8+0.001*(a_{10}(j)-10) & \text{if } a_{10}(j)>10\end{cases}$		

SFL is widely accepted and studied as a promising automated technique in the debugging community, and the research has empirically proven that SFL is effective to correlate faulty statements with failures in terms of suspiciousness [3-13]. According to the popularity and ability, this study chooses SFL to evaluate the suspiciousness of contexts and their elements.

III. The Approach

Program slicing as a debugging aid was introduced by Mark Weiser [14]. Korel and Laski afterwards proposed dynamic slicing to focus on an execution in a specific input [15]. Because the localization result of SFL is based on the executions of a set of test cases instead of a specific execution, this study adopts static slices for our approach. There are two types of static slices: static backward slices (SBS) and static forward slices (SFS). The SBS of a variable at a statement includes all those statements which affect the value of the variable at that statement through chains of static data and/or control dependence [14]. In contrast, the SFS of a variable at a statement includes all those statements that are affected by the value of the variable at that statement through chains of static data and/or control dependence [19]. It can be seen that SBS can find a set of statements affecting a statement while SFS can identify a set of statements affected by a statement.

The basic idea of our approach is to apply program slicing to SFL by constructing different suspicious contexts and their elements for discovering and understanding faults. Our approach uses both SBS and SFS to construct a context showing how a statement affects and is affected by other statements in a program, and then utilizes SFL to evaluate the suspiciousness of each context and its elements.

The Algorithm 1 describes our approach. First, there are some explanations for Algorithm 1. This section adopts the program *P* and set of test cases *T* defined in Section II. The program *P* consists of a set of statements *S* that is denoted as $\{s_1, s_2, \ldots, s_N\}$. *FSlice*($s_i$) represents the union of the SFS of each variable at statement s_i. *BSlice*(s_i) denotes the union of SBS of each variable at statement s_i. *contextSet* stores all constructed contexts.

Algorithm 1 The proposed approach

```
1: Step 1: Compute the suspiciousness of each statement.
2:        SFLAnalyze(S,T);
3: Step 2: Construct different suspicious contexts and their statements.
4:        while(S is not empty){
5:          stmSet=GetMostSuspiciousStm(S);
6:          context=Φ;
7:          for(i=0; i<stmSet.size; i++)
8:            context=context ∪ BSlice(stmSet[i]) ∪ FSlice(stmSet[i]) ∪ stmSet[i];
9:          context.suspiciousness=stmSet[0].suspiciousness;
10:         S=S−context;
11:         if(mode=="weak-contexts")
12:           for(i=0;i<contextSet.size;i++)
13:             context=context−contextSet[i];
14:         Add(context,contextSet);
15:       }
16: Step 3: Output the localization results.
17:      if(visualization == true)
18:        Visualize(contextSet);
19:      else
20:        Text(contextSet);
```

Step 1: Compute the suspiciousness of each statement. The function *SFLAnalyze*(*S*,*T*) analyzes the statement coverage information and test results of test cases *T*, and adopts SFL to compute the suspiciousness of each statement in *S*. The format of statement coverage information and test results of test cases *T* is a matrix as shown in Fig. 1. In this study, for the statements with the same assigned suspiciousness, SFL ranks them in descending order of their line numbers in the source code. This strategy is also adopted by the *text-form* of localization results mentioned in the following step 3.

Step 2: Construct different suspicious contexts and their elements. This step iteratively constructs suspicious contexts until each statement is classified into a specific context. As the suspiciousness of some statements may be the same, the function *GetMostSuspiciousStm*(*S*) may return a set of most suspicious statements in *S* more than one statement. Lines 6 to 8 represent that besides the most suspicious statements, *context* consists of a set of statements that can affect or are affected by at least one of the most suspicious statements in *S*. As a context is constructed from the most suspicious statements in *S*, line 9 assigns the suspiciousness of the most suspicious statements to its corresponding context. Line 10 excludes the statements of the new context from the set of statements *S*.

There are two modes in our approach: *weak-contexts* and *strong-contexts*. The mode of *weak-contexts* demands that a statement can be only classified into one context. Therefore, it can cause some less suspicious contexts miss those statements presented in a more suspicious context. Lines 11 to 13 exclude those statements of all constructed contexts from the new context when the mode is *weak-contexts*. In contrast, the mode of *strong-contexts* maintains the integrity of each context. In this mode, a statement may be classified into different contexts. The step 3 presents a strategy to give some hints of the repetitive appearance of a statement in different contexts. Line 14 inserts the new context into *contextSet* that contains all constructed contexts.

Step 3: Output the localization results. This step outputs localization results in two different forms. One is *text-form* and the other is *visualization-form*. The functions *Text*(*contextSet*) and *Visualize*(*contextSet*) process the *text-form* and *visualization-form* of the localization results *contextSet* respectively. The *text-form* firstly ranks all contexts in descending order of their suspiciousness and then sequences the statements of each context in descending order of suspiciousness given by SFL. The ranking list is outputted in a text form. The *visualization-form* uses program dependence graphs [26,27] or lists the statements in a context along the program dependence edges from the starting point to show each context and its statements, and maps color to them according to the suspiciousness. This form can visually assist developers in understanding and locating faults. Some information is attached to each statement, such as *suspiciousness* and *role*. There are three types of *role*, namely "*root*", "*affect*" or "*affected*". The "*root*" denotes the statement is chosen to be sliced to construct the context. The "*affect*" and "*affected*" represents the statement affects or is affected by the "*root*" statement respectively. As mentioned in step 2, a context may be constructed from several statements with the same highest suspiciousness. Hence, a context may contain several "*root*" statements. In this case, the algorithm will number "*root*" statements to associate each "*root*" statement with its corresponding "*affect*" and "*affected*" statements.

The key idea of the color mapping algorithm is based on the visualization algorithm of Tarantula [5]. The color of a context or statement can be anywhere in the continuous spectrum of colors from red to yellow to green in descending order of suspiciousness. The contexts or statements are colored red to denote "danger" and indicate high likelihood of containing faults; those contexts or statements are specified green to denote "safety" and suggest little correlation with the failure; the contexts or statements are marked yellow to denote "caution" and imply a medium circumstance between "danger" and "safety". The visualization algorithm of Tarantula is implemented by *GIMP* and limits the suspiciousness to belong to [0, 1]. However, some metrics of SFL can produce a value of suspiciousness out of this range, such as Wong2, Wong3 and Optimal. In addition, this study chooses *GTK+* to implement the color mapping algorithm. Hence, a new color mapping equation is defined in Eq. (1).

$$color(s)=\begin{cases} Red=\begin{cases} Range, \text{ if } Rate \geq 0.5 \\ (\frac{Rate-0}{0.5})*Range=2*Rate*Range, \text{ if } Rate<0.5 \end{cases} \\ Green=\begin{cases} (\frac{1-Rate}{0.5})*Range=(2-2*Rate)*Range, \text{ if } Rate>0.5 \\ Range, \text{ if } Rate \leq 0.5 \end{cases} \\ Blue=0 \\ where,\ Rate=\frac{s.suspiciousness-minSuspiciousness}{maxSuspiciousness-minSuspiciousness} \text{ and } Range=65535 \end{cases} \quad .(1)$$

In Eq. (1), *s* represents a context or statement and *s.suspiciousness* denotes the suspiciousness of *s*. The *minSuspiciousness* and *maxSuspiciousness* are the minimum and maximum suspiciousness in all statements respectively. *GTK+* uses RGB model to produce different colors and specifies the values of each of the three basic colors at the range from 0 to 65535. In addition, the color mapping algorithm just needs red and green to generate the spectrum of colors for contexts and statements. As a result, *Range* and

Blue equal to 65535 and 0 in Eq. (1) respectively. It can be found that Eq. (1) can handle any range of the suspiciousness and map color to contexts and statements from red to yellow to green in descending order of suspiciousness. Note that if *maxSuspiciousness* equals to *minSuspiciousness*, all contexts and their statements will be colored green by the color mapping algorithm.

As mentioned in the step 2, although the mode of *strong-contexts* maintains the integrity of each context, checking those repetitive statements in different contexts may increase the workload of developers. A strategy is proposed to address this problem. Following the ranking list of all contexts in descending order of suspiciousness, this strategy examines the statements of each context in turn. When checking the statements of a context, some additional information is attached to those statements presented in previous examined contexts, such as *rank/total* and *pre-contexts*. The *rank/total* means the *rank* of the statement in *total* statements of the program in descending order of suspiciousness. The *pre-contexts* denote the set of higher ranked contexts that contains the statement before this context. The *rank/total* shows an indication of how suspicious a repetitive statement is in total statements of the program and the *pre-contexts* provides the connections of a repetitive statement in different contexts. In addition, the *visualization-form* colors gray to the repetitive statements. The above strategy offers some useful information of the repetitive statements and can alleviate the burden on developers for checking those repetitive statements.

As described above, it can be found that our approach can construct different suspicious contexts and their statements, and offer examining guidance of these contexts and statements in both *text-form* and *visualization-form*. It implies that the proposed approach equips SFL with contexts to further assist in discovering and understanding bugs.

IV. An Experimental Study

A. Experimental Setup

As SFL outputs a ranking list of all statements without repetition, this experiment use the *weak-contexts* mode of our approach to be compared to SFL. More concretely, the experiment study compares our approach in the *weak-contexts* mode with nine ranking metrics of SFL, namely Ochiai [3], Jaccard [4], Tarantula [5], Wong2 [6], Wong3 [6], Ample [7], CBI [8], Optimal and Optimal^P [9]. The formulas of these metrics are illustrated in TABLE I. This study chooses the *Siemens suite* and *Space* as the benchmarks because they are two widely used benchmarks in the field of software debugging with high quality. The two benchmarks can be obtained from the Software artifact Infrastructure Repository [28]. The *Siemens suite* contains 7 programs and 132 faulty versions of these programs. The *Space* contains 38 faulty versions. We select "*universe*" suite that contains all test cases in TABLE II. TABLE II lists the programs, the number of faulty versions of each program, lines of statements, lines of executable statements, number of all test cases, as well as the functional descriptions of the corresponding program.

TABLE II. Description of the *Siemens Suite* and *Space*

Program	Versions	LOC	Ex	Test	Description
print_tokens	7	563	203	4130	Lexical analyzer
print_tokens2	10	508	203	4115	Lexical analyzer
replace	32	563	289	5542	Pattern recognition
schedule	9	410	162	2650	Priority scheduler
schedule2	10	307	144	2710	Priority scheduler
tcas	41	173	67	1608	Altitude separation
tot_info	23	406	136	1052	Information measure
Space	38	9564	6218	13585	ADL interpreter

Although there are 170 versions in total, we were unable to adopt all of them. Because there was no failed test case in version 32 of *replace*, version 9 of *schedule2* and versions 1, 2, 34 of *Space*, we excluded the five versions. Additionally, we focus on executable statements, so the modifications of header files and definition/declaration errors were ignored. Hence, versions 4 and 6 of *print_tokens*, version 12 of *replace*, versions 13, 14, 36, 38 of *tcas* and versions 6, 10, 19, 21 of *tot_info* were also discarded. Finally, 154 faulty versions were used for the experiment.

In the experiment, the coverage information is gathered by using *Gcov* tool. We use FEMA (Failure Modes and Effects Analysis) slicing tool [26] developed by our group to perform program slicing. In addition, we adopt the *GTK+* to implement the algorithm of the visualization of our approach.

B. Evaluation metrics

The effectiveness of debugging techniques is widely evaluated by the percentage of code that needs to be examined (or not examined) to find the fault [6]. This evaluation assumes that developers will follow the ranking list to examine all statements from top to bottom until they encounter the faulty statement. Following this notion, we define *fault-localization accuracy* (referred as *Acc*) as the percentage of executable statements to be examined before finding the actual faulty statement [10]. A lower value of *Acc* indicates higher effectiveness.

For a more detailed comparison, we adopt *relative improvement* (referred as *Imp*) [10]. The *Imp* is to compare the total number of statements that need to be examined to find all faults using our approach versus the number that need to be examined by using the SFL. A lower value of *Imp* shows better improvement that our approach obtains.

C. Results and analysis

Figure 2 illustrates the *Acc* comparison between SFL and our approach in all faulty versions. The x-axis represents the percentage of executable statements to be examined. The y-axis denotes the percentage of faulty versions. A point in Fig. 2 represents when a percentage of executable statements is examined in each faulty version, the percentage of faulty versions has located their faults.

As shown in Fig. 2, the curves of our approach are usually higher than those of the corresponding metrics of SFL. It suggests that our approach improves the effectiveness of the nine metrics of SFL.

For a more detailed comparison, Fig. 3 presents the *Imp* of our approach over each metric of SFL in each program. The x-axis represents the name of each program. The y-axis

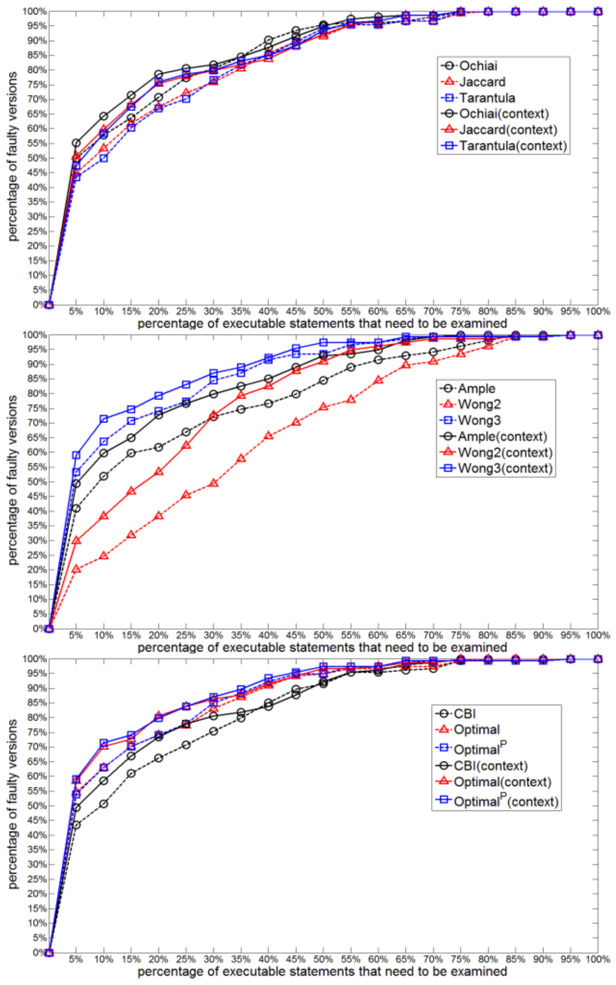
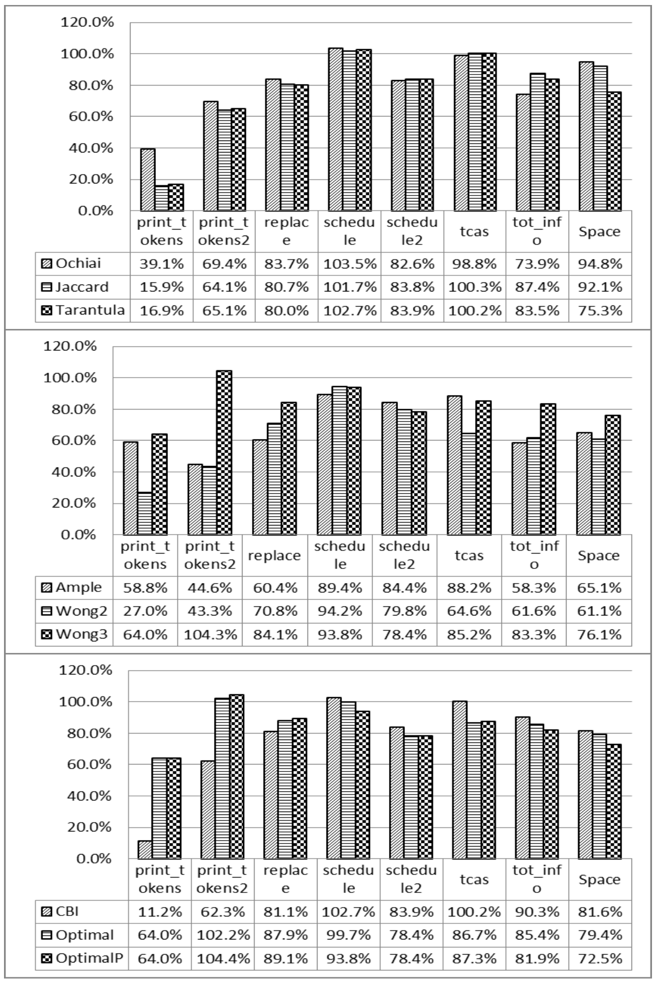


Figure 2. *Acc* comparison between SFL and our approach.

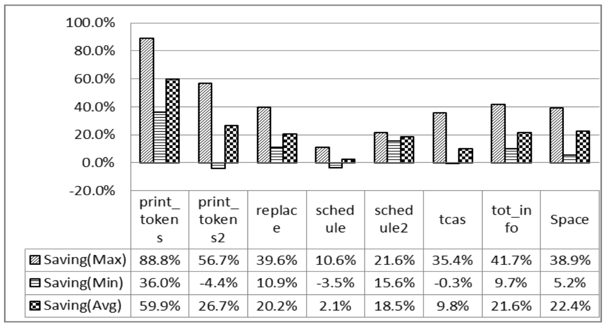


	print_tokens	print_tokens2	replace	schedule	schedule2	tcas	tot_info	Space
▨ Ochiai	39.1%	69.4%	83.7%	103.5%	82.6%	98.8%	73.9%	94.8%
⊟ Jaccard	15.9%	64.1%	80.7%	101.7%	83.8%	100.3%	87.4%	92.1%
⊠ Tarantula	16.9%	65.1%	80.0%	102.7%	83.9%	100.2%	83.5%	75.3%

	print_tokens	print_tokens2	replace	schedule	schedule2	tcas	tot_info	Space
▨ Ample	58.8%	44.6%	60.4%	89.4%	84.4%	88.2%	58.3%	65.1%
⊟ Wong2	27.0%	43.3%	70.8%	94.2%	79.8%	64.6%	61.6%	61.1%
⊠ Wong3	64.0%	104.3%	84.1%	93.8%	78.4%	85.2%	83.3%	76.1%

	print_tokens	print_tokens2	replace	schedule	schedule2	tcas	tot_info	Space
▨ CBI	11.2%	62.3%	81.1%	102.7%	83.9%	100.2%	90.3%	81.6%
⊟ Optimal	64.0%	102.2%	87.9%	99.7%	78.4%	86.7%	85.4%	79.4%
⊠ OptimalP	64.0%	104.4%	89.1%	93.8%	78.4%	87.3%	81.9%	72.5%

Figure 3. *Imp* of our approach over each metric of SFL on each program.



	print_tokens	print_tokens2	replace	schedule	schedule2	tcas	tot_info	Space
▨ Saving(Max)	88.8%	56.7%	39.6%	10.6%	21.6%	35.4%	41.7%	38.9%
⊟ Saving(Min)	36.0%	-4.4%	10.9%	-3.5%	15.6%	-0.3%	9.7%	5.2%
⊠ Saving(Avg)	59.9%	26.7%	20.2%	2.1%	18.5%	9.8%	21.6%	22.4%

Figure 4. The maximum, minimum and average saving of our approach over each program.

denotes the *Imp* in a specific metric of SFL. The tables in Fig. 3 show the detailed values of *Imp* on each program. If the value of *Imp* is less 100%, it means that our approach promotes the effectiveness of SFL. Otherwise it indicates our approach decreases the effectiveness of SFL.

As shown in Fig. 3, the values of *Imp* over each metric of SFL are less than 100% in most of programs. This indicates that the effectiveness of SFL is improved by our approach in most of programs. Take Ochiai as an example. The lowest *Imp* is 39.1% in *print_tokens*. This implies that our approach obtains the maximum improvement over Ochiai in *print_tokens*. It also means that when locating all faults in *print_tokens*, our approach only requires the examination of 39.1% of the number of statements that Ochiai requires the examination of. This represents a 60.9% saving in terms of effort, which is the maximum saving that our approach obtains in Ochiai. However, the highest *Imp* is 103.5% in *Schedule*, which implies that our approach requires an extra 3.5% effort to locate all faults in *Schedule* compared to Ochiai. This represents the minimum saving, -3.5%, that our approach obtains in Ochiai.

Fig. 4 illustrates the maximum, minimum and average saving of our approach in each program. As shown in Fig. 4, the average maximum saving that our approach obtains is 41.7% and the average minimum saving is 8.6%. On average, the saving of our approach is 22.7%, which indicates our approach is more effective than SFL.

Because the high suspicious statements evaluated by SFL are usually relevant to the faulty statements, our approach can classify the faulty statements into more suspicious contexts and they finally obtain higher ranks compared to those ranks in SFL. However, the high suspicious statements sometimes may be irrelevant to the faulty statements. Thus, it can cause some less suspicious statements that are relevant to the high suspicious statements surpass the faulty statements in more suspicious contexts. This reveals the reason why our approach slightly decreases the effectiveness of some metrics of SFL in several programs.

We also found that the decrease of effectiveness, to some extent, is caused by the vulnerability of SFL. For example, SFL utilizes coverage information that cannot identify those statements whose execution affects the program output. Suppose that a non-faulty statement that is irrelevant to the faulty statement and has little contribution to the faulty output. Suppose further that the number of failed test runs executing the non-faulty statement is larger than that of failed test runs covering the faulty statement, whereas the number of passed test runs covering the non-faulty statement is less than that of passed test runs executing the faulty statement. In this case, SFL usually assigns higher suspiciousness to the non-faulty statement than that to the faulty statement. If a context is constructed from this non-faulty statement, it is more probable for some less suspicious statements associated with this non-faulty statement to surpass the faulty statement in this more suspicious context. However, if SFL can identify those statements whose execution affects the program output, SFL can rank this non-faulty statement lower than the faulty statement. Under this circumstance, it reduces the possibility of decreasing the effectiveness of SFL when using our approach.

D. *Threats to Validity*

A threat to the validity of our experiment is the subject programs used by the study. The experiment chooses the *Siemens suite* and *Space* because they are two de-facto benchmarks in the field of software debugging. Apparently, the results obtained may not apply to all programs. For instance, a program, in reality, usually ships with multiple faults rather than a single fault as used in our experiment. The recent research [2] has found that multiple faults pose a negligible effect on the effectiveness of fault localization, and even in the presence of many faults, at least one fault is found by the fault localization technique with high effectiveness. Although these findings increase our confidence in the effectiveness of our approach for locating multiple faults, they cannot guarantee that multiple faults create a negligible effect on the effectiveness of our approach. It is necessary to use more subject programs to further investigate the effectiveness of our approach.

Another threat is the metrics of SFL adopted by our experiments. The experimental study selects nine metrics of SFL to empirically evaluate the effectiveness and applicability of our approach. However, SFL is a big family and contains many metrics [3-13]. Our approach may not be applicable to some other metrics of SFL. It is vital to apply our approach to a much broader spectrum of SFL to further evaluate its effectiveness and applicability.

V. Related Work

Spectrum-based fault localization (SFL) has motivated plenty of debugging techniques over recent years. The effectiveness of SFL highly depends on the ranking metrics that measure the correlations between program entities and failures. Hence, many metrics of SFL are proposed, such as the nine metrics of SFL adopted by the experiment [3-9]. In addition, there are many types of program entities presented for SFL, such as statements [5,6,9,10], blocks [3,7], branches [8], etc. Some new complex coverage types of program entities using dependences or flow are also proposed to strengthen the relationship among the elements of a program entity, such as mixed coverage [11], information flow coverage [12] and control flow edge coverage [13]. Although all of the above approaches have delivered the promising ability in correlating program entities and failures, they usually ignore the fact that the contextual information is useful for discovering and understanding the bugs. To enhance contexts for SFL, our approach applies program slicing to SFL to construct different suspicious contexts and their statements.

Program slicing technique [14-16] has also been widely studied in the field of debugging. Kusumoto et al. [17] conducted an experimental evaluation of program slicing for fault localization and Zhang et al. [18] investigate the effectiveness of dynamic slicing in locating faults. Their research shows program slicing is useful for fault localization. To further narrow down the searching scope, Gupta et al. [19] present failure inducing chops that intersect the forward dynamic slices of inputs with the backward dynamic slices of outputs. Zhang et al. [20] study the probable missing dependencies in dynamic slices and use an effective slicing approach to locate execution omission errors. Xin et al. [21] present a data-centric dynamic slicing technique that focuses on the dependencies in memory locations. Zhang et al. [22] propose an event-centric dynamic slicing technique that removes the irrelevant events from the sets of events to narrow down the searching scope of events. Although slicing-based debugging techniques have made great progress in these years, the size of a slice is still large. In addition, the elements of a slice are always treated with same suspiciousness to be faulty and no checking order is recommended to developers. Therefore, the slicing-based debugging techniques are rarely used in practice [2]. To alleviate this problem, our approach uses SFL to quantify the suspiciousness of a slice and its statements, and provides the guidance as to how the statements in a slice should be examined.

Baah et al. [23] uses the conditional probability in a dependence graph of a failed run to compute the suspiciousness of each node, and associate a state configuration with each node to construct a context and understand the problem. In contrast to their approach, our approach uses program slicing to iteratively construct different suspicious contexts and their statements according to the location results given by SFL.

Jiang et al. [24] proposes a context-aware statistical debugging approach by constructing and ranking the control-flow paths. The control-flow path is a context showing how a faulty predicate behave in a program. HOLMES [25] statistically analyzes path profiles of both passed runs and failed runs to isolate bugs that correlate with failure, and also uses paths to show a context where bugs occur. Unlike these two approaches, our approach uses program slicing constructs contexts and their elements, and utilizes SFL to assign suspiciousness to them. A context in our approach is essentially a slice showing how the most suspicious statement affects and is affected by other statements.

VI. CONCLUSION

This paper proposes a debugging approach to enhance contexts for a promising automated debugging technique, namely spectrum-based fault localization (SFL). The proposed approach applies program slicing to SFL by constructing different suspicious contexts and their elements for assist in understanding and locating faults. In addition, our approach offers two modes to different experienced developers, and uses the visualization and program dependence to further help understand the problem. The experimental study on two standard benchmarks shows that the proposed approach outperforms all nine metrics of SFL.

In future work, we plan to evaluate the effectiveness of our approach across a much broader spectrum of programs. We will also further study the applicability of our approach to more metrics of SFL and other automated debugging techniques.

ACKNOWLEDGMENT

This work is partially supported by the National Natural Science Foundation of China under Grant No.91118007, 90818024 and 61133001, the National High Technology Research and Development Program of China (863 program) under Grant No.2011AA010106 and 2012AA011201 and the Program for New Century Excellent Talents in University.

REFERENCES

[1] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?," in *the 2011 International Symposium on Software Testing and Analysis*, Toronto, Canada, 2011, pp. 199-209.

[2] N. DiGiuseppe and J. Jones, "On the influence of multiple faults on coverage-based fault localization," in *the 2011 International Symposium on Software Testing and Analysis*, Toronto, Canada, 2011.

[3] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "On the accuracy of spectrum-based fault localization," in *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, Windsor, UK, 2007, pp. 89-98.

[4] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic internet services," in *the 32nd International Conference on Dependable Systems and Networks*, Maryland, USA, 2002, pp. 595-604.

[5] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *the 24th International Conference on Software Engineering*, Orlando, USA, 2002, pp. 467-477.

[6] W. E. Wong, Y. Qi, L. Zhao, and K. Y. Cai, "Effective fault localization using code coverage," in *the 31st Annual International Computer Software and Applications Conference*, Beijing, China, 2007, pp. 449-456.

[7] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "An evaluation of similarity coefficients for software fault localization," in *the 12th Pacific Rim International Symposium on Dependable Computing*, Riverside, USA, 2006, pp. 39-46.

[8] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, "Scalable statistical bug isolation," in *the ACM SIGPLAN Conference on Programming Language Design and Implementation*, NY, USA, 2005, pp. 15-26.

[9] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Transactions on Software Engineering and Methodology,* vol. 20, p. 11, 2011.

[10] V. Debroy, W. E. Wong, X. Xu, and B. Choi, "A Grouping-Based Strategy to Improve the Effectiveness of Fault Localization Techniques," in *the 10th International Conference on Quality Software*, Zhangjiajie, China, 2010, pp. 13-22.

[11] R. Santelices and J. A. Jones, "Lightweight fault-localization using multiple coverage types," in *the 31st International Conference on Software Engineering*, Vancouver, Canada, 2009, pp. 56-66.

[12] W. Masri, "Fault localization based on information flow coverage," *Software Testing, Verification and Reliability,* vol. 20, pp. 121-147, 2010.

[13] Z. Zhang, W. Chan, T. Tse, B. Jiang, and X. Wang, "Capturing propagation of infected program states," in *the ESEC/FSE 2009,* Amsterdam, The Netherlands, 2009, pp. 43-52.

[14] M. Weiser, "Program slicing," *IEEE Transactions on Software Engineering*, vol. 10, pp. 352-357, 1984.

[15] B. Korel and J. Laski, "Dynamic Program Slicing," *Information Processing Letters,* vol. 29, pp. 155-163, 1988.

[16] T. Gyimóthy, Á. Beszédes, and I. Forgács, "An efficient relevant slicing method for debugging," in *the ESEC/FSE 1999*, Toulouse, France, 1999, pp. 303-321.

[17] S. Kusumoto, A. Nishimatsu, K. Nishie, and K. Inoue, "Experimental evaluation of program slicing for fault localization," *Empirical Software Engineering*, vol. 7, pp. 49-76, 2002.

[18] X. Zhang, N. Gupta, and R. Gupta, "A study of effectiveness of dynamic slicing in locating real faults," *Empirical Software Engineering,* vol. 12, pp. 143-160, 2007.

[19] N. Gupta, H. He, X. Zhang, and R. Gupta, "Locating faulty code using failure-inducing chops," in *the 20th International Conference on Automated Software Engineering*, Long Beach, USA, 2005, pp. 263-272.

[20] X. Zhang, S. Tallam, N. Gupta, and R. Gupta, "Towards locating execution omission errors," *ACM Sigplan Notices,* vol. 42, pp. 415-424, 2007.

[21] B. Xin and X. Zhang, "Memory slicing," in *the 18th International Symposium on Software Testing and Analysis*, Chicago, USA, 2009, pp. 165-176.

[22] X. Zhang, S. Tallam, and R. Gupta, "Dynamic slicing long running programs through execution fast forwarding," in *the 14th International Symposium on Foundations of Software Engineering*, Portland, USA, 2006, pp. 81-91.

[23] G. K. Baah, A. Podgurski, and M. J. Harrold, "The probabilistic program dependence graph and its application to fault diagnosis," *IEEE Transactions on Software Engineering,* vol. 36, pp. 528-545, 2009.

[24] L. Jiang and Z. Su, "Context-aware statistical debugging: from bug predictors to faulty control flow paths," in *the 22nd International Conference on Automated Software Engineering*, Atlanta, Georgia, 2007, pp. 184-193.

[25] T. M. Chilimbi, B. Liblit, K. Mehra, A. V. Nori, and K. Vaswani, "HOLMES: Effective statistical debugging via efficient path profiling," in *the 31st International Conference on Software Engineering*, Vancouver, Canada, 2009, pp. 34-44.

[26] W. Dong, J. Wang, C. Zhao, X. Zhang, and J. Tian, "Automating software FMEA via formal analysis of dependence relations," in *the 32nd Annual International Computer Software and Applications Conference*, Turku, Finland, 2008, pp. 490-491.

[27] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems*, vol. 9, pp. 319-349, 1987.

[28] SIR, http://sir.unl.edu/portal/index.php

Leveraging Traceability between Code and Tasks for Code Review and Release Management

Nitesh Narayan, Jan Finis, Yang Li
Institute of Computer Science
Technical University of Munich
Boltzmannstrasse 3, 85748 Garching, Germany
{narayan, finis, liya}@in.tum.de

Alexander Delater
Institute of Computer Science
University of Heidelberg
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
delater@informatik.uni-heidelberg.de

***Abstract*—The software maintenance process relies on traceability information captured throughout the development of a software product. Traceability from code to software engineering artifacts like features or requirements has been extensively researched. In this paper, we focus on traceability links between code and tasks. Tasks can be further linked to other artifacts such as features or requirements. In this paper, we present an approach for (semi-) automatic creation of traceability links between code and tasks. The core idea is to let the developers create the links themselves while they use a version control system. We use these traceability links to improve the processes of code review and release management. A prototype based on this work has been implemented and integrated into the model-based CASE tool UNICASE. We applied the developed prototype in the open-source project UNICASE itself and report about our significant experiences.**

***Keywords*-traceability; code review; release management; patch; branch.**

I. Introduction

Software configuration management (SCM) is the discipline of managing the evolution of large and complex software systems to assist software development and maintenance processes [1]. According to the IEEE standard [2], SCM covers several activities such as identification of product components and their versions, audit, review, as well as change control (by establishing procedures to be followed when performing a change). Practicing SCM in a software project has several benefits, including increased productivity, better project control, identification and fixes of bugs, and improved customer satisfaction [3]. Especially in projects with increased complexity, efficient handling of SCM requires tool support. Standard SCM tools exist for various activities e.g. version control systems (VCS). However, other SCM activities still lack proper tool support because of the involved traceability challenges, especially the review of changes during a code review and the building of a software product during release management. In this paper, we present an approach for (semi-) automatic creation of traceability links between code and tasks to improve the processes of code review and release management by providing tool support. Tasks represent a unit of work, which describe changes to be performed to the code or new developments and they are used in many software development projects. In the remainder of the paper, we use the term *work item* instead of task to avoid misunderstandings with the term task used in requirements engineering.

The paper is structured as follows: in Section II, we provide background information. In Section III, we describe the processes of code review and release management and benefits from using traceability links between code and work items. The approach is presented in Section IV and the prototype implementation is shown in Section V. In Section VI, we describe our experiences in using the prototype in the open-source project UNICASE. Related work is presented in Section VII and a discussion and future work conclude the paper in Section VIII and Section IX, respectively.

II. Background

Traceability links between requirements and work items have previously been researched in the MUSE model (Management-based Unified Software Engineering) [4], which integrates the system model and the project model. The system model describes the system under construction, such as requirements, features, use cases or UML artifacts. The project model describes the on-going project, such as work items, the organizational structure, sprints or meetings. The MUSE model is implemented in the model-based CASE tool UNICASE [5] [6], which we use to implement our approach.

The presented approach in this paper is dealing with the (semi-) automatic creation of traceability links between work items and code. Previous studies have shown that links between system elements and project elements provide useful information for the work (by shortening the navigation paths of the developers) and that based on such links system elements are kept more up-to-date [7]. Thus, we are extending the MUSE model by introducing traceability to code. Various development activities can benefit from the traceability links between requirements, work items and code. In this work, we concentrate on code review and release management activities.

III. Code Review and Release Management

This section explains the processes of code review and release management. We discuss how these two processes could be improved by using traceability links between requirements, work items and code. Furthermore, we specify requirements for an approach improving these two processes.

A. *Code Review*

Code review is a process where a team member reviews code written by another member to ensure quality and consistency, as well as to share knowledge in a software development project. It assists in improving the quality by identifying defects at an early stage. However, code review by its nature is a labour-intensive process. This is further affected by the lack of effective tool support. The problem is not overwhelming tool complexity, as the goals of such a tool are rather simple. Goals of such a tool are:

- Enable the reviewer to quickly transfer the changes to be reviewed to his local workspace as well as highlight the changes so that s/he can review the interplay of the changes with the entire code base.
- After the code review, it should allow the reviewer to add a review summary to the associated work item.

A specific question during requirements validation is to ensure that the implementation of every requirement or feature is reviewed. Thus, one needs to be able to associate changes in the code to its corresponding work item in the project management documents. Therefore, work items themselves are associated to a requirement or feature as well as to the changes in the code. This requires extended tool support to aid the code review process.

B. *Release Management*

Another important activity in SCM is the release management process. This process involves deciding on which configuration a product is released and which features it includes. During the release management it needs to be verified that the code, from which the release is actually built, includes all features or requirements the release should embody.

Most of the existing literature fails to highlight this activity and its importance in release management. Van der Hoek et al. [8] even describe the release management as a “poorly understood and underdeveloped part of the software process”. Instead, it is generally assumed that a configuration of code already exists and that it contains all required content. Thus, two important aspects are:

- Is the implementation of each feature or requirement, which are part of this release, included in the code? Is the implementation not finished, or finished but only stored in the local workspace of the implementor?
- Are there any other changes in the code e.g. undocumented bug fixes or accidentally introduced changes?

If it can be validated which features or requirements are included in the code of a release, the release management process can further benefit from extended functionalities provided by the tool. For example, assembling the code for the release autonomously. By specifying the base version of code and a set of features or requirements to be included into the release, the system should be able to merge the implementation of all features or requirements into the code, ignoring already included features or requirements.

Other activities can also benefit from tracking the changes in the release. During release management, there is a need to capture a list of features and included bug fixes in the form of a change log for every release. This log is usually shipped with the product and/or published to inform users about changes. However, the change log could be assembled automatically as it is possible to identify the list of features or bug fixes implemented in the code.

C. *Leveraging Traceability between Code and Work Items*

For the purpose of improving the processes of code review and release management, we capture all changes made to the code in a so-called *change package*. The change package is *created* by accumulating the changes a developer performs in the code within the context of a certain work item. Hence, every change package is associated with only one work item.

During a code review, the reviewer needs to quickly transfer all changes to be reviewed to his/her local workspace. Suppose the reviewer has to review some changes in the code. For this task, s/he gets assigned a work item to be reviewed. The original work item was linked by a developer to a change package containing all changes that s/he performed to the code. So, while viewing the work item, the reviewer could quickly *apply* all the changes in the change package to his/her local workspace. After applying the changes, the reviewer could start immediately with the code review. The code review process would be improved due to automatic transfer of all changes to the local workspace and reduced setup time. Previously, these tasks had to be performed manually.

For the release management process, the base version of code and a set of features or requirements to be included into the release need to be specified. Using our approach, it is possible to merge all the change packages for every work item associated with the selected features or requirements over the base version of the code, ignoring already included change packages. Furthermore, it needs to be *validated* whether a change package is already included in the given code. Additionally, one could assemble the change log for the release automatically, as it is possible to identify the set of features, requirements or bug fixes implemented in the code. The release management process would benefit from automated assembling of code and change log creation. Previously, these tasks had to be performed manually, as well.

D. Requirements

Based on the ideas above on how to improve the processes of code review and release management, we have identified the following three requirements for a change package:

1) *Change package creation:* A change package must be producible from the current changes the developer has in his/her workspace.
2) *Change package application:* A change package must be applicable to a given set of code files.
3) *Change package validation:* It must be possible to validate if the changes in a package are already applied to a given code configuration.

In the following section, we discuss how to satisfy these three requirements to improve the processes of code review and release management.

IV. Approach

This work proposes an approach to (semi-) automatically establish traceability links between code and work items. Our approach links the exact changes which were made in the code to a work item. We assume that developers only work on one work item at a time and do not switch between different work items. The *core idea* for (semi-) automatically creating traceability links is letting the developer build the links himself/herself while using a VCS within the project. Whenever s/he finishes the implementation of a work item, the developer does not immediately commit the changes to the repository. Instead, before the commit, s/he orders the system to create a *change package* containing all changes in the code and associate it to a work item (see Figure 1).

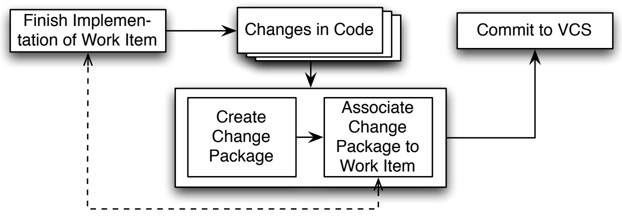

Finish Implementation of Work Item
Changes in Code
Commit to VCS
Create Change Package
Associate Change Package to Work Item

Figure 1. Process of creating Change Packages

The following subsections describe the proposed approach in detail with regard to two different ways of change package representation: a patch and a branch.

A. Patches as Change Packages

A patch is a file containing a set of changes between two versions of the code. The changes are stored in a specified format (e.g., unified diff) which allows to apply the changes contained in them to files (e.g., code), thus reproducing the patched version. Patches can be created and applied by almost all VCSs and even without a VCS, common programs like `diff` and `patch` under Unix can be used to create patches. The following mechanisms are used to fulfill the requirements for a change package:

1) *Change package creation:* A change package is created by creating the patch file.
2) *Change package application:* A change package is applied by applying the patch.
3) *Change package validation:* This is where patches reach their limits: It is rather difficult to check if a patch is already included in a given code. If the code was not changed afterwards, a simple check for the changes in the patch file can yield a result. However, if the code was changed afterwards (which is the more common case), comparing the content and the changes in the patch will not yield a result. Thus, relying only on a comparison of the patch content with the current file content is not suitable.

While the first two requirements are straightforward, the last requirement is challenging. There exist several possible approaches to implement the last requirement. For example, one is keeping a list of patches applied onto the code and linking this list with the version history in the repository. The problem with this approach is that it only works if all patches are applied using the system which tracks their application. If a patch is applied using common tools like the `patch` Unix command or the commands provided by the used VCS, this patch will be un-tracked.

A prototype for the patch representation of change packages was implemented. It is based on Subversion. Because it only uses the functionality of patch creation, it can also be adapted to other VCSs. The prototype currently does not support the check whether a patch is contained in the current version of the code. Therefore, the *patch-based approach can only support the code review process, but not the release management process*.

B. Lightweight Branches as Change Packages

VCSs ensure that no changes, besides the ones introduced by commits (and reliably logged), are done to the repository. Thus, a good approach for providing change package validation is to make them reside directly in the repository. By tying the representation closer to the repository, more of its tracking features can be leveraged.

The obvious choice for a change package representation which resides in the repository is a branch. A branch represents changes done to the code since the revision from which the branch started to diverge. The three required properties could be implemented for branches as follows:

1) *Change package creation:* A branch is created and changes are committed to this branch.
2) *Change package application:* A branch can be applied onto another branch of the repository by merging it into the other one.
3) *Change package validation:* By checking the revision graph, it can be deducted whether a branch has been

merged into another one (see details below). However, the repository has to support a revision graph to allow this approach.

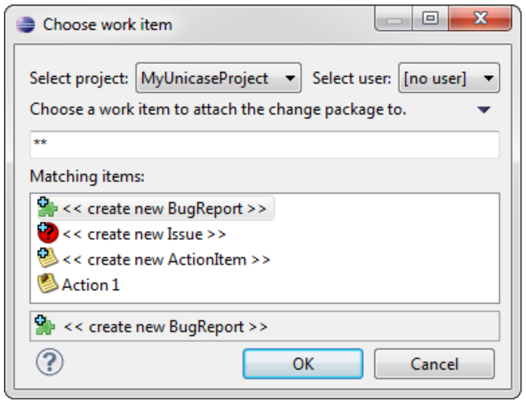
The basic idea of checking if a change package is already merged into a given branch is using the revision graph and performing a backward search (i.e., a search from a revision into the direction of its predecessor revisions). The search starts from the head revision of the given base branch which is to be checked for included change packages. If a revision identifying a change package branch (hereinafter called *indicator revision*) is found, a positive answer is given. If the search does not yield the indicator revision of a change package, a negative answer for this package is given. The representation of the indicator revision depends on the VCS. If the branch head of a merged-in branch is kept, this branch head can be used as indicator revision and stored in the change package. Otherwise, for example, the first commit on the branch can be used.

A drawback of this approach is that it restricts the VCS to be used. First, the VCS must support branches. This is no big restriction as most modern VCS do support this feature. The next limitation is more severe: The VCS must support a revision graph which reveals all predecessors of a revision which was created by merging. Subversion, for example, is not able to deliver this information and is therefore unsuitable for this approach. Finally, the branches must be *lightweight*: Since one change package is represented by one branch, numerous branches will exist concurrently in the repository. A branch is considered lightweight, if a large number of branches can be created without reducing the performance of the system and without taking too much space in the VCS. Additionally, the creation and merging of branches should be fast and the merge algorithms used should be sophisticated. It must be able to resolve most conflicts automatically, because this is cumbersome and error-prone.

One system that actively advertises its ability to maintain a large number of lightweight branches is Git. It also incorporates the use of sophisticated merge algorithms which reduce the amounts of conflicts propagated to the user. Thus, Git was chosen for the prototype implementation of the lightweight branch representation of change packages. In contrast to the patch-based prototype, the branch-based one is able to support all required three requirements. Thus, the *branch-based approach can support both processes of code review and release management*.

V. Prototype

A prototype of the presented approach has been implemented and integrated into UNICASE. After performing changes to the code, the developer selects a work item for the created change package (see Figure 2). For the code review, a reviewer gets assigned a work item to be reviewed. S/he can easily apply the linked change packages to his/her local


Choose work item

Select project: MyUnicaseProject Select user: [no user]

Choose a work item to attach the change package to.

**

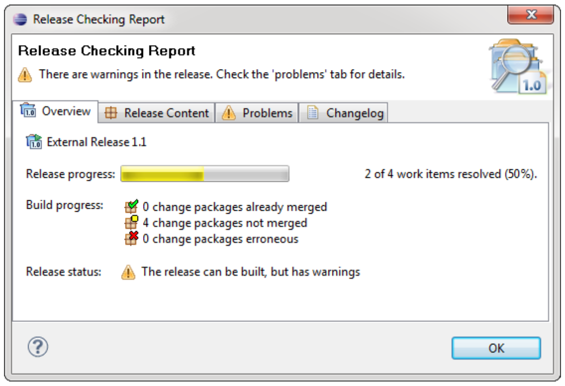
Matching items:

<< create new BugReport >>
<< create new Issue >>
<< create new ActionItem >>
Action 1

<< create new BugReport >>

Figure 2. Choosing a Work Item to associate to a Change Package

workspace. This process is eased in UNICASE by providing user-specific change notifications [9]. Once the developer finishes his/her work, the reviewer gets notified whether s/he can start with the review process. After the review, s/he can add a review summary to this work item and share it with the developer.


Release Checking Report

Release Checking Report

There are warnings in the release. Check the 'problems' tab for details.

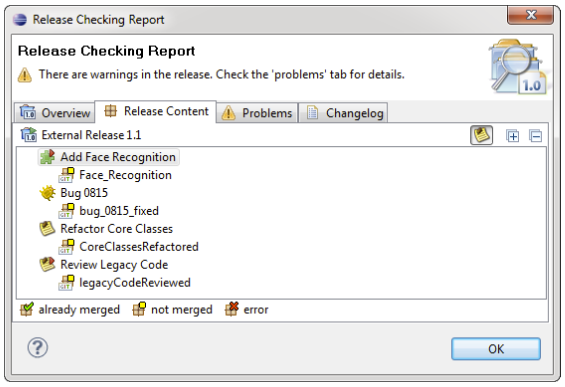
External Release 1.1

Release progress: 2 of 4 work items resolved (50%).

Build progress: 0 change packages already merged
4 change packages not merged
0 change packages erroneous

Release status: The release can be built, but has warnings

Figure 3. Release Management Support in Prototype: Overview


Release Checking Report

Release Checking Report

There are warnings in the release. Check the 'problems' tab for details.

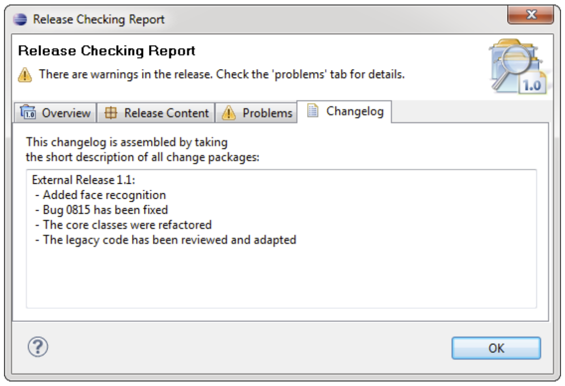
External Release 1.1

Add Face Recognition
Face_Recognition
Bug 0815
bug_0815_fixed
Refactor Core Classes
CoreClassesRefactored
Review Legacy Code
legacyCodeReviewed

already merged not merged error

Figure 4. Release Management in Prototype: Release Content

In Figures 3-5, an insight into the prototype implementation in UNICASE for the support of the release management



There are warnings in the release. Check the 'problems' tab for details.

This changelog is assembled by taking
the short description of all change packages:

External Release 1.1:
- Added face recognition
- Bug 0815 has been fixed
- The core classes were refactored
- The legacy code has been reviewed and adapted

Figure 5. Release Management Support in Prototype: Changelog

is provided. The tool is able to show the progress of the build process as well as all already merged, not merged or erroneous change packages (see Figure 3). The release content tab shows all included work items in this particular release and their linked change packages (see Figure 4). The change log is automatically created (see Figure 5).

VI. Application of Prototype

We used the prototype during one sprint (4 weeks) in the development of the UNICASE project. In the following, we report on the processes of code review and release management with and without using the prototype as well as our experiences.

A. Code Review

The code review process *without* using the prototype was as follows: a developer got assigned a work item describing his/her work for implementation. After implementation, the developer created manually a patch file containing all changes. Then s/he send the patch via e-mail to a reviewer for code review. The reviewer downloaded the patch and applied it manually to his/her local workspace. Afterwards, the reviewer had to find the work item belonging to the patch. After the reviewer had reviewed all changes and agreed to all performed changes, s/he committed the changes to the VCS. Finally, the reviewer had to write a review comment to the work item indicating that the changes have been reviewed and applied.

The code review process *with* using the prototype was as follows: a developer got assigned a work item describing his/her work for implementation. After implementation, the developer selected the assigned work item (see Figure 2). If no work item existed for the performed changes, e.g., for a hot fix, the developer just created a work item on demand. After that, the changes were committed to the VCS in a separate branch. A change package was created containing a link to a branch and it was linked to the selected work item. The developer marked the work item as *done* and assigned a reviewer to it. The reviewer was automatically notified that new code changes were waiting for review using user-specific change notifications [9] in UNICASE. S/he opened the work item and selected the option to automatically download and apply the linked change package. All changes were automatically fetched/pulled from the branch in the VCS and applied to the local workspace of the reviewer. After reviewing all changes, the reviewer committed the changes to the main trunk of the VCS. Finally, the reviewer had to write a review comment to the work item indicating that the changes have been reviewed and applied.

B. Release Management

The release management process *without* the prototype was as follows: A release manager went through the list of all the work items and their linked features/requirements that the release should embody. Subsequently, the release manager went through each work item based on their priority and verified if their is a corresponding commit from the developer in the VCS assigned to the work item (switching back and forth between two different tools). Often, s/he also noticed that the patches were not applied to the branch from where the new release has to be made but rather at other place. In this scenario, s/he would have created a diff and applied it to the release branch that was checked out in his/her local workspace. If s/he was unsure whether a commit belongs to a work item, s/he first tried to contact and confirm with the assignee of the work item. Once the branch is ready by including all the code changes, the release manager committed all the local changes to the VCS and moved on to do the release manually or using a build server.

The release management process *with* the prototype was as follows: A release manager created a release model and added all the features/requirements the release should embody. Once s/he was done with a VCS checkout of the latest version for the release branch, s/he selected the “Build Release” option provided in the prototype. The prototype applied all change packages linked with every work item that are related to the features/requirements included in the release model automatically. Next, the prototype presented a release checking report to the release manager that showed a summary of the release process with various information, e.g. whether all the work items are resolved (Figure 3), all change packages merged to the release branch (Figure 4) and the automatically created change log from the work item descriptions (Figure 5). Once the release manager selected to do the release, all the change packages were merged and committed to the VCS repository.

C. Experiences

We noticed several advantages while using the prototype. First, the developers were able to automatically assemble all code belonging to the sprint and its planned work items and check if code could be assembled successfully. Second, if problems occurred during the release, these problems were reported. An overview about already merged, not merged or erroneous change packages helped the developers looking into the specific change packages. Third, the change log was automatically assembled. We learned that the presented approach significantly increased the productivity of our development team, e.g. the release management process using the prototype is now up to 4 times faster than before (30 minutes with and 2 hours without prototype).

VII. Related Work

In the following, related work for code traceability, code review and release management is discussed.

A. Code Traceability

Maintaining traceability links between code and other artifacts is a challenging task and therefore a field of intense research. Most approaches relate structures in the code like classes, methods, modules, files, or lines of code to other artifacts like requirements or features. Our approach tracks changes instead of structures in the code, instead.

A very simple, yet effective approach is presented by Treude et al. [10] embedding the links directly into code comments, which can be read by their proposed tool TagSEA. They use tags in comments to refer to other artifacts. They go further into the direction by creating links between tasks (equal to our definition of work item) and code. This is accomplished by connecting their tool with MyLyn [11], which can be used to express tasks. They connect code to MyLyn tasks by using special tags. A major drawback of using tags in the source code is the overhead of keeping the tags updated. In a complex project with a large number of artifacts it gets difficult for a developer to recall which code files should be connected to which work item. In our approach, we use the abstraction of patches over individual code artifacts to overcome this issue. Fischer et al. [12] link VCS with release data, which is in contrast to our approach. They use the version history in the repository, in addition to bug tracking data, to automatically build the release history of a project and allow viewing and querying it to retrieve information about the evolution of the project. While they do post-mortem analysis, our approach is focused on creating links between code and tasks during development to benefit code review and release management.

Marcus et al. [13] establish links between code and the corresponding documentation. In contrast to the approaches mentioned above, they try to recover the links automatically using information retrieval, namely Latent Semantic Indexing (LSI). This is different to our approach, as we establish links between code and work items (semi-) automatically while the developers use a VCS. Qusef et al. [14] use their SCOTCH tool for dynamic slicing and conceptual coupling to recover traceability links between unit tests and tested classes. Antoniol et al. [15] link code locations to object-oriented design artifacts. Like Marcus et al., they establish all the links without additional information, by performing different static and dynamic analyses. They use, amongst others, vector space indexing and probabilistic indexing techniques (comparable to LSI). All these methods are based on the textual similarity of artifact content, code identifiers and comments. This results in unreliable traceability links, as the created traceability links cannot ensure a high precision as well as a high recall at the same time.

B. Code Review

Several researchers contributed to the tool support of code review. Brothers et al. [16] proposed the ICICLE tool, which embodies different functionalities aiding code reviews. Examples are a human interface for preparing comments on the code under inspection and hypertext-based browsers for referring to various kinds of knowledge associated with code inspection, thus achieving a certain degree of traceability. Harjumaa et al. [17] proposed a web based tool, which features the distribution of the document to be inspected, annotation of it, searching of related documents, a checklist, and inspection statistics. Belli et al. [18] described an approach for the automatic handling, checking, and updating of check lists used in reviews. A similar approach of improving artifact quality by distributed artifact inspection was presented by De Lucia et al. [19]. They focused on general aspects of the artifacts life-cycle and presented a distributed inspection process consisting of seven phases implemented in a tool called WAIT.

All these approaches are in contrast to ours, as we are able to transfer the changes to be reviewed automatically as well as providing traceability from the code over work items to other artifacts, e.g., features or requirements. There are further tools for code review which can do a lot of other things that our tool does not support. However, these tools do not support the functionality discussed here.

C. Release Management

Among numerous research work aiming at improving the release management process, we identified two major contributions related within the scope of our work. Van der Hoek et al. [8] identified the basic issues in release management and developed a tool to aid the release management process. In contrast to our approach, their approach does not include assembling and building components to be released. It is merely a database of releases, components to be released and their dependencies among each other. Saliu et al. [20] contributed research in the field of release planning, especially for evolving systems which are built incrementally. Our approach supports release planning as well, since work items linked to changes in the code are included in a release.

VIII. Discussion

The proposed approach works with certain constraints and assumptions. For example, it is also possible that a change belonging to a work item has been committed to the repository without the creation of a change package. This can happen if the code is committed with the sources of another change package, which happens if a developer was working on two work items and committed the changes for both in one change package only. However, this can be noticed by the developer if s/he realizes that one of the work items has a missing change package. The approach leaves the task of creating links between the code changes and the work items to the developer himself/herself. So, the approach suffers from mistakes a developer does while performing this activity (like linking code to the wrong work item). The general problem is that artifacts in human readable text or code can not be linked together with full reliability using any existing technique. Our approach of letting the developers create the links themselves as part of their usual work is expected to be better in comparison to automatic approaches.

IX. Conclusion

This paper proposed an approach for (semi-) automatically creating traceability links between code and work items by using VCSs. The created traceability links were used to improve the processes of code review and release management. The idea to automate the code review process was to use traceability links between code and work items to apply the code to the reviewers machine, highlight the changes and add the reviewers feedback to the work items. For the release management, the traceability links between code and work items were used to check which work items are contained in the code of a release and to build a release automatically by merging in the missing features or requirements.

The techniques and applications discussed in this paper, like the establishment of links between code and work items, are still not explored in depth. Therefore, there are a lot of possibilities for future research in this area. We are aware that our approach currently only allows developers to work at one work item at a time. Therefore, support for working on several work items at once is subject to future work. Furthermore, we want to study further possibilities for the application of our proposed approach. The work presented in this paper has been evaluated so far only with a small user group during one sprint in the UNICASE project. Thus, we plan to conduct a representative user experiment. A study where the prototype is used throughout a development project would be very beneficial to evaluate the benefits offered by the proposed approach to the code review and release management processes.

References

[1] Tichy, W.F. Tools for software configuration management. In Proceedings of the International Workshop on Software Version and Configuration Control, pp. 1-20 (1988)

[2] IEEE. IEEE standard for software configuration management plans: ANSI/IEEE std. 828-1983 (1983)

[3] Leon, A. Software configuration management handbook. Artech House, Inc. Norwood, MA, USA (2004)

[4] Helming, J., Koegel, M., and Naughton, H. Towards traceability from project management to system models. In TEFSE 09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering, pp. 11-15, IEEE Computer Society (2009)

[5] Bruegge, B., Creighton, O., Helming, J., and Koegel, M. Unicase - an Ecosystem for Unified Software, In ICGSE 08: Distributed software development: methods and tools for risk management, pp. 12-17 (2008)

[6] UNICASE Open Source Project. http://www.unicase.org/

[7] Helming, J., David, J., Koegel, M., and Naughton, H. Integrating System Modeling with Project Management - A Case Study. In COMPSAC 09: International Computer Software and Applications Conference, pp. 571-578 (2009)

[8] Van der Hoek, A., Hall, R., Heimbigner, D., and Wolf, A. Software release management. Software Engineering - ESEC/FSE, pp. 159-175 (1997)

[9] Helming, J., Koegel, M., Naughton, H., David, J., and Shterev, A. Traceability-Based Change Awareness. In MODELS 09: International Conference on Model Driven Engineering Languages and Systems, pp. 372-376 (2009)

[10] Treude C. and M.A. Storey. How tagging helps bridge the gap between social and technical aspects in software development. In ICSE 09: International Conference on Software Engineering, pp. 12-22 (2009)

[11] Eclipse MyLyn. http://www.eclipse.org/mylyn/ [retrieved: September, 2012]

[12] Fischer, M., Pinzger, M., and Gall, H. Populating a Release History Database from version control and bug tracking systems. In ICSM 03: International Conference on Software Maintenance, pp. 23-32 (2003)

[13] Marcus, A., Maletic, J.I., and Sergeyev, A. Recovery of traceability links between software documentation and source code. International Journal of Software Engineering and Knowledge Engineering, vol. 15, no. 5, pp. 811-836 (2005)

[14] Qusef, A., Bavota, G., Oliveto, R., De Lucia, A., and Binkley, D. SCOTCH: Slicing and Coupling Based Test to Code Trace Hunter. In 18th Working Conference on Reverse Engineering (WCRE), pp. 443-444 (2011)

[15] Antoniol, G., Canfora, G., Casazza, G., and De Lucia, A. Maintaining traceability links during object-oriented software evolution. Software: Practice and Experience, vol. 31, no. 4, pp. 331-355 (2001)

[16] Brothers, L.R. Multimedia groupware for code inspection. International Conference on Discovering a New World of Communications (ICC), pp. 1076-1081 (1992)

[17] Harjumaa, L. and Tervonen, I. A WWW-based tool for software inspection. In HICSS, Published by the IEEE Computer Society, pp. 379-388 (1998)

[18] Belli, F. and Crisan, R. Towards automation of checklist-based code-reviews. In ISSRE 96: International Symposium on Software Reliability Engineering, IEEE Computer Society, pp. 24-33 (1996)

[19] De Lucia, A., Fasano, F., Scanniello, G., and Tortora, G. Improving artefact quality management in advanced artefact management system with distributed inspection. Software, IET, vol. 5, no. 6, pp. 510-527 (2011)

[20] Saliu, O. and Ruhe, G. Supporting software release planning decisions for evolving systems. 29th Annual IEEE/NASA Software Engineering Workshop, pp. 14-26 (2005)

A Multiple View Environment for Collaborative Software Comprehension

Glauco de F. Carneiro
Computer Science Department
Salvador University (UNIFACS)
Salvador, Bahia, Brazil
glauco.carneiro@unifacs.br

Carlos F. R. Conceição
Computer Science Department
Salvador University (UNIFACS)
Salvador, Bahia, Brazil
carlos.conceicao@unifacs.br

José Maria N. David
Computer Science Department
Federal University of Juiz de Fora
Juiz de Fora, Minas Gerais, Brazil
jose.david@ufjf.edu.br

***Abstract*—Collaboration is an important issue for software comprehension activities which are performed in distributed development environments. Several studies have pointed to the relevance of visualization to provide support to these activities. Enriching visual metaphors with awareness elements can enhance collaboration in such environments. This paper presents a multiple view interactive environment to support collaborative software comprehension. A case study was carried out to analyze the effectiveness of the proposed environment considering that awareness elements are visually represented to support the collaborative software comprehension.**

Keywords- collaboration; software comprehension; software visualization; distributed development environments.

I. Introduction

Humans rely more on vision than all the other senses [25]. For this reason, the use of visual resources is relevant for software engineering. Software visualization uses perceptible cues to visually represent several software systems properties. The goal is to unveil patterns and structures that otherwise would remain hidden during software comprehension activities [24]. Software comprehension in distributed development environments requires collaboration support. Awareness plays an important role in software comprehension activities in a collaborative environment since it supports programmers to find meaningful information to their activities [11]. Supporting awareness in a distributed environment enables, for example, the identification of who is working in the project, what participants are doing, why they are doing, which artifacts they are manipulating and how their actions might impact others [16].

Visual resources have been used to support programmers to perform their activities in distributed development environments [1][2]. However, there are still some open questions in this area, specially related to awareness. In this paper we focus on two of these questions. The first is related to the inclusion of visual representation of awareness elements in integrated development environments (IDEs) in order to increase the effectiveness of software comprehension. The goal is to provide programmers with information related to what has been done in the context of a given project. The second question is related to the use of multiple view interactive environments as a mean to enhance software comprehension. The goal is to support awareness due to the use of three important concepts used in the information visualization domain: i) navigational slaving – multiple views systems should enable actions in one view to be automatically propagated to the others [22]; linking – multiple views systems should connect data in one view with data in the other views [17]; brushing – multiple views should enable corresponding data items in different views to be simultaneously highlighted [17].

A view is a particular visual representation of a data set. Complex data sets typically require multiple views, each revealing a different aspect of the data [19]. Multiple view systems have been proposed to support the investigation of a wide range of information visualization topics [20]. SourceMiner [3][28] is a multiple view interactive environment (MVIE) from which the collaborative environment was developed and now is described in this paper. It was implemented as an Eclipse IDE plug-in to interactively visualize Java projects, complementing the native views and resources provided by the IDE. It uses code as its main data source and provides a set of features to support programmers to configure the visual scenario that best fit a software comprehension goal. Examples of features to interact with the views are: (i) filters to visually present information that match filtering criteria; (ii) semantic and geometric zooming to better adjust views to the canvas; (iii) flexibility to arrange views in accordance with the preference of the programmer; and (iv) transparent navigation from the visual representation to the source code. SourceMiner has been used in different software engineering studies such as code smells identification [3] and characterization of strategies adopted by programmers in software comprehension activities [4].

The Collaborative SourceMiner [5] is a collaborative version of SourceMiner. It combines the use of a multiple view interactive environment with collaboration elements such as chat and bullets that inform which parts of the software have been analyzed by each programmer . This paper focuses on awareness support of the environment. The goal is to enhance software comprehension in distributed development, for this reason we use the term collaborative software comprehension.

This paper is structured as follows. In Section II we briefly review concepts related to collaborative software comprehension. Next, we present the proposed conceptual model of Collaborative SourceMiner. In Section IV we present a case study to analyze the effectiveness of the use of awareness elements in MVIEs to support collaborative software comprehension. Finally, in Section V, a discussion is presented, followed by conclusions about our work and avenues for future work

II. COLLABORATIVE SOFTWARE COMPREHENSION

In distributed software development environments, the geographic distance can hinder and limit the interaction opportunities due to the temporal distance [6]. It also hampers the understanding level of actions and efforts of group participants due to the cultural differences [7]. Moreover, the fact that the participants could have different native languages is a potential obstacle to communication [8]. According to Dix [9], two important aspects that benefit programmers in distributed development environment are: (i) explicit communication, where one programmer can inform others about his or her activities, and (ii) consequential communication where programmers can obtain useful information to accomplish activities by observing others´ actions.

The approach used in this paper is based on a collaboration model known as 3C+P (communication, coordination and cooperation plus perception) proposed in [17]. According to Fuks and Assis [12], awareness is the key element to support collaboration activities. However, the way these elements interact with each other depends on the project in which they have been used [18]. Awareness provides information to enhance collaboration due to the following: a) it enables the coordination of activities; b) it promotes the discussion of tasks through communication; c) it enhances interaction with others participants in the shared workspace through cooperation [10][13]. The workspace has an important role in collaboration activities [14]. Through the shared workspace participants can gain knowledge about group activities. This fact enhances awareness. The way awareness is supported in shared workspaces is essential in the cases where time and space need to be considered in the collaboration process definition [15].

In a distributed context, visual resources could also support awareness. These resources can be combined with collaboration elements (communication, coordination and cooperation) represented in the IDE to enhance software comprehension. This results in the proposed conceptual model that is discussed in the next section.

Researchers have already used visualization to support awareness. For example, Lanza et all. [26] proposed an approach to augment awareness by recovering development information in real time and broadcasting it to developers in the form of three lightweight visualizations. Treude and Storey [27] conducted a study about the use of a community portal by software project members. However, to the best of our knowledge, these researches do not consider examples of the use of awareness elements associated with collaboration elements in order to support software comprehension activities in MVIEs.

III. THE PROPOSED CONCEPTUAL MODEL

The proposed conceptual model was based on the definition of awareness presented in [11]. The main goal is to enable programmers from the same group to collaborate in a shared workspace and hence obtain knowledge to perform software comprehension activities. The conceptual model has as its start point the scenario illustrated in Figure 1. According to the figure, programmers perform software comprehension activities in different places. In the figure, the circle illustrates programmers accessing the source code (triangle) using the IDE (square). Considering this situation, we can conclude that collaboration occurs using resources (for example, chat on line) that are not integrated into the IDE. This scenario does not necessarily explore the potentiality of visual resources to support software comprehension activities. Moreover, a considerable cognitive effort will be needed due to the fact that the collaboration resources are not integrated into the IDE. This situation can also hinder convergence to perform a collaborative software comprehension.

Based on the scenario illustrated in Figure 1, we present the Figure 2 with the proposed conceptual model for collaborative software comprehension. The difference between Figure 1 and the part A of Figure 2 is that the IDE now has the Collaborative SourceMiner plug-in, represented by the red circle in the Figure 2. Moreover, the visual resources provided by the Collaborative SourceMiner have the goal to support awareness in a distributed software development. The views are enriched by awareness elements to enhance communication, coordination and cooperation.

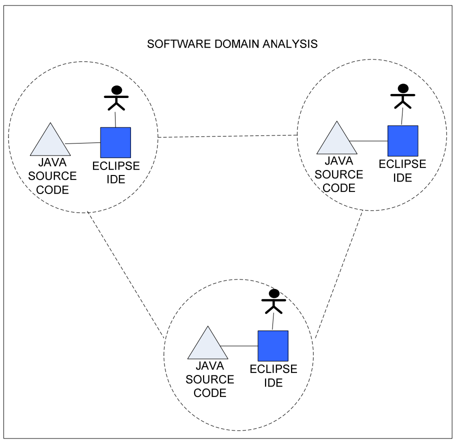


Figure 1. Collaborative Software Comprehension

For example, consider a set of classes that had its source code most frequently accessed by the members of a team while performing a given task. The result is that they can have its visual representation highlighted in the views. In this same example, a programmer can add a note to the visual representations of a class reporting information that needs to be considered relevant to the execution of the same task.

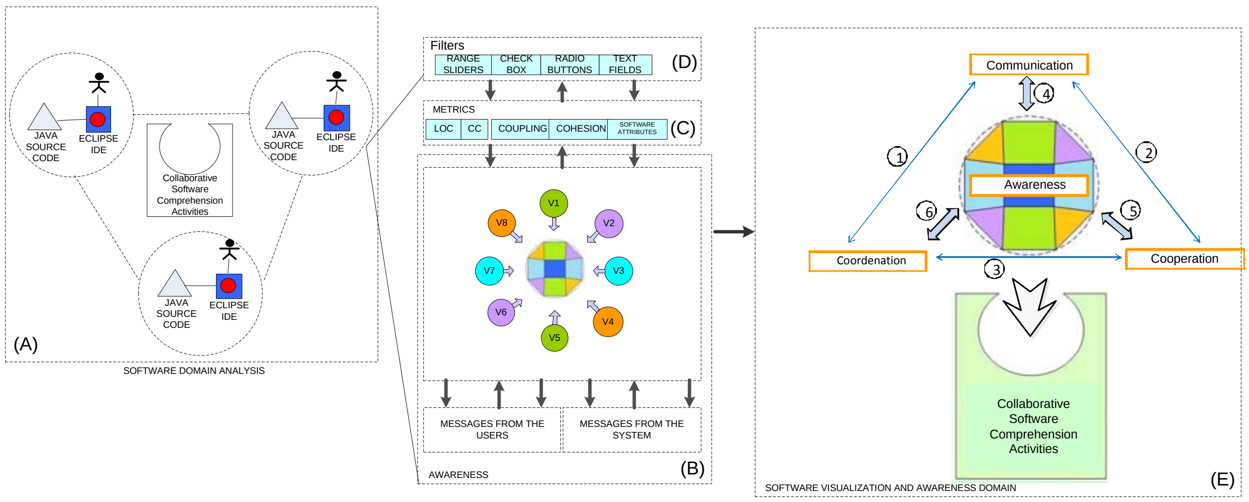


Figure 2. The Proposed Conceptual Model for Collaborative Software Comprehension

The fact that the representation of a set of entities is highlighted by awareness elements does not necessarily imply that these entities are relevant. These awareness elements are an initial suggestion of what should be analyzed by the group. For this end, the group can use the shared workspace to discuss and converge to the set of entities that are really of interest to the task at hand.

The combined use of multiple views enriched by awareness elements is conveyed in the part B of the Figure 2. Each view is represented by a colored circle (V1 to V8, for example). These views when used together and combined aim at providing features of a multiple view interactive environment (MVIE). The awareness elements are the result of information that programmers find useful to share with others from the same team (marked as messages from the user in the Figure 2 and implemented in Figure 6) and information regarding classes and methods accessed while performing a specific task (marked as messages from the system in the Figure 2 and implemented in Figure 7). These messages are represented in the views using visual attributes such as icons and colors that can vary in tonality depending on the type and numbers of messages related to a specific software entity (see Figure 7). The part C of the figure shows that software entities (packages, classes, methods, attributes and interfaces) obtained from the Abstract Syntax Tree (AST) are enriched by metrics such as size, cyclomatic complexity, and coupling. The model allows the inclusion of new metrics that appear to be relevant in the shared workspace. The part D illustrates that the interaction with the multiple views is supported by the filters, semantic and geometric zooms and other interaction resources.

In fact, the model considers the influence of coordination, cooperation and communication elements to enrich the shared workspace with awareness information. This is represented in part E of Figure 2, where the result is a visual scenario composed of multiple views and their corresponding awareness elements.

The model considers both synchronous and asynchronous interaction support. Interactions in Collaborative SourceMiner result from two types of messages: those from the user (Figure 6) and messages that are automatically collected by the system registering what programmers are doing in the IDE. The first one is the kind of messages that can be sent by the programmers to register information that is considered relevant to a specific task asynchronously. The second type of message is sent automatically by the Collaborative SourceMiner. The goal of this set of messages is to enrich the visual representation in the multiple views in order to contextualize programmers synchronously. When a programmer starts the execution of a task his or her actions are registered and automatically sent to a server, as illustrated in Figure 3.

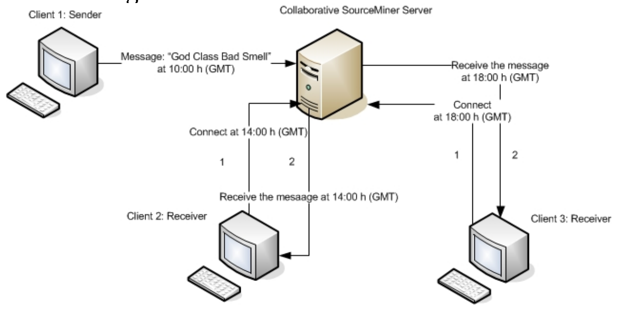
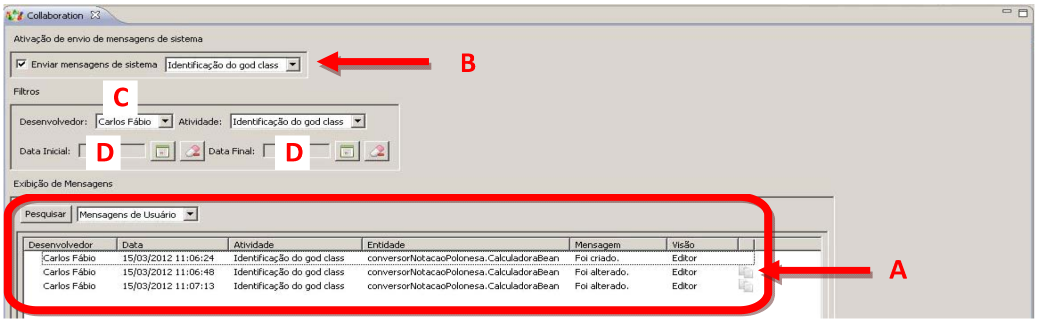

Client 1: Sender
Collaborative SourceMiner Server
Message: "God Class Bad Smell" at 10:00 h (GMT)
Receive the message at 18:00 h (GMT)
Connect at 18:00 h (GMT)
Connect at 14:00 h (GMT)
1
2
1
2
Receive the mesaage at 14:00 h (GMT)
Client 2: Receiver
Client 3: Receiver

Figure 3. Implemented Topology of the Conceptual Model

A web service is available to receive and send messages from and to the Collaborative SourceMiner clients which are configured in the team. The client of this service is the IDE Eclipse with the Collaborative SourceMiner plug-in. The messages contain the following parameters: project, user, and, optionally, the activity in which the programmer is working. Before recording the message, Collaborative SourceMiner checks if the user who sent the message is in fact registered in the project.



Ativação de envio de mensagens de sistema

☑ Enviar mensagens de sistema: Identificação do god class

Filtros

Desenvolvedor: Carlos Fábio Atividade: Identificação do god class

Data Inicial: Data Final:

Exibição de Mensagens

Mensagens de Usuário

Desenvolvedor	Data	Atividade	Entidade	Mensagem	Visão
Carlos Fábio	15/03/2012 11:06:24	Identificação do god class	conversorNotacaoPolonesa.CalculadoraBean	Foi criado.	Editor
Carlos Fábio	15/03/2012 11:06:48	Identificação do god class	conversorNotacaoPolonesa.CalculadoraBean	Foi alterado.	Editor
Carlos Fábio	15/03/2012 11:07:13	Identificação do god class	conversorNotacaoPolonesa.CalculadoraBean	Foi alterado.	Editor

Figure 4. Messages from the System Registering Actions Executed by a Programmerc

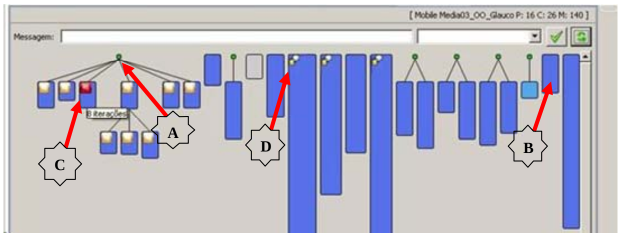
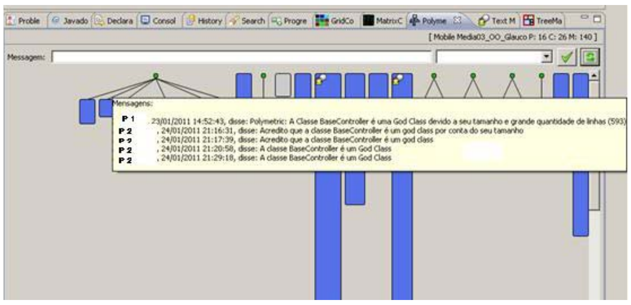


Figure 5. Polymetric View enriched by Messages from the System Registering Actions Executed by a Programmer



Mensagens:

P 1 23/01/2011 14:52:43, disse: Polymetric: A Classe BaseController é uma God Class devido a seu tamanho e grande quantidade de linhas (593)
P 2 , 24/01/2011 21:16:31, disse: Acredito que a classe BaseController é um god class por conta do seu tamanho
P 2 , 24/01/2011 21:17:39, disse: Acredito que a classe BaseController é um god class
P 2 , 24/01/2011 21:20:58, disse: A classe BaseController é um God Class
P 2 , 24/01/2011 21:29:18, disse: A classe BaseController é um God Class

Figure 6. Communication among Participants through the Polymetric View

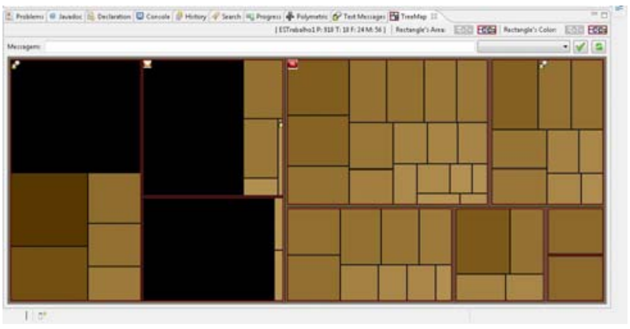


Figure 7. Treemap View displaying Icons to represent Messages from the System

Figure 4 displays examples of messages that are automatically registered (marked as A in the Figure). Each register has the following format: programmer, date/hour, activity, entity (class, interface, or method) and view. The messages can be filtered using as a parameter the programmer(s) that performed the actions (marked as C in the figure) and the task that it was associated with (marked as B in the figure).

Another possibility to filter messages is by the period in days that it occurred (marked as D). These data aim at characterizing actions performed by programmers while performing a specific task. When a software entity is modified, an icon is presented. Different versions of this entity can be shown if the programmer click in this icon. In Figure 5 the polymetric view [3] illustrates the inheritance hierarchy of entities (classes and interfaces) of a software system. It portrays inheritance relationships between the software entities (class/interface) as a forest of round rectangles.

Originally proposed for this purpose, polymetric views help to understand the structure and detect problems of a software system in the initial phases of a reverse engineering process [3]. Interfaces are represented as green circles (arrow A) and classes as blue rectangles (arrow B). In the same figure, arrows C and D indicate icons that represent messages that can be relevant to the understanding of a specific entity. In this case, arrow C indicates messages from system while arrow D indicates messages from the users. The icons can vary in tonality to highlight software entities with which programmers most interacted. This is related to the coordination support. It has the goal to indicate entities that at first glance are somehow related to the activity performed by the group. This can, for example, motivate the group to know which entities programmers with more experience were interacting with. In this case, specific pieces of code are relevant when the team knows that experienced programmers worked on them.

This can promote faster convergence for the identification of these parts of the code that are probably related to the software comprehension activity. Moreover, this scenario can also be used to stimulate interactions among participants so that they can make a decision. The difference between this scenario and the one described in Figure 1 is that now collaboration occurs through the use of the Collaborative SourceMiner shared workspace. Another difference is that the collaborative software comprehension is based in a multiple view interactive environment.

As already discussed, visual resources have the potential to support collaborative software comprehension. This potential can be better exploited when the views are enriched with information from every participant. For example, the way the shared workspace has been used by group members of the group (part B of conceptual model presented in Figure 2) encourage the sharing of knowledge.

Another example is related to the importance of expert programmers in a team. They can lead the convergence to the strategy to be applied in a given activity. This fact can occur when the shared workspace indicates software entities that expert programmers have selected to perform the activity. This enables other programmers to analyze the same entities through the same views as suggested and registered by the experts. Moreover, this situation motivates interaction among programmers so that decisions can be taken together and the group has access to a wider pool of ideas and possibilities regarding the activity to be performed then they would have when working alone. The expected result is the combined use of collaboration elements (communication, coordination and cooperation) in a multiple view interactive environment to support software comprehension activities. This is represented in Part E of Figure 2.

IV. The Case Study

A case study was conducted to analyze the following research question: "How awareness elements provided by Collaborative SourceMiner support software comprehension considering that programmers work collaboratively in a distributed environment?"

Null hypothesis: Awareness elements provided by Collaborative SourceMiner do not effectively support the identification of code smells considering that the participants work collaboratively in a distributed environment.

Alternative hypothesis: Awareness elements provided by Collaborative SourceMiner effectively support the identification of code smells considering that the participants work collaboratively in a distributed environment.

Six participants took part in the study. They worked in two groups of three participants each. This number of participants offered a reasonable tradeoff between the cost of the study and detailed qualitative analysis and the generalizability of the results. To be eligible for inclusion, participants were required to have the following skills: experience with the object-oriented programming Java; and in the use of the Eclipse IDE. This experience was verified by asking them to fill in questionnaire forms. No current member of our research groups took part in this study. They were all volunteers and no compensation was provided for their participation in this study.

Prior to the study tasks, the participants were required to complete a tutorial session on how to use the multiple views approach implemented by Collaborative SourceMiner. In this training session, the participants had 24 hours to familiarize themselves with the tool. They were asked to analyze a program, called Health Watcher [21], and to answer 28 basic questions regarding the tool functionalities. During the tutorial session, the second author of this paper was available online (email and chat) to provide complementary guidance and detailed explanation on how to use Collaborative SourceMiner. After the tutorial participants were asked to execute the code smells identification.

This study relies on a software product line, called MobileMedia (MM) [22] that manipulates photo, music, and video on mobile devices, such as mobile phones. It has about 4 KLOC distributed in 18 packages and 50 classes. We selected MobileMedia due to several reasons. First, its Java implementation is available. Second, its key concerns were previously identified by the developers and mapped to the source code [22].

We relied on two experts to build a reference list for each analyzed code smell (Feature Envy - FE, God Class - GC, and Divergent Change - DC[23]). The experts are researchers that participated in the development, maintenance, and assessment of the target system. The goal was to detect actual instances of each code smell in versions 3 to 7 of MobileMedia.

We collected direct and indirect data based on questionnaires answered by the participants and provided by an instrumentation system. The questionnaires described the MobileMedia main functionalities, the code smells with examples, and the tasks to be performed by the participants. Participants were asked to list classes suspected of manifesting code smells as well as the strategies they use to identify them. They were also asked to describe which of the Collaborative SourceMiner resources, such as views, concerns, filters, and colors, they found helpful to perform the task at hand. A logging functionality of Collaborative SourceMiner automatically records data describing the environment usage at a fine-grained detailing level. This functionality sends the data automatically to the server (see Figure 3) and is used to monitor how frequently a view or a feature of the tool is used, the transitions among views, and the time each action happened. The goal is a better understanding of the participants' strategies based on their recorded actions.

Two important roles of this study were the coordinator and programmer. The first had the following responsibilities: register the project to be analyzed, the activities to be performed and the participants of the study. The goal was to configure the environment for the study. The coordinator did not perform any of the software comprehension activities. Each team had 48 hours to perform the asked tasks. Each group was asked to identify the code smells Feature Envy, God Class and Divergent Change [23] in a software system called Mobile Media.

Table 1 presents the values of precision (p) and recall (r) of the identification of code smells of each participant. The precision metric quantifies the rate of correctly identified code smells by the number of detected code smell candidates. Recall quantifies the rate of correctly identified code smells by the totally number of actual code smells. PA1, PA2, and PA3 represent the participants 1, 2 and 3 from the first group. PA4, PA5, and PA6 represent the participants 1, 2, and 3 from the second group. The values of PA5 were not considered in the study due to the fact that he did not answer the questionnaires.

TABLE I. PRECISION AND RECALL IN CODE SMELLS IDENTIFICATION

	PA1		PA2		PA3		PA4		PA6	
	r	p	r	p	r	p	r	p	r	p
GC	0,9	1,0	0,8	0,9	0,9	0,7	0,2	0,7	0,2	0,7
DC	0,2	0,6	0,1	0,4	0,1	0,1	0,1	0,1	0,1	0,4
FE	0,4	0,3	0,1	0,1	0,2	0,2	0,2	0,4	0,2	0,6

The analysis of the data from Table 1 shows that participants from the first group had greater variation of precision and recall than participants from group 2. The lesser variation of precision and recall of group 2 can be justified by the collaboration among participants. All the participants obtained higher values and lesser variation of precision and recall in the God Class identification when compared with the other two code smells. The analysis of the messages provided by the Collaborative SourceMiner and the questionnaires show that participants used the communication feature (internal chat) provided by the proposed plug-in to indicate the code smells candidates.

Messages among participants revealed evidences of communication during the execution of the asked tasks. Figure 6, for example, shows evidences of this communication where PA1 and PA2 comment the case of the class BaseController as a God Class candidate. PA2 also informed us in the questionnaires that s/he had clicked in the icons in the views to read messages sent by others about the relationship of specific software entities and the asked task.

PA2 also mentioned that used the filters presented in Figure 4 to analyze the messages of interest to the task. According to PA2, "when I noticed that PA1 interacted several times with the class UnavailablePhotoAlbumException, I analyzed the class in more details in order to verify if it should be a Feature Envy candidate. However, after this analysis I concluded that it was not a Feature Envy occurrence". PA1 registered in the questionnaire that "the comments from the other participants helped me to identify certain particularities in the classes and methods of the analyzed software system that I would not be able to identify without collaboration". PA3 informed that: "The messages, especially the ones from PA1, were of great relevance to guide me in the execution of the asked tasks. PA4 also mentioned that: "the indication of the BaseController class was in accordance with the suggestion of PA6". This comments provided by the participants show initial evidences that enriching the visual representations with Information provided by the participants of a group contributed to the convergence of which should be done in the asked tasks.

Due to the values of precision and recall obtained by PA1 and the comments registered in the questionnaires, there are initial evidences that PA1 guided PA2 and PA3 in the tasks execution using the collaboration resources provided by the Collaborative SourceMiner.

Based on the analysis of the research questions analysis, we present the observations as follows. Observation 1: during the execution of the asked activities programmers collaborated among themselves and to some extent converged in the indication of code smells. Observation 2: there are initial evidences that participants considered the actions and comments of others from the same group to decide how to proceed in the asked activities. Observation 3: there are initial evidences that participants adopted similar strategies to identify code smells, hence they collaborate while performing the asked tasks. These observations and the results obtained in the study presented in this paper show evidences that the alternative hypothesis is true.

A. *Threads to Validity*

The use of only one object (Mobile Media) as well as its size and complexity are far inferior when compared with typical software systems. However, MobileMedia has already been used in other studies to characterize the use of software visualization tools. An important limit to the generalizability of our findings comes from the fact that we have based our observations on the analysis of the behavior of only five subjects. However, as already mentioned, the number of participants accepted in the study was based on a tradeoff between the cost of the study and qualitative analysis of the results to derive the observations.

V. CONCLUSION AND FUTURE WORK

This paper presented a multiple view environment to support collaborative software comprehension. The coordinated views integrated into the IDE provide mechanisms that enable the use of awareness to perform software comprehension activities in a distributed development. The results of the study presented in this paper show initial evidences about how programmers use Collaborative SourceMiner to perform code smells identification. Moreover, the results showed how the features provided by the proposed environment enable the use of awareness elements to perform software comprehension activities. Another important result was the use of visual representation of software entities combined with awareness elements in the context of software comprehension. Differently from other studies like the ones presented in [26] and [27], the focus of this paper was to present initial evidences on how programmers could interact and collaborate to foster software comprehension using the visual metaphors available in SourceMiner. We are planning the inclusion of other mechanisms of cooperation, communication and coordination in the Collaborative SourceMiner to support software comprehension activities in a distributed development.

A version of CollaborativeSourceMiner is available at [28] as well as instructions to configure its environment.

REFERENCES

[1] Biehl, J.T., Czerwinski, M., Smith, G., and Robertson, G.G. (2007) "Fastdash: a visual dashboard for fostering awareness in software teams". In: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM 1313-1322.

[2] de Souza, C.R.B., Quirk, S., Trainer, E. and Redmiles, D. Supporting Collaborative Software Development through the Visualization of Socio-Technical Dependencies. ACM Conference on Supporting Group Work, ACM Press, Sanibel Island, FL, 2007.

[3] Carneiro, G., Silva, M., Mara, L., Figueiredo, E., Sant'Anna, C., Garcia, A., and Mendonca, M. Identifying Code Smells with Multiple Concern Views. In proceedings of the 24th Brazilian Symposium on Software Engineering (SBES), 2010.

[4] Fernandes, J. M.; Carneiro, G. Strategies and Profiles of Novice Programmers while Identifying Code Smells. In: IX Brazilian Workshop on Software Maintenance (WMSWM 2012), Fortaleza/CE. In portuguese.

[5] Conceição, C.F.R. Analyzing the Use of Awareness Elements to Support Software Comprehension Activities in a Distributed Development Environment. Master Thesis. Computer Science Department. Salvador University (UNIFACS), 2012. In portuguese.

[6] Gerfalk, P. J. Fitzgerald B. Flexible and distributed software processes: old petunias in new bowls? Communications of the ACM, v. 49, n.10, p.26-34, 2006.

[7] Casey V. Leveraging or exploiting cultural difference? In: IEEE International Conference on Global Software Engineering (ICGSE 2009), Limerick, Ireland: IEEE Computer Society, 2009. p. 8-17.

[8] Carmel, E.; Tjia, P. Offshoring Information tecnhnology: sourcing and outsourcing to a global workforce. Cambridge: Cambridge University Press, Cambridge, U.K., 2005.

[9] Dix, A.; Finlay, J.; Abowd, G.; and Beale, R. Human-Computer Interaction, Prentice Hall. 1993.

[10] Ellis, C. A.; Gibbs, S. J.; and Rein, G. L. Groupware - Some Issues and Experiences. Communications of the ACM, v. 34, n. 1, p. 38-58, 1991.

[11] Dourish, P.; Bellotti, V. Awareness and coordination in shared workspace. Conference on Computer-Supported Cooperative Work. pp. 107-114, Toronto, Canada, Nov. 1992.

[12] Fuks, H.; Assis, R. L. Facilitating perception on virtual learningware-based environments. The Journal of Systems and Information Technology. Edith Cowan University. Austrália, v. 5, n. 1, p. 93-113, 2001.

[13] Gutwin, C.; Greenberg, S. A descriptive framework of workspace awareness for real-time groupware. Journal of Computer-Supported Cooperative Work. Issue 3-4, p. 411-446, 2002.

[14] Gutwin, C. Workspace awareness in real-time distributed groupware. 1997. PhD Thesis. Department of Computer Science, University of Calgary, 1997.

[15] Omoronyia, J. Ferguson, M. Roper, and M. Wood. A Review of Awareness in Distributed Collaborative Software Engineering. Software Practice and Experience, 40 (12). November 2010. pp. 1107-1133.

[16] Storey,M. Theories, Tools and Research Methods in Program Comprehension: Past, Present and Future. Software Quality Journal, 2006.

[17] Fuks, H.; Raposo, A.; Gerosa, M.A.; Pimentel, M.; and Lucena, C.J.P. The 3C Collaboration Model. The Encyclopedia of E-Collaboration, Ned Kock (org), 2007, pp. 637-644.

[18] Fuks, H., Raposo, A., Gerosa, M.A., Pimentel, M., Filippo, D., and Lucena, C.J.P. Inter- and Intra-relations among Communication, Coordination and Cooperation. In IV Brasilian Symposium on Collaborative Systems, Rio de Janeiro – RJ. 2007, pp. 57-68. (In Portuguese).

[19] Pattison, T. and Phillips, M. View Coordination Architecture for Information Visualization. In Proceedings of the Australian Symposium on Information Visualization, 2001, Sydney, Australia. pages 165-171.

[20] M. Baldonado, A. Woodruff, and A. Kuchinsky. Guidelines for Using Multiple Views in Information Visualization. In ACM AVI 2000; Palermo, Italy. 110-119.

[21] Greenwood, P. On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study. ECOOP, Germany, 2007.

[22] Figueiredo, E. Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability. ICSE, May 2008.

[23] Fowler, M. Refactoring: Improving the Design of Existing Code. Addison Wesley, 1999.

[24] Petre, M. Mental imagery and software visualization in high-performance software development teams. J. Vis. Lang. Comput., v. 21, n. 3, p. 171-183, 2010.

[25] Ware, C. Information Visualization, Second Edition: Perception for Design (Interactive Technologies). 2. ed.Morgan Kaufmann, 2004.

[26] Lanza, M., Hattori, L., and Guzzi, A. Supporting Collaboration Awareness with Real-time Visualization of Development Activity. In Proceedings of the 14th IEEE European Conference on Software Maintenance and Reengineering (CSMR), pp. 207 - 216. IEEE CS Press, 2010.

[27] C. Treude and M.-A. Storey. Effective Communication of Software Development Knowledge Through Community Portals. In Proceedings of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE ’11). ACM, New York, NY, 91-101.

[28] SourceMiner. A Multiple View Interactive Environment Implemented as an Eclipse Plug-in. Available at http://www.sourceminer.org.

Assisting bug Triage in Large Open Source Projects Using Approximate String Matching

Amir H. Moin and Günter Neumann
Language Technology (LT) Lab.
German Research Center for Artificial Intelligence (DFKI)
Saarbrücken, Saarland, Germany
$\{$*amir.moin, neumann*$\}$*@dfki.de*

***Abstract*—In this paper, we propose a novel approach for assisting human bug triagers in large open source software projects by semi-automating the bug assignment process. Our approach employs a simple and efficient n-gram-based algorithm for approximate string matching on the character level. We propose and implement a recommender prototype which collects the natural language textual information available in the summary and description fields of the previously resolved bug reports and classifies that information in a number of separate inverted lists with respect to the resolver of each issue. These inverted lists are considered as vocabulary-based expertise and interest models of the developers. Given a new bug report, the recommender creates all possible n-grams of the strings, evaluates their similarities to the available expertise models concerning a number of well-known string similarity measures, namely Cosine, Dice, Jaccard and Overlap coefficients. Finally, the top three developers are recommended as proper candidates for resolving this new issue. Experimental results on 5200 bug reports of the Eclipse JDT project show weighted average precision value of 90.1% and weighted average recall value of 45.5%.**

***Keywords*-software deployment and maintenance; semi-automated bug triage; approximate string retrieval; open source software.**

I. Introduction

Open source software projects often provide their developer and user communities with an open bug repository for reporting the software defects in order to be tracked by developers and users. Each bug report usually undergoes a triage process in which a group of developers, known as triagers, check whether it contains sufficient amount of information for the developers, whether it is not a duplicate of a previously reported bug and if the bug is reported at the right place. Only if the bug report passes these filters successfully then they would assign a priority degree and a severity degree to it from the business perspective and from the technical point of view, respectively. Last but not least, the triagers should assign each bug report to a developer in order to hopefully resolve the issue. This latter part, i.e., bug report assignment defines the scope of our work.

In large open source projects where hundreds or thousands of developers are collaborating with each other the main question is which person would be the best candidate for fixing a newly reported bug. Human triagers often take developers' fields of expertise and interest into consideration in order to reduce the bug resolution cost for the project. However, since the number of bug reports and the rate of their production could become very large, the bug triage process itself might become labor intensive when performed manually.

In recent years, there have been a number of valuable contributions in order to address this problem. We overview most of the works which we are aware of in Section IV. One common approach for semi-automated bug assignment is to employ a supervised machine learning algorithm through which a classifier is trained and used to categorize new bug reports. In such text categorization problems, documents are usually considered as word vectors and the term weights (Term Frequency-Inverse Document Frequencies, abbreviated as TF-IDF) are calculated. Support Vector Machines (SVMs) have turned out to be the best supervised machine learning algorithms in applying such an approach [1]. Furthermore, there have been other approaches already applied to several subfields of the bug triage problem area such as the duplicate bug report recognition problem [2] based on the natural language processing techniques.

However, we clearly make several distinctions between our work and those contributions. Our approach is novel in that it considers n-gram representations for strings and works on the character level rather than on the token level (term level). Furthermore, we perform approximate string matching, i.e., approximate vocabulary look-up, with a flexible similarity threshold parameter rather than a fixed exact match. Considering the application domain, this brings in a noticeable capability. For example, one does not have to concern about the misspelled words in the bug reports too much anymore. Finally, we do not directly consider the similarity between the vector representation of two text documents, but, instead, the approximate similarity between the n-gram representations of strings are taken into account.

Regardless of the applied techniques, almost all prior work in this field could be categorized in three main groups based on the source of information which they use in order to extract the developers' expertise and interest models. One major source of information is the previously resolved bug reports which are archived in the repositories of bug tracking systems like Bugzilla [3], JIRA [4], etc. ([5] [6] [1] [7]). Another one is the log information of the source code revision control system, i.e., comments of developers in each source code commit (for example to the Subversion [8] or CVS [9] source code repository) ([10] [6] [7]). Finally, one could explicitly use the vocabulary of the committed source code by each developer as a means of extracting such an expertise model ([11]).

Our recommender is efficient and powerful enough to benefit from all of the above mentioned resources in large scales in order to make its recommendation more precise while performing fast. However, in the current implementation we have only included the first one, i.e., the previously resolved bug reports in the open bug repository.

This paper makes two main contributions:

1) It proposes a novel approach which employs a simple and efficient approximate string matching algorithm [12] in order to find appropriate developers who are more likely to have sufficient expertise and interest levels to resolve a new issue.
2) It provides an implementation of the proposed approach as well as experimental results on a large dataset of 5200 bug reports from the Eclipse Java Development Tools (JDT) project including the achieved information retrieval evaluation metrics, namely, accuracy, weighted average precision, weighted average recall and F-measure for the top three developers recommendation.

The paper is structured as follows: Section II presents the proposed approach. In Section III, we describe the implementation of our recommender prototype and present experimental results. Related work in this field are discussed in Section IV. Finally, conclusion and possible future work are stated in Section V.

II. The Proposed Approach

The core idea is to apply an n-gram vocabulary-based approach with approximate string matching. We start with a dataset of previously resolved bug reports where each instance contains the free text available in the summary and description of the bug report and the name of the developer who has resolved the bug. For each developer p_i we automatically extract a vocabulary from all bug reports assigned to p_i. This gives us a set D of vocabularies d_i ($1 \leq i \leq n$, where n is the number of different developers p_i). Then, for a new bug report, we automatically extract its vocabulary d_j, and compute its overlap with D. We compute a ranked list of names of developers $\{p_i\}$ with respect to the degree of overlap of d_j and the corresponding vocabulary $d_i \in D$, and finally recommend the top three developers as possibly good candidates for resolving this new bug.

Our approach is semi-automated, since a human triager would need to choose one of the recommended developers by our system in order to finally assign the bug to that person.

A. Background: SimString

Since our approach is vocabulary-based, we need to address the problem of spelling variations (e.g., spelling errors like “Pyhton” instead of “Python” or name variations like “FileOpenDialog” vs. “’File Open’ Dialog”). Furthermore, since we are extracting a number of different large-scale vocabularies, we need fast and scalable approximate string matching algorithms.

SimString [13] is a C++ library that uses the CPMerge algorithm [12] for fast approximate string matching. The idea is to construct an n-gram-based inverted index from the entries of a vocabulary (basically a list of strings of instances of some common semantic class). The n-gram representation of a string s is just the set of all substrings of length n of s. For example, for n=3, the n-grams for the string “python” are {pyt, yth, tho, hon}, and for the string “pathon” they are {pat, ath, tho, hon}. Matching is then realized by defining a similarity function that applies a τ-overlap join between the n-gram representation of a query and the n-gram representation of the inverted index of the vocabulary. Actually, τ is a function of the given approximate similarity threshold (a value between 0 and 1.0), the given query string, and the provided vocabulary. For example, for the cosine function, when using the similarity threshold of 1.0, the two above entries would not match. However, setting the similarity threshold to 0.7, they would match.

For our purpose, the most important properties of SimString are:

1) It allows finding matches in a collection of millions of entries in only a few milliseconds, e.g., using cosine similarity function and a similarity threshold of 0.7 only about 1 millisecond is needed for a query on standard PC hardware.
2) Beside the cosine similarity function, SimString utilizes additional well-known similarity functions, namely, Jaccard, Dice and Overlap coefficients [12].
3) When constructing the n-gram-based inverted index, the exact value of n can be parametrized.
4) SimString uses efficient disk-based hashing for maintaining the inverted index, and hence, it has a very good memory footprint.

We will now describe in detail how we are employing SimString in our recommender prototype, which is called Approxricom.

B. Our Bug Resolver Recommender: Approxricom

A bug report, i.e., an issue, in an open bug repository, i.e., an issue tracking system, often has a life-cycle. It is initially created by any registered user or developer. In this very beginning stage, its status is set to NEW. Later, during its life-cycle, its status might change to other possible values such as ASSIGNED, RESOLVED DUPLICATE, RESOLVED INVALID, RESOLVED WORKSFORME, RESOLVED WONTFIX, RESOLVED FIXED, VERIFIED FIXED or CLOSED FIXED. No matter the bug is currently at which state, it always resides in the open bug repository of the open source project. Our recommender only uses issues which are currently in the RESOLVED FIXED, VERIFIED FIXED and CLOSED FIXED states in order to create its vocabulary-based expertise and interest model.

Furthermore, each bug report is a structured file which consists of a number of fields in addition to its current state, such as the bug ID, summary, description, product, component, importance, assignee, reporter, date of report, date of the latest modification, CC (Carbon Copy) list (i.e., the list of people who are interested in being updated about this issue), comments, etc. Among all these fields, our recommender is only concerned with the bug ID, product name, status, summary and description of the bug reports.

As depicted in figure 1, the workflow of our recommender comprises three main parts:

1) Collecting and preprocessing the required data including the bug IDs, the resolver developer for each bug and the summary and description text of each bug report in order to create the vocabulary-based expertise and interest models (corresponds to step 1 in Figure 1).
2) Creating the n-gram-based inverted lists of the textual information in summaries and descriptions of previously resolved bug reports (corresponds to step 2 in Figure 1).
3) Recommending the three most similar vocabulary databases to each query which contains the textual information extracted from the summary and description of a newly reported bug (corresponds to steps 3 to 5 in Figure 1).

Concerning the first part, our recommender connects to the open bug repository of an open source project and retrieves a large set of successfully resolved bug reports for a specific product. The second and third parts of the recommendation workflow deal with employing a fast approximate string matching algorithm to store the inverted lists (inverted indexes) in an efficient way and later retrieve them w.r.t. the similarity level of the query, i.e., the vocabulary of the new bug report to each of those lists which we actually consider them as the vocabulary-based expertise and interest models of developers. Given a new bug report as a query, our recommender assigns a score to each vocabulary-based expertise model database, i.e., to each developer, and finally, recommends the top three databases (developers) as proper candidates for this query, i.e., for the resolution of this new bug report.

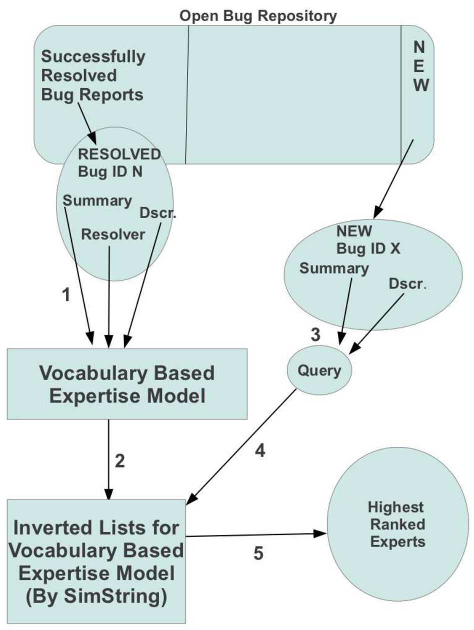

Open Bug Repository
Successfully Resolved Bug Reports
N E W
RESOLVED Bug ID N
Summary
Dscr.
Resolver
NEW Bug ID X
Summary
Dscr.
1
3
Query
Vocabulary Based Expertise Model
2
4
Inverted Lists for Vocabulary Based Expertise Model (By SimString)
5
Highest Ranked Experts

Figure 1. Bug Resolver Recommendation by Approxricom using SimString

III. Implementation & Experimental Results

We implement our prototype using Java, C++ and Python. The interface for interacting with the open bug repository is developed in Java using XML Remote Procedure Call (XML-RPC) [14]. The core of Approxricom which employs SimString is developed in C++. Finally, we take advantage of Python for our data preparation using the python-based Natural Language Toolkit (NLTK) [15].

A. Parameters

The approximate string matching algorithm which we have employed in Approxricom, i.e., SimString is configurable with various parameters. In particular, one could set the parameter *n* for n-gram size, the similarity measure for vocabulary look-up (Cosine, Dice, Jaccard and Overlap coefficients) and the degree of flexibility in the approximate matching, known as the similarity threshold which is between 0 and 1.0 (1.0 corresponds to exact matching instead of approximate matching).

After doing some experiments with various values of configuration parameters, we decided to use the following setting for our recommender:

1) The size of n-grams (n) is 5. In other words, all possible 5-grams are created for string representations.
2) All these four similarity measures are taken into account: Cosine, Dice, Jaccard, Overlap.
3) The similarity threshold is set to 0.95. While maintaining a high degree of accuracy (since it is close to

Table I
EVALUATION USING WELL-KNOWN INFORMATION RETRIEVAL METRICS (W.A. STANDS FOR WEIGHTED AVERAGE)

Fold /	Accuracy /	W.A. Precision /	W.A. Recall /	F-measure
1	50.19	85.8	46.02	59.91
2	50.86	83.16	44.39	57.89
3	46.73	75.05	42.62	54.37
4	49.8	86.35	41.46	56.03
5	51.92	94.98	44.81	60.89
6	55	96.85	50.4	66.3
7	57.11	97.16	50.25	66.24
8	57.11	93.76	53.29	67.95
9	55.57	100	42	59.15
10	53.37	88.14	39.82	54.86
Average	52.76	90.12	45.50	60.35

1.0) this value prevents the recommender from trivially doing exact string matching, which is in contrast to our core idea of approximate string matching in order to handle spelling errors and different writing forms.

B. Scoring Strategy

In order to rank our inverted lists based on the degree of their similarity to a query, we store the maximum number of similarities which are reported by each of the four similarity functions, Cosine, Dice, Jaccard and Overlap coefficients. This maximum number is considered as the score of that inverted list, given a query string and a similarity threshold.

C. Experimental Results

In order to evaluate our approach, we did experiments on 5200 resolved bug reports of the Eclipse JDT project [16]. This project has been chosen because of its popularity and rich history of bug resolution. After preparing the dataset, we performed a 10-folded random cross validation on this dataset which led to the results of Table I (values are in percent). As shown in the table, we have achieved average weighted precision of 90.12% and average weighted recall of 45.50%. Moreover, the accuracy is 52.76% on average.

We consider the top three developers, which are recommended by our bug resolver recommender as proper candidates for resolving a new bug report. Therefore, it is clear that if the developer who has already fixed a bug in practice is present among the three top-ranked developers which are recommend by Approxricom, we consider this as a success. Otherwise, it is considered as a failure.

The True Positive (TP) rate, also known as recall, is a measure of completeness which shows how much part of a specific class is captured. In other words, recall is the proportion of the bug reports, which are recommended to be assigned to developer p, among all bug reports which have been resolved by p in practice.

The precision of our recommendation is reflected by the proportion of the bug reports which have been resolved by developer p in practice, among all those bug reports which are recommended to be assigned to p.

Since there is often a trade-off between precision and recall, it is common to measure the final performance via a mixture of both, called F-Measure.

$$F - Measure = \frac{2*Precision*Recall}{Precision+Recall}$$

Finally, accuracy is the proportion of the total number of correctly assigned bug reports among all of the bug reports in the test dataset.

IV. RELATED WORK

One related contribution is Mockus and Herbsleb’s Expertise Browser [10]. They have introduced a tool which uses source code change data from a revision control repository to determine appropriate experts to work on various elements of software projects.

Cubranic and Murphy [5] have trained a Bayesian classifier using descriptions of fixed bug reports in open bug repositories as machine learning features, and names of developers as class labels. They have reported the accuracy value of 30% for Eclipse projects. However, they fairly believe that even this level of classification accuracy could significantly lighten the load that the human triagers face in large open source projects.

Canfora and Cerulo [6] have proposed an information retrieval approach. They have used textual descriptions of fixed change requests stored in open source software repositories, both bug repository and revision control system (Bugzilla and CVS), to index developers and source files as documents in an information retrieval system. The indices are utilized to suggest the most appropriate developers to resolve a new change request. They have reported recall values of 10-20% and 30-50% for Mozilla projects and KDE, respectively.

Their approach is similar to ours in that they also index documents and developers to solve an information retrieval problem. However, from their very brief explanation it is clear that they do not apply any approximate n-gram-based string retrieval algorithm.

Anvik et al. [1] have presented an approach which expands on Cubranic and Murphy’s previous work [5]. They have used additional textual information of bug reports beyond the bug description, to form the machine learning features. They have also applied a non-linear Support Vector Machines (SVMs) and C4.5 algorithms in addition to the Naive Bayes classifier which their predecessor work had used. They have compared the achieved results using the three classifiers and found SVM the best one for this purpose.

Anvik and Murphy [7] have presented an empirical evaluation of two approaches for determining who has the implementation expertise for a bug report using data from two types of repositories: The source repository check-in logs and the bug repository. They have found that different repositories are useful in different situations, based on what is wanted.

Jeong et al. [17] have introduced a graph model based on Markov chains to capture bug tossing history. Bug tossing refers to the process of re-assignment of bug reports among the developers of an open source project. The model could be used both to reveal team structures to find suitable experts for a new task and also to better assign developers to bug reports. Their model has reduced bug tossing by up to 72% and improved the accuracy of automatic bug assignment by 23% comparing the common manual bug triage process.

Baysal et al. [18] have presented a theoretic framework for automating assignment of bug-fixing tasks with an emphasis on learning developer preferences. They have proposed to apply a vector space model to recommend experts for resolving bug reports based on the level of expertise, current workload and preference of developers which are inferred from the previously fixed bugs by each developer. Implementation of their novel model has remained as future work.

Matter et al. [11] have stood in an outstanding position, since they model developer expertise by explicitly comparing the vocabulary found in the source code contributions of developers with the vocabulary of bug reports. The advantage of their approach is that no previous activity of a developer in the current project is necessarily required. Instead, any prior activity of a developer through interacting with a source code revision control system would be enough to model his or her expertise. They report 33.6% top-1 precision and 71% top-10 recall.

Bhattacharya and Neamtiu [19] have improved triaging accuracy and reduced tossing path lengths by employing several techniques such as refined classification using additional attributes and intra-fold updates during training.

Finally, Servant and Jones [20] have presented a new technique which automatically selects the most appropriate developers for fixing the fault represented by a failing test case. Their technique is the first to assign developers to execution failures without the need for textual bug reports. On one hand, they do fault localization to map the current software bug to the corresponding line of code. On he other hand, they perform history mining to find out who has committed that buggy line of code. They have reported 81% of success (accuracy) for the top-three developer suggestions.

While their contribution is a valuable one, it is basically different than our point of view. Actually, they try to find the developer who has introduced a bug to the software, so that he himself fixes the bug too. In contrast, we believe that the developer who has caused a bug might not necessarily be the best one to resolve it.

V. Conclusion and Future Work

In this paper, we presented a novel approach to semi-automate the bug assignment task of human triagers in large open source projects using an expert recommender system. We create vocabulary-based expertise and interest model of developers based on the history of their contributions in resolving previous bug reports. Using a fast and efficient algorithm for approximate string matching, we store inverted lists of the mentioned vocabulary-based models and efficiently query them to find the most similar ones to a new bug report.

Our promising experimental results transparently show that our proposed approach is competitive with other approaches and our recommender system performs more efficiently than prior semi-automated bug triagers.

Our approach is novel in that it uses the n-gram-based string representation and works on the character level rather than on the token (term) level. Moreover, it performs approximate string matching rather than an exact match. Therefore, it is capable of handling spelling errors and different forms of writing, and it is extremely fast and efficient.

In the current implementation, before using the textual information of the summaries and descriptions of the bug reports, some kind of data preparation is performed. This step comprises setting all characters to lower case, removing all stop-words, and performing lemmatization. However, since we are working on the character-level rather than the token-level, one should further investigate whether these steps, especially removing the stop-words and lemmatisation are necessary in practice or not.

Also, one may use more textual information such as the archives of the mailing lists or the comments in the bug reports which are often produced during the bug tossing processes among the developers in order to make the expertise and interest models richer and thus, make the recommendation more efficient.

Like other related works, the implicit assumption is that for a resolved bug report, the person who has finally resolved the issue in practice is the one whose name is mentioned as the assignee of that bug report. While this assumption is in most cases true, there exist open source projects in which this is not always the case.

Similar to many other related works, one limitation of our approach is that it needs the history of previously resolved bug reports in order to be able to perform well. One future work could be to use the history of bug resolution for other similar products which are hosted in the same bug repository and have several developers in common with each other, in order to suggest bug resolvers for a new project which does not yet have such a rich bug resolution history.

Furthermore, the prototype which we have implemented currently works only with the Bugzilla open bug repositories [3], which support the XML-RPC protocol [14]. Extending our prototype to support more bug tracking systems and evaluating our approach with other open source projects could be done as future work.

Applying this approach on other parts of the bug triage problem such as duplicate bug reports detection remains as a possible future work. In addition, empirical research work is needed to find the best parameter configurations for the approximate string matching algorithm as well as the effect of those parameter values on the achieved values for the information retrieval evaluation metrics. Further, we believe that the scoring strategy which we have proposed in Section III is itself a matter of further research work. It might turn out that one could rank the databases based on the results of the similarity measures in a smarter way.

Finally, we plan to investigate deeper text exploration methods for our vocabulary-based modeling approach, namely to explore textual entailment technology to gain better text understanding. Textual entailment can be understood as a mechanism to deal with the variability of expressions, where the same meaning can be expressed by, or inferred from different texts. This might lead to a better degree of text similarity and hence, a better modeling and comparison of new and existing bug reports.

Acknowledgment

This work has been partially funded by the European Union (EU) project EXCITEMENT (FP7-IST) [21].

References

[1] J. Anvik, L. Hiew, and G. C. Morphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering (ICSE)*, 2006.

[2] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th international conference on Software engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 461–470.

[3] (2012, July) The official website of bugzilla. [Online]. Available: http://www.bugzilla.org/

[4] (2012, July) The official website of jira. [Online]. Available: http://www.atlassian.com/software/jira/overview/

[5] D. Cubranic and G. C. Murphy, "Automatic bug triage using text categorization," in *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE 2004)*, 2004.

[6] G. Canfora and L. Cerulo, "How software repositories can help in resolving a new change request," in *Proceedings of the workshop on Empirical Studies in Reverse Engineering*, 2005.

[7] J. Anvik and G. C. Morphy, "Determining implementation expertise from bug reports," in *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR)*, 2007.

[8] (2012, July) The official website of the apache subversion project. [Online]. Available: http://subversion.apache.org/

[9] (2012, July) The official website of cvs. [Online]. Available: http://www.nongnu.org/cvs/

[10] A. Mockus and J. D. Herbsleb, "Expertise browser: A quantitative approach to identifying expertise," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2002.

[11] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, ser. MSR '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 131–140.

[12] N. Okazaki and J. Tsujii, "Simple and efficient algorithm for approximate dictionary matching," in *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, Beijing, China, August 2010, pp. 851–859.

[13] (2012, July) Naoaki okazaki's website. [Online]. Available: http://www.chokkan.org/software/simstring/

[14] (2012, July) The xml-rpc website. [Online]. Available: http://xmlrpc.scripting.com/

[15] (2012, July) The official website of nltk. [Online]. Available: http://nltk.org/

[16] (2012, July) The official website of the eclipse jdt project. [Online]. Available: http://www.eclipse.org/jdt/

[17] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the 7th joint meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, 2009.

[18] O. Baysal, M. W. Godfrey, and R. Cohen, "A bug you like: A framework for automated assignment of bugs," in *Proceedings of the 17th IEEE International Conference on Program Comprehension (ICPC)*, 2009.

[19] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, ser. ICSM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[20] F. Servant and J. A. Jones, "Whosefault: automatic developer-to-fault assignment through fault localization," in *Proceedings of the 2012 International Conference on Software Engineering*, ser. ICSE 2012. Piscataway, NJ, USA: IEEE Press, 2012, pp. 36–46.

[21] (2012, July) The official website of project excitement. [Online]. Available: http://www.excitement-project.eu/

Using Normalized Systems Patterns as Knowledge Management

Peter De Bruyn, Philip Huysmans, Gilles Oorts,
Dieter Van Nuffel, Herwig Mannaert and Jan Verelst
Normalized Systems Institute (NSI)
University of Antwerp
Antwerp, Belgium
{*peter.debruyn,philip.huysmans,gilles.oorts,dieter.vannuffel,*
herwig.mannaert,jan.verelst}*@ua.ac.be*

Arco Oost
Normalized Systems eXpanders factory (NSX)
Antwerp, Belgium
{*arco.oost*}*@nsx.normalizedsystems.org*

***Abstract*—The knowledge residing inside a firm is frequently considered to be one of its most important internal assets to obtain a sustainable competitive advantage. Also in software engineering, a substantial amount of technical know-how is required in order to successfully deploy the organizational adoption of the technology. In this paper, we focus on the wide-spread approach of using design patterns for knowledge management purposes. It is discussed how they facilitate the transfer and (re)use of state-of-the-art knowledge in three dimensions: (1) more efficient documentation, (2) the development of new applications, and (3) the incorporation of new knowledge in existing applications. More specifically, we show how the use of Normalized Systems elements can be considered as an advanced form of design patterns for the development of highly evolvable software architectures, further enhancing the inherent design patterns advantages. Normalized Systems captures software engineering knowledge in a limited set of theorems and patterns, and enables the application of this knowledge through systematic pattern expansion. Because of the highly structured way of working, the reuse of knowledge can be significantly improved.**

***Keywords*-*Normalized Systems*; *Design Patters*; *Knowledge Management*.**

I. Introduction

As an important movement within the strategic management literature, the resource-based view of the firm (RBV) states that internal resources (e.g., money, patents, buildings, geographical location, etcetera) are the key elements for organizations in order to obtain a sustainable competitive advantage [1]. More specifically, the *knowledge* residing inside a firm is frequently considered to be its most important internal asset [2]. Further, focusing on the case of software adoption and development within organizations, the prevalence of the available knowledge becomes even more clear and the need for knowledge management practices in this respect have been acknowledged frequently [3]. Indeed, information technology in general can be considered as a knowledge-intensive or complex technology innovation, requiring a substantial amount of know-how and technical knowledge by the adopting firm [4]. As such, the degree of expertise or advanced knowledge of best-practices regarding a certain software technology becomes a decisive factor in the chances for an organization to successfully deploy and manage it. Consequently, a firm should either already (i.e., prior to the adoption) possess the advanced knowledge required to operate the software technology or engage in *organizational learning* during exploitation.

Organizational learning is generally regarded as the result of individual learning experiences of members of an organization, which become incorporated into the behavior, routines and practices of the organization the individuals belong to [4]. According to Levitt and March [5], such an organizational learning can occur in two general ways: (1) "learning by doing", which involves a learning process by self-experienced trial-and-error and (2) learning from the direct experiences of other people. While the first type of learning is typically a very profound and thorough way of knowledge gathering, it can be time-consuming, expensive and error-prone in the earliest stages. At this point, know-how, experiences and best-practices formulated by other users (i.e., the second type of organizational learning) come into play. Inside organizations, such knowledge transfers in software development can occur in many different ways, including for example explicit knowledge bases or experience repositories [6], "yellow pages" enabling search actions for accessible knowledgeable people [7] and mentoring programs [8]. At the inter-organizational or industrial level, the gathered knowledge can benefit from experience based on many different development projects. Design patterns are a wide-spread approach to achieve this goal [9]. In this paper, we explore three types of benefits when using knowledge captured by design patterns:

- Improved documentation;
- Using the captured knowledge to build new applications;
- and Incorporating new knowledge into existing applications.

We introduce the Normalized Systems (NS) theory, which proposes more concrete and structured patterns. We argue that the use of knowledge captured in such patterns can further enhance the discussed benefits of applying design patterns.

II. Knowledge Management in Software Engineering

The specific use of design patterns in object-orientation during the 90's, exemplified by the seminal work of Gamma et al. [10], was incited by the fact that modern computer literature regularly failed to make tacit (but success determining) knowledge regarding low-level principles of good software design explicit [11]. Patterns provide high-level solution templates for often-occurring problems. The patterns proposed by Gamma et al. [10] were conceived as the bundling of a set of generally accepted high-quality and best-practice solutions to frequently occurring problems in object-orientation programming environments. For instance, in order to create an one-to-many dependency between objects so that when the state of one object changes, all its dependents are notified and automatically updated, the observer pattern (i.e., an overall structure of classes giving a description or template of how to solve the concerned problem) was proposed [10]. As a consequence, the use of these patterns can be considered as specifically aimed at facilitating (inter-)organizational learning by learning from direct experiences of other people — in this case experienced software engineers —, and being one specific way of knowledge base distribution.

According to Schmidt [12], design patterns have been so successful because they explicitly capture knowledge that experienced developers already understand implicitly. The captured knowledge is called implicit because it is often not captured adequately with design methods and notations. Instead, it has been accumulated through timely processes of trial and error. Capturing this expertise allows other developers to avoid spending time rediscovering these solutions. Moreover, the captured knowledge has been claimed to provide benefits in several areas [13]. In this paper, we focus on the usage of patterns to (a) document software code, (b) build new applications, and (c) incorporate new knowledge in existing software applications.

A. Documentation

Patterns provide developers with a vocabulary which can be used to document a design in a more concise way [10], [13], [14]. For example, pattern-based communication can be used to preserve design decisions without elaborate descriptions. By delineating and naming groups of classes which belong to the same pattern, the descriptive complexity of the design documentation (e.g., a UML class diagram) can be reduced [14]. Consequently, the vocabulary offered by patterns allows a shift in the abstraction level of the discussions. This usage of design patterns is mostly applied at the conceptual level, and neglects the source code documentation. However, the abstract nature of patterns, i.e., as a solution template, means that it is possible to implement a certain design pattern using different alternatives. Therefore, it has been argued that the addition of source-code level documentation of the pattern usage is required to perform coding and maintenance tasks faster and with fewer errors [15].

B. Using knowledge to build new applications

Several authors propose the usage of design patterns to create new software applications (e.g., [16]). We discussed above how patterns provide high-level solution templates, and, as such, do not dictate the actual source code. Consequently, knowledge concerning the implementation platform remains important. A correct and efficient implementation of a design pattern requires a careful selection of language features [12]. Clearly, design patterns alone are not sufficient to build software. As a result, the implementation of a design pattern during a software development process remains essentially a complex and activity [12]. Developing software for a concrete application then requires the concrete experience of a domain and the specifics of the programming language, as well as the ability to abstract away from details and adhere to the structure prescribed by the design pattern. Nevertheless, certain companies and researchers attempt to integrate the knowledge available in design patterns in other approaches, in order to create automated code generation. For example, so-called software factories attempt to create software similar to automated manufacturing plants [17]. This should drastically improve software development productivity. However, such approaches have not yet reached wide-spread adoption.

C. Incorporating new knowledge in existing applications

Because of the increasing change in the organizational environment in which software applications are used, adaptability is considered to be an important characteristic. However, adapting software remains a complex task. Various studies have shown that the main part of the software development cost is spent after the initial deployment [18]. Several design patterns focus on incorporating adaptability into their solution template. Empirical observations have been reported which confirm the increased adaptability when using design patterns [19]. Adaptations could be made easier in comparison with an alternative which was programmed using no design patterns, and achieved adaptability was retained more successfully because of the prescribed structure. Nevertheless, some researchers also report negative effects on adaptability, caused by the added complexity of the design patterns. By prescribing additional classes in comparison to simpler solution, more errors have been introduced in some cases [19].

III. Normalized Systems

The Normalized Systems (NS) theory starts from the postulate that software architectures should exhibit *evolvability* due to ever changing business requirements, while many indications are present that most current software implementations do not conform with this evolvability requisite. Evolvability in this theory is operationalized as being the absence of so-called *combinatorial effects*: changes to the system of which the impact is related to the size of the system, not only to the kind of the change which is performed. As the assumption is made that software systems are subject to unlimited evolution (i.e., both additional and changing requirements), such combinatorial effects are obviously highly undesirable. In case changes are dependent on the size of the system and the system itself keeps on growing, changes proportional to the systems size become ever more difficult to cope with (i.e., requiring more efforts) and hence hampering evolvability. Normalized Systems theory further captures its software engineering knowledge by offering a set of four theorems and five elements, and enables the application of this knowledge through pattern expansion of the elements. The theorems consist of a set of formally proven principles which offer a set of necessary conditions which should be strictly adhered to, in order to obtain an evolvable software architecture (i.e., in absence of combinatorial effects). The elements offer a set of predefined higher-level structures, primitives or "building blocks" offering an unambiguous blueprint for the implementation of the core functionalities of realistic information systems, adhering to the four stated principles.

A. Theorems

Normalized Systems theory proposes four theorems, which have been proven to be necessary conditions to obtain software architectures in absence of combinatorial effects:

- *Separation of Concerns*, requiring that every change driver (concern) is separated from other concerns in its own construct;
- *Action Version Transparency*, requiring that data entities can be updated without impacting the entities using it as an input or producing it as an output;
- *Data Version Transparency*, requiring that an action entity can be upgraded without impacting its calling components;
- *Separation of States*, requiring that each step in a workflow is separated from the others in time by keeping state after every step.

In terms of knowledge management, as mentioned explicitly in [20], it must clearly be noted that the design theorems proposed are not new themselves; in fact, they relate to well-known (but often tacit or implicit) heuristic design knowledge of experienced software developers. For instance, well-known concepts such as an integration bus, a separated external workflow or the use of multiple tiers can all be seen as manifestations of the Separation of Concerns theorem [20]. As such, the added value of the theorems should then rather be situated in the fact that they (1) make certain aspects of that heuristic design knowledge explicit, (2) offer this knowledge in an unambiguous way (i.e., violations against the theorems can be proven), (3) are unified based on one single postulate (i.e., the need for evolvable software architectures having no combinatorial effects) and (4) have all been proven in a formal way.

B. Normalized Systems Elements as Patterns

The above stated theorems illustrate that typical software primitives do not offer explicit mechanisms to incorporate the principles. Also, the systematic application of the principles leads to a very fine-grained modular structure, which could form an additional design complexity on its own when performed "from scratch". Therefore, NS theory proposes a set of five elements as encapsulated higher-level patterns complying with the four theorems:

- *data elements*, being the structured encapsulation of a data construct into a data element (having get- and set-methods, exhibiting version transparency, etcetera);
- *action elements*, being the structured encapsulation of an action construct into an action element;
- *workflow elements*, being the structured encapsulation of software constructs into a workflow element describing the sequence in which a set of action elements should be performed in order to fulfill a flow;
- *connector elements*, being the structured encapsulation of software constructs into a connector element allowing external systems to interact with the NS system without calling components in a stateless way;
- *trigger elements*, being the structured encapsulation of software constructs into a trigger element controlling the states of the system and checking whether any action element should be triggered accordingly.

Each of the elements is a pattern as they represent a recurring set of constructs: besides the intended, encapsulated core construct, also a set of relevant cross-cutting concerns (such as remote access, logging, access control, etcetera) is incorporated in each of these elements. For each of the patterns, it is further described in [20] how they facilitate a set of anticipated changes in a stable way. In essence, these elements offer a set of building blocks, offering the core functionalities for contemporary information systems.

Regarding these patterns, it can be noted that their separate definition and identification is based on the implications of the set of theorems. For instance, the theorems Separation of Concerns and Separation of States indicate the need to formulate a workflow element next to an action element, in order to allow for the stateful invocation of action elements in a (workflow) construct other than action elements containing functional tasks. Next, each of the five patterns themselves contain knowledge concerning all the implications of the theorems referred to in Section III-A. Finally, each of these patterns has been described in a very detailed way. Consider for instance a data element in a JEE implementation [21]. In [20] it is discussed how a data element `Obj` is associated with a bean class `ObjBean`, interfaces `ObjLocal` and `ObjRemote`, home interfaces `ObjHomeLocal` and `ObjHomeRemote`, transport classes `ObjDetails` and `ObjInfo`, deployment descriptors and EJB-QL for finder methods. Additionally, methods to manipulate a data element's bean class (create, delete, etcetera) and to retrieve the two serializable transport classes are incorporated. Finally, to provide remote access, an agent class `ObjAgent` with several lifecycle manipulation and details retrieval methods is included. It can be argued that these elements incorporate the main concerns which are relevant for their function.

Moreover, the complete set of elements covers the core functionality of an information system. Consequently, as such detailed description is provided for each of the five elements, an NS application can be considered as an aggregation of a set of instantiations of the elements. Consider for example the implementation of an observer design pattern [10]. In order to implement this pattern in NS, three data elements (i.e., `Subscriber`, `Subscription` and `Notification`) are required. A `Notifier` connector element will observe the subject, and create instances of the `Notification` data element. These `Notification` data elements will be sent to every `Subscriber` that has a `Subscription` through a `Publisher` connector element. The sending is triggered by a `PublishEngine` trigger element which will periodically activate a `PublishFlow` workflow element. Consider that each (NS) element consists of around ten classes [22]. The seven identified elements therefore result in around seventy classes used to implement the design pattern, whereas the original implementation of the design pattern consists of two classes and two interfaces. Consequently, it is clear that, in order to prevent combinatorial effects, a very fine-grained modular structure needs to be adhered to.

C. Pattern Expansion

As stated before, in practice, the very fine-grained modular structure implied by the NS principles seems very unlikely to arrive at without the use of higher-level primitives or patterns. Consequently, as NS proposes a set of five elements which serve for this purpose, the actual software architecture of NS conform software applications can actually be generated relatively straightforward. For example, in case of the data element pattern structure, the pattern expansion mechanism would need a set of parameters including the basic name of the data element (e.g., `Invoice`), context information (e.g., component and package name) and data field information (e.g., data type). Next, based on these parameters, the pattern expansion mechanism will generate the predefined structured (i.e., the set of classes and data fields) as illustrated above: the bean class `InvoiceBean`, interfaces `InvoiceLocal` and `InvoiceRemote`, etcetera.

However, in terms of knowledge management, it should be noted that the patterns and the expansion mechanism should not be considered as separate knowledge reuse mechanisms: rather, the pattern expansion facilitates the re-use of knowledge embedded in the patterns, as each expansion of the patterns results in a new application of the knowledge encapsulated in the pattern. Through this, pattern expansion facilitates both types of learning discussed earlier (i.e., "learning by doing" and learning from experience of other people) by utilizing the knowledge contained in the patterns.

Also, the information codified in a pattern may not be sufficient to adequately transfer the intended knowledge. This was already the case when using the design patterns proposed by Gamma et al. [10]. For example, it has been claimed that the *Dependency Inversion Principle* helps to gain a better understanding of the *Abstract Factory* pattern [23]. Similarly, the structure of the NS patterns can only be understood when the NS theorems are taken into account.

IV. Normalized Systems Patterns as Knowledge Management

In the previous sections, we explained how traditional design patterns entail knowledge management related benefits regarding documentation, the development of new applications and the adaptation of existing applications. We also argued how NS patterns represent a fine-grained modular structure which can be expanded to provide an evolvable software architecture. In this section, we discuss how the use of NS patterns seems to even further enhance the reported design pattern benefits, when compared to design patterns.

A. Documentation

As NS-compliant applications based on the NS elements have basically five recurrent elementary structures, only these five elements have to be understood to grasp the structure of each instantiated element throughout the application. Because the patterns are detailed enough to be instantiated, no manual implementation of the patterns (as is the case with the design patterns proposed by Gamma et al. [10]) is required. Consequently, an identical code structure reoccurs in every application which is created using the NS expanders. The commonality of the structure of the patterns makes that once one understands the patterns, one understands all its instantiations as well. In this way, it could be argued that — at least partially — the pattern structure becomes the documentation. Therefore, no source code level documentation is required.

B. Using knowledge to build new applications

As (1) each violation of the NS theorems during any stage of the development stage results in a combinatorial effect, and (2) the systematic application of these theorems results in very fine-grained structures, it becomes extremely challenging for a human developer to consistently obtain such modular structures. Indeed, the fine-grained modular structure might become a complexity-issue on its own. In this sense, the NS patterns might offer the necessary simplification by offering pre-constructed structures ("building blocks"), which can be parameterized during implementation efforts. This way the NS patterns do dictate the source code for implementing the pattern, contrary to the patterns of Gamma et al. [10].

An important characteristic of these structures is that they separate technology-dependent aspects from the actual implementation, resulting in the fact that one can easily switch the underlying technology stack of the software. One transition that has been performed, is changing the underlying implementation architecture from EJB 2 to EJB 3. Because these standards use a different way of communicating between agents and beans, this transition normally is a labor-intensive and difficult task. Using the architecture described in this paper, this transition can however be achieved rather easily by using the pattern expansion mechanism. This is because the expanders that perform the expansion are very similar for different technologies. This is done by clearly separating functional requirements of the system (i.e., input variables, transfer functions and output variables) from constructional aspects of the system (i.e., composition of the system). Whereas all constructional aspects are described in patterns and expanders, functional aspects are separately included in descriptor files (such as data elements, action elements, etc.). As each pattern can be conceived a recurring structure of programming constructs in a particular programming environment (e.g., classes), one can conclude that the functional/constructional transformation then becomes located at one abstraction level higher than before.

C. Incorporating new knowledge in existing applications

The purpose is to easily incorporate new knowledge of improvements, intrinsically by the use of the elements. This can be interpreted from two distinct perspectives. First, improvements or changes (e.g., typical bug fixing or a new kind of algorithm) regarding the actual functional parts of the system (i.e., the so-called 'tasks') are easily to be incorporated in the whole system as the properly separated change driver is the only place where any modifications have to be made and the remainder of the system can easily interact with the new task (and hence, use this knowledge). In NS terms, we could call these kind of changes and expertise inclusions, knowledge dispersion at the "*sub-modular level*" as only changes and new knowledge are incorporated at the sub-modular level of the tasks (and not in the modular structure of the elements). Second, however, knowledge can be incorporated at the "*modular level*" as well. This kind of knowledge inclusion would include change (e.g., an extra separated class in the pattern) and modifications (e.g., improved persistence mechanism) regarding the internal structure of an element (the pattern). Indeed, once the basic structure or cross-cutting concern implementation of an element is changed due to a certain identified need or improvement, the new best-practice knowledge can be expanded throughout the whole (existing) modular structure and used for new (i.e., additional) instantiations of the elements. In order to further illustrate this second kind of knowledge dispersion based on NS patterns, consider the following example, based on real-life experience from developers using NS.

For instance, one way to adopt a model-view-controller (MVC) architecture in a JEE distributed programming environment is by adopting (amongst others) the Struts framework. In such MVC architecture, a separated controller is responsible for handling an incoming request from the client (e.g., a user via a web interface) and will invoke (based on this request) the appropriate model (i.e., business logic) and view (i.e., presentation format), after which the result will eventually be returned to the client. Struts is a framework providing the controller (ActionServlet) and enabling the creation of templates for the presentation layer. Obviously, security issues need to be handled properly in such architecture as well. Applied to our example, these security issues in Struts were handled in the implementation of the Struts Action itself in a previous implementation of our elements. In other words, the implementation class itself was responsible for determining whether or not a particular operation was allowed to be executed (based on information such as the user's access rights, the screen in which the action was called, etcetera). As such, this "security function" became present in all instantiations of an action element type (i.e., each session). Moreover, this resulted in a combinatorial effect as the impact of a change such as switching towards an equivalent framework (i.e., handling similar functions as Struts), would entail a set of changes dependent on the number of instantiated action elements (and hence, on the size of the system). In order to solve the identified combinatorial effect, the Separation of Concern theorem has to be applied: separating the part of the implementation class responsible for the discussed security issues (i.e., a separate change driver) in its own module within the action element. In our example, a separate interceptor module was implemented, next to the already existing implementation class. This way, not only the combinatorial effect was excluded, but the new knowledge in terms of a separate interceptor class was applied to all action elements after isolating the relevant implementation class parts and executing the pattern expansion. Additionally, all new applications using the new action element structure automatically incorporate this new knowledge.

Hence, compared to traditional design patterns, the NS patterns offer a formally proven evolvable software architecture as well as an convenient knowledge distribution mechanism.

V. Conclusion and Future Work

In this paper, we indicated the application and usefulness of patterns in software development. It was also shown that Normalized Systems theory can readily be considered as a method of building stable and large-scale information systems. Furthermore it has been demonstrated how Normalized Systems theory uses patterns to facilitate the transfer and use of knowledge on software development. But far most we showed in this paper that the NS elements can be considered to be enhanced patterns for software development with benefits on three dimensions (i.e., less need for explicit documentation, more deterministic development of new applications and more convenient incorporation of new knowledge into existing applications). From interviews with developers, these benefits have shown to enhance the transfer of knowledge, success rate and the overall quality of NS developments. Although the discussion in this paper was limited to Normalized Systems theory for software, the theory has recently been applied to both Business Process Management and Enterprise Architecture domains. As part of future research, the possible formulation of patterns on the level of business processes and enterprise architecture will be studied.

Acknowledgment

P.D.B. is supported by a Research Grant of the Agency for Innovation by Science and Technology in Flanders (IWT).

References

[1] B. Wernerfelt, "A resource-based view of the firm," *Strategic Management Journal*, vol. 5, no. 2, pp. 171–180, 1984.

[2] R. M. Grant, "Toward a Knowledge-Based Theory of the Firm," *Strategic Management Journal*, vol. 17, pp. 109–122, 1996.

[3] F. O. Bjørnson and T. Dingsøyr, "Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used," *Information and Software Technology*, vol. 50, no. 11, pp. 1055–1068, 2008.

[4] P. Attewell, "Technology diffusion and organizational learning: The case of business computing," *Organization Science*, vol. 3, no. 1, pp. 1–19, 1992.

[5] B. Levitt and J. G. March, "Organizational learning," *Annual Review of Sociology*, vol. 14, pp. 319–340, 1988.

[6] C. Chewar and D. McCrickaerd, "Links for a human-centered science of design: integrated design knowledge environments for a software development process," in *Proceedings of the Hawaii International Conference on System Sciences*, 2005.

[7] T. Dingsøyr, H. K. Djarraya, and E. Røyrvik, "Practical knowledge management tool use in a software consulting company," *Communications of the ACM*, vol. 48, no. 12, pp. 96–100, 2005.

[8] F. Bjørnson and T. Dingsøyr, "A study of a mentoring program for knowledge transfer in a small software consultancy company," in *Product Focused Software Process Improvement*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005, vol. 3547, pp. 245–256.

[9] I. Rus and M. Lindvall, "Knowledge management in software engineering," *IEEE Software*, vol. 19, pp. 26–38, 2002.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Professional, 1994.

[11] J. Coplien, "The culture of patterns," *Computer Science and Information Systems*, vol. 1, no. 2, pp. 1–26, 2004.

[12] D. C. Schmidt, "Using design patterns to develop reusable object-oriented communication software," *Commun. ACM*, vol. 38, no. 10, pp. 65–74, Oct. 1995.

[13] D. Riehle, "Transactions on pattern languages of programming ii," J. Noble and R. Johnson, Eds. Berlin, Heidelberg: Springer-Verlag, 2011, ch. Lessons learned from using design patterns in industry projects, pp. 1–15.

[14] G. Odenthal and K. Quibeldey-Cirkel, "Using patterns for design and documentation," in *ECOOP*, 1997, pp. 511–529.

[15] L. Prechelt, B. Unger-Lamprecht, M. Philippsen, and W. F. Tichy, "Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance," *IEEE Trans. Softw. Eng.*, vol. 28, no. 6, pp. 595–606, 2002.

[16] C. Larman, *Applying UML and Patterns*. Prentice Hall, 1997.

[17] J. Greenfield and K. Short, "Software factories: assembling applications with patterns, models, frameworks and tools," in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ser. OOPSLA '03, 2003, pp. 16–27.

[18] R. L. Glass, "Maintenance: Less is not more," *IEEE Software*, vol. 15, no. 4, pp. 67–68, 1998.

[19] L. Prechelt, B. Unger, W. Tichy, P. Brossler, and L. Votta, "A controlled experiment in maintenance: comparing design patterns to simpler solutions," *Software Engineering, IEEE Transactions on*, vol. 27, no. 12, pp. 1134 –1144, dec 2001.

[20] H. Mannaert, J. Verelst, and K. Ven, "Towards evolvable software architectures based on systems theoretic stability," *Software: Practice and Experience*, vol. 42, pp. 89–116, 2011.

[21] Oracle. Java platform, enterprise edition. [Online]. Available: http://www.oracle.com/technetwork/java/javaee/overview/index.html

[22] H. Mannaert and J. Verelst, *Normalized systems: re-creating information technology based on laws for software evolvability*. Koppa, 2009.

[23] L. Welicki, J. Manuel, C. Lovelle, and L. J. Aguilar, "Patterns meta-specification and cataloging: towards knowledge management in software engineering," in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, ser. OOPSLA '06, 2006, pp. 679–680.

An UML-based Authoring Approach of S1000D Procedural Data Modules and Tool Support

Youhee Choi, Jeong-Ho Park, Byungtae Jang, DongSun Lim
Vehicle&Defense-IT Convergence Research Department, ETRI
Daejeon, KOREA
e-mail: {yhchoi, parkjh, jbt, dslim}@etri.re.kr

***Abstract*— The S1000D specification is developed as the standard format to describe technical publications in aerospace and military field. The S1000D supports systematically classifying technical information about various equipments in XML format. Most technical information about equipments contains procedural information about installation, operation, and maintenance. The procedural information can be effectively described using graphical description methods rather than using textual description like XML. Also, In UML, activity diagrams can be used to describe the business and operation step-by-step workflows. In the S1000D, there are many types of data modules and procedural data modules regarding procedural information. In this paper, we propose an approach to authoring S1000D procedural data modules using UML.**

Keywords- S1000D; XML; UML; Shipdex.

I. INTRODUCTION

The “S1000D International Specification for technical publication utilizing a common source database” is an international specification for the procurement and production of technical publications [1]. The S1000D covers technical publication activities in support of any airline projects and military projects. However, in the shipbuilding field, most technical manuals are supplied today on paper or in different formats, different structures and different data quality. This situation caused several problems in terms of information comprehension and electronic usage. Therefore, to utilize enterprise resource planning (ERP) and manage various technical manuals effectively, a common and standardized protocol for exchanging technical data was deemed necessary. For this reason, some European shipping companies agreed to develop the Shipdex protocol that is a common and shared data exchange protocol based on ASD S1000D issue 2.3 [2]. The Shipdex protocol allows data exchange, update, and search by standardizing technical information publication format.

The S1000D that is the basis of the Shipdex defines a “Data Module” which is defined as “the smallest self contained information unit” and a data module is described in XML format. XML is a markup language that defines a set of rules for encoding documents [3]. It is a textual data format with strong support via Unicode. XML was designed to carry data, not to display data and XML tags are not predefined. Naturally, there are various XML editors that have facilities like tag completion and on-the-fly XML validation with XML schema [4, 5, 6, 7]. Also, some XML editors allow rendering XML documents using XSLT stylesheets to show them close to the final output. However, producing an XML document using these XML editors requires learning XML schemas or DTDs. In the same way, due to these features of XML, it means that producing a S1000D data module requires pre-learning the S1000D schemas or tags. Also, it is difficult to understand data contained in data modules for non-technical users without transforming data modules supplied in XML format into other format. In addition, there are many studies to model XML with UML[10, 11, 12, 13, 14]. However, the existing researches focus on representing static structures of XML schema elements. There are few studies to model contents of XML using UML in view of semantics of XML elements.

We note that most technical publications for equipments are manuals that contain procedural information about installation, operation, and maintenance. The procedural information can be effectively described using graphical description methods. In this respect, we propose an approach to authoring a S1000D procedural data module using behavioral modeling method of UML that is the de facto standard modeling language. Applying our approach to describing S1000D procedural data modules, it allows to easily produce S1000D data modules and to easily understand contents of the S1000D data modules without knowing XML DTD schema and tags.

The remainder of this paper is organized as follows. The outline of the S1000D and the Shipdex is presented in Section 2. Section 3 presents our approach to authoring S1000D XML data using UML. The approach is applied to an example in section 4. The paper concludes with future work and conclusions in section 5.

II. S1000D AND SHIPDEX

A. *S1000D*

The S1000D is an international standard for the production and procurement of technical publications. It has been initially developed by the AeroSpace and Defence Industries Association of Europe (ASD). In the S1000D, a data module is the smallest information unit. Data modules may be stored and managed in the CSDB(Common Source Data Base). The purpose of the CSDB is to manage the data modules so that information is not duplicated, link relationships are maintained, and version control is applied to content [8]. Because S1000D is modular, it facilitates content reuse and seamless data interchange between organizations [9]. There are many data module types which are appropriate for use in the production of all technical information required in operation and maintenance of the product. The S1000D defined various types of data modules- Descriptive, Procedural, Crew/Operator, Fault information, Maintenance planning, Illustrated parts data, Process, Wiring data, Wiring data description, Technical repository, Container, Product cross-reference table, Technical conditions cross-reference table, Business rules exchange. A data module has a basic structure which is comprised of two sections:

•Identification and status (IDSTATUS) section

•Content section

The IDSTATUS section of all data modules contains identification data (data module code, title, issue number, issue date, language) and status data (security classification, responsible partner company and originator, applicability, technical standard, quality assurance status, skill, reason for update). The Content section of a data module must be structured in accordance with data module types. The Content section of a procedural data module that we focus on contains the following elements.

•Data module title (<dmtitle>)

•Table of contents

•References (<refs>)

•Preliminary requirements (<prelreqs>) including safety conditions(<safety>)

•Procedure (<step>)

•Requirements after job completion (<closereqs>)

B. *Shipdex*

The Shipdex protocol is the international business rules developed to standardize the development and the exchange of technical and logistic data within the shipping community. It applies to ASD S1000D at issue 2.3. The Shipdex protocol has been developed by the following companies- Grimaldi Compagnia di Navigazione s.p.a, Intership Navigation Co. Ltd., Alfa Laval, MacGREGOR a part of Cargotec Group, MAN Diesel & Turbo, SpecTec Group Holdings Ltd., Yanmar Co. Ltd. The scope of this protocol is to cover the data exchange related to the information currently supplied in the form of technical manuals. The reason to develop the protocol is that shipping companies are receiving from manufactures technical manuals in different formats, different structures and different data quality. The data module types that the Shipdex protocol makes use are Descriptive, Procedural, and Illustrated parts data (IPD).

C. *Modeling XML schemas with conceptual models*

Several approaches of modeling XML schemas by using existing conceptual models, such as ER, UML have been proposed [10, 11, 12, 13, 14]. The goals of most researches are to represent a XML schema using a UML class diagram, even if the modeler has no familiarity with the XML schema syntax. In this respect, the authors have used the UML extension mechanisms to represent XML schema elements. They focus on representing static structures and data relationships of XML schema elements. The existing researches hardly consider semantics of values which would be represented using XML schema elements.

III. AN UML-BASED AUTHORING APPROACH OF S1000D PROCEDURAL DATA MODULES AND TOOL SUPPORT

A. *Process for authoring S1000D procedural data modules*

This section describes the process for authoring the content section of S1000D procedural data modules as shown in Fig. 1.

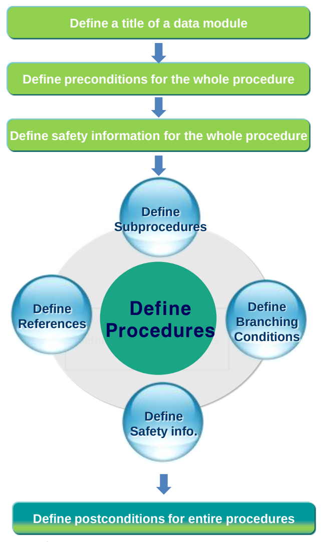

Define a title of a data module
Define preconditions for the whole procedure
Define safety information for the whole procedure
Define Subprocedures
Define References
Define Procedures
Define Branching Conditions
Define Safety info.
Define postconditions for entire procedures

Figure 1 The process for authoring the S1000D procedural data modules

First of all, the title of a data module that represents the whole procedure must be defined. Second, preconditions for the whole procedure that are carried out before starting the procedure must be defined. Third, safety information such as warnings, cautions for the whole procedure must be defined. Then, the main procedure must be defined. To define the main procedure, a subprocedure that is a unit of an action must be defined. For each subprocedure, if they exist, referenced elements that the subprocedure refers and safety information that is related to the subprocedure should be defined. Then, if conditional flows are needed, branching conditions that determine which actions are carried out should be defined. Lastly, postconditions for the whole procedure must be defined.

B. Method for describing S1000D procedural data modules

This section describes the method for describing the content section of S1000D procedural data modules using UML. The procedural data module is used to describe procedural information. In the UML, activity diagrams can be used to describe the business and operation step-by-step workflows of components in a system [15]. Thus, we suggest using activity diagrams to describe procedural data modules.

Major elements and attributes of the procedural data module can be classified as shown in Table I.

TABLE I. MAJOR ELEMENTS AND ATTRIBUTES OF THE PROCEDURAL DATA MODULE

Procedural data module DTD/Schema			
Element	***Subelement***	***Subelement***	***Attribute***
<dmtitle>	<techname>		
<refs>	<refdm> or <reftp> <avee> or <pubcode> <dmtitle> or <pubtitle> <issno> or <pubdate>		
<prelreqs>	<supequip> <supplies> <spares>	<nomen> <qty>	id uom
<safety>	<warning> <caution>		
<step>			
<closereqs>			
<xref>			xidtype xrefid

Firstly, the 'Data module title (<dmtitle>)' must give meaning to identification of the product and it has the mandatory element <techname>. The content of the element <techname> must reflect name of the hardware or function. Since a procedural data module can be described as a unit of activity diagram, the element <techname> can be described with the name of the activity diagram or an activity that represents the whole procedure without extending the UML.

Secondly, the 'Table of contents' element doesn't contain specific contents of the data module. Thus, we can exclude it.

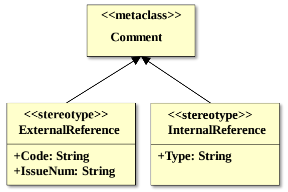
Thirdly, in case of the 'References' element, there are two types of references used in the S1000D.

•Internal references: References to other places within the same data module(<xref>)

•External references: References to other data modules (<refdm>) or other technical publications (<reftp>)

The former (<xref>) type of references corresponds to Figures, tables, multimedia, procedures (steps), and so on. Although, in the DTD/Schema, this type is optional, it gives the detail information about the referenced targets. In this respect, it is necessary to describe this type of references, so identification information about the referenced target should be described. Thus, we define the stereotype <<InternalReference>> by extending UML Comment element with tags that represent ID and type of the referenced target as shown in Fig. 2.

In case of the latter (<refdm> or <reftp>) type of references, these references are the mandatory elements and they should be presented on the references table. The table presents the data module/technical publication code, data module/technical publication title, and issue number. Thus, we define the stereotype <<ExternalReference>> by extending UML Comment element with tag definitions that represent code, title, and issue number as shown in Fig. 2.



S1000D				UML element		
Element	Subelement	Subelement	Attribute	Stereotype	Tag	Properties
<refs>	<refdm> or <reftp>			ExternalReference		
		<dmtitle> or <pubtitle>				body
		<avee> or <pubcode>			Code	
		<issno> or <pubdate>			IssueNum	
<xref>				InternalReference		
			xrefid			body
			xidtype		Type	

Figure 2 S1000D references element vs. UML elements

In turn, the 'Preliminary requirements' element consists of the following sub-elements.

•Required Conditions: actions to be done and/or conditions that must be satisfied before doing procedure(<prelreqs>) and any actions that are required after the procedure is complete(<closereqs>)

•Support equipment: A list of any support equipment including special tools, required to accomplish the procedure(<supequip>)

•Supplies: A list of any consumables, materials and expendables required to accomplish the procedure(<supplies>)

•Spares: A list of any spares required to accomplish the procedure(<spares>)

•Safety: A list of any safety requirements(<safety>)

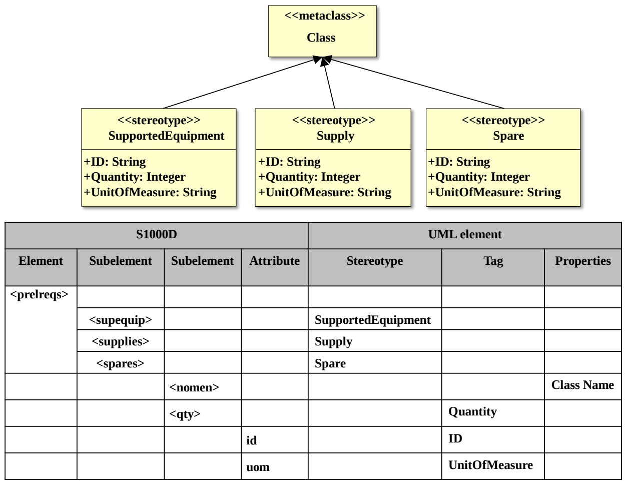
Firstly, in case of the 'Required Conditions', there are two types of Required Conditions.

•Required Conditions with no reference(<reqcond>)

•Required Conditions with external references(<reqcondm>, <reqcontp>)

In case of the former (<reqcond>) type of the 'Required Conditions', since the semantics of the sub-element (<prelreqs>) corresponds to the semantics of the pre-defined stereotype <<precondition>> for the activity, the sub-element (<prelreqs>) can be described with the stereotype <<precondition>> for the activity that represents the whole procedure. In case of the latter (<reqcondm>, <reqcontp>) type of the 'Required Conditions', we define the stereotype <<PreconditionRef>> for precondition and the stereotype <<PostconditionRef>> for postcondition by extending the UML Comment element. In addition, relationships between 'References' and 'Required Conditions' can be described as the linkage between the UML Comment element with the stereotype <<PreconditionRef>> or <<PostconditionRef>> and the UML Comment element with the stereotype <<ExternalReference>>.

Secondly, in case of the sub-elements (<supequip>, <supplies>, <spares>), each element is mandatory in procedural data modules and can be referenced within the same data module. And each element has a <nomen> element which indicates the functional item nomenclature and a <qty> element which is used to identify the quantity of items. Also, each element has an 'id' attribute and an 'uom' attribute that indicates the unit of measure. To describe each element, we define the stereotype <<SupportedEquipment>>, <<Supply>>, and <<Spare>> by extending the UML Class element with tags that indicate ID, the quantity of items, and the unit of measure as shown in Fig. 3.



S1000D				UML element		
Element	Subelement	Subelement	Attribute	Stereotype	Tag	Properties
<prelreqs>						
	<supequip>			SupportedEquipment		
	<supplies>			Supply		
	<spares>			Spare		
		<nomen>				Class Name
		<qty>			Quantity	
			id		ID	
			uom		UnitOfMeasure	

Figure 3 S1000D 'Preliminary requirements' elements vs. UML elements

In case of the sub-element (<safety>), the type of safety conditions can be warnings or cautions. Thus, we define the stereotype <<Warning>> and <<Caution>> by extending the UML Comment element and body of the comment can be safety conditions as shown in Fig. 4.

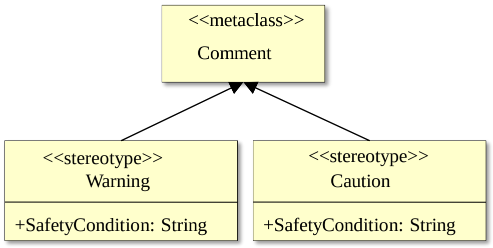


Figure 4 The stereotypes for S1000D 'Safety' elements

In case of the 'Procedure (<step>)', each sub-procedure (step) can be described with an UML Activity or an UML Action element. If a step is broken down into sub-steps, this step should be described with an UML Activity element that can include other activities that are sub-steps. Otherwise, a step can be described with an UML Action element. In addition, conditional flows of procedure should be described. First of all, to describe conditions, the type of condition should be classified. If the condition affects selecting resources that are required to accomplish a procedure, the condition can be described using an UML Comment element with the stereotype <<selection>> on the UML ObjectFlow. Otherwise, the condition affects selecting the next procedure (step), the condition can be described using an UML DecisionNode.

Finally, since the semantics of the 'Requirements after job completion (<closereqs>)' corresponds to the semantics of the pre-defined stereotype <<postcondition>> for the activity, the element (<closereqs>) can be described with stereotype <<postcondition>> for the activity that represents the whole procedure.

Table II shows that each rows' items of the left part (S1000D) can be described using the right part (UML)'s items .

TABLE II. SUMMARIES OF S1000D ELEMENTS VS. UML ELEMENTS

S1000D				UML		
Element	Subelement	Subelement	Attribute	Element	Stereotype	Tag
<dmtitle>	<techname>			Activity name		
<refs>	<refdm> or <reftp>			Comment	ExternalReference	
		<dmtitle> or <pubtitle>		Comment body		
		<avee> or <pubcode>				Code
		<issno> or <pubdate>				IssueNum
<xref>				Comment	InternalReference	
			xrefid	Comment body		
			xidtype			Type
<prelreqs>	<reqcond>			Activity	precondition	
	<reqcondm> or <reqcontp>			Comment	PreconditionRef	
	<supequip>			Class	SupportedEquipment	
	<supplies>			Class	Supply	
	<spares>			Class	Spare	
		<nomen>		Class name		
		<qty>				Quantity
			id			ID
			uom			UnitOfMeasure
<step>				Activity or Action		
	Conditional flow			DecisionNode or Comment with the stereotype <<selection>>		
<safety>	<warning>			Commnet	Warning	
	<caution>			Comment	Caution	
<closereqs>	<reqcond>			Activity	postcondition	
	<reqcondm> or <reqcontp>			Comment	PostconditionRef	

C. *Transformation rules*

This section describes the rules that transform from an UML model to a S1000D procedural data module XML file according to the proposed UML metamodel.

•The name of the outermost Activity can be transformed into the <techname> value of the <dmtitle> tag.

•The body of the stereotype <<precondition>> can be transformed into the <reqcond> values of the <prelreqs>/<reqconds> tag.

•The body of the stereotype <<postcondition>> can be transformed into the <reqcond> values of the <closereqs>/<reqconds> tag.

•In case of the stereotype <<PreconditionRef>> or <<PostconditionRef>>, the body of the stereotype <<PreconditionRef>> or <<PostconditionRef>> can be transformed into a value of the <reqcond> tag and if the tag <code> value of the linked <<ExternalReference>> element is a data module code, it can be transformed as follows:

<reqconds>*<reqcondm>*<reqcond>….</reqcond> *<reqdm><avee>* *….</avee></reqdm></recondm>*</reqconds>

Otherwise, it can be transformed as follows:

<reqconds>*<reqcondtp>*<reqcond>……</reqcond> *<reqtp>……</reqtp></recondtp>*</reqconds>

•In case of the stereotype <<ExternalReference>>, if the tag <code> value is a data module code, it can be transformed as follows:

<refs>*<reftp>……</reftp>*</refs>

Otherwise, it can be transformed as follows:

<refs>*<refdm>……</refdm>*</refs>

Also, if the <<ExternalReference>> element is linked with Activity or Action element, it can be transformed as follows:

<step~><para> ……. *<reftp>……</reftp>*</para></step~>

or

<step~><para> ……. *<refdm>……</refdm>*</para></step~>

•In case of the stereotype <<InternalReference>>, the part that the <<InternalReference>> element is used can be transformed as follows:

<step~><para>*…<xref xrefid=…. xidtype=…>…*</para></step~>

•In case of the stereotype <<Warning>> or <<Caution>>, if the element is linked with the outermost Activity, it can be transformed as follows:

<safety>*<safecond><caution>……</caution></safecond>*</safety>

or

<safety>*<safecond><warning>……</warning></safecond>*</safety>

If the element is linked with other Activity elements or Action elements, it can be transformed as follows:

<step~><warning>……</warning></step~>

•In case of Activity elements and Action elements, it can be transformed as follows:

<step~><para>[Activity body|Action body]</para></step~>

If the source end of an incoming control flow is the UML DecisionNode, it can be transformed as follows:

<step~>
<para>[guard condition of the control flow], [Activity body|Action body]</para>
</step~>

D. Structure of the S1000D Procedural Data Module Authoring Tool

This section shows the structure of the S1000D Procedural Data Module Authoring Tool. The tool supports automatically generating major contents of a S1000D procedural data module XML file by modeling an UML activity diagram.

Fig. 5 shows the relationships between subblocks of the tool and their artifacts. First of all, contents of primary S1000D procedural elements can be defined using a S1000D procedural data module editor even though not knowing the particular S1000D procedural data module structure or the UML diagram syntax. Then, an UML model generator generates an UML activity diagram on the basis of the primary S1000D procedural elements' contents. An UML model property editor allows a user to define additional specific properties of the UML activity diagram that don't have to be externally represented. A S1000D procedural data module XML file generator generates a S1000D procedural data module XML file on the basis of the UML activity diagram. A S1000D procedural data module XML file editor allows a user to define optional elements' contents.

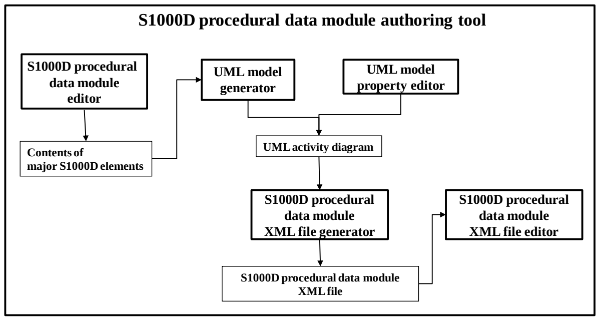

S1000D procedural data module authoring tool
S1000D procedural data module editor
UML model generator
UML model property editor
Contents of major S1000D elements
UML activity diagram
S1000D procedural data module XML file generator
S1000D procedural data module XML file editor
S1000D procedural data module XML file

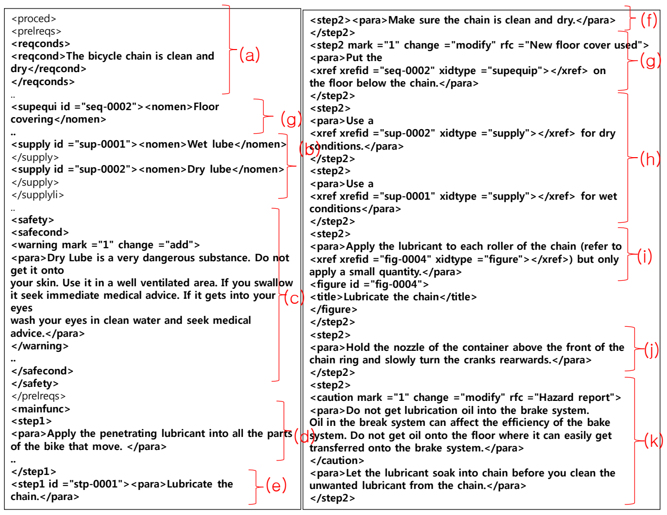
Figure 5. Subblocks of S1000D procedural data module authoring tool

IV. An Example

To address the practical applicability and features of our approach, we have chosen the procedure of lubricating the bicycle chain that is a typical example of the S1000D specification.

Fig. 6 shows the mark-up example of the S1000D procedural data module. In Fig. 6, the part (a) indicates the sub-element (<prelreqs>) of the 'Preliminary requirements' element. The part (b) of Fig. 6 indicates the sub-element (<supplies>) of the 'Preliminary requirements' element. The part (c) of Fig. 6 indicates the sub-element (<safety>) of the 'Preliminary requirements' element. The parts (d) ~ (k) of Fig. 6 indicate the 'Procedure (<step>)' elements.

Fig. 7 shows the UML activity diagram that describes the major elements of the mark-up example in Fig. 6. The part (a) of Fig. 6 can be described using the pre-defined stereotype <<precondition>> as shown in the part (a) of Fig. 7. The part (b) of Fig. 6 can be described using the UML Class element with the stereotype <<Supply>> as shown in the part (b) of Fig. 7. The part (c) of Fig. 6 can be described using the UML Comment element with the stereotype <<Warning>> as shown in the part (c) of Fig. 7. As shown in the part (d) of Fig. 7, the part (d) of Fig. 6 can be described using the UML Action element because it doesn't have any sub-step. On the other hand, since the part (e) has sub-steps, it can be described using the UML Activity element as shown in Fig. 7. In case of the part (g) of Fig. 7, since the step needs the supported equipment (Floor covering), the step can be described using the UML action element linking with the stereotyped object (<<SupportedEquipment>>). The part (h) of Fig. 7 shows that the step contains the condition affects determining the next procedure (step). As shown in the part (h) of Fig. 7, the condition can be described using an UML DecisionNode. The part (i) of Fig. 7 shows that the step with the internal reference can be described using an UML action element linking with the stereotyped comment element (<<InternalReference>>). The part (k) of Fig. 7 shows that safety condition can be described using an UML Comment element with the stereotype <<Caution>>.



```
<proced>
<prelreqs>
<reqconds>
<reqcond>The bicycle chain is clean and
dry</reqcond>
</reqconds>
..
<supequi id ="seq-0002"><nomen>Floor
covering</nomen>
..
<supply id ="sup-0001"><nomen>Wet lube</nomen>
</supply>
<supply id ="sup-0002"><nomen>Dry lube</nomen>
</supply>
</supplyli>
..
<safety>
<safecond>
<warning mark ="1" change ="add">
<para>Dry Lube is a very dangerous substance. Do not
get it onto
your skin. Use it in a well ventilated area. If you swallow
it seek immediate medical advice. If it gets into your
eyes
wash your eyes in clean water and seek medical
advice.</para>
</warning>
..
</safecond>
</safety>
</prelreqs>
<mainfunc>
<step1>
<para>Apply the penetrating lubricant into all the parts
of the bike that move. </para>
..
</step1>
<step1 id ="stp-0001"><para>Lubricate the
chain.</para>
```

```
<step2><para>Make sure the chain is clean and dry.</para>
</step2>
<step2 mark ="1" change ="modify" rfc ="New floor cover used">
<para>Put the
<xref xrefid ="seq-0002" xidtype ="supequip"></xref> on
the floor below the chain.</para>
</step2>
<step2>
<para>Use a
<xref xrefid ="sup-0002" xidtype ="supply"></xref> for dry
conditions.</para>
</step2>
<step2>
<para>Use a
<xref xrefid ="sup-0001" xidtype ="supply"></xref> for wet
conditions</para>
</step2>
<step2>
<para>Apply the lubricant to each roller of the chain (refer to
<xref xrefid ="fig-0004" xidtype ="figure"></xref>) but only
apply a small quantity.</para>
<figure id ="fig-0004">
<title>Lubricate the chain</title>
</figure>
</step2>
<step2>
<para>Hold the nozzle of the container above the front of the
chain ring and slowly turn the cranks rearwards.</para>
</step2>
<step2>
<caution mark ="1" change ="modify" rfc ="Hazard report">
<para>Do not get lubrication oil into the brake system.
Oil in the break system can affect the efficiency of the bake
system. Do not get oil onto the floor where it can easily get
transferred onto the brake system.</para>
</caution>
<para>Let the lubricant soak into chain before you clean the
unwanted lubricant from the chain.</para>
</step2>
```

Figure 6 The S1000D mark-up example

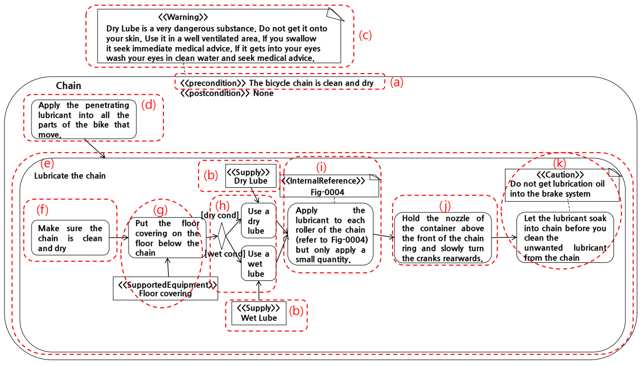

<<Warning>>
Dry Lube is a very dangerous substance. Do not get it onto your skin. Use it in a well ventilated area. If you swallow it seek immediate medical advice. If it gets into your eyes wash your eyes in clean water and seek medical advice.
(c)
Chain
<<precondition>> The bicycle chain is clean and dry
(a)
<<postcondition>> None
Apply the penetrating lubricant into all the parts of the bike that move.
(d)
(e)
Lubricate the chain
(f)
Make sure the chain is clean and dry
(g)
Put the floor covering on the floor below the chain
<<SupportedEquipment>> Floor covering
(b)
<<Supply>> Dry Lube
(h)
[dry cond]
[wet cond]
Use a dry lube
Use a wet lube
<<Supply>> Wet Lube
(b)
(i)
<<InternalReference>> Fig-0004
Apply the lubricant to each roller of the chain (refer to Fig-0004) but only apply a small quantity.
(j)
Hold the nozzle of the container above the front of the chain ring and slowly turn the cranks rearwards.
(k)
<<Caution>> Do not get lubrication oil into the brake system
Let the lubricant soak into chain before you clean the unwanted lubricant from the chain

Figure 7 The example UML activity diagram

V. CONCLUSIONS

S1000D specification is developed as the standard XML format to describe technical publication. It is difficult to author XML documents without learning XML schemas or tags. Most technical publications for equipments are manuals that contain procedural information. In this respect, we proposed an approach to authoring S1000D procedural data modules using behavioral modeling method of UML. The proposed approach allows to easily produce S1000D data modules and to easily understand contents of the S1000D data modules without knowing XML DTD schema and tags. In the future, we will evaluate the proposed approach's substantiality by applying to more practical examples.

ACKNOWLEDGMENT

This work was supported by the IT R&D program of MKE/KEIT.[KI10038619, Development of Solution for Ship Safety Navigation based Maritime Ad-hoc Network].

REFERENCES

[1] ATA, ASD, and AIA, "S1000D: International Specification for Technical Publications Utilizing A Common Source Database", Issue 2.3, Air Transport Association, AeroSpace and Defence Industries Association of Europe, AreoSpace Industries Association[S], 2007.
[2] Shipdex Organization, http://www.shipdex.com [retrieved: Sep, 2012]
[3] W3C, "XML Tutorial", Available: http://www.w3schools.com/xml/default.asp [retrieved: Sep., 2012]
[4] PTC, Arbotext CSDB for S1000D, Available: http://www.ptc.com/product/arbortext/csdb-for-s1000d/ [retrieved: Sep., 2012]
[5] CORENA, CORENA S1000D solutions, Available: http://www.corena.com/what_we_offer/products/corena_s1000d/ [retrieved: Sep., 2012]
[6] Web-x, UltraCSDB S1000D suite, Available: http://www.webxsystems.com/. [retrieved: Sep., 2012]
[7] Siberlogic, SiberSafe S1000D Edition, http://www.siberlogic.com/index.html. [retrieved: Sep., 2012]
[8] Crowell Solutions, "S1000D introduction", http://www.crowsol.com/s1000d/s1000d-introduction [retrieved: Sep., 2012]
[9] CORENA, "Understanding S1000D business rules", CORENA white paper, 2010.
[10] Carlson, D. A., "Modeling XML Vocabularies with UML: Part 1", XML.com, Aug. 2001. Available: http://www.xml.com/pub/a/2001/08/22/uml.htm [retrieved: Sep., 2012]
[11] Carlson, D. A., "Modeling XML Vocabularies with UML: Part 2", XML.com, Sep., 2001. Available: http://www.xml.com/pub/a/2001/09/19/uml.html [retrieved: Sep., 2012]
[12] Carlson, D. A., "Modeling XML Vocabularies with UML: Part 3", XML.com, Oct., 2001. Available: http://www.xml.com/pub/a/2001/10/10/uml.html [retrieved: Sep., 2012]
[13] Booch, G., Christerson, M., Fuchs, M., and Koistinen, J., "UML for XML Schema Mapping Specification. Rational White Paper, Dec. 1999.
[14] Conrad, R., Scheffner, D., and Freytag. J., "XML conceptual modeling using UML", Proceedings of ER'2000, pp. 558-571, 2000.
[15] Farhad, J., "The UML Extension Mechanisms", Department of Computer Science, University College London, Dec., 2002.

Algorithmic Software Adaptation Approach in Mobile Augmented Reality Systems

Oleksii Vekshyn
Computer Aided Management Systems department
National Technical University "Kharkov Polytechnic Institute"
Kharkov, Ukraine
alexeyvekshin@gmail.com

Mykola Tkachuk
Computer Aided Management Systems department
National Technical University "Kharkov Polytechnic Institute"
Kharkov, Ukraine
tka@kpi.kharkov.ua

Abstract—**Growing complexity of mobile software systems leads to problems with productivity and multiple devices support in such applications, especially in augmented reality systems. In this paper, the new approach to algorithmic adaptation for mobile augmented reality systems is proposed. This approach is based on complexity estimation of business logic and separation of computation loading between client and server sides that increases software performance and usability in augmented reality systems. This paper illustrates the main concept and provides some design issues of the proposed approach.**

Keywords-augmented reality; software; mobile systems; algorithmic adaptation

I. INTRODUCTION

Nowadays, mobile information systems become more and more popular. One of the most complex and dynamically grown type of these systems are augmented reality systems (ARS) [1]. Such systems require more hardware resources than standard mobile applications for social networks, and this fact leads to supporting problems of different devices such as mobile phones and tablets. One of the possible solutions is an execution of complex business logic on the server side, where computational capabilities are higher than on the mobile client side; but on the other hand this, could lead to problems with application response time and energy efficiency because of more intensive usage of wireless networking technologies. In this paper we, propose an approach which is based on computational complexity estimation on design-time and analysis of ARS (Augmented Reality System) [1] state in run-time for algorithmic adaptation of mobile ARS. The elaborated approach helps to define which part of business logic should be executed on mobile the client side, and which one on the server side, depends on CPU-performance (Central Processing Unit) of a mobile device.

The paper is structured in the following way: Section 2 depicts briefly some modern trends in this research domain, w.r.t. software adaptation issues; Section 3 provides the review of mobile ARS reference architectures; the formal definitions of the proposed approach are introduced in Section 4; in Section 5, the elaborated algorithm and the appropriate software architectural solution for proposed approach are represented. Finally, Section 6 concludes the paper and gives a short outlook on some future works on this research.

II. SURVEY OF AUGMENTED REALITY SYSTEMS AND ADAPTATIONAL SOLUTIONS FOR MOBILE SYSTEMS

An Augmented reality (AR) is a representational form for a real physical environment, which is extended by adding of computer generated data [2]. AR registers physical objects in three dimensions and combines them with the virtual ones. Unlike the concept of virtual reality, which completely replaces the real world with the virtual one, AR uses a combination of them both.

The ARS operate with such data sources as: two-dimensional markers; data received from GPS-modules (Global Positioning System) [3] and from build-in gyroscopes; they use technologies like images recognition without any markers and GPS data [4].

For implementation of mobile ARS's several frameworks could be used, e.g., Metaio Mobile SDK (Software Development Kit) [5], D'Fusion Mobile [6] and Qualcomm [7].

Nowadays, software adaptation is one of the common trends in modern software engineering (see, e.g., in [8]), and especially in mobile application development. There are several approaches to adaptation in mobile systems, some of them are represented in projects like Q-CAD (QoS and Context Aware Discovery), MADAM (Mobility and Adaptation Enabling Middleware), IST-MUSIC (Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environments), etc [9].

Q-CAD is a resource discovery framework which enables mobile applications to discover and to select resources best satisfied the user's needs. MADAM and ITS-MUSIC frameworks provide model-driven development approach enabling to assemble applications through a recursive composition process. In this case, variability is achieved by plugging into the same component type different component's implementation with similar functional behavior [9]. In [10], a new approach to the composition of mismatching components in context-aware systems is introduced.

Summarizing the described approaches, we can conclude they do not take into account possibility of ARS algorithmic adaptation based on their complexity business logic estimation. This way, it is possible to separate computation loading between client and server sides in order to increase ARS productivity in run-time mode.

III. Reference Architectures for Mobile ARS with Respect to Adaptation Issues

Nowadays, mobile ARS could have two types of reference architectures. The simplest type of Mobile ARS architecture is Standalone architecture [11], which is presented in Figure 1. Such systems work in the following way: mobile devices register the environment using the camera, camera handler processes obtained images and adds data to images with information from local database or GPS-data. This architecture contains several components as: 1) Camera handler is a component, which process data from mobile device's camera; 2) AR objects handler is a component, which processes data from the Camera handler and injects information in images; 3) GPS component is a component, which provides information from GPS controller of the mobile device; 4) Local Database is a component, which provides access to application-specific data stored on mobile devices locally. Standalone architecture has the following features: problems with scalability, maintainability, updating information and extensibility; denial of a whole system in case of single element's denial, high performance in case of low-complex calculations and on the other hand: low performances in case of high computational load; low level of security and easier to manage.

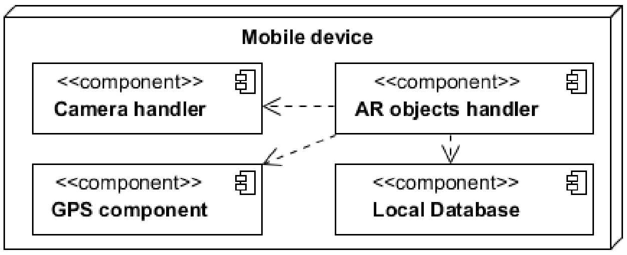

Mobile device
<<component>> Camera handler
<<component>> AR objects handler
<<component>> GPS component
<<component>> Local Database

Figure 1. Stand-alone architecture of mobile ARS [11]

Second type of Mobile ARS implementation is Two-tier client-server architecture [12] which is shown in Figure 2 as a UML 2.0 [13] deployment diagram. There are two tiers in this architecture: application or user interacts with client-side software, in which most cases just provides interface to a server-side application. The client side application invokes server on demand, in some cases for modern systems (e.g., all necessary data are available in local database) data could be processed on the mobile client side. For mobile system's client node is a mobile device, which in case of standalone architecture registers objects from an environment. Two-tier architecture in addition to standalone contains following components: 1) AR objects processing web-service is a component witch provides functionality to process AR object on the server side with better productivity; 2) Server-side Database is a centralized storage of AR objects. Two-tier architecture have the following features: high scalability, maintainability, update of information and extensibility, high performance in case of complex calculations and on the other hand: low performance in case of slow network connection.

All ARS types could be designed with either Standalone or Two-tier architecture, so, the reference architecture for Mobile ARS could be selected with respect to software requirements and available resources.

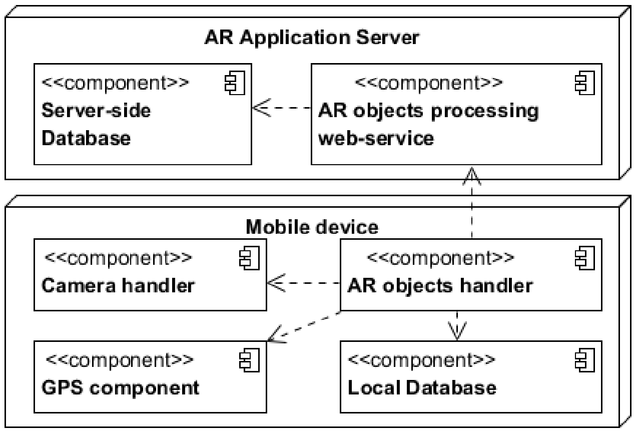

AR Application Server
<<component>> Server-side Database
<<component>> AR objects processing web-service
Mobile device
<<component>> Camera handler
<<component>> AR objects handler
<<component>> GPS component
<<component>> Local Database

Figure 2. Two-tier architecture of mobile ARS [12]

To design and implement the proposed approach, we choose Two-tier architecture, because of some constraints of Standalone architecture such as: scalability problems and server node's absence, where complex business logic should be executed and where the centralized database should be deployed.

IV. Formal Definition of Algorithmic Adaptation for Mobile ARS

The proposed approach to algorithmic adaptation of Mobile ARS can be represented in a formal way using the following definitions.

To measure time of processor work the special parameter: CPU-time has been used. One of the measurement units of CPU-time is MIPS (Million Instructions per Second) [14]. This value has some restrictions, but it could be used in the proposed approach because it is necessary to compare performance of algorithms on the one and the same device.

Definition 1. Precision of calculation for AR is a parameters vector

$$\bar{a} = (f, \quad r, \quad s), \tag{1}$$

where f – a number of decimal places after comma;

r – image resolution (in pixels);

s – ratio of image compression (possible values are from 0 to 1).

These values should be taken into account in the procedure of computation complexity estimation. They implicitly describe possible amount of data for business methods and these influence on amount of operations and calculation time.

Definition 2. A calculation complexity is a vector

$$\bar{c} = (c_0, \ c_t), \tag{2}$$

where c_0 – estimated amount of operations;

c_t – estimated calculation time (is seconds).

Definition 3. A coefficient of a mobile device loading can be estimated with following expression:

$$P = \frac{D_P \cdot 10^6}{\frac{c_0}{c_t}}, \tag{3}$$

where D_p – estimated mobile device performance, MIPS.

Need to notice, that D_p is measured in MIPS, so this value should be multiplied with 10^6 to transform its value to c_0 measurement. This equation illustrates the coefficient of a mobile device load as a ratio of device performance to required amounts of operations in a second.

We consider two different client side application types, which can be classified with UML class-diagram given in Figure 3.

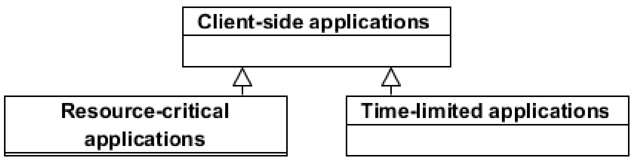


Figure 3. Client-side applications classification

In case of resource-critical applications, we consider the availability of resources for execution of application on the mobile device. For this type of application a calculation on a the mobile device is possible if and only if $P \geq 1$; otherwise it is necessary to execute this on a server side.

The second type of client applications, namely so-called time-limited applications (see Figure 3) could be executed on client side with respect to existing recourses, but due to, time constraints they have to be transferred to server side. In case of time-limited applications calculation on a mobile device side possible if the equation (4) is satisfied.

$$t_{mobile}(\bar{a}) > t_0, \tag{4}$$

where $t_{mobile}(\bar{a})$ – data processing time on mobile client side (in seconds);

t_0 – time constraint for calculation (in seconds).

Additionally, for time-limited applications time efforts should be estimated for network communication. If a network connection speed is slow, transferred calculations could reduce application performance. To estimate the time in such a situation, equation (5) is introduced:

$$t_{mobile}(\bar{a}) > t_{request}(\bar{a}) + t_{server}(\bar{a}) + t_{response}(\bar{a}) \tag{5}$$

where $t_{mobile}(\bar{a})$ – data processing time on mobile client side (in seconds);

$t_{request}(\bar{a})$ – time for request sending (in seconds);

$t_{server}(\bar{a})$ – data processing time on server side (in seconds);

$t_{response}(\bar{a})$ – time for response getting (in seconds).

Taking into account the time constraints in time-limited applications we can make the conclusion: calculation transfer is possible if and only if then the equation (4) and the equation (5) are both satisfied.

Below, we consider only resource-critical applications, the time-limited applications and its constraints do not take into account in the proposed algorithm and approach.

V. Algorithm of Adaptation Process and Prototype Architecture

Based on the given definitions (see Definition (1) - (3)) the algorithm of the adaptation process in mobile ARS has been elaborated and presented as UML 2.0 activity diagram in Figure 4.

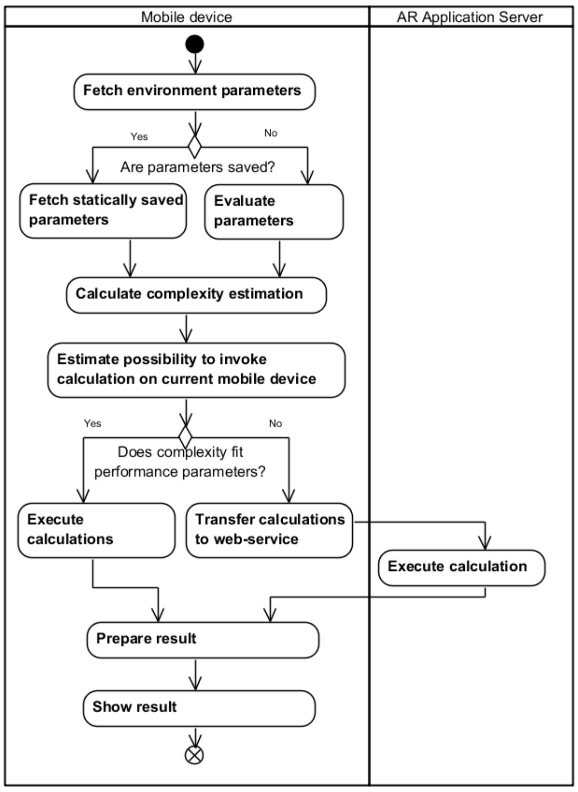

Mobile device
AR Application Server
Fetch environment parameters
Yes
No
Are parameters saved?
Fetch statically saved parameters
Evaluate parameters
Calculate complexity estimation
Estimate possibility to invoke calculation on current mobile device
Yes
No
Does complexity fit performance parameters?
Execute calculations
Transfer calculations to web-service
Execute calculation
Prepare result
Show result

Figure 4. Algorithm of the adaptation process

This algorithm illustrates the process of a node selection where calculation will be executed.

With respect to proposed algorithm, environment's parameters should be obtained and evaluated. Basing on values of this parameters complexity estimation should be calculated (see definition (2)). If complexity estimation requires more resources, than the mobile device performance (see definition (3)), calculations should be transferred to AR Application Server, in other case calculations should be executed on mobile client side. After execution a result will be prepared to visualization and presented to an user.

In case of mobile system, it is possible to apply algorithmic adaptation for selection of node where business logic should be executed. This could improve data processing speed because of complex calculations will be transferred to web-service which provides better computational facilities.

Additionally, it should be noticed that the transfer of calculations requires an estimation of business logic complexity and definition of possibility to execute these calculations on the mobile client side.

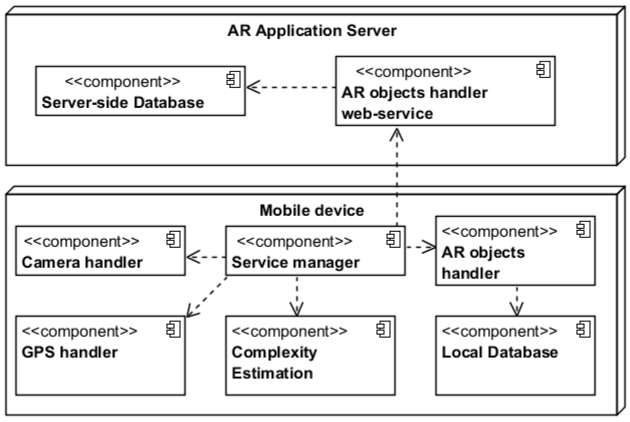


Figure 5. Adaptive mobile ARS architecture prototype

From an architectural point of view, the proposed approach is shown in Figure 5 in the form of UML 2.0 deployment diagram.

The selected two-tier reference architecture has been extended with new two components: 1) Service manager, which analyses context of mobile device, obtains data about business logic complexity and basing on these values generates decision about suitable target node (mobile client or server) for given calculations, and 2) Complexity Estimation, which calculates business logic complexity based on time complexity estimation gathered on mobile application's design time.

To prove the proposed approach, the following example could be used: let CPU-time's value is $D_p = 0.00961\ MIPS$, the precision of calculations is $\bar{a} = (3,\ \ 96000,\ \ 0.6)$, and the estimated complexity of algorithm calculated by Complexity Estimation component is $\bar{c} = (3241,\ \ 0.4)$. In this case, according to the equation (3), value of coefficient of mobile device load is $P = 1.186$, according to proposed approach this calculation could be executed on the mobile client side.

VI. Conclusion and Future Work

We have presented the approach to algorithmic adaptation in mobile ARS, which increases performance and usability of mobile ARS. The proposed approach allows use of mobile systems in more efficient way with respect to system resources. For now, it is not a fully specified process to measure complexity of business logic in application design time and component to fetch complexity estimation on run-time is considered as "black box". In future work, we plan to specify the procedure to estimate a computational complexity of business logic for mobile ARS and integrate it in the proposed approach. Also, formal definitions will be extended with such mobile device characteristics, like free Random-Access Memory (RAM) size and battery capacity.

References

[1] H. Lopez, A. Navarro, J. Relano, An Analysis of Augmented Reality Systems", Proc. of the 2010 Fifth International Multi-conference on Computing in the Global Information Technology (ICCGI '10), 2010, pp. 245-250

[2] B. Furth, Handbook of Augmented Reality, Springer New York Dordrecht Heidelberg London, 2011, p. 746.

[3] Official U.S. Government information about the Global Positioning System (GPS) and related topics, http://www.gps.gov/ 01.11.2012

[4] A. A. Macwilliams Decentralized Adaptive Architecture for Ubiquitous Augmented Reality Systems, 2005, 186 p.

[5] Metaio Mobile SDK, http://www.metaio.com/software/mobile-sdk/ 23.09.2012.

[6] D'Fusion Mobile, http://www.t-immersion.com/products/dfusion-suite/dfusion-mobile 23.09.2012.

[7] Qualcomm AR SDK, http://www.qualcomm.com/solutions/augmented-reality 23.09.2012.

[8] S. Kell, A Survey of Practical Software Adaptation Techniques, J.UCS, vol. 14, 2008, pp. 2110-2157.

[9] K. Kakousis, N. Paspallis, G. A. Papadopoulos, "A survey of software adaptation in mobile and ubiquitous computing," Enterprise Information Systems, vol. 4, Nov. 2010, pp. 355–389, doi:10.1080/17517575.2010.509814.

[10] J. Camara, G. Salaun, C. Canal: "On run-time behavioural adaptation in context-aware systems," Proc. 1st Workshop on Model-driven Software Adaptation (M-ADAPT'07 at ECOOP 2007), 2007, pp. 26–34.

[11] B. Butchart: "Architectural Styles for Augmented Reality in Smartphones", Third International AR Standards Meeting, 2011.

[12] C. Woodward, M. Hakkarainen, M.Billinghurst, "A Client/Server Architecture for Augmented Assembly on Mobile Phones", Research on Mobile Software Engineering: Design Implementation and Emergent Applications, IGI Global Publishing, 2010.

[13] UML 2.0 Specification, http://www.omg.org/spec/UML/2.0/ 23.09.2012.

[14] D. J. Lilja, Measuring Computer Performance: A Practitioner's Guide, Cambridge University Press, 2005, p. 278.

A Case Study in Modeling a Fault-tolerant Satellite System through Implementation of Dynamic Reconfiguration via Handshake

Kashif Javed
Turku Centre for Computer Science (TUCS)
Department of Information Technologies
Abo Akademi University
Turku, FIN-20520, Finland
Kashif.Javed@abo.fi

Elena Troubitsyna
Department of Information Technologies
Abo Akademi University
Turku, FIN-20520, Finland
Elena.Troubitsyna@abo.fi

***Abstract*— Fault tolerance of satellite systems is critical for ensuring the success of the space mission. To minimize redundancy of the on-board equipment, the satellite systems should rely on dynamic reconfiguration in case of failures of some of their components. In this paper, modeling and implementation of a handshake procedure has been presented that becomes a crucial part of the dynamic reconfiguration process of a satellite subsystem for data processing. The model for handshake methodology is specialized software for quickly and successfully recovering from the crisis and failure situation of the satellite system.**

Keywords – dynamic reconfiguration; fault tolerance; advanced software for handshake procedure; modeling and verification.

I. Introduction

To ensure high reliability during long-term missions, the satellite systems rely on redundancy to achieve fault tolerance and guarantee that the system would be able to deliver its services despite component failures. However, the use of redundancy in the satellites is restricted by the constraints put on the weight and volume of the on-board equipment.

Despite a careful analysis performed to ensure the desired degree of reliability, recently one of the satellites has experienced a double-failure problem with a system that samples and packages scientific data [6]. The system consisted of two identical modules. When one of the subcomponents of the first module failed, the system switched to the use of the second module. However, after a while a subcomponent of the spare module also failed, so it became impossible to produce scientific data. In order to avoid failure of the entire mission, the company controlling the operation of the system has invented a solution that relies on healthy subcomponents of both modules and provides complex communication mechanism based on the handshake procedure to restore functioning and to resume production of scientific data.

In this paper, we present a case study in modeling and implementation of Control and Data Management Unit (CDMU) [1] - a generic subsystem of satellites. In particular, we focus on modeling fault tolerance aspect of the system that is implemented as a handshake procedure between two redundant systems. This mechanism is introduced to achieve the dynamic reconfiguration. For this purpose, a formal model of the handshake procedure has been designed and implemented in Promela. Handshake modeling is an advanced software application to deal with dynamic reconfiguration for ensuring fault-tolerance when the mission-critical satellite system encounters faults in its component and errors in data communication.

This paper is structured as follows. Section II describes the state-of-the-art model of CDMU and Section III presents the architecture of the control and data management unit. Section IV describes the handshake procedure performed to reconfigure the system from simple redundant two-module architecture to the Master-Slave architecture. The proposed system model for handshake is explained in Section V covering all relevant details of master and slave modules. Section VI discusses the handshake model between the two reconfiguration modules that has been implemented and verified using SPIN/PROMELA. Finally, conclusions and future work are summarized in Section VII.

II. State-of-the-Art Model

CDMU is a state-of-the-art platform to monitor and control the satellites system and to organize the collected on-board data. The major objective of CDMU is to acquire and transmit the data to the ground after carrying out appropriate processing. Moreover, it also distributes and decodes the given commands to its all redundant systems consisting of processor, reconfiguration and telemetry modules. Whenever any failure or data error takes place during the operation of the satellite system, there is an emergent requirement to dynamically reconfigure the components of CDMU for its smooth and crisis-free control and data management. Processing and storing of satellite data at the right time is of top-most importance during the working and recovery procedure of the proposed system. In case of experiencing any failure, the implemented CDMU structure and the developed model of handshake procedure immediately adapts to the well-defined and specialized switchover mechanism for shifting from one redundant processor to another in order to reconfigure and provide safe operation of the satellite system during its critical mission.

III. ARCHITECTURE

The CDMU consists of two Processor Modules (PM1 and PM2), two Reconfiguration Modules (RM1 and RM2), and two Telemetry Modules (TMM1 and TMM2). It their own turns, each PM consists of Random Access Memory (RAM), Integer Unit (IU), Floating Point Unit (FPU), and Erasable Electrically Programmable Memory (EEPROM). Each Reconfiguration Module (RM) has two components -- Mass Memory (MM) and On-Board Reference Time (OBRT). Telemetry Modules generate Telemetries (TMs) that are processed by Processor Modules.

In CDMU, only one Processor Module (PM1 or PM2) is in active mode and can access one or both RM1 and RM2. TMs are received by the active processor module and accumulated only in MM of its local RM. However, TMs can be retrieved from the MM of partner RM after switching is done from one processor module to another. When each particular PM has experienced a failure, the Master and Slave policy is introduced for error recovery. It aims at ensuring that the CDMU functionality can be preserved even when failures are present in the system.

In our case study, we consider the following two consecutive errors in CDMU that might occur during the execution of the system:

1) PM1 fails due to the failure in FPU.
2) TM ceases to function due to the failure in the link between TMM2 and PM2.

The basis of the Master and the Slave is to prepare a work-around in order to address above mentioned failures. In this case, PM1 and PM2 are converted into the Slave and the Master respectively. Similarly, Master and Slave comprise of the functional program running in PM2 and PM1 respectively and it is mainly established to execute the system without the FPU and connection link.

At a time, both the Master and the Slave interface with RM1 and RM2, respectively, as shown in the CDMU structure. However, RM1 and RM2 are not capable to hold simultaneous access to both of them.

Despite the error in the connection link of PM2, the PM2 is still in operational mode and stores TM in the MM. Similarly, PM1 is also in operational mode by using only IU program (without FPU) that recovers TM from the MM and sends to the operator. The operator interacts with the Master and the Slave by sending Tele-Commands (TCs). Figure 1 shows that each processor module is connected to both RM1 and RM2 and to both TMM1 and TMM2. The TeleCommand (TC) receiver is also linked to both PM1 and PM2.

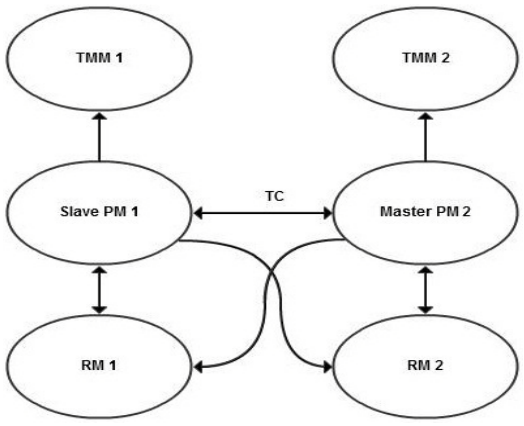


Figure 1: CDMU Structure [1]

IV. FACTORS CONTRIBUTING IN HANDSHAKE

The important key factors that are involved in the handshake procedure are as follows:

1) Time Event Register (TER) is used for messaging between the Master and the Slave. As there is no direct link between the Master and the Slave, so TER is used as a shared device. Both can access TER to read and write messages. RM1 and RM2 have their own TER devices.
2) The two interrupts -- Time Event Interrupt (TEI) and Time Synchronization Interrupt (TSI) caused by RM1 and RM2 are sent to the Slave and the Master respectively. If the Master uses RM1 and interrupt triggers, then interrupt is only sent to the Slave because it is a local processor module of RM1.
3) The interrupts can be used as a signal from the Master to the Slave for the acknowledgement of the messages because the Master has a charge of the interrupt timing.
4) OBRT Status Register is used to find out that interrupt has triggered in the system. The Master holds the check of this register and clears the interrupt flag for allowing the coming up interrupts.
5) The Master and the Slave cannot use the same RM at a time. However, both the Master and the Slave are informed through handshake procedure in order to choose required RM at a given time interval.
6) Handshaking is done through Communication Channel (CCH) between the Master and the Slave. RM1 or RM2 is used as CCH. The TER in the CCH is expressed as Communication Time Event Register (CTER).
7) The selection of RM1 or RM2 as CCH depends on the Master as it utilizes both RM1 and RM2. On getting the TC instruction from the operator, it

switches to one module of RM (RM1 or RM2) and releases the other RM for CCH. If the Master is using only one RM module initially, the unused RM will be selected as CCH. The Master can switch the RM at the end of the handshake procedure.

8) The handshake message contains the phase content and timing of the message that is encoded in the CTER. The timing of the interrupt is slightly affected by the phase content that is encoded in the four Least Significant Bits (LSB) of the CTER, but this affect of interrupt timing is less than 0.3 ms and is, therefore, ignored.
9) The phase content in the four least significant bits of the CTER is as under:
   i. When 4 LSB of CTER has value ‘1’, then the Master informs the Slave to communicate through RM1. Similarly, when 4 LSB of CTER has value ‘2’, then the Master informs the Slave to communicate through RM2. This phase is known as “Select Communication RM”.
   ii. If the value is ‘4’ in the 4 LSB of CTER, the Slave updates the Master to confirm the communication through RM1. Likewise, if the value is ‘5’ in the 4 LSB of CTER, then the Slave informs the Master that it confirms the communication through RM2. This phase of the handshake procedure is called “Confirm Communication RM”.
   iii. Upon setting the value of ‘10’ in 4 LSB of CTER, the Slave is informed by the Master that if RM1 is not in use then switch to it and use it. For the value ‘11’, the Slave has to switch to use RM2. When the value is ‘14’, then the Master instructs the Slave to release both RM1 and RM2. This phase is named as “Command Slave”.
   iv. The Master sends a message to the Slave in which it verifies the RM1 or RM2 selection by putting the value ‘8’ in 4 LSB of CTER. This phase is entitled as “Confirm Command”.

10) The encoding of the handshake messages is done within one second (s) - Pulse Per Second (PPS). The interrupts according to the PPS time slot are given below:
   i. When interrupts occur from 0.10 to 0.40 s, RM1 and RM2 are not selected in this time slot. It means that the Master instructs the Slave to confirm the change to use no RM.
   ii. For the selection of RM1, interrupts take place in the time slot ranging from 0.42 to 0.70 s. The Master orders the Slave either to communicate with RM1 or confirm change to use RM1 during the handshake procedure.
   iii. In the 0.72 - 1.00 s time slot, interrupts are taken into account. This selection is encoded for RM2 where master notifies the Slave either to communicate with RM2 or confirm change to use RM2 during the handshake procedure.
   iv. The purpose of the remaining unused slots 0.00 – 0.10 s, 0.40 – 0.42 s and 0.70 – 0.72 s is to avoid overlaps. Any interrupts appearing in these timing slots will be ignored.

11) The minimum time between two TSIs is greater than 0.3s to ensure that two TSIs do not trigger during the same time slot. On the other hand, interrupt can be triggered two times during the same time slot.

V. PROPOSED SYSTEM MODEL FOR HANDSHAKE

The handshake procedure [2] has been modeled for the Master and the Slave as shown in Figure 2. Handshake is a procedure in which the Master communicates with the Slave to update the selection of RM1 and RM2. It is a complicated process as there is no direct communication link between them.

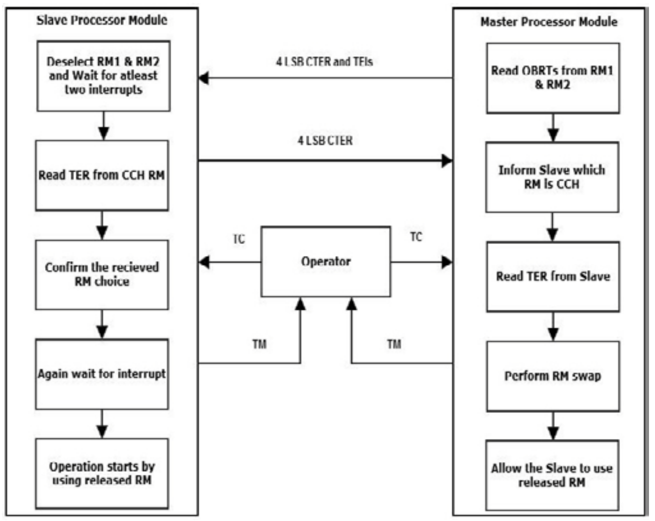

Slave Processor Module
Deselect RM1 & RM2 and Wait for atleast two interrupts
Read TER from CCH RM
Confirm the recieved RM choice
Again wait for interrupt
Operation starts by using released RM
4 LSB CTER and TEIs
4 LSB CTER
TC
Operator
TC
TM
TM
Master Processor Module
Read OBRTs from RM1 & RM2
Inform Slave which RM is CCH
Read TER from Slave
Perform RM swap
Allow the Slave to use released RM

Figure 2: Model of Handshake Procedure

A. *Master Handshake Procedure*

The handshake procedure that is executed by the Master Module is shown in Figure 2. Below we give its brief description:

Upon the reception of TC from the operator, the handshake procedure is started by the Master. The Master informs the Slave that other RM will be used as CCH by updating the value of 4 LSB TER. If the Master is using RM2 and storing TM, then the Slave will be informed to make RM1 as CCH. Likewise, if RM1 is operated by the Master, then the Slave has to use RM2 as CCH. When CCH is RM1, then system operation is performed from 0.42 to 0.70 s PPS slot. Similarly, for RM2, 0.72 to 1.00 s, PPS slot is used for the system operation. System has to wait for starting of the right PPS slot according to the CCH.

In order to send information to Slave, interrupts are triggered from the Master after setting the value of OBRT Status Register to zero. For accuracy, the value of TER for the Slave RM is set to 0.04 s. The interval between two interrupts is 0.06 s. The Master ensures by reading the CTER value from the Slave that selection of CCH is done. The Master can swap the CCH selection at the end of handshake procedure. The Master commands the Slave by setting the future CCH selection value in the 4 LSB CTER and triggers a TEI only. The time value of TEI is not relevant to the CTER, so the time slot of TEI makes no changes in the end result of the system. Only operator is responsible for the new RM selection and determining which RM is used as CCH as stated in Section IV. In the system, operator initially notifies the RM selection to the Master, it changes CCH selection from used RM to other RM according to the swapping information that is encoded in 4 LSB CTER and also confirms the RM selection. The confirmation message is also forwarded to the Slave by sending two interrupts within the correct time slot. At this moment, the Master ends the handshake procedure and updates the operator for successful working by sending the corresponding TM.

B. Handshake Procedure: Slave Behaviour

When the operator starts the handshake, the following operations are carried out by the Slave as shown in Figure 2.

If the Slave is using RM1 or RM2, then it will deselect the current RM on the reception of TC command from the operator. When RM is discontinued from the Slave, then OBRT Status Register will be set to zero and no more interrupts will be triggered. The Slave waits for 0.03 s to get the new command along with two interrupts (i.e. TEI and TSI) which will be generated from the Master during the expected PPS slot. When the Slave receives a message from the Master, then it decodes it from the interrupts time slot as mentioned in Section IV (para # 10). For verification, the Slave also interprets the value of 4 LSB CTER as described in Section IV (para # 9). If the values derived from the interrupts time slot and 4 LSB CTER are the same, then the Slave achieves the specified CCH selection. After that, the Slave sends acknowledgement of confirmation to the Master by setting the value of 4 LSB CTER according to Section IV. Now, the Slave has to wait again for 0.02 s for the new response or interrupt from the Master according to the PPS slot. On the arrival of message from the Master, the Slave is triggered by TEI. The Slave has no opportunity to change the decision of new selection and waits for 10s for the confirmation message from the Master. Again, the Slave receives two interrupts with the CTER message and compares the time slot of interrupts with previous CTER value. If both are same, then the Slave begins the operation with released RM. Finally, the Slave also completes the handshake procedure by sending TM to the operator.

VI. Verification of the Handshake Model

The handshake model has been implemented by using PROMELA (PROcess MEta LAnguage) high level modeling language with SPIN model checker for verifying the required results. SPIN [3,4] is extensively used in formal verification of distributed and parallel processing systems. SPIN has greatly facilitated the process of verification in the areas of mission-critical algorithmic applications, message and data communication in the client-server environment, synchronization and coordination of large number of processes in the parallel and distributed systems, deadlock handling methodologies in the modern multi-tasking operating systems, verification of the mission-oriented control models for space aircrafts, utilization of intelligent models for determining most suitable and economical paths over wide area networks, checking performance of routing protocols [5], testing of fault-tolerant strategies and implementation of a wide variety of switching techniques. The literature review reveals that most of the software-based systems/models are checked and verified by the SPIN model checker.

The handshake model between two processors in control and data management unit has been successfully implemented and verified using SPIN/PROMELA. The flow chart for handshake procedure model is shown in Figure 3. The following algorithm along with description of each condition of the processes shows part of the implemented SPIN/PROMELA model.

```
/*Variable Declarations */
active proctype Slave_starts_HP()
{S_TC=true;
if
::(S_TC==true)->RM1=0;RM2=0;
::( S_TC!=true)-> printf("\n\nExit Handshake Procedure.\n\n");
fi
S_TM=true;}
```

The above code depicts that when TC command is received to Slave from the operator, Slave starts handshake procedure by deselecting the RM selection. After successful execution of the TC command, Slave sends TM to operator and waits for Master’s response. In any other condition, handshake procedure will be terminated.

```
active proctype Master_starts_HP()// time value is taken in (ms)
{M_TC=true; RM1=0;RM2=1; // set by the operator
if
::(RM1==0 && RM2==1)->// I_time denotes timing of interrupts
{CTER_4_LSB=1;I_time=500;TEI=true;TSI=true;OBRT_SR=1;
run Slave_read_wrtie_operation(CTER_4_LSB,I_time,TEI,TSI);}
::(RM1==1 && RM2==0)->
{CTER_4_LSB=2;I_time=800;TEI=true;TSI=true;OBRT_SR=1;
run Slave_read_wrtie_operation(CTER_4_LSB,I_time,TEI,TSI);}
fi}
```

The code associated with the above process describes that Master starts handshake on the operator command. When operator selects RM2 for Master, then Master uses RM2 and notifies Slave (by sending CTER and interrupts) to use RM1 as CCH. Likewise, if operator selects RM1, then Master uses RM1 and updates the Slave (through CTER and interrupts) to use RM2 as CCH. After that, it waits for Slave's response.

```
proctype Slave_read_wrtie_operation(int CTER_4_LSB,I_time;bool
TEI,TSI)
{if
::((CTER_4_LSB==1) && (TEI==true && TSI==true) && (I_time>=420
&& I_time<=700))->
{CTER_4_LSB=4;run Master_decides_future_selection(CTER_4_LSB);}
::((CTER_4_LSB==2) && (TEI==true && TSI==true) && (I_time>=720
&& I_time<=1000))->
{CTER_4_LSB=5;run Master_decides_future_selection(CTER_4_LSB);}
::((CTER_4_LSB!=1) || !(I_time>=420 && I_time<=700))->
{printf("\n\nExit Handshake Procedure.\n\n");}
::((CTER_4_LSB!=2) || !(I_time>=720 && I_time<=1000))->
{printf("\n\nExit Handshake Procedure.\n\n");}
fi}
```

The above piece of code illustrates that when timing of interrupts is in line with the information that is encoded in CTER 4 LSB, then Slave confirms the selection to Master and waits for 0.02 s in order to get Master's response. So, when interrupts occurs between 0.42 to 0.70 s time slot and CTER 4 LSB is '1', it means Slave confirms to use RM1 as CCH by encoding the value '4' in CTER 4 LSB. Similarly, if time slot for interrupt is 0.72 to 1.00 s and CTER 4 LSB is '2' then RM2 is confirmed as CCH by the Slave through updating the value '5' in CTER 4 LSB. If timing of the interrupts is not compatible with the encoded information in CTER 4 LSB, handshake procedure exits at this stage.

```
proctype Master_decides_future_selection(int CTER_4_LSB)
{if
::(CTER_4_LSB==4)->
{OBRT_SR=0;CTER_4_LSB=11;TEI=true;OBRT_SR=1;
if
::(CTER_4_LSB==11)->
{RM1=1;RM2=0;aa= CTER_4_LSB;OBRT_SR=0;CTER_4_LSB=8;
I_time=800;TEI=true;TSI=true;OBRT_SR=1;M_TM=true;
run Slave_interprets_message(aa,I_time,TEI,TSI);}
::(CTER_4_LSB==14)->
{RM1=0;RM2=0;OBRT_SR=0;aa=CTER_4_LSB;CTER_4_LSB=8;
I_time=200;TEI=true;TSI=true;OBRT_SR = 1;M_TM=true;
run Slave_interprets_message(aa,I_time,TEI,TSI);}
fi;}
```

The above fragment of the code describes that when Slave is using RM1, Master updates the up-coming selection of RM by placing the value '11' or '14' in CTER 4 LSB with only TEI. If Master selects RM1, it releases RM2 to be used as CCH by putting the value '11' in CTER 4 LSB. When Master picks RM1 and does not release RM2 to be used as CCH, it writes the value '14' in CTER 4 LSB. After a half second to give the Slave sufficient time to read value of CTER, the Master confirms the selection to the Slave by encoding the value '8' in CTER 4 LSB on the specified time

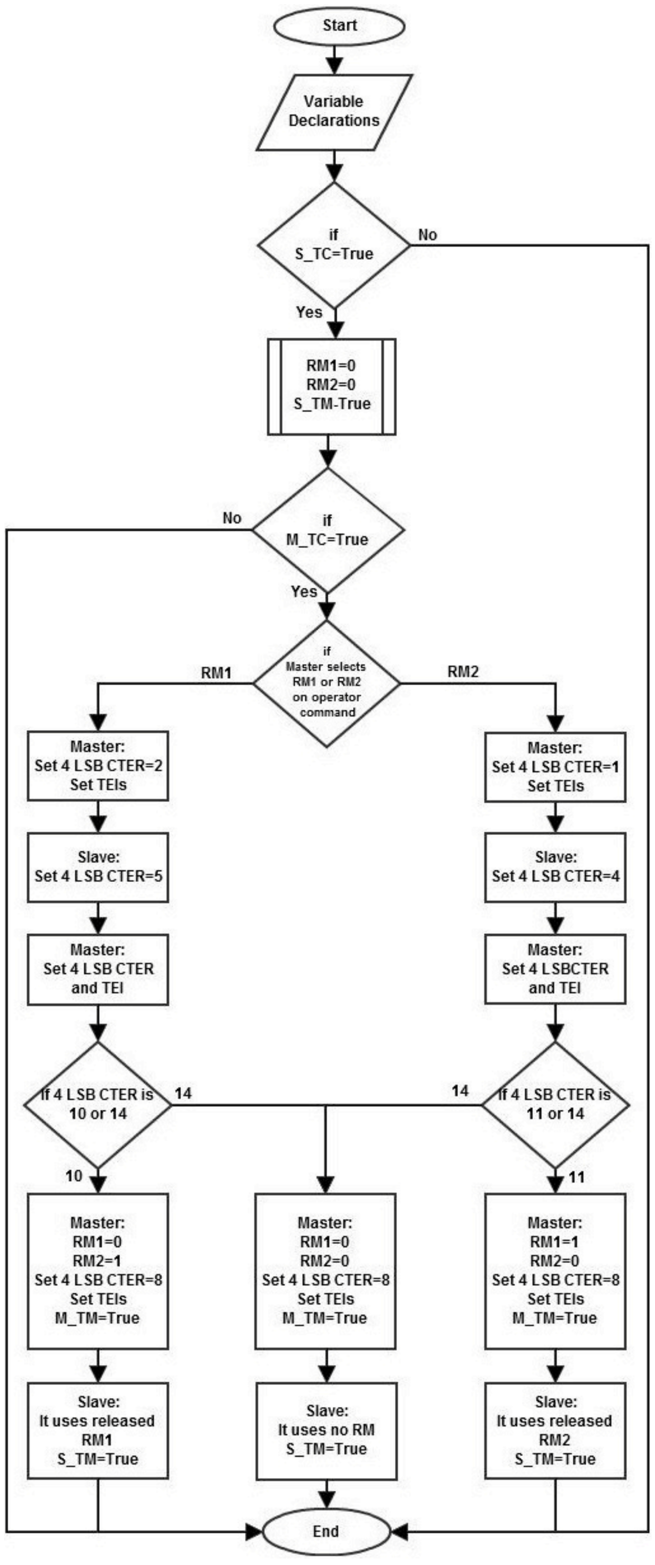

Start
Variable Declarations
if S_TC=True
No
Yes
RM1=0
RM2=0
S_TM-True
if M_TC=True
No
Yes
if Master selects RM1 or RM2 on operator command
RM1
RM2
Master: Set 4 LSB CTER=2 Set TEIs
Master: Set 4 LSB CTER=1 Set TEIs
Slave: Set 4 LSB CTER=5
Slave: Set 4 LSB CTER=4
Master: Set 4 LSB CTER and TEI
Master: Set 4 LSBCTER and TEI
If 4 LSB CTER is 10 or 14
If 4 LSB CTER is 11 or 14
14
14
10
11
Master: RM1=0 RM2=1 Set 4 LSB CTER=8 Set TEIs M_TM=True
Master: RM1=0 RM2=0 Set 4 LSB CTER=8 Set TEIs M_TM=True
Master: RM1=1 RM2=0 Set 4 LSB CTER=8 Set TEIs M_TM=True
Slave: It uses released RM1 S_TM=True
Slave: It uses no RM S_TM=True
Slave: It uses released RM2 S_TM=True
End

Figure 3: Flow Chart of Handshake Procedure Model

slot according to Section IV and exits the handshake procedure.

```
::(CTER_4_LSB==5)->
{OBRT_SR=0;CTER_4_LSB=10;TEI=true;OBRT_SR=1;
```

The associated code with above condition illustrates this.

```
if
::(CTER_4_LSB==10)->
{RM1=0;RM2=1;OBRT_SR=0;bb=CTER_4_LSB;CTER_4_LSB=8;
I_time=800;TEI=true;TSI=true;OBRT_SR=1;M_TM=true;
run Slave_interprets_message(bb,I_time,TEI,TSI);}
::(CTER_4_LSB==14)->
{RM1=0;RM2=0;OBRT_SR=0;bb=CTER_4_LSB;CTER_4_LSB=8;
I_time=200;TEI=true;TSI=true;OBRT_SR = 1;M_TM=true;
run Slave_interprets_message(bb,I_time,TEI,TSI);}
fi;}
fi}
```

The above part of the code shows that when the Master is using RM1, it updates the up-coming selection of RM by setting the value ‘10’ or ‘14’ in CTER 4 LSB with only TEI. If the Master selects RM2, it releases RM1 to be used as CCH by putting the value ‘10’ in CTER 4 LSB. When the Master picks RM2 and does not release RM1 to be used as CCH, it writes the value ‘14’ in CTER 4 LSB. After a half second to give the Slave sufficient time to read value of CTER, the Master confirms the selection to the Slave by encoding the value ‘8’ in CTER 4 LSB on the specified time slot according to Section IV and exits the handshake procedure.

```
proctype Slave_interprets_message(int previous_CTER,I_time;bool
TEI,TSI)
{if
::((I_time>=420 && I_time<=700) && (previous_CTER==10) &&
(TEI==true && TSI==true))->
{S_TM=true;}
::((I_time>=720 && I_time<=1000) && (previous_CTER==11) &&
(TEI==true && TSI==true))->
{S_TM=true;}s
::((I_time>=100 && I_time<=400) && (previous_CTER==14) &&
(TEI==true && TSI==true))->
{S_TM=true;}
::(!(I_time>=420 && I_time<=700) || (previous_CTER!=10))->
{ printf("\n\nExit Handshake Procedure.\n\n");}
::(!(I_time>=720 && I_time<=1000) || (previous_CTER!=11))->
{ printf("\n\nExit Handshake Procedure.\n\n");}
::(!(I_time>=100 && I_time<=400) || (previous_CTER!=14))->
{ printf("\n\nExit Handshake Procedure.\n\n");}
fi}
init
{atomic// Atomic is used to reduce the complexity.
{run Slave_starts_HP();
run Master_starts_HP();}
}
```

The code given above indicates that after waiting for 10 s, Slave receives the confirmation message with two interrupts from Master. The timing of interrupts is matched with the information that is encoded in previous CTER 4 LSB as mentioned in Section IV. Therefore, when timing of the interrupts lies between 0.42 to 0.70 s time slot and previous CTER 4 LSB is ‘10’, it notifies that Slave uses RM1 as CCH that is released by the Master. Similarly, timing of the interrupts lies between 0.72 to 1.00 s time slot and previous CTER 4 LSB is ‘11’, it notifies that Slave uses RM2 as CCH that is released by the Master. Also, when interrupts timing lies between 0.10 to 0.40 s and the value of previous CTER 4 LSB is ‘14’, then Slave uses neither RM1 nor RM2 as CCH. After then Slave exits the handshake procedure. If interrupts timing is not in line with the information that is encoded in earlier CTER 4 LSB, handshake procedure exits at this stage too.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a formal approach for modeling a fault-tolerant satellite system that relies on the handshake procedure for dynamic reconfiguration. We have demonstrated how to create a Promela model of the handshake and carry out its analysis. Since the handshake procedure has a number of non-trivial properties caused by the distributed nature of the system, such a model allows the designers to ensure correctness of the handshake implementation. In our future work, we are planning to extend the proposed approach to derive the generic modeling patterns. Moreover, it would be interesting to explore the handshake in the presence of more complex network architecture.

REFERENCES

[1] “DEPLOY – Software Requirement Specification, Master/Slave Software”, Space Systems Finland, Ltd., July 2011.

[2] J. Kashif, and E. Troubitsyna, “Designing a Fault-Tolerant Satellite System in SystemC”, ICONS 2012, The Seventh International Conference on Systems, IEEE Computer Press, pp. 49–54, March 2012.

[3] C.Baier and J.-P. Katoen. “Principles of Model Checking”. MIT Press, 2008.

[4] N. A. S. A. Larc, “What is Formal Methods?", NASA Langley Methods, http://shemesh.larc.nasa.gov/fm/fmwhat.html, formal methods program, 2001.

[5] J. Kashif, A. Kashif, and E. Troubitsyna,, “Implementation of SPIN Model Checker for Formal Verification of Distance Vector Routing Protocol”, International Journal of Computer Science and Information Security (IJCSIS), Vol 8, No 3, USA, ISSN 1947-5500, pp. 1-6, June 2010.

[6] A. Tarasyuk, I. Pereverzeva, E. Troubitsyna, T. Latvala, and L. Nummila, Formal Development and Assessment of a Reconfigurable On-board Satellite System, In: Frank Ortmeier, Peter Daniel (Eds.), Proceedings of 31st International Conference on Computer Safety, Reliability and Security (SAFECOMP 2012), LNCS 7612, pp.210-222, Springer, 2012.

Transformation of Medical Service Ontology to Relational Data Models

Osamu Takaki[a], Izumi Takeuti[b], Koichi Takahashi[b], Noriaki Izumi[b], Koichiro Murata[c], Mitsuru Ikeda[a] and Kôiti Hasida[b]
[a]School of Knowledge Science, Japan Advanced Institute of Science and Technology (JAIST), Nomi, Japan, {takaki, ikeda}@jaist.ac.jp
[b]National Institute of Advanced Industrial Science and Technology (AIST), Ibaraki, Japan, {takeuti.i, k.takahashi, n.izumi, hasida.k}@aist.go.jp
[c]School of Medicine, Kitasato University, Sagamihara, Japan, murata-k@kitasato-u.ac.jp

***Abstract*—For assessment of medical service quality in hospitals, it is important to define quality indicators for evaluating medical services and to calculate their values based on data in hospitals databases. Thus, one needs a proper method to correctly calculate the value of a given quality indicator based on data in medical databases. This paper introduces a method to transform Medical Service Ontology (MSO) to relational data models in medical databases, where MSO is an ontology that provides vocabulary words for developing quality indicators. To this end, this paper defines a virtual data model called the Global Data Model (GDM) and a transformation of MSO to GDM by grouping concepts and properties in MSO. Based on the transformation, a quality indicator defined with MSO can be transformed into queries on GDM automatically. Moreover, by developing mappings from GDM to relational data models in medical databases, a quality indicator on MSO can be transformed into queries on the data models, using which the value of the quality indicator is calculated based on data in medical databases.**

Keywords-component; quality indicator; ontology; data model; medical database

I. INTRODUCTION

A quality indicator is a measure of medical service quality that is represented numerically. For example, “in-hospital mortality after stomach cancer surgery” is a quality indicator that indicates the quality of surgeries of stomach cancers in a hospital [1, 2]. For assessment of medical service quality in hospitals, it is important to define quality indicators for evaluating medical services and to calculate their values based on data in hospital databases. Many hospitals and institutes in medical science have developed and published quality indicators and their data [1, 2, 3, 4, 5].

However, despite these efforts, an established framework to develop or maintain quality indicators has not yet been developed. In fact, even though certain groups of hospitals have begun to give quality indicators to collaborating hospitals, to gather their data, and to compare them, it is not easy to compare them fairly. For example, mortality, re-hospitalization rate, incidence of bedsores, and incidence of complications are difficult to compare, since there are no suitable standard definitions of related concepts. Moreover, there often exist various types of gaps in the development of quality indicators and in the calculation of their values. Such gaps arise from a difference in the vocabulary of medical services and/or its interpretations, and they often make the results of evaluations of the quality of medical services unsuitable.

To realize sharing of quality indicators among multiple hospitals and to compare data fairly, a proper framework that helps develop and/or improve quality indicators according to their own environment is needed. We call such a framework QI-framework and describe it by QI-FW. To realize QI-FW, it is desirable to develop a system that creates quality indicators with self-checking vocabulary words and constructs quality indicators and their interpretations. To this end, we developed a representation system of quality indicators, which we describe by QI-RS [6, 7, 8]. QI-RS gives a graph-based representation of a quality indicator that is rigorous and easily understandable, and the system is based on an ontology called “Medical Service Ontology (MSO).” QI-FW helps to develop quality indicators in QI-RS and to calculate their values based on medical databases.

The main purpose of this paper is to present a method to calculate the value of a given quality indicator in QI-RS based on data in medical databases. To this end, this paper introduces a method to transform MSO to data models for relational databases, where MSO is an ontology that provides vocabulary words to define quality indicators. We first define a virtual data model called the Global Data Model (GDM) as a standard model for calculating the values of quality indicators, and introduce transformation of MSO to GDM by grouping concepts and properties in MSO. Based on the transformation, a quality indicator defined using MSO is transformed to queries in the GDM. Moreover, by developing mappings from GDM to data models on given medical databases, a quality indicator in MSO is transformed to queries on the data models, and the value of the quality indicator can be calculated based on the data in the given medical databases.

The remainder of this paper is organized as follows. Section II explains quality indicators and an overview of the QI-FW. Section III explains QI-RS based on our previous studies. Section IV explains a method to transform MSO to GDM and to generate queries on GDM from quality indicators in QI-RS. This section contains the main contribution of the paper. Section V extends the method to transform quality indicators in QI-RS to queries on data models of medical databases. The last section explains related works and conclusions.

II. Outline of QI-Framework

A. Quality Indicator

A quality indicator consists of a *name* (or a *label*) and a *calculating formula*. For example, "five-year survival rate of stomach cancer patients" is the name of a quality indicator, and its calculating formula is given as follows [2].

Calculating Formula: Data for the above quality indicator is obtained by calculating the rate of the following numerical values.

Numerator: Number of inpatients that satisfy the condition defined in the denominator and that survived more than five years since they were diagnosed with stomach cancer

Denominator: Number of patients that were diagnosed with stomach cancer

The value that is obtained from a quality indicator by using the calculating formula and data in a hospital (or hospitals) is called "the value of a quality indicator (in a hospital (or hospitals))" or "the data of a quality indicator (in a hospital (or hospitals))." We assume that the values of quality indicators are essentially calculated from data in medical databases.

Even though herein we distinguish between a quality indicator and its calculating formula, we will often refer to the calculating formula of a quality indicator simply as "a quality indicator," unless stated otherwise.

B. Overview of the QI-Framework

Here, we briefly explain an overview of the framework QI-FW to develop quality indicators and to calculate their values based on medical databases.

As mentioned in Section I, proper sharing of the definition of a quality indicator is not straightforward. To address the above problems, it is significant to establish a way to unify the vocabulary of quality indicators and their interpretation including their whole structures. Thus, toward solving this problem, we are in the process of developing QI-FW. (At this time, QI-FW has not yet been completed implementation, and it is currently under development.)

QI-FW consists of (1) a representation system QI-RS of quality indicators (Section III), (2) medical databases in hospitals, and (3) mapping systems (Section V). Moreover, QI-RS has MSO as its main component (Section III.A). Medical staff and system engineers who administrate medical databases (and knowledge engineers, if necessary) collaborate in developing and improving MSO.

QI-FW users are assessors of the medical service quality of a hospital (or hospitals) based on data in medical databases, who are patients, medical staff, and so forth (Fig. 1). They can develop quality indicators in QI-RS via some interface of QI-FW. A quality indicator *Q* in QI-RS is expressed as a graph (Section III.B). Some nodes in *Q* are concepts in MSO, while edges in *Q* are properties in MSO.

On the other hand, system engineers who manage medical databases are responsible for developing and improving mapping systems between GDM and data models in medical databases. Concepts and properties in MSO are translated to tables in the Global Data-Model (GDM), which is a virtual relational data model (Section IV). According to the translation and mapping systems described above, *Q* is translated to queries on the data models and an algorithm on the data retrieved by the queries. Through the queries and the algorithm, the user can calculate the value of *Q* based on data in the medical databases.

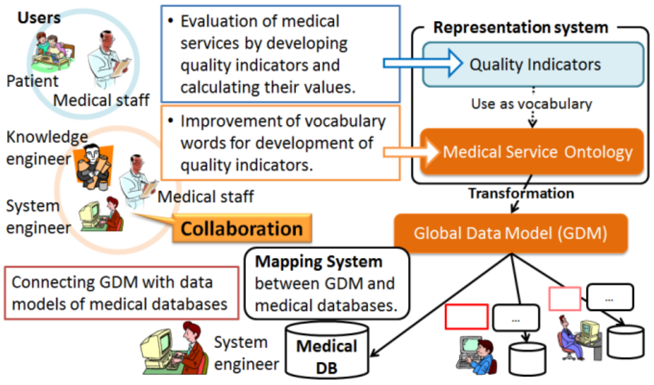

Users
Patient
Medical staff
Knowledge engineer
System engineer
Medical staff
Collaboration
• Evaluation of medical services by developing quality indicators and calculating their values.
• Improvement of vocabulary words for development of quality indicators.
Representation system
Quality Indicators
Use as vocabulary
Medical Service Ontology
Transformation
Global Data Model (GDM)
Connecting GDM with data models of medical databases
Mapping System between GDM and medical databases.
System engineer
Medical DB

Figure 1. Overview of QI-framework.

III. Representation Based on MSO

In this section, we explain a graph-based representation system of quality indicators, which we call QI-RS. QI-RS is developed based on the concept of considering a quality indicator as a combination of the quantification target and the quantification and development method of the target and the method independently. We represent the target of the calculation or quantification as a graph that we call an objective graph, and we represent the way to calculate or quantify it by a concept that we call a quantifying concept. Furthermore, objective graphs are constructed based on ontology that we call Medical Service Ontology (MSO).

A. Medical Service Ontology

In this sub-section, we define MSO as a vocabulary for calculating formulas of quality indicators. For example, the formal calculation in Section II.A uses the words "patients," "hospitalize (hospitalization)," and "aged." In fact, to define quality indicators, we need words for describing characteristics of patients, events (medical services) in hospitals, predicates about patients, and so forth. MSO is developed in the ontology developing tool Semantic Editor [9].

1) Patients

First, we describe the basic concepts related to patients and their attributes in Fig. 2. Yellow rounded rectangles denote concepts, and pink rounded rectangles denote attributes. In general, pink rounded rectangles in diagrams in Semantic Editor denote properties.

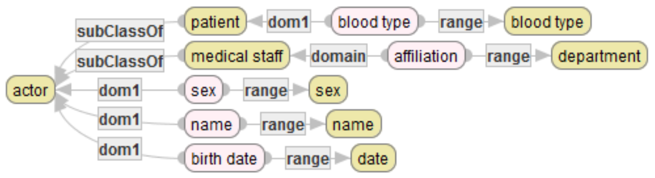


Figure 2. Basic concepts and attributes related to patients.

In this paper, we classify properties between concepts into attributes of concepts and relations between concepts. The concept [patient] has attributes ⟨blood type⟩, ⟨sex⟩, ⟨name⟩, and ⟨birth⟩, where we describe a concept by brackets and labels and an attribute by angle brackets and labels. The values of these attributes are supposed to be immutable for a patient.

2) Events

Next, we explain concepts related to events in a hospital. An event is defined to be what a medical staff or a hospital executes for a patient or what happens to him/her (Fig. 3).

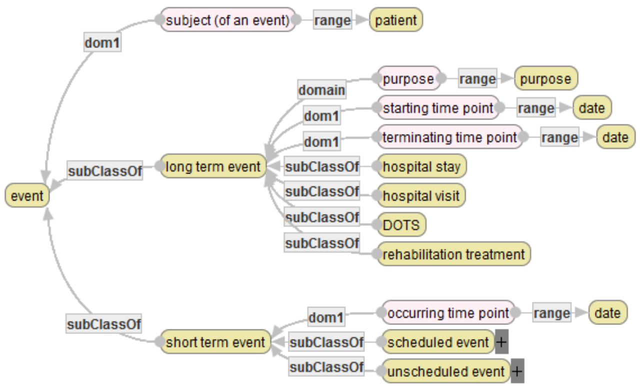


Figure 3. Basis concepts and attributes related to events (partial).

Events are classified into long-term and short-term events. While a long-term event such as a hospital stay (hospitalization) usually takes multiple days, a short-term event does not usually take more than two days. Moreover, short-term events are further classified into scheduled and unscheduled events. Usual medical services are regarded as scheduled events, while accidents such as deaths are regarded as unscheduled events. For example, "admission," "discharge," "diagnosis," "examination," and "operation (surgery)" are typical scheduled short-term events, while "death," "falling," and "bone fracture" are typical unscheduled short-term events. Each typical event is further classified into detailed classes. For example, examination events are classified into about thirty types of examinations.

Each long-term event has attributes including the subject (target patient), purposes, the starting date, and the ending date, while each short-term event has attributes including the subject and occurring time (Fig. 3). We omit a detailed explanation of them due to space limitations.

3) States of Patients

The state of a patient denotes the health state or a condition of a patient at a time point. The diagram in Fig. 4 defines the following main states: age, state of life or death, state of disease that a patient suffers from, and basic body properties. These states are used to describe a feature of a patient as a target of a medical service or an outcome of a patient that cannot be represented by any event.

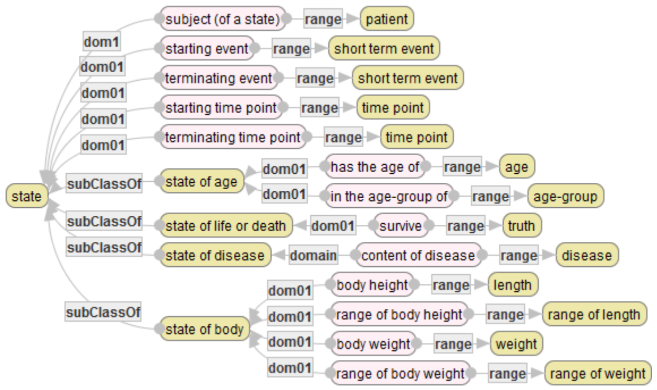


Figure 4. Basis concepts and attributes related to states of patients.

Remark. We often identify a concept with its extension, which is the set of instances of the concept. For example, in a hospital *H*, the concept "patient" is identified with the set of patients of *H*.

4) Main Relations in MSO

In this sub-section, we define properties in MSO that are not attributes. We call them *relations*. We define the primary relations between concepts as follows.

Relations of patients and events: These relations are defined between the concept [patient] and concepts of events. For example, the following relation denotes the relations between patients and their diagnosis (we describe a relation by angle brackets and a label).

⟨subject (of an event)⟩ ⊆ [diagnosis] × [patient].

Note that these relations share the same name, "subject (of an event)." We omit the explanation of the relations between patients and other events due to limitations of space.

Relations of patients and states: These relations are defined between [patient] and concepts of patients' states. For example, the following relation denotes the relationship between patients and their state of disease.

⟨subject (of a state)⟩ ⊆ [patient] × [state of disease].

These relations also share the same name "subject (of a state)" and all concepts of patients' states have the common attributes of starting time points and terminating time points. We omit the explanation of the relations between patients and other states.

Relations of time ordering: These relations are defined between the concepts of events and patients' states. For example, the following relations denote the relationships between operations.

⟨before more than <p>⟩ ⊆ [operation] × [operation],
⟨before less than <p>⟩ ⊆ [operation] × [operation],
⟨after less than <p>⟩ ⊆ [operation] × [operation], and
⟨after more than <p>⟩ ⊆ [operation] × [operation].

Here, "<p>" denotes a parameter. For example, the relation ⟨before more than <2 weeks>⟩ consists of a pair <op_1, op_2> if op_1 and op_2 are performed and if op_1 is performed more than two weeks before op_2.

Belonging relations of events: These relations are defined between concepts of events with no term and events with terms. For example, the following relation denotes the relations between operations and hospital stays that have operations.

⟨belonging⟩ ⊆ [operation] × [hospital stay].

The relation contains a pair (op, sty) of an event of an operation op and that of a hospital stay sty if op is performed in the duration of sty.

5) Special property OR

MSO has a special property with the label "OR." The domain and range of the OR property is the most general concept "class," which is a superclass of all concepts in MSO. Thus, we can give this property to all pairs of concepts in MSO. The property is used in the definition of objective graphs in the next section (Case 3 for the definition).

B. Objective Graphs

In the first part of Section III, we explained that QI-RS represents a quality indicator as a combination of a target concept of quantification and a quantification method. In this sub-section, we define the representation of a quantification target as a graph.

To define a quality indicator such as the example in Section II.A, one needs to define special concepts such as "inpatients that were diagnosed with stomach cancer and that survived more than five years since they received the diagnosis" as a quantification target. In most cases, such a special concept can be defined with a basic concept [patient] or [event] and some conditions on it that specializes the basic concept. Furthermore, such conditions can be defined with properties and (other) basic and/or special concepts. MSO defines the basic concepts and properties, while objective graphs represent special concepts and conditions to define special concepts based on MSO.

1) Syntax of Objective Graphs

Definition 1. An objective graph $\mathbb{G}$ consists of five components (N($\mathbb{G}$), R($\mathbb{G}$), E($\mathbb{G}$), L($\mathbb{G}$), C($\mathbb{G}$)), where

(i) N($\mathbb{G}$) is a set of nodes,
(ii) R($\mathbb{G}$) is the root node,
(iii) E($\mathbb{G}$) is a set of edges,
(iv) L($\mathbb{G}$) is a label function on N($\mathbb{G}$) ∪ E($\mathbb{G}$), and
(v) C($\mathbb{G}$) is a concept.

We define $\mathbb{G}$ by the induction on the structure of the node labels, as follows.

Case 1. Assume that the following data are given:
(a) concept C,
(b) attributes $A_1,\ldots, A_n$ of C, and
(c) values $a_1,\ldots, a_n$ of $A_1,\ldots, A_n$, respectively.
Then, we define an objective graph $\mathbb{G}$, as follows.
(i) N($\mathbb{G}$) := $\{*_0, \ldots, *_n\}$,
(ii) R($\mathbb{G}$) := $*_0$,
(iii) E($\mathbb{G}$) := $\{f_1,\ldots, f_n\}$, where each f_i is an edge from $*_0$ to $*_i$.
(iv) L($\mathbb{G}$)($*_0$) := C,
L($\mathbb{G}$)($*_i$) := a_i for $i = 1,\ldots, n$, and,
L($\mathbb{G}$)(f_i) := A_i for $i = 1,\ldots, n$,
(v) C($\mathbb{G}$) := C.
Note that if n = 0, then N($\mathbb{G}$) is the singleton set $\{*_0\}$ and E($\mathbb{G}$) is an empty set.

Case 2. Assume that the following data are given:
(a) an integer n with $n \geqq 1$,
(b) a set of objective graphs $\{\mathbb{G}_0, \ldots, \mathbb{G}_n\}$,
(c) a set of relations $\{R_1,\ldots, R_n\}$, where each R_i is a relation between C($\mathbb{G}_i$) and C($\mathbb{G}_0$),
(d) a function $n(i,j)$: $\{1,\ldots,n\}^2 \rightarrow \mathbb{N}$, where $\mathbb{N}$ is the set of integers.
(e) a set of relations $\{R^{i,j}_1,\ldots, R^{i,j}_{n(i,j)}\}$, where $0 \leqq i \leqq n$ and j with $0 \leqq j \leqq n$, and each $R^{i,j}_k$ is a relation between C($\mathbb{G}_i$) and C($\mathbb{G}_j$). (Note: if $n(i,j) = 0$, then $\{R^{i,j}_1,\ldots, R^{i,j}_{n(i,j)}\}$ is an empty set).
Then, we define an objective graph $\mathbb{G}$, as follows.
(i) N($\mathbb{G}$) := $\{*_0, \ldots, *_n\}$,
(ii) R($\mathbb{G}$) := $*_0$,
(iii) E($\mathbb{G}$) := $\{f_1,\ldots, f_n\} \cup (\cup_{0\leqq i\leqq n,\ 0\leqq j\leqq n}\{f^{i,j}_1,\ldots, f^{i,j}_{n(i,j)}\})$, where each f_i is an edge from $*_i$ to $*_0$ and each $f^{i,j}_k$ is an edge from $*_i$ to $*_j$.
(iv) L($\mathbb{G}$)($*_i$) := $\mathbb{G}_i$ ($i = 0,\ldots, n$),
L($\mathbb{G}$)(f_i) := R_i ($i = 1,\ldots, n$) and,
L($\mathbb{G}$)($f^{i,j}_k$) := $R^{i,j}_k$ ($i,j = 0,\ldots, n$ and $k = 1,\ldots, n(i, j)$).
(v) C($\mathbb{G}$) := C($\mathbb{G}_0$).

Case 3. Assume that the following data are given:
(a) an integer n with $n\geqq 1$,
(b) a set of objective graphs $\{\mathbb{G}_0, \ldots, \mathbb{G}_n\}$, where each C($\mathbb{G}_i$) ($i = 1,\ldots, n$) is a subclass of C($\mathbb{G}_0$).
Then, we define an objective graph $\mathbb{G}$, as follows.
(i) N($\mathbb{G}$) := $\{*_0, \ldots, *_n\}$,
(ii) R($\mathbb{G}$) := $*_0$,
(iii) E($\mathbb{G}$) := $\{f_1,\ldots, f_n\}$, where each f_i is an edge from $*_0$ to $*_i$.
(iv) L($\mathbb{G}$)($*_i$):=$\mathbb{G}_i$ ($i=0,\ldots, n$) and
L($\mathbb{G}$)(f_i):= OR ($i=1,\ldots, n$), where OR is a special property defined in Section IV.
(v) C($\mathbb{G}$):= C($\mathbb{G}_0$).

Each f_i and each $f^{i,j}_k$ in Case 2.(iii) are called a *main edge* of $\mathbb{G}$ and an *optional edge* of $\mathbb{G}$, respectively.

Let $\mathbb{G}$ be an objective graph. If $\mathbb{G}$ is defined in Case 1 of the above definition, then it is called an *atomic* objective graph. If $\mathbb{G}$ is defined in Case 2, then it is called a *complex* objective graph. Finally, if $\mathbb{G}$ is defined in Case 3, then it is called an *OR-type* objective graph.

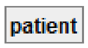
2) Example of an Objective Graph

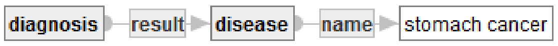
Here, we give an example of an objective graph. Let us consider the quality indicator "five-year survival rate of stomach cancer patients". The definition of the quality indicator is the ratio of the number of patients surviving five years to all stomach cancer patients, where a "stomach cancer patient" is a patient who was diagnosed with stomach cancer, and a "five-year surviving patient" is a patient diagnosed with stomach cancer but who is alive five years after the diagnosis. Thus, we will first express the set of five-year surviving patients in Fig.5. To this end, we construct three objective graphs $\mathbb{G}_0$, $\mathbb{G}_1$, and $\mathbb{G}_2$, as follows.

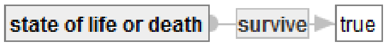
(1) $\mathbb{G}_0 = (\{*\}, *, \emptyset$ (empty set), L_0, [patient]), where $L_0(*) =$ [patient].

(2) $\mathbb{G}_1 = (\{*_0, *_1\}, *_1, \{f_1{:}*_0 \rightarrow *_1\}, L_1,$ [diagnosis]), where $L_1(*_0) =$ [diagnosis], $L_1(*_1) =$ ⟪stomach cancer⟫, $L_1(f_1) =$ ⟨result⟩ and [diagnosis] denotes an event concept, ⟪stomach cancer⟫ denotes an instance of the concept of diseases, and ⟨result⟩ denotes an attribute of the concept [diagnosis]. Note that the range of ⟨result⟩ is the concept of diseases.

(3) $\mathbb{G}_2 = (\{*_0, *_1\}, *_1, \{f_1{:}*_0 \rightarrow *_1\}, L_2,$ [state of life or death]), where $L_2(*_0) =$ [state of life or death], $L_2(*_1) =$ ⟪true⟫, $L_2(f_1)$ = ⟨survive⟩, [state of life or death] denotes the viability status of a patient, ⟪stomach cancer⟫ denotes an instance of the concept of diseases, and ⟨result⟩ denotes an attribute of the concept [diagnosis]. Note that the range of ⟨result⟩ is the concept of diseases.

(1) $\mathbb{G}_0$= 

(2) $\mathbb{G}_1$= 

(3) $\mathbb{G}_2$= 

We next construct an objective graph $\mathbb{G}$ of "five-year surviving stomach cancer patients", as follows.

(i) $N(\mathbb{G}) = \{*_0, *_1, *_2\}$,

(ii) $R(\mathbb{G}) = *_0$,

(iii) $E(\mathbb{G}) = \{f^1{:}*_1 \rightarrow *_0, f^2{:}*_2 \rightarrow *_0, f^{21}{:}*_2 \rightarrow *_1\}$,

(iv) $L(\mathbb{G})(*_i) = \mathbb{G}_i$ ($i = 0, 1, 2$),

$L(\mathbb{G})(f^1)$ = ⟨subject (of the event)⟩,

$L(\mathbb{G})(f^2)$ = ⟨subject (of the state) ⟩,

$L(\mathbb{G})(f^{21})$ = ⟨after more than <5 years>⟩,

(v) $C(\mathbb{G}) = C(\mathbb{G}_0) =$ [patient].

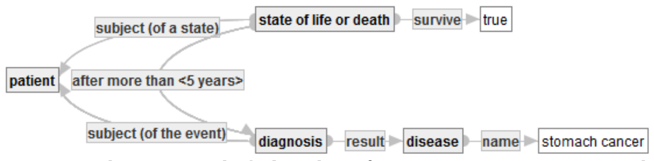


Figure 5. Objective graph $\mathbb{G}$ describing five-year surviving patients with stomach cancer.

3) *Semantics of Objective Graphs*

An objective graph $\mathbb{G}$ can be regarded as a concept denoted by $C(\mathbb{G})$ and modified by other concepts and properties that are denoted by $L(\mathbb{G})$. If each concept is identified with its extension, i.e., the set of instances of the concept, an objective graph can be identified with a subset of the set denoted by $C(\mathbb{G})$ that is obtained from $C(\mathbb{G})$ by restricting it by sets and functions denoted by $L(\mathbb{G})$. To clarify this identification, we define an interpretation of an objective graph, as follows.

Definition 2. For an objective graph $\mathbb{G}$, we define a set $[[\mathbb{G}]]$, which is called the *interpretation* of $\mathbb{G}$, as follows.

Case 1. Let $\mathbb{G}$ be an atomic objective graph. Then, $[[\mathbb{G}]] := \{c \in C(\mathbb{G}) | c.A_1 = a_1 \wedge \ldots \wedge c.A_n = a_n\}$, where $c.A_i$ is the value of the attribute A_i on c, and symbol $\wedge$ denotes the logical connective symbol "and."

Case 2. Let $\mathbb{G}$ be a complex objective graph. Then, $[[\mathbb{G}]] := \{x_0 \in [[\mathbb{G}_0]] | \exists x_1 \in [[\mathbb{G}_1]], \ldots, \exists x_n \in [[\mathbb{G}_n]] (\wedge_{i=1,\ldots,n} R^i(x_i, x_0)) \wedge (\wedge_{i,j=0,\ldots,n} (\wedge_{k=1,\ldots,n(i,j)} R^{i,j}_k(x_i, x_j)))\}$.

Case 3. Let $\mathbb{G}$ be an OR-type objective graph. Then, $[[\mathbb{G}]] := [[\mathbb{G}_0]] \cap ([[\mathbb{G}_1]] \cup \ldots \cup [[\mathbb{G}_n]])$.

Note that for all concepts C and D we assume that if D is a subclass of C then the extension of D is a subset of that of C.

4) *Segments of an Objective Graph*

To define some quantifying concepts in the next section, we will use an objective graph $\mathbb{G}^*$, which is a kind of a subgraph of $\mathbb{G}$. We call such an objective graph a *segment* of $\mathbb{G}$ and describe it by $\mathbb{G}^*$. $\mathbb{G}^*$ is obtained from $\mathbb{G}$ by reducing some conditions of $\mathbb{G}$ that are represented by nodes or edges in $\mathbb{G}$. In fact, $\mathbb{G}^*$ satisfies that $[[\mathbb{G}]] \subseteq [[\mathbb{G}^*]]$.

Due to space limitations, we omit the definition of segments, and just give an example in Fig.6, which is a segment of the objective graph $\mathbb{G}$ in Fig.5.

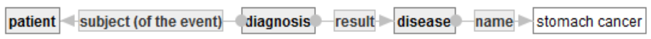


Figure 6. Segment $\mathbb{G}^*$ of $\mathbb{G}$.

C. *Graph Representation of Quality Indicators*

The main purpose of this study is to introduce transformation of MSO to relational data models and a method to transform quality indicators in QI-RS to queries on the relational data models. However, we will focus on only the transformations of MSO and objective graphs (see Sections IV and V for the reason). Thus, we explain quantification concepts and quality indicators in QI-RS very briefly.

1) *Quantifying Concepts*

A quantifying concept is a function that extracts the value from a given target that is expressed to be an objective graph. The function may have input data that are some attributes of the base concept of a given objective graph and another function on the values of the above attributes. Here, each concept is regarded as a set obtained from the concept under some situation. For example, in a hospital *A*, the concept "inpatient" is regarded as the set of inpatients in *A*.

There are generally three types of quantifying concepts:

a) *Total Numbers*

For a finite set *S*, the summation of numbers obtained from elements of *S* is called the total number of *S*. For example, if each element is assigned 1 denoting its existence, then the total number is the same as the cardinality of *S*. The quantifying concept ⋘cardinality⋙ is regarded as a function that has an objective graph $\mathbb{G}$ as input data and that outputs the cardinality of $[[\mathbb{G}]]$.

For a concept *S* and attributes A_1, …, A_n of *S*, a real-valued function on the product set of (extensions of) A_1,…, A_n is called an *attribute-quantifying function*. Moreover, for a concept *S*, attributes A_1,…, A_n of *S*, and an attribute-quantifying function f of $A_1,\ldots, A_n$, the summation $\Sigma_{s\in S}$ $f(s.A_1,..., s.A_n)$ is called the *total attribute number* of S with respect to $A_1,\ldots, A_n$ and f, where $s.A_i$ denotes the value of an instance s with respect to A_i.

The quantifying concept ≪total attribute number≫ is regarded as a function that has the following data as input data:

1. an objective graph $\mathbb{G}$,
2. attributes $A_1,..., A_n$ of C($\mathbb{G}$), and
3. an attribute-quantifying function f of $A_1,..., A_n$.

Moreover, the ≪total attribute number≫ outputs the total attribute number of [[$\mathbb{G}$]] with respect to $A_1,..., A_n$ and f.

b) Rate

For a finite set S and its subset S^*, the rate of the total number of S^* among the total numbers of S is obtained in the same way as that of calculating the total number of S^* is called a rate of S^* among S. In particular, the rate of the cardinality of S^* among that of S is called the cardinality rate of S^* among S. Moreover, the rate of the total attribute number of S^* with respect to $A_1,..., A_n$ and f among that of S with respect to the same attributes and the same attribute-quantifying function is called the total attribute number rate.

The quantifying concept ≪cardinality rate≫ is regarded as a function that has the following data as inputs:

1. an objective graph $\mathbb{G}$, and
2. a segment $\mathbb{G}^*$ of $\mathbb{G}$.

In contrast, the quantifying concept ≪total attribute number rate≫ is regarded as a function that has the following data as inputs:

1. an objective graph $\mathbb{G}$,
2. a segment $\mathbb{G}^*$ of $\mathbb{G}$,
3. attributes $A_1,..., A_n$ of C($\mathbb{G}$), and
4. an attribute quantifying function f of $A_1,..., A_n$.

Moreover, ≪total attribute number rate≫ outputs the rate of the total attribute number of [[$\mathbb{G}$]] with respect to $A_1,..., A_n$ and f among that of [[$\mathbb{G}^*$]] with respect to the same attributes and the same attribute-quantifying function.

c) Average

For concept S, attributes $A_1,..., A_n$ of S, and the attribute-quantifying function f, the ratio of the total attribute number of S with respect to $A_1,..., A_n$ and f and the cardinality of S is called the average of the value of S with respect to $A_1,..., A_n$ of f. The quantifying concept ≪average≫ is regarded as a function that has the same input data as that of ≪total attribute number≫ and that outputs the average of the value of S with respect to $A_1,..., A_n$ of f.

2) Outline of Quality Indicator Graphs

We now explain the graph representation of a quality indicator in QI-RS by showing examples of quality indicator graphs. We call such a graph-represented quality indicator a *quality indicator graph*.

In Figs. 6 and 7, we showed an objective graph $\mathbb{G}$ representing five-year surviving patients with stomach cancer and its segment $\mathbb{G}^*$, respectively. Now, we connect them with a quantifying concept ≪cardinality rate≫ in Fig.7. Then, the labeled graph that consists of $\mathbb{G}$, $\mathbb{G}^*$, and the new node is a quality indicator graph that represents the quality indicator in Section II.A.

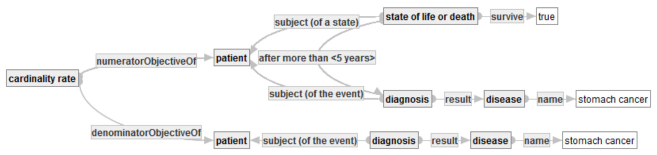


Figure 7. Quality indicator graph of five-year survival rate of stomach cancer patients.

IV. Transformation of MSO to GDM

In this section, we introduce a transformation of MSO to GDM.

A. *GDM and Transformation of MSO to GDM*

GDM is a relational data model [10] that is obtained from MSO by transforming MSO with a standard technique, where MSO is regarded as an entity relationship model (ERM) [11] in the following manner.

1) Classification of Concepts in MSO

Concepts in MSO are grouped into *entity-type* and *dataset-type*, as follows. Let C be a concept in MSO.

i. If C has some attribute(s), then C is an entity-type concept.
ii. Otherwise, C is a dataset-type concept.

Basically, a concept that is an objective of quantification is an entity-type concept. For example, subclasses of [actor] (essentially [patient]), [event], and [state (of a patient)] are entity-type concepts. In fact, there exist entity-type concepts that are not objectives of quantification but are used for defining quality indicators, but we do not explain them in this paper. For example, the concept [disease] in Fig.8 below is an auxiliary concept that has attributes, and hence, it is an entity-type concept.

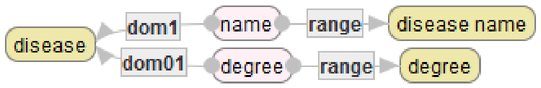


Figure 8. Concepts related to diseases.

On the other hand, a dataset-type concept is basically defined as a range of attributes of other concepts. Dataset-type concepts are further classified into *general dataset-type* and *medical knowledge-type*. For example, the concept [patient] has four attributes ⟨name⟩, ⟨birth⟩, ⟨sex⟩, and ⟨blood type⟩, while [disease] has two attributes ⟨name⟩ and ⟨degree⟩. The ranges of the attributes are concepts [human name], [birth], [sex], [blood type], [disease name], and [degree], respectively. These concepts are dataset-type. Specifically, [blood type] and [disease name] are medical knowledge-type concepts, while the others are general dataset-type concepts. The extension of a dataset-type concept is the same across hospitals. For example, the extensions of the above concepts are the same in every hospital. On the other hand, the extension of an entity-type concept such as [patient] is considered different among hospitals.

2) Classification of Properties in MSO

Properties in MSO are grouped into *relationship-type* and *attribute-type*, as follows. Let *P* be a property in MSO.

i. If *P* is a relation defined in Section III.A.4, then *P* is a relationship-type.
ii. If *P* is an attribute of a concept *C* in MSO and for every instance *c* contained in the extension of *C c* has at most one value with respect to *P*, then *P* is an attribute-type property.
iii. Otherwise, *P* is a relationship-type property.

From the MSO that can be regarded as the ERM by the above groupings, one can obtain a relational data model, which we describe by $\mathrm{GDM}_{\mathrm{MSO}}$ or simply GDM, using the standard transformation of ERMs to relational data models [11]. In fact, the relational data model GDM can be obtained from MSO by the following transformation of concepts and properties in MSO. We describe the transformation by | - |.

1. An entity-type concept *C* is transformed to an entity table |*C*|. Here, we set the primary key of |*C*|, which we describe by *C*-ID.
2. A dataset-type concept *D*, which has some concept(s) having attributes whose range is *D*, is transformed to a dataset (data-type) |*D*|.
3. A relationship-type property *R* between concepts C_1 and C_2 is transformed to a relationship table |*R*|. Here, we set a pair of C_1* and C_2* as the primary key of |*R*|, where, for *i* = 1 or 2, C_i* is C_i-ID (if C_i is an entity-type concept) or |C_i| (if C_i is a dataset-type concept).
4. An attribute-type property *A*, which has a concept *C* having *A* as its attribute, is transformed to an attribute |*A*| of the entity table |*C*|. Here, the data-type of |*A*| is the dataset |*D*| of the range *D* of *A* (if *D* is a dataset-type concept) or the set of values of the primary key of |*D*| (if D is an entity-type concept).

3) *Example of Tables in GDM*

We consider the concepts [patient] in Fig. 2 and [diagnosis] in Fig.9 below.

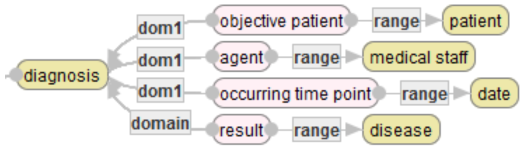


Figure 9. Diagnosis-related Concepts.

In Fig.9, the label "dom1" of the edge between ⟨objective patient⟩ and [diagnosis] indicates that each diagnosis, which is an instance *d* of [diagnosis], has a single patient that is the value of *d* with respect to ⟨objective patient⟩. Therefore, ⟨objective patient⟩ is an attribute-type property. Similarly, ⟨agent⟩ and ⟨occurring time point⟩ are attribute-type properties. On the other hand, ⟨result⟩ is a relationship-type property, since the label "domain" indicates that it is possible that some diagnosis has multiple diseases. On the other hand, [diagnosis], [patient] [medical staff] and [disease] are entity-type concepts, while [date] is a dataset-type concept.

From the above concepts and properties, one obtains the following entity and relationship types.

(Entity table of [diagnosis])
<u>diagnosis-ID</u>
objective patient: patient-ID
agent: medical staff-ID
occurring time point: |date|

(Entity table of [patient])
<u>patient-ID</u>
name: |name|
sex: |sex|
birth date: |date|
blood type: |blood type|

(Entity table of [medical staff])
<u>Medical staff-ID</u>
name: | human name|
sex: |sex|
birth date: |date|
affiliation: |department|

(Entity table of [disease])
<u>disease-ID</u>
name: |disease name|
degree: |degree|

(Relationship table of [result])
<u>diagnosis-ID</u>
<u>disease-ID</u>

Here, the keys with an underscore denote primary keys of tables. Moreover, "objective patient: patient-ID" indicates that the attribute "objective patient" has the data-type that is the set of values of patient-ID as a foreign key, while "occurring time point: |date|" indicates that the attribute "occurring time point" has the data-type |date| that is the dataset obtained from [date]. Other attributes have similar data-types.

In what follows, we often abbreviate brackets "[" and "]". For example, we abbreviate "[patient]-ID" as "patient-ID" and "|[date]|" as "|date|."

B. *Transformation of QI to Queries on GDM*

In this sub-section, we define a method of generating queries from quality indicators in QI-RS based on the transformation in the previous section and the semantics of objective graphs in Definition 2. A quality indicator graph consists of (an) objective graph(s) and a quantifying concept, and an objective graph represents a set of patients or events that is a target of the quantification represented by the quantifying concept. On the other hand, a quantifying concept represents a method of calculating numerical values from sets of patients or events represented by objective graphs. The calculation method is simple and independent of objective graphs. Thus, the transformation of a quantifying concept to the corresponding algorithm can be performed uniquely. Due to limitations of space, we define only a transformation of an objective graph to a query on GDM.

Definition 3. For an objective graph $\mathbb{G} = (\mathrm{N}, \mathrm{R}, \mathrm{E}, \mathrm{L}, \mathrm{C})$, we define an SQL query on GDM, which is described by $Q_{\mathbb{G}}$, as follows.

Case 1. Let $\mathbb{G}$ be an atomic objective graph with attributes $A_1,\ldots,A_n$ of C and values $a_1,\ldots,a_n$ of $A_1,\ldots,A_n$, respectively. Then, $Q_{\mathbb{G}}$:= SELECT ∗ FROM |C($\mathbb{G}$)|

WHERE $cond_1$ AND … AND $cond_n$.

Here, $cond_i$ is defined as follows.

i. If $|A_i|$ is an attribute of |C($\mathbb{G}$)|, then $cond_i$ is defined to be $(|A_i| = a_i)$.

ii. Otherwise, $cond_i$ is defined to be

(a_i IN (SELECT $|D_i|$ FROM $|A_i|$)),

where $|A_i|$ is a relationship table obtained from A_i and $|D_i|$ is a data-type obtained from the range of A_i.

Case 2. Let $\mathbb{G}$ be a complex objective graph with the same components as those in Case 2 of Definition 1. Then,

```
Q𝔾 := SELECT * FROM |C(𝔾)|
        WHERE
          EXISTS(SELECT * FROM | Q𝔾1 |,…, | Q𝔾n |
                   WHERE (
                    cond1 AND ⋯ AND condn
                    AND
                    (cond0,0,1 AND ⋯ AND cond0,0,n(0,0))
                    AND
                    (cond0,1,1 AND ⋯ AND cond0,1,n(0,1))
                   AND ⋯ AND
                   (condn,n,1 AND ⋯ AND condn,n,n(n,n)))).
```

Here,

$|Q_{\mathbb{G}i}|$ is the table obtained by the query $Q_{\mathbb{G}i}$,

$cond_i$:= (|C($\mathbb{G}$)|. C($\mathbb{G}$)-ID = $|R^i|$. C($\mathbb{G}$)-ID AND $|Q_{\mathbb{G}i}|$. C($\mathbb{G}_i$)-ID = $|R^i|$. C($\mathbb{G}_i$)-ID)

and

$cond_{i,j,k}$:= ($|Q_{\mathbb{G}i}|$. C($\mathbb{G}_i$)-ID = $|R^{i,j}_k|$. C($\mathbb{G}_i$)-ID AND $|Q_{\mathbb{G}j}|$. C($\mathbb{G}_j$)-ID = $|R^{i,j}_k|$. C($\mathbb{G}_j$)-ID).

Case 3. Let $\mathbb{G}$ be an OR-type objective graph with the same components as those in Case 3 of Definition 1. Then,

```
Q𝔾 := SELECT * FROM |C(𝔾)|
        WHERE
          EXISTS(SELECT * FROM | Q𝔾1 |,…, | Q𝔾n |
                    WHERE ( cond1 OR …OR condn)).
```

Here, $|Q_{\mathbb{G}i}|$ is the table obtained from the query $Q_{\mathbb{G}i}$ and $cond_i$:= (|C($\mathbb{G}$)|. C($\mathbb{G}$)-ID = $|Q_{\mathbb{G}i}|$. C($\mathbb{G}_i$)-ID).

Note that C($\mathbb{G}$) = C($\mathbb{G}_0$) in the above Cases 2 and 3.

Remark. The relation of time ordering defined in Section III.A.4 has a parameter of time length. Thus, in Case 2 of Definition 3, if R^i or $R^{i,j}_k$ is a relation of time ordering, $cond_i$ or $cond_{i,j,k}$ may have the third condition for such a parameter. For example, if $|R^i|$ is a relation of time ordering with a year parameter, then $cond_i$ may have an additional condition

$|R^i|$.|year parameter| = p,

where p has the value of year length.

1) *Example of queries on GDM*

We now construct a query on GDM from the objective graph $\mathbb{G}$ in Example 0 by Definition 3. By the transformation | - |, we obtain the following tables from the concept [state of life or death] and relations ⟨subject (of an event)⟩, ⟨state object⟩, and ⟨after more than <p>⟩ (Section 3.A.4), which are used to compose $\mathbb{G}$.

(Entity table of [state of life or death])
<u>state of life or death-ID</u>
subject (of a state): patient-ID
starting event: short term event-ID
terminating event: short term event-ID
starting time point: |time point|
terminating time point: |time point|
survive: |truth|

(Relationship table of ⟨subject (of an event)⟩)
<u>diagnosis-ID</u>
<u>patient-ID</u>

(Relationship table of ⟨state object⟩)
<u>state of life or death-ID</u> (We abbreviate it as “LorD-ID.”)
<u>patient-ID</u>

(Relationship table of ⟨after more than <p>⟩)
<u>diagnosis-ID</u>
<u>state of life or death-ID</u>
year parameter: |number|

By using the above tables and Example 1, we can obtain the query $Q_{\mathbb{G}}$ as follows.

```
Q𝔾 := SELECT * FROM |patient|
      WHERE
      EXISTS(
       SELECT * FROM | Q𝔾1 |, | Q𝔾2 |
       WHERE
        |patient|.patient-ID=|subject (of event)|.patient-ID
         AND
        |Q𝔾1|.diagnosis-ID=|subject (of event)|.diagnosis-ID
       AND
        |patient|.patient-ID=|state object|.patient-ID
       AND
        |Q𝔾2|.LorD-ID=|state object|.LorD-ID
       AND
       |Q𝔾1|.diagnosis-ID
                    =|after more than <p>|.diagnosis-ID
       AND
        |Q𝔾2|.LorD-ID=|after more than <p>|.LorD-ID
       AND
        |after more than <p>|.|year parameter|=5)
```

```
Q𝔾1 := SELECT * FROM |diagnosis|
       WHERE
       EXISTS(
        SELECT * FROM |Q𝔾11|
        WHERE
        |diagnosis|.diagnosis-ID=|result|.diagnosis-ID
        AND
        |Q𝔾11|.disease-ID=|result|.disease-ID)

Q𝔾11 := SELECT * FROM |disease|
         WHERE name = "stomach cancer"

Q𝔾2 := SELECT * FROM |state of life or death|
         WHERE survive = "true"
```

C. Elimination of Relationship Tables Obtained from MSO-Relations

Let us call a relationship table in GDM that is obtained from a relation in MSO an *MSO-relation-based table*. In this sub-section, we explain how MSO-relation-based tables can be eliminated. In fact, one can give a relation in MSO a canonical definition by using the attributes of domain concepts, as follows.

1. *Relations of patients and events* are determined by using the attribute ⟨objective patient⟩ of events.
2. *Relations of patients and states* are determined by using the attribute ⟨subject (of a state)⟩ of patient states.
3. *Relations of time ordering* are determined by using the attributes of starting, terminating, and occurring time points of events and patient states.
4. *Belonging relations of events* are determined by the attributes of time points, objective patients, and types of events.

Thus, one can regard MSO-relation-based tables as view tables defined by using other tables in GDM, and hence, for a query $Q_{\mathbb{G}}$ in Definition 3, one can replace all conditions in Q defined using relation-based tables from other conditions without an MSO-relation-based table.

However, it is still meaningful to consider MSO-relation-based tables. The reason is that some hospitals may need their own definitions of such relations. Especially, it is possible that the definitions of belonging relations of events have slight differences across hospitals that need to modify the conditions for the table to contain an instance. To calculate proper indicators of medical service quality, it would be useful to consider a framework of both canonical ways and special ways to deal with it.

V. Transformation of MSO to a Local Data Model

Based on the transformation of MSO to GDM, we briefly explain a transformation of MSO to a data model on a medical database in a hospital. We call such a data model a *Local Data Model (LDM)*. To realize transformation of MSO to LDM, it is necessary to develop a mapping between GDM and LDM. From the remark in the previous section, one can effectively omit MSO-relation-based tables from GDM. Thus, to develop a mapping between GDM and LDM, we need to develop entity tables and relationship tables (besides MSO-relation-based tables) as view tables on LDM, which we call proper relationship tables. Through the queries on LDM that are definitions of the view tables, one can obtain queries on LDM to calculate the value of a given quality indicator in QI-RS. In fact, for a given $\mathbb{Q}$ consisting of one or two objective graph(s) $\mathbb{G}$ (and $\mathbb{G}^*$), one can obtain a query $Q^+_{\mathbb{G}}$ (and $Q^+_{\mathbb{G}^*}$), as follows.

1. First, obtain a query $Q_{\mathbb{G}}$ on GDM, which is defined based on entity tables and proper relationship tables.
2. Then, replace the tables above by sub-queries on LDM that define the tables as view tables on LDM.
3. Then, one can obtain a nested query $Q^+_{\mathbb{G}}$ on LDM to calculate the values of the objective graph $\mathbb{G}$.

On the other hand, one can easily define an algorithm that is obtained from the quantifying concept in $\mathbb{Q}$. For example, if the quantifying concept is ≪cardinality rate≫, the algorithm only counts the numbers of rows in the tables that are calculated by $Q^+_{\mathbb{G}}$ and $Q^+_{\mathbb{G}^*}$, respectively, and shows the ratio between the numbers. Such an algorithm can be defined independently from the input data, the objective graphs. Thus, one can obtain the value of $\mathbb{Q}$ from the algorithm above based on $Q^+_{\mathbb{G}}$ and $Q^+_{\mathbb{G}^*}$.

VI. Related Works

A. Transformation of Queries on GDM into Those on LDM

The concept of a GDM and the transformation of GDM into LDMs in this paper have already been developed in previous research on distributed databases. Especially in the 1990s, many productive algorithms were developed for the transformation of queries on a GDM into those on LDMs (see, for example, chapter 9 of [12] or [13]). However, we need a representation system of quality indicators that satisfies compatibility of formality and understandability of quality indicator representation. Moreover, to guarantee the formality of the representation system, one needs to establish a concrete way to calculate values of quality indicators represented by the system. This paper shows a solution to the problem by transforming quality indicators (more precisely, objective graphs) in QI-RS into queries on LDMs.

B. Ontology-Based Information Retrieval

Ontology-based information retrieval has been actively investigated. In particular, research results on transformations between ontologies in RDF and relational data models on RDBs (see, for example, [14] or [15]) are closely related to the results in this paper. Moreover, ontology-matching (see, for example, [16] or [17]) has been investigated as the basis of ontology-based information retrieval.

The results of this paper are based on the transformation of MSO into GDM, and previous research can be consulted to reproduce these results. However, in order to actually realize a transformation from ontology such as MSO, which is developed for a special purpose such as the assessment of medical services, to a suitable data model, one still requires special techniques. For example, although D2RQ [18] is one of most useful tools to connect ontologies in RDF with relational data models, we found that it requires considerable customization and extension of the functions to translate MSO into GDM. The results in this paper can be regarded as the special requirements based on common techniques of transformation on RDF-ontologies into relational data models to transform MSO into GDM (or LDMs).

Moreover, to calculate the value of a quality indicator in QI-RS based on medical databases, one also needs to deal with the transformation of objective graphs and quantifying concepts, particularly the transformation of objective graphs into queries on GDM (and LDMs). This paper solves this problem by employing the semantics (interpretations) of objective graphs (Definitions 3 and 4).

VII. Conclusion and Future Work

This paper introduces a method to transform a quality indicator represented by Medical Service Ontology (MSO) to queries on a virtual relational data model called a Global Data Model (GDM). To this end, concepts and properties in MSO are grouped and accordingly transformed to entity and relationship tables in GDM. Moreover, based on a mapping from GDM to each relational data model on a medical database in a hospital, this study extends the method to transform a quality indicator to queries on the data models on medical databases. Thus, the value of the quality indicator based on MSO is automatically calculated based on data in the medical databases.

We have still not explained any method to perform mapping between GDM and the relational data model on a given medical database in detail. Moreover, we need to develop a method of generating queries on relational data models that are efficient from the viewpoint of computation. These issues will be solved in future works.

Acknowledgment

This work was supported by a JSPS KAKENHI Grant Number 24500167.

References

[1] OECD: Health Care Indicators Project Initial Indicators Report, OECD Health Working Papers 22 (2006).

[2] Nihon Hospital Organization: Clinical Indicators 2009, http://www.hosp.go.jp/7,7018,61.html, (in Japanese), [retrieved: 9, 2012].

[3] B. T. Collopy: Clinical indicators in accreditation: An effective stimulus to improve patient care, International Journal for Quality in Health Care 12(3) (2000), pp.211-216.

[4] J. Mainz, B. R. Krog, B, Bjørnshave, and P. Bartels: Nationwide continuous quality improvement using clinical indicators: The Danish national indicator project, International Journal for Quality in Health Care 16(1) (2004), pp.i45-i50.

[5] J. Mainz, A. M. Hansen, T. Palshof, and P. D. Bartels: National quality measurement using clinical indicators: The Danish national indicator project, Journal of Surgical Oncology 99(8) (2009), pp.500-504.

[6] O. Takaki, I. Takeuti, K. Takahashi, N. Izumi, K. Murata, and K. Hasida: Representation system of quality indicators towards accurate evaluation of medical services based on medical databases, Proceedings of the 4th International Conference on eHealth, Telemedicine, and Social Medicine (eTELEMED2012) (2012), pp.249-255.

[7] O. Takaki, I. Takeuti, K. Takahashi, N. Izumi, K. Murata, and K. Hasida: Representation system for quality indicators by ontology, Semantics - Advances in Theories and Mathematical Models, InTech, April (2012), pp.193-212.

[8] O. Takaki, I. Takeuti, K. Takahashi, N. Izumi, K. Murata, M. Ikeda, and K. Hasida: Evaluation of a representation system of quality indicators, Proceedings of the 10th Conference on Knowledge-Based Software Engineering (JCKBSE2012), Frontiers in Artificial Intelligence and Applications 240, IOS Press, (2012), pp.144-153.

[9] K. Hasida: Introduction to Semantic Editor, http://i-content.org/semauth/intro/index.html, (in Japanese), [retrieved: 9, 2012].

[10] E. F. Codd: A relational model of data for large shared data banks, Commun. ACM 26, 1 (1983), pp.64-69.

[11] P. P. S. Chen: The entity-relationship model: A basis for the enterprise view of data, In Proceedings of National Computer Conference (AFIPS '77), ACM, (1977), pp.77-84.

[12] M. T. Özsu and P. Valduriez: Principles of Distributed Database Systems (3rd Ed.), Springer, (2011).

[13] A. Y. Halevy: Answering queries using views: A survey, The VLDB Journal 10 (2001), pp.270-294.

[14] N. Konstantinou, D.-E. Spanos, and N. Mitrou: Ontology and database mapping: A survey of current implementations and future directions. J. Web Eng., 7(1) (2008), pp.1-24.

[15] S. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. Thibodeau Jr, S. Auer, J. Sequeda, and A. Ezzat: A Survey of Current Approaches for Mapping of Relational Databases to RDF, http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf, (2009), [retrieved: 9, 2012].

[16] N. F. Noy: Semantic integration: A survey of ontology-based approaches. SIGMOD Record, 33(4) (2004), pp.65-70.

[17] J. Euzenat and P. Shvaiko: Ontology Matching. Springer (2007).

[18] D2RQ: Accessing Relational Databases as Virtual RDF Graph, http://d2rq.org, [retrieved: 9, 2012].

BrainTool

A Tool for Generation of the UML Class Diagrams

Oksana Nikiforova, Konstantins Gusarovs, Olegs Gorbiks, Natalja Pavlova
Faculty of Computer Science and Information Technology
Riga Technical University
Riga, Latvia
e-mail: oksana.nikiforova@rtu.lv, konstantins.gusarovs@rtu.lv, olegs.gorbiks@rtu.lv, natalja.pavlova@rtu.lv

***Abstract*—Object-oriented system modeling enables the sharing of responsibilities between system objects at a high level of system abstraction. The UML class diagram is the central part of the object-oriented system model and serves as a "bridge" between the information about the problem domain at the customer's side and the software components at the developer's side. However, UML is not a methodology for how to model the system, but just a notation for "drawing" of model elements. This paper demonstrates the functionality of the BrainTool, which enables the generation of the UML class diagram from the so called two-hemisphere model, where the problem domain is presented as a concatenation of the problem domain processes, incoming and outgoing information flows and their types. BrainTool is developed using Visual Studio .NET for modeling of the two-hemisphere model, the Python programming language for definition of transformation rules and XMI for model interchange with Sparx Enterprise Architect.**

Keywords-BrainTool v1.0; UML class diagram; two-hemisphere model; model transformation.

I. Introduction

The object-oriented approach is widely used in software development. One of the tasks of software development is to present different aspects of the system for the implementation of the software solution for the required system. In solving this task, system modeling became an important activity in software development. The goal of system modeling is to represent the system graphically, in a form understandable to analysts, developers and at least partly understandable to the customer. A systematic approach to the derivation of the system model from information about the problem domain plays an important role in completing the task of system modeling.

K. Vollmer, C. Richardson, and Clair C. research [1] confirms that tools to support models and modeling at the initial stage of software development are the modern trend in business process modeling and analysis. Therefore, the focus of the automation of software development is shifted from automatic code generation from the UML diagrams to the automatic modeling of the UML diagrams and further code generation from them. Here, the valuable diagram became the UML class diagram, which specifies the structure of the developed system and static information about system behavior.

Moreover, the increasing interest towards software development within the framework of Model Driven Development (MDD) [2] turns the focus again to the area of model transformation at different levels of abstraction. Unified Modeling Language (UML) [3] is an industry standard for software specification and modeling in an object-oriented manner. The UML class diagram is used to model class specification and serve as a "bridge" between the information about the problem domain and the information required for definition of the software components and their architecture. Researchers are trying to achieve a high enough level of automation in creation of the UML class diagram and derivation of the diagram from information about the problem domain. Currently, an increasing number of developers admits the necessity to model system at the initial stage of the software development project, and the models are increasingly used to specify the system and its processes at the business level [1], [4].

BrainTool [5], developed by researchers of the Riga Technical University, is a step forward in the area of automation of the modeling process. There exist a number of tools which generate the UML class diagram. Some of them enable to define several elements of class structure based on data presentation of the problem domain. Others generate a class diagram from existing source code, to display the structure of the developed system. However, the problem of automatic generation of the UML class diagram from the formal and still customer-friendly presentation of problem domain is not solved yet. Authors of BrainTool propose to generate UML class diagram from the so-called two-hemisphere model [6] of the problem domain, which presents information about processes, information flows between these processes and pre-defined types of these information flows.

In 2004, when the main idea of the two-hemisphere model were published, the lack of the appropriate languages and standards eliminated the ability to support the two-hemisphere modeling of the system and to implement the transformations defined for it by tool. The evolution of the idea of model driven software development and appearance of different techniques for development of such modeling environment with embedded abilities for implementation of transformations gave us a powerful means to create such the tool.

The goal of this paper is to solve the task of tool development to support generation of the UML diagram from the initial model of the problem domain expressed in terms of the two-hemisphere model and to discuss about technical abilities of current solutions to create such a tool.

The paper is structured as follows. The next section describes the related work in the area of UML class diagram generation and tools supporting this generation. Section 3 explains the main principles of the transformations used for generation of the UML class diagram from the two-hemisphere model. The essence of the two-hemisphere model is also described in the third section. Section 4 gives several explanations on the solutions used for implementation of BrainTool and shows its main components. Several conclusions on the application of BrainTool and directions for future research are stated in the fifth section.

II. RELATED WORK

Since the beginning of the 1980s, a great number of modeling tools and model generating software systems have been offered to attack problems regarding software productivity and quality [7]. Modeling tools developed since that time were oversold on their "complete code-generation capabilities" [8]. Nowadays, similar situation is observed in modeling tools, using and integrating UML models at different levels of abstraction and automation of software development [9].

Most of today's tools combine a number of modeling and code generation functions in a more or less open fashion. The traditional modeling tools provide a model editor and a model repository. A code generator based on a scripting language and plugged into a modeling tool provides the transformation tool and transformation definition editor. In that case, the transformation repository is simply text files [10].

The variety of the "model-driven" tools can be divided into tools created for defining the system model itself and tools to support code generation from the UML model. The first group of tools is so-called "UML editors", where the developers of these tools provide different levels of automation of the actual model creation. BrainTool demonstrated in this paper can be classified as a tool for automatic creation of UML class diagrams, where the result of the generation – the UML class diagram – is importable either into UML editors for further refinement and working with model or into code generation tools for further generation of software components.

Loniewski et al. in [11] describe the results of a survey about different approaches used for transformation of system requirements to system design and implementation. The survey shows the result of analysis of different approaches to transformation of the problem domain description into the UML class diagram during the last 10 years, published in four digital libraries (IEEEXplore, ACM, Science Direct, Springerlink). The survey states that there exist many approaches with different types of solutions for the generation of a UML class diagram. Moreover, the authors analyze the approach based on several criteria, one of them is tool support. Analysis of the automation level in these approaches shows that 25 out of 71 approaches described in corresponding papers are supported with a tool. However, Loniewski et al. stress that these tools are academic tools and are not widely practically used as far as they are created to approve the automation level of the approach offered by their vendors [11].

One more kind of the related tools are tools that generate the class diagram from a data structure or a data model. These tools require a solid contribution from a software specialist to define all these structures. It is already the modeling of the UML class diagram itself. In contrast to these tools, BrainTool generates the class diagram from initial information about the system, which is understandable for the business analyst and doesn't require software knowledge for its modeling. Therefore, a tool that generates the class diagram in the initial stages of the project is very useful. It allows to automatically create a static structure of the developed system and serves as a base for further code, avoiding mistakes and mismatches between requirements and implementation.

As far as for the evolution of the two-hemisphere model, which is a base model for generation of the UML class diagram in BrainTool, the main idea of displaying the initial information about the system with two interrelated models– the business process model and the concept model – and the hypothesis about how to use two interrelated model to share the responsibilities between object classes was demonstrated on the abstract example in [6] and later in several real projects. In all cases, the two-hemisphere model was created manually, in the first one by authors and in others by an independent problem domain expert. Successful application of two-hemisphere model transformation into the UML class diagram served as a motivation to support these transformations by software system. The first software prototype of tool supporting two-hemisphere model based transformation was introduced in 2008 [12]. The prototype used textual information in special format as a source and produced a text file containing description of the resulting UML class diagram as a specification, where classes, attributes, methods and relationships were listed in pre-defined format. Analysis of these generated text files gave authors an ability to refine transformations for definition of relationships between classes; the results are published in [13]. Currently, the ability to apply the two-hemisphere model for generation of the UML sequence diagram with attention to the timing aspect is investigated and preliminary results are published in [14]. So far, the continuing research in the area of model-driven software development and an increasing demand in the industry for automation of the ability to bridge the gap between problem domain and software components, can serve as a motivation to develop the first version of BrainTool, which gives an ability to draw the two-hemisphere model in the manner suitable for the problem domain expert and to generate the UML class diagram from it.

Moreover, instead of manual creation of the UML class diagram directly from information about problem domain based on principles of object-oriented analysis, the proposed BrainTool gives an ability to use already existing business artifact – a business process diagram is widely used in many enterprises, and the structure of information flows between processes is definable under description of user stories. A lot of organizations are using different tools for business process analysis and therefore they have complete and consistent models of their organizational structure, employer responsibilities, business processes and the structure of documentation flows, in other words, well-structured initial business knowledge, which can serve as a basis for even automatic creation of the two-hemisphere model.

The main benefit of the two-hemisphere model is that it can be created and often already is created by the business analyst at the customer's side. A Standish group survey shows that about 83% of companies are engaged in business process improvement and redesign. This implies that many companies are very familiar with business process modeling techniques or at least they employ particular business process description frameworks [4], [15]. On the other hand, the practice of software development shows that functional requirements can be derived from the problem domain description as much as 7 times faster than if trying to elicit them directly from users [16]. Both facts mentioned above and the existence of many commercial and open source business modeling tools are a strong motivation to base software development on the business process model, rather than on any other soft or hard models.

Therefore, with minimal efforts, the two-hemisphere model, which is created and intuitively understandable by the customer, can be used to automatically generate class diagram prototypes that can be later reviewed and used in software development.

III. TRANSFORMATION OF TWO-HEMISPHERE MODEL TO THE UML CLASS DIAGRAM

The two-hemisphere model driven approach uses the transformation of graphs, where nodes of one graph become the edges of the other graph, and edges of the first graph become the nodes of the other. These two initial interrelated graphs are: business process model (shortly – process model), which displays behavior of the system and the model of conceptual classes (shortly – concept model), which displays a skeleton of system's static structure. The meaning of objects in an object-oriented philosophy gives a possibility to share responsibilities between objects based on the direct graph transformation, where the data flow outgoing from the internal process in the process model becomes the owner of this process for performing it as an operation in object communication.

The essence of the transformation is illustrated on the left side of Figure 1. The business process model (graph G1 in Figure 1) is interrelated with the concept model (graph G2 in Figure 1) as follows. A certain concept in the concept model defines the data type for one or several data flows between business processes. The business process model is transformed into an object communication diagram (graph G3 in Figure 1), where edges (i.e., data flows) of the business process model became nodes (i.e., objects) of communication diagram, and nodes of business process model (i.e., processes) became edges (i.e., messages to perform the operation) of the communication diagram. The communication diagram itself serves as a base for the definition of classes-owners of methods in the UML class diagram. Details about the application of the two-hemisphere model are expressed in [12].

The right side of Figure 1 shows the interpretation of the transformations defined by the approach after the transformations have been studied for the implementation.

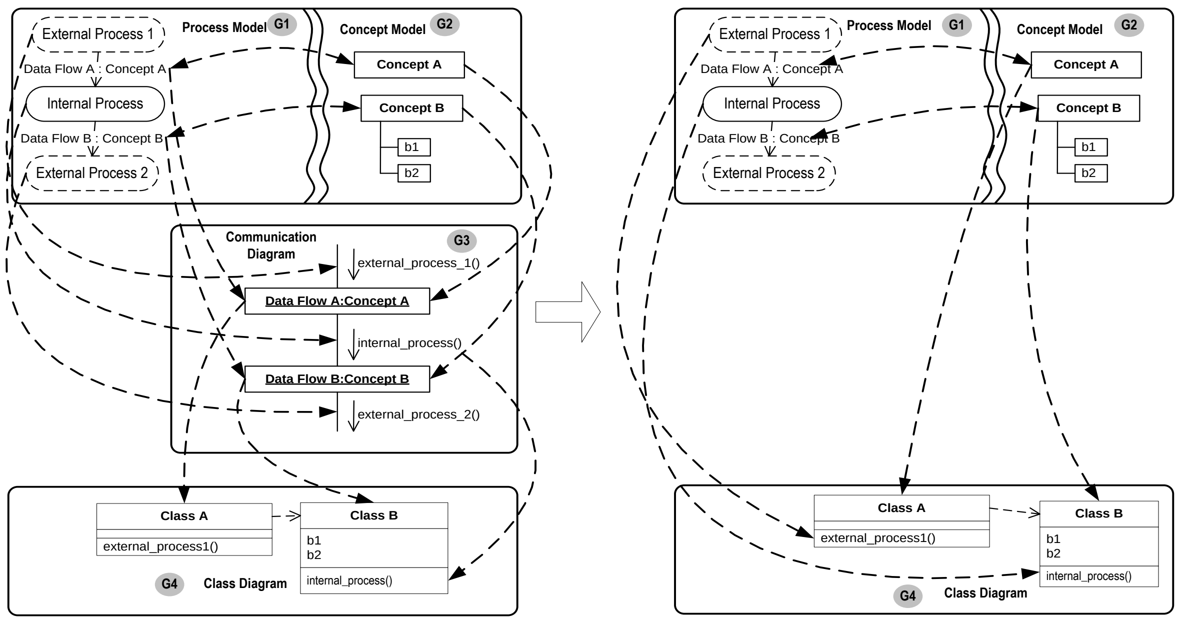


Figure 1. Interpretation of the transformations from two-hemisphere model to the UML class diagram.

Therefore, the left side of Figure 1 shows the transformations defined by the approach of the theoretical investigation of the sharing responsibilities among objects and the right side of Figure 1 shows the situation as it is simplified for tool development. The elements of the business process model are transformed into the UML class diagram directly. The edges of the business process model became the nodes of the UML class diagram. The nodes of the business process model became the methods of the classes, which were the outgoing data flows of the exact business process.

The analysis of different situations, which may appear in drawing the process model, i.e., a number of incoming and outgoing data flows, a variety of their types and so on, has given a possibility to define various transformation cases depending on the number of input and output processes and cardinality (a set of different concepts assigned to a certain data flow). These transformation cases are implemented according to the definition of relationships between generated classes, which are expressed in [13].

IV. Implementation of Two-Hemisphere Model in BrainTool

The goal of the prototype tool implementation in 2008 [12] was to examine the efficiency of the proposed method and to confirm that transformations offered by the two-hemisphere model driven approach can be automated. The current version of the implementation of the two-hemisphere model driven approach can be stated as a standalone tool entitled BrainTool in correspondence with the title of the approach, which in turn is derived from cognitive psychology [17] by analogy with human brain consisting of two interrelated hemispheres.

According to [10], a modern trend in system modeling tools is having the components to implement a model editor, a repository, its validation and transformation to another model. BrainTool gives a possibility to create the two-hemisphere model, to save it in the defined repository, to apply all the defined transformations for generation of the UML diagram and to export it in XMI format.

The Model Editor is a part of the tool providing model creation and modification possibilities. Model Repository is the "database" for models, where they are stored. The Transformation Definition Editor is used for transformation definition construction and modification. Currently, the Python interpreter is being used to support this component. However, it is possible to define the transformation in any programming language.

Finally, The Model Validator is a component used to check if the model is well-enough defined and has no potential problems that can affect the transformation result. This component is implemented as a standalone transformation using the Python programming language. The next subsections describe the main components of BrainTool and technical solutions for their implementation in detail.

A. Model Editor

The two-hemisphere model can be designed and then transformed using BrainTool model editor shown in Figure 2.

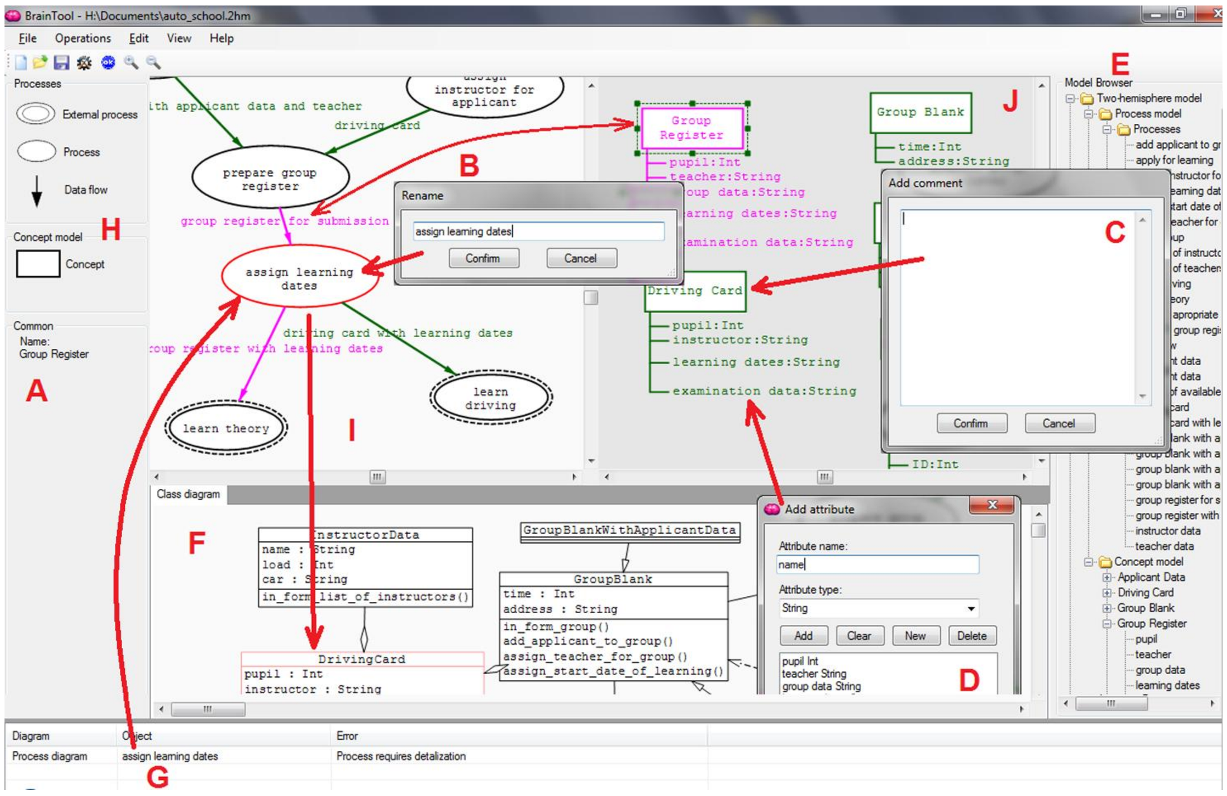


Figure 2. Model editor view in BrainTool.

The screen of BrainTool is divided into three parts. The information panel (highlighted as H in Figure 2 on the left side of the screenshot) shows the list of elements defined by the two-hemisphere model, where panel (A) provides basic information on the currently selected element.

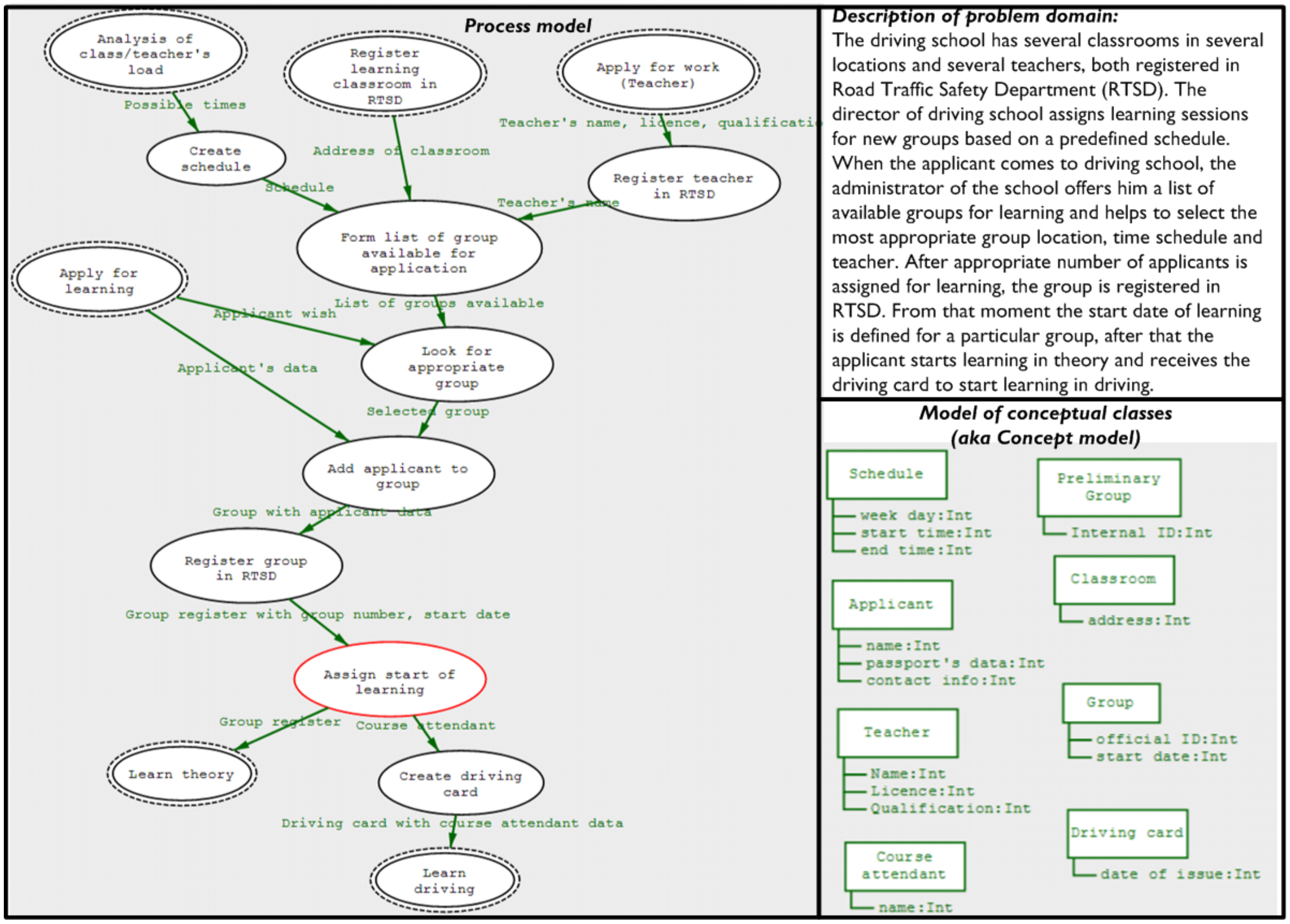
The central part is divided into three drawing frames – the process model (I), the concept model (J) and the resulting class diagram (F). Any element of the two-hemisphere model can be entitled (highlighted as B for processes and D for attributes in Figure 2) or commented (C). The tree view of all objects defined in all models (including the resulting model) is shown in the right part of the screen shot under the letter E in Figure 2. The diagram elements causing transformation problems are listed at the bottom of the screen (G). Such problematic elements are also highlighted in the model perspective.

The simplified version of the business process for the driving school is reflected in Figure 3, where the process model is presented on the top side and the model of conceptual classes (so called concept model) is presented on the bottom side of the figure, which presently is the screen shot from BrainTool.

B. *Transformation Definition*

According to the transformation definition – a transformation is the automatic generation of a target model from a source model [10]. In the case of BrainTool, the source model is presented as a two-hemisphere model consisting of the process diagram, a set of concepts and concepts assigned to data flows. The target is the UML class diagram, which is a set of classes, class methods, class attributes, interfaces and relationships between classes and interfaces. The first transformation task is to generate classes of the resulting UML class model. Classes are created from concepts and retain their attributes. Cardinalities (number of different concepts linked to separate data flows) of process' inputs and outputs are used to determine different types of the relationships between classes in the UML class diagram.

For example, outgoing multiple data flows assigned to the same concept give an ability to define the generalization. The transformation rules give a possibility also to define aggregation, dependency or at least simple association. The following high-level pseudocode expresses the idea of the transformation for class creation:



Description of problem domain:

The driving school has several classrooms in several locations and several teachers, both registered in Road Traffic Safety Department (RTSD). The director of driving school assigns learning sessions for new groups based on a predefined schedule. When the applicant comes to driving school, the administrator of the school offers him a list of available groups for learning and helps to select the most appropriate group location, time schedule and teacher. After appropriate number of applicants is assigned for learning, the group is registered in RTSD. From that moment the start date of learning is defined for a particular group, after that the applicant starts learning in theory and receives the driving card to start learning in driving.

Figure 3. Two-hemisphere model of a driving school.

```
func generate_classes(process_model pm,
concept_model cm, class_model clm)
  for each concept in cm do
    clm.create_class_from(concept)
  for each process in pm do
    u_inputs = node.input_set().cardinality
    outputs = len(node.output_set())
    u_outputs = node.output_set().cardinality

    if u_inputs = 1 and u_outputs = 1 and outputs != 1 then
      for each output in node.output_set() do
        clm.create_class_from(output)
        clm.define_generalization(output, node.input_set())
```

Processes from the process model become the class methods as a result of the transformation expressed in such pseudocode fragment:

```
func assign_methods(process_model pm,
concept_model cm, class_model clm)
  for each process in pm do
    classes = node.get_classes(clm)
    inter = null
    if len(classes) > 1 then
      inter = clm.create_or_get_interface(classes, node)

    for each c in classes do
      c.add_method(node)
      if inter != null then
        clm.define_realisation(c, inter)
```

The method assignment to classes allows to also define interfaces and realization relationships in the UML class diagram. So, as a result, the target model consists of classes with methods and attributes, interfaces with methods and five kinds of relationships: generalization, dependency, aggregation, association and realization. An example of generated UML class diagram for the problem domain of a driving school described in Section 4.1 is shown in Figure 4.

C. *Export of the UML Class Diagram to the UML Compliant Tool*

After elements of the two-hemisphere model are transformed into the class diagram, BrainTool gives the possibility to export it in XMI format to be later used in other UML editor or code generators that are able to import UML class diagrams in XMI format. Currently, most UML compatible tools use their own modifications of the XMI format and a developer cannot be sure about the result of import/export [18]. Therefore, the authors were forced to adjust the exported XMI for the requirements of a specific corresponding tool. The Sparx Enterprise Architect [19] is selected for the experiment, and the result of the implemented chain is shown in Figure 5. It is not a problem to define the elements of the UML class diagram according to the specific requirements for import in any other UML tool.

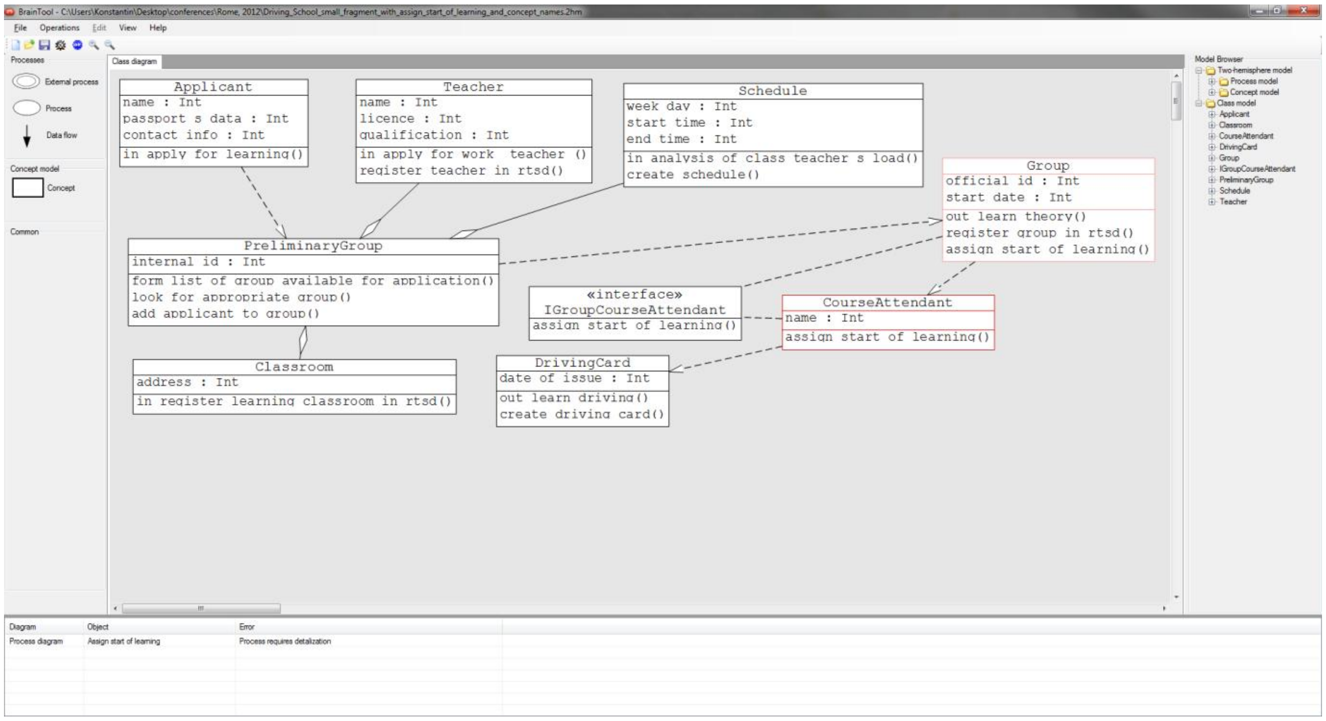
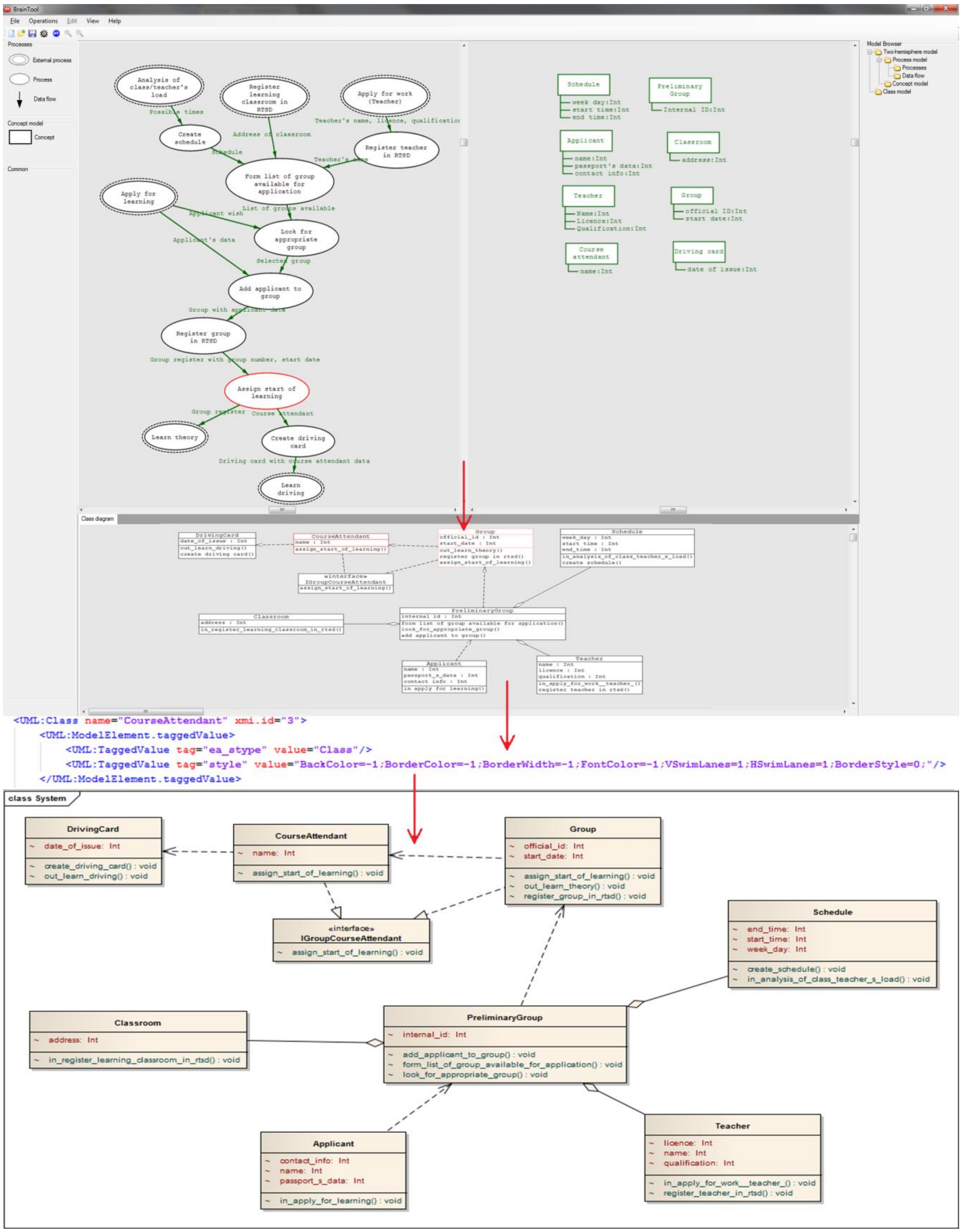


Diagram	Object	Error
Process diagram	Assign start of learning	Process requires detalization

Figure 4. Resulting UML class model for driving school.



```
<UML:Class name="CourseAttendant" xmi.id="3">
    <UML:ModelElement.taggedValue>
        <UML:TaggedValue tag="ea_stype" value="Class"/>
        <UML:TaggedValue tag="style" value="BackColor=-1;BorderColor=-1;BorderWidth=-1;FontColor=-1;VSwimLanes=1;HSwimLanes=1;BorderStyle=0;"/>
    </UML:ModelElement.taggedValue>
```

Figure 5. Export to SPARX Enterprise Architect.

D. Model Validator

The two-hemisphere model driven approach has several limitations in definition of the UML class diagram. They are expressed in several combinations of incoming and outgoing data flows of certain process. In this case, BrainTool highlights the problematic process in the two-hemisphere model and the potential owner in the UML class diagram.

The modeler is then required to create the sub-process diagram for the highlighted process, as it is shown in Figure 6.

The problematic process is being detailed in the following manner:

A – identify the problem.

B – receive the working area for creation of sub-process diagram

C – create sub-process diagram.

D – confirm sub-process diagram.

E – transfer sub-process diagram into main model.

For example, if there are at least two data flows outgoing from the process, which are typified by different conceptual classes and are differently typified from incoming data flows, the two-hemisphere model driven approach offers to refine the problematic process by dividing it into sub-processes. In order to support these treatments BrainTool gives the possibility to validate the two-hemisphere model developed by the modeler and to define processes, which do not give the clear ability to define a corresponding method's owner for the UML class diagram.

What is more the preliminary structure of the sub-process diagram already contains the incoming and outgoing information flows derived from the main model and the concerned external processes. The modeler is asked to divide the problematic process into a number of separated sub-processes and to define outgoing information flows more precisely (see area B in Figure 6).

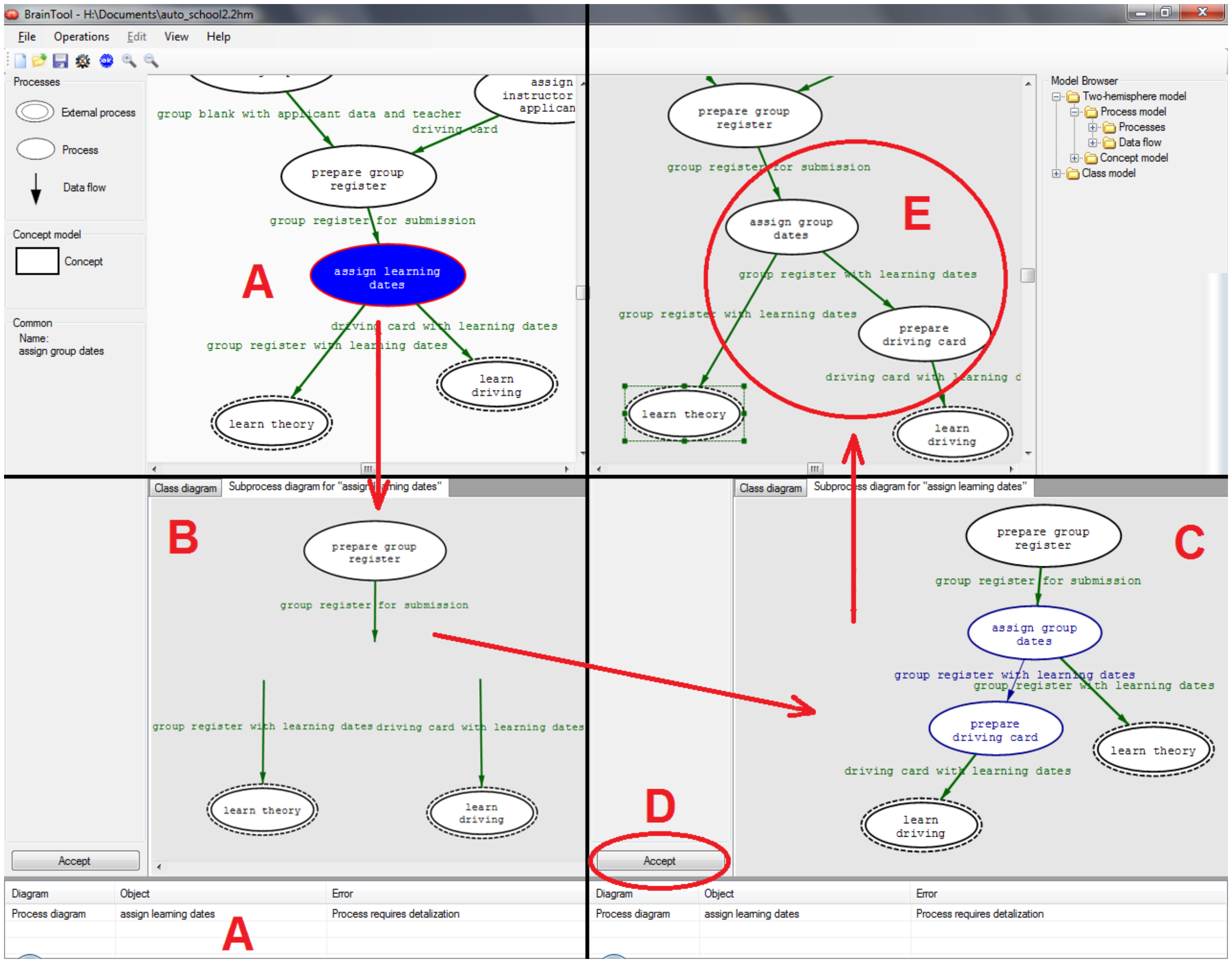


Diagram	Object	Error
Process diagram	assign learning dates	Process requires detalization

Diagram	Object	Error
Process diagram	assign learning dates	Process requires detalization

Figure 6. Model validation for the necessity to define the sub-process diagram.

V. CONCLUSION

Nowadays, the usage of model transformations has become a widespread practice and tools supporting such transformations have become increasingly popular. The main goal of the research presented in this paper was to implement a tool, which can generate the UML class diagram from the initial presentation of problem domain. BrainTool gives a possibility to automatically generate the UML class diagram and to export it to any UML compliant modeling environment supporting the XMI [20] format for model interchange. Due to the fact that modern modeling tools use their own variations for model interchange, the authors this time have chosen SPARX Enterprise Architect for integration with BrainTool in XMI import domain and have tuned the output of BrainTool suitable to import it into SPARX Enterprise Architect. The wide use of the unified standard for model interchange in modeling tools will increase the number of tools that can be integrated with BrainTool.

When the essence of the transformation from the source model to the target model is clearly defined, the creation of a tool supporting such a transformation becomes a programming task and it can be solved in two ways. The first way is to use a special transformation language and environments supporting model-driven software development. Another way, which was applied by authors of this paper, is to use general purpose programming languages for regular software implementation and consider the task of developing such a tool as a software development task. The tool specification, including description of the required use cases, was defined.

The definition of source and target models was used to define corresponding data structures; transformation definition was automated using a scripting language. As a result, the authors have implemented a tool that supports creation, editing and validation of the two-hemisphere model and its transformation into the UML class model, where the generated class diagram can further be imported into some UML editor or code generator.

Despite the successful project and the expected result of having a working tool, which enables to draw the initial information about problem domain in the form of the two-hemisphere model and to generate from it the UML class diagram, several problems are left unresolved. One of these problems is a cross-tool model exchange. Various modeling tools support XMI export and import, but, unfortunately, in most cases, the tool defines its own XMI-based format and thus common model interchange standard needs to be defined and implemented. As for BrainTool, the problem was temporarily solved by choosing one concrete tool, namely, Sparx Enterprise Architect, and adjusting the exported XMI schema in correspondence with its requirements.

Another unresolved problem is the layout of the generated diagram. There is no complete algorithm for automatic layout of the UML class diagram, therefore for now BrainTool requires manual layout of the resulting UML class diagram. However, this problem is not tool-specific and the layout algorithm can be integrated with BrainTool at any moment.

The main contributions of the research in comparison with authors' previous papers in the area are as follows:

1) *BrainTool has a standalone modeling editor for the two-hemisphere model. We improved the lack of supporting software prototypes developed in 2008, which required import of text files describing the elements of the model.*

2) *A set of transformations for identification of the elements of the UML class diagram from the two-hemisphere model was refined during the programming to simplify the transition from processed in the process model into operations in classes. Several corrections of the transformation for relationships identification were made during implementation. The approach was improved by the implementation of transformations into the tool.*

3) *BrainTool has its own model validator, which had not been implemented in the software prototype. It allows identifying processes needed to be refined to complete transformations. This ability gave authors a base for further research of transformation capacity from the two-hemisphere model.*

4) *An import of the developed UML class diagram into the UML compatible tools bridges the gap between computation independent modeling of the system and software components could be generated from the UML class diagram.*

Within the development process, several new possibilities of two-hemisphere approach were investigated. Authors believe that current transformation rules can be improved in order to reduce the number of limitations currently existing in the two-hemisphere model driven approach, to generate a more precise UML class model and more complete set of the class diagram elements for further using this model for code generation. In turn, several new facilities of code generation directly from the two-hemisphere model were also stated.

Authors' future work will be focused on the implementation of a refined version of BrainTool with respect to creation of two-hemisphere model of BrainTool itself and generating the UML class diagram for its development. We expect interesting results in comparison of the UML class diagram “as is” in the current version of BrainTool with the one generated by the tool.

ACKNOWLEDGMENT

The research presented in the paper is partly supported by Grant of Latvian Council of Science No. 09.1245 "Methods, models and tools for developing and governance of agile information systems".

The travel costs and participation fee to conference was supported by the European Regional Development Fund project «Development of international cooperation projects and capacity in science and technology Riga Technical University», Nr. 2DP/2.1.1.2.0/10/APIA/VIAA/003

REFERENCES

[1] K.Vollmer, C. Richardson, and C. Clair, “The importance of matching BPM tools to the process,” 2010. Available at: http://www.forrester.com/ (last access in March, 2012).

[2] T. Stahl and M. Volter, “Model-Driven Software Development,” Wiley & Sons, 2006.

[3] OMG “UML Unified Modeling Language Specification”. 2011. Retrieved from: http://www.omg.org (last access in March, 2012).

[4] P. Rittgen, “Quality and perceived usefulness of process models,” 25th Symposium On Applied Computing. ACM, 2010, pp. 65-72.

[5] Website of BrainTool. Available at http://brain-tool.org/ (last access in March, 2012).

[6] O. Nikiforova and M. Kirikova, “Two-Hemisphere Model Driven Approach: Engineering Based Software Development,” CAiSE 2004 16th International Conference on Advanced Information Systems Engineering June 7-11, Proceedings, 2004, pp. 219-233. Riga, Latvia.

[7] R. Balzer, “A 15 year perspective on automatic programming,” IEEE Transactions on Software Engineering, 11 (No 11), 1985, pp. 1257-1268.

[8] J. Krogstie, “Integrating enterprise and IS development using a model driven approach,” 13th International Conference on Information Systems Development – Advances in Theory, Practice and Education. Vasilecas O. et al. (Eds). 2005. Springer Science+Business media, Inc. pp.43-53.

[9] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, “MDA Distilled. Principles of Model-Driven Architecture,” Addison-Wesley, 2004.

[10] A. Kleppe, J. Warmer, and W. Bast, “MDA Explained: The Model Driven Architecture – Practise and Promise,” Addison-Wesley, 2003.

[11] G. Loniewski, E. Insfran, and S. Abrahao, “A Systematic Review of the Use of Requirements Techniques in Model-Driven Development,” 13th Conference, MODELS 2010, Model Driven Engineering Languages and Systems, Part II, Oslo, Norway, 2010, pp. 213—227.

[12] O. Nikiforova and N. Pavlova, “Development of the Tool for Generation of UML Class Diagram from Two-Hemisphere Model,” Proceedings of The Third International Conference on Software Engineering Advances (ICSEA), International Workshop on Enterprise Information Systems (ENTISY). Mannaert H., Dini P., Ohta T., Pellerin R. (Eds.), Published by IEEE Computer Society, Conference Proceedings Services (CPS), 2008, pp. 105-112.

[13] O. Nikiforova and N. Pavlova, “Foundations on generation of relationships between classes based on initial business knowledge,” Information Systems Development: Towards a Service Provision Society. Springer US, 2009, pp. 289–297.

[14] O.Nikiforova, “Object Interaction as a Central Component of Object-Oriented System Analysis,” International Conference „Evaluation of Novel Approaches to Software Engineering” (ENASE 2010), Proceedings of the 2nd International Workshop „Model Driven Architecture and Modeling Theory Driven Development” (MDA&MTDD 2010), Osis J., Nikiforova O. (Eds.), Greece, 2010, pp. 3-12. SciTePress.

[15] H. Peyret and D. Miers, “The Shifting Market For Business Process Analysis Tools,” Forester research, 2010.

[16] W. M. P. van der Aalst, “Trends in business process analysis: From validation to process mining,” International Conference on Enterprise Information Systems, 2007.

[17] J. Anderson, “Cognitive psychology and its implications,” W.H. Freeman and Company, New York, 1995.

[18] O.Nikiforova, N. Pavlova, A. Cernickins, and T. Jakona, “Certification of Model-Driven Architecture Tools: Vision and Application,” Proceedings of the ICSEA 2011 - The Sixth International Conference on Software Engineering Advances. Lavazza L. et al (eds.), IARIA ©, Barcelona, Spain, October 23-29, 2011, pp. 393-398.

[19] Sparx, 2012. Sparx Enterprise Architect. Official page of Sparx Enterprise Architect tool. Available at: http://www.sparxsystems.com (last access in March, 2012).

[20] Information Technology, 2005. XML Metadata Interchange (XMI), International Standard ISO/IEC 19503:2005(E), ISO/IEC.

Decoupled Model-Based Elicitation of Stakeholder Scenarios

Gregor Gabrysiak, Regina Hebig, and Holger Giese
Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
{gregor.gabrysiak|regina.hebig|holger.giese}@hpi.uni-potsdam.de

***Abstract*— Requirements engineers iteratively elicit scenarios by capturing and combining individual stakeholder perspectives into a consistent overall scenario model. This model has to be validated to exclude elicitation errors and check whether all alternatives are covered. While involving all stakeholders at once is considered beneficial, it is usually not feasible due to scheduling and resource constraints. Consequently, techniques that permit all stakeholders to be involved in the elicitation and validation independently, i.e., temporally and locally decoupled, are required. In this paper, we present an approach that enables stakeholders to participate in the elicitation of their collaborative scenarios remotely and decoupled from other stakeholders. The resulting fragmentation of the elicited scenarios is overcome by allowing stakeholders to express their expectation on how a scenario is usually complemented by activities of other stakeholders. Our approach systematically combines these decoupled perspectives to establish the overall scenario model.**

Keywords-decoupled requirements elicitation; scenario synthesis; incomplete scenarios.

I. Introduction

A requirements engineer *gets people to tell the stories of what their systems are meant to do*, as pointed out by Alexander and Maiden [2]. For complex systems with multiple, collaborating stakeholder groups, a requirements engineer needs to listen to all of their stories and to synthesize these stories into suitable scenario models. Among other aspects, these scenarios have to capture how the involved stakeholders interact to achieve their common goals.

The requirements engineers start by eliciting scenarios from individual stakeholder perspectives. After combining these separate scenarios into a consistent overall scenario model, they validate this model to exclude elicitation errors and check whether they covered all alternatives. Then, these initial activities continue iteratively with additional elicitation activities, updates of the scenario model, and subsequent validation activities until the result stabilizes. The overall scenario model is the crucial element to ensure that a consistent understanding of the different stakeholder perspectives can be established.

The elicitation and validation of such scenarios requires less effort if all stakeholders are involved simultaneously. By directly commenting on whether they agree with the statements of other stakeholders, the requirements engineer might obtain a commonly agreed-upon scenario model directly within an elicitation session [15]. However, due to scheduling and resource constraints such a setting is usually not feasible, if not less efficient compared to elicitations with individual stakeholders [17]. Also, experience shows that in case of group meetings social effects can result in suppressing opinions and observations of stakeholders positioned lower in the hierarchy. Furthermore, the stakeholders who participate are sometimes chosen based on who is noncritical for the daily work to continue without interruptions [1]. To limit such effects, techniques that permit all stakeholders to be involved in the elicitation, consistency, and validation without the necessity to be present in person at the same location and at the same time are required.

In our former work we developed a model-based approach [9]. After initial elicitation interviews, an overall scenario model is set up and can be validated by stakeholders in an interactive simulation (*play-out*, cf. [12]). Stakeholders can complement each other's activities through refinement or playing in additional activities, which result in consistent model updates. Still, the initial elicitation of new scenarios remained a problem. Since the simulator cannot know how to react if no suitable response was observed before, stakeholders run into dead ends during their elicitation sessions (referred to as *stalemates*). Arrange a meeting of all involved stakeholders to elicit the new scenarios in one simulation session is a complex, time-consuming solution. If this is not feasible, the only alternative is to perform multiple simulation sessions with individual stakeholders. To play-in her parts of one scenario, a stakeholder is enforced to participate in multiple sessions, waiting in-between for other stakeholders to play-in their continuations for the scenario. Especially for stakeholders playing coordinating roles, this can lead to numerous sessions.

In this paper, we present an extension of the simulation approach, that overcomes this challenge and can reduce the number of necessary sessions. Therefore, we use the simulator's property to not capture scenarios in an explicit process view, but in form of reachable states and possible transitions between them. The idea is to empower stakeholders to explicitly express the responses they usually expect when interacting with other stakeholders in form of partial states. Based on a partial state, a stakeholder can continue playing-in her parts of the scenario, without requiring additional elicitation sessions. The extension of the simulator is able to recognize the fulfillment of such expectations, thus, iden-

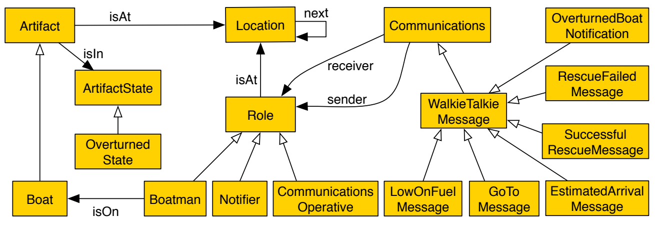


Figure 1. (Abridged) ontology $\mathcal{D}_W$ elicited from a lifeguard service

tifying a suitable continuation of incomplete interactions. Consequently, the results of multiple simulation sessions can be combined automatically.

The paper is structured as follows: At first, we present our model-based approach for the validation, additional elicitation and model update of scenario models and discuss its limits in Section II. Based on practical examples elicited from a lifeguard service, we discuss our approach on how stakeholders can describe their *expected continuations* by simply answering three questions, thereby overcoming these limitations in Section III. Then, two different types of *triggers* are presented in Section IV. Section V explains how such triggers are fulfilled and how the overall scenarios can be synthesized automatically. The paper closes with a discussion of related work in Section VI and a conclusion.

II. Model-Based Requirements Validation

Requirements engineers can hope to solve the stakeholders' problems only if they capture the concepts of the stakeholders' domain correctly [4]. This can be achieved by a domain ontology $\mathcal{D}$, which is gathered by the requirements engineers similar to a glossary of commonly agreed upon definitions of concepts. By collecting all concepts of the domain under investigation, the requirements engineers obtain a model suitable to describe scenarios in that domain. Similar to Artifact Models [3], $\mathcal{D}$ also captures how these concepts relate to each other. As outlined in [7], a domain ontology $\mathcal{D}$ contains the roles involved and identified from their scenarios, the artifacts that they use, and the specific information that they share. For our example, Figure 1 illustrates the abridged version $\mathcal{D}_W$ of the domain ontology elicited from a lifeguard service. In this example, the communication between a bystander notifying the lifeguards (referred to as *Notifier*) about an emergency, the corresponding communications operative (*ComOp*), and a boatman are elicited.

Kühne [14] argues, that metamodels such as $\mathcal{D}$ specify a language that can be used to describe instance situations. Thus, $\mathcal{D}_W$ provides a language for describing states as they can be observed during the scenarios of the lifeguards.

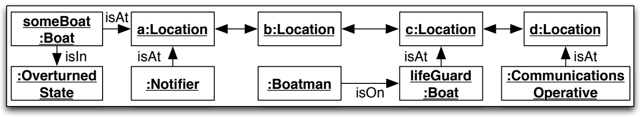
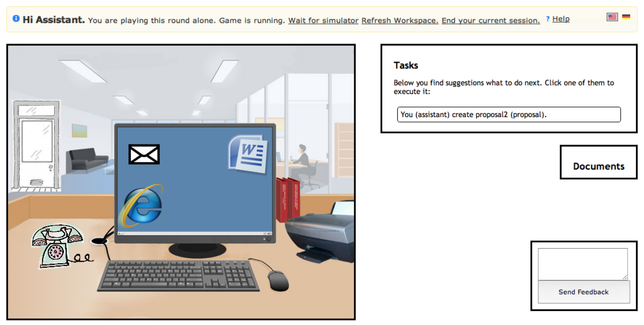


Figure 2. An initial state s_{init} of the lifeguard service scenarios


Hi Assistant. You are playing this round alone. Game is running. Wait for simulator Refresh Workspace. End your current session. ? Help

Tasks

Below you find suggestions what to do next. Click one of them to execute it:

You (assistant) create proposal2 (proposal).

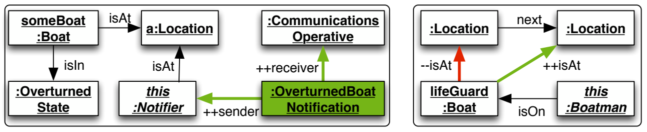
Figure 3. An example of a workplace visualization that allows stakeholders to interact with other, simulated roles to validate their interactions [6]

Similar to a Unified Modeling Language (UML) Class Diagram, $\mathcal{D}_W$ prescribes all possible states of the scenarios. States, in turn, can be specified using UML Object Diagrams. Such a state is illustrated in Figure 2. It is also the initial state s_{init} referred to in the sequence diagrams throughout the paper. In the following, all state labels in sequence diagrams refer to complete or partial states represented by such object diagrams.

Based on the idea of Harel and Marelly's *play-out* [12], our approach includes a simulator, which is able to play-out behavioral specifications to simulate the behavior observed beforehand from specific stakeholders. This simulator allows participating stakeholders to experience interactions with other stakeholders who are not participating in the same simulation session. Our simulator [9] decouples these interactions temporally by replaying them using the specifications to complement activities of participating stakeholders and, thus, allows stakeholders to validate each other's behavior without having to be in the same session or the same room.

Each stakeholder participating in our simulation has an individual interactive visualization (Figure 3, cf. [6]) of their distinct perspective on the current state of the simulation. Depending on the considered domain, different concepts are visible at different points in time. In case of the lifeguards, boats can only be seen if the stakeholder is at the same location (cf. Figure 1). The same holds for different artifacts or even other stakeholders. Thus, what has to be visualized to reproduce a distinct stakeholder's individual perspective is domain-specific. However, the requirements engineers can prototype what has to be shown to quickly get the details right. Then, the same rules usually apply for all stakeholders.

By using the visualization to interact with the simulation, participating stakeholders can change the state of the simulation according to what they would normally do in a corresponding situation, e.g., a boatman might move on to the next location or upturn an overturned boat. This visualization allows stakeholders to play through scenarios by interacting either directly with their colleagues or with roles that are simulated based on prior observations. In a related experiment, the state visualization has been evaluated successfully [8]. Thus, the play-out enables stakeholders to validate what has been observed and captured so far.



(a) Notifying the life guard service (b) Going to the next location

Figure 4. Story patterns of a $Notifier$ informing a CommunicationsOperative about an overturned boat he sees *(a)* and of a Boatman going from one location to the next *(b)*

After stakeholders have played through one of the valid scenarios, all states observed in-between their initial state s_{init} and the final state describe the activities of individual stakeholders and how they interacted. To formally capture what happened between succeeding states, our approach relies on graph transformations (specifically *story patterns*). Graph transformations, as described by Heckel [13], consist of a precondition and a postcondition. Similar to an object diagram, the precondition is a structural specification. If a state contains an exact match for the precondition of a story pattern, the matching elements are restructured through the addition or deletion of the corresponding elements and associations to match the story pattern's postcondition, which is also a structural specification. In story patterns, the elements that need to change are color-coded. Additions, i.e., instances and associations which have to be added, are marked using green and "++", while "– –" in red indicates removals. For example, Figure 4 illustrates a Notifier notifying a ComOp (*a*) and a Boatman changing locations (*b*).

By observing scenarios, i.e., sequences of states, the simulator can automatically derive story patterns based on the changes between two succeeding states. Each story pattern is then assigned to a specific role, i.e., to the one represented by the stakeholder, who was observed executing the behavior that changed the state of the simulation. In each story pattern, the instance of the specific role it belongs to is named *this* (cf. Figure 4).

As mentioned before, each stakeholder participating in a simulation has a unique perspective on a state s_i during a simulation session. After each activity of another role, the stakeholder might be affected by the result. Still, only some of these activities and their results are even visible to the role that this stakeholder represents. Consequently, every time a stakeholder participating in the simulation is affected by a change of the current state of the simulation, e.g., when a boat arrives at his location or an artifact is brought to his attention, this change has to be reflected in the stakeholder's visualization. This visualization illustrates the current state s_i of the simulation reduced to what a stakeholder's role is able to perceive. We refer to the reduction of a state s_i to what is visible for a role $Role_T$ as *projection*. Thus, a partial state $s_i|_{Role_T}$ can be derived from a state s_i using the visibility information, which apply for a domain while $s_i|_{Role_T} \subseteq s_i$ has to hold. Per default, $s_i|_{Role_T}$ contains an instance of $Role_T$ itself as well as all artifacts and information this role has access to in s_i.

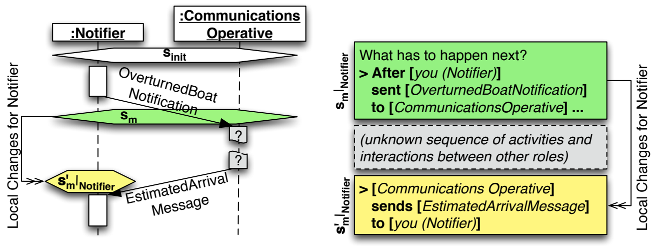

:Notifier
:Communications Operative
s_{init}
OverturnedBoat Notification
s_m
?
?
$s'_m|_{Notifier}$
EstimatedArrival Message
Local Changes for Notifier
$s_m|_{Notifier}$
What has to happen next?
> After [you (Notifier)] sent [OverturnedBoatNotification] to [CommunicationsOperative] ...
(unknown sequence of activities and interactions between other roles)
$s'_m|_{Notifier}$
> [Communications Operative] sends [EstimatedArrivalMessage] to [you (Notifier)]
Local Changes for Notifier

Figure 5. While the $Notifier$ knows how she informs the $ComOp$ (*green*), she does not know how the $ComOp$ continues this scenario (*gray*); still, the $Notifier$ can validate his expectations, i.e., how he is affected (*yellow*), in a natural language representation (*right*)

To allow stakeholders to comment on story patterns or describe what they can perceive or expect directly, the underlying object diagrams offer a *Natural Language* representation that is easier to understand for stakeholders as illustrated in Figure 5.

Based on stakeholder observations of *what* they do and *how* they do it, the story patterns derived from these observations can be used to replay and simulate the behavior observed from the individual stakeholders. This, in turn, enables the simulator to employ strategies to simulate other stakeholders and, thus, to steer the simulation into conflicting or unresolved states. Consequently, stakeholders can validate the behavior of other stakeholders by either agreeing to it or pointing out errors. Through the simulation of other roles, it becomes unnecessary to get all interacting stakeholders together in one room at the same time. Attending the simulation does not even require the attendees to be at the same location, since the visualization is web-based and, thus, allows for remote sessions [6]. Since the stakeholders can play-in incomplete scenarios, which can then be used to simulate them during simulation sessions with other stakeholders, the simulator also decouples the stakeholders temporally. By providing a model-based validation for behavioral models describing collaborative scenarios, our simulator tries to solve the problem of elicitation and validation for complex systems with multiple collaborating stakeholders.

If a stakeholder is observed starting an alternative scenario the simulator cannot respond appropriately, i.e., cannot offer any reasonable continuations. Since no behavior is available that completes this unknown situation, the requirements engineers have to talk to other stakeholders first to get to know a reasonable continuation for this scenario. Still, the remainder of the scenario might be unknown as well. Consequently, in the worst case, the requirements engineers have to go back and forth between different stakeholders to complete this scenario. In the worst case, to elicit a simple scenario between two stakeholders, each of them might have to be interviewed once for each interaction they have.

III. Approach: Changes in Partial States

As illustrated in Figure 6, the requirements engineers have to deal with individual perspectives as well as handovers. It

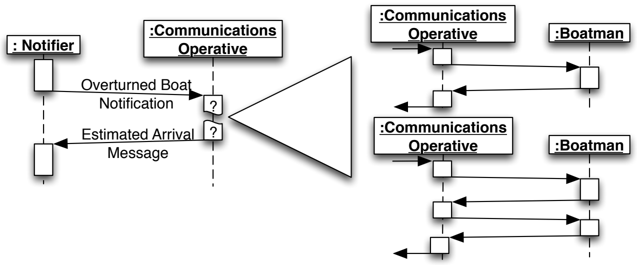

: Notifier
:Communications Operative
Overturned Boat Notification
?
Estimated Arrival Message
?
:Communications Operative
:Boatman
:Communications Operative
:Boatman

Figure 6. The scenario on the left (a *Notifier* calling in an emergency) is incomplete and the Notifier cannot know, which of the possible continuations (*right*) will occur

TABLE I
BY ANSWERING THESE QUESTIONS, STAKEHOLDERS DESCRIBE, WHICH INTERACTIONS USUALLY OCCUR AND HOW THEY END

	Question for Stakeholder	*Answer of Notifier*	*Named*
Q_1	Who did you interact with?	CommunicationsOperative	$Role_{Req}$
Q_2	Who, if not [$Role_{Req}$], do you expect to get an answer from?	CommunicationsOperative *(default: $Role_{Req}$)*	$Role_{Resp}$
Q_3	How are you affected by the outcome? What do you expect [to get]?	"An Estimated Arrival Message is sent to me"	$s'_m\|_{Role_T}$

happens quite often, that a stakeholder $Role_T$ telling his story cannot continue after he hands over a critical artifact, requests information or starts any other form of interaction with another stakeholder $Role_{Req}$. Since $Role_T$ does not know what Desai et al. [5] refer to as local and usually *private policies*, which dictate how $Role_{Req}$ acts or reacts in a specific situation, we can never be completely sure what happens. We refer to such a potential dead end during the elicitation as a *stalemate* – without information on how another role continues the scenario, requirements engineers and the stakeholder can only guess what happens next.

Generally, a stalemate can only be overcome if the requirements engineers gather the other side of the story. Similar to a black box, we can only assume how $Role_{Req}$ continues after being triggered. Still, while $Role_{Req}$'s actions are not yet known, $Role_T$ can describe most of the possible outcomes, i.e., how he is affected by the result, based on experience. Similar to a jigsaw puzzle, many pieces of information exist, however, only few of them are required to complete individual scenarios. Thus, it is essential that individual stakeholders do not give up at the first stalemate, but are able to continue to describe their expectation(s) as well as their follow-up actions. Even if a stalemate occurs during an elicitation of a scenario, a stakeholder can still answer the questions in Table I.

Q_1 provides the requirements engineers with the information of who to talk to next to complete the scenario. Only a stakeholder identified as role $Role_{Req}$, i.e., someone who usually receives $Role_T$'s request, knows how to continue. For the Notifier, this would be the CommunicationsOperative (ComOp) who he notifies about an emergency (Figure 6). After this operative passed on the information, the Notifier expects to hear from her again – consequently, the Q_2 would be answered the same. Still, in other cases, the person being interacted with is not the one who responds. To be able to distinguish between different responses from different stakeholders, Q_2 provides information on who else might provide a response $Role_T$ expects.

The simulation has a specific state s_m, in which the stalemate occurred. In this state, an interaction has been started that results in a change for $Role_T$ – although he does not know how anyone else might be affected as a side effect, the stakeholder can still describe what changes for him (Q_3). Since stakeholders can only describe changes that affect them directly and that are visible for them, the expectation can only be a partial state based on the perspective of an individual stakeholder. In the example provided in Table I, the Notifier expects to receive an estimation on when someone will arrive. Thus, using the vocabulary already established as part of the domain model $\mathcal{D}_W$, this expectation can be described explicitly.

IV. EXPECTATION TRIGGERS & FOLLOW-UP ACTIONS

We define an *expectation* as the partial state a role expects to perceive between a pair of states s_m and s'_m. Since a single stakeholder cannot know how the overall state of all stakeholders changed in-between, he can only specify his perspectives of the respective states. Thus, in case of the Notifier, he expects $s_m|_{Notifier}$ and $s'_m|_{Notifier}$ to be identical, except that he has to receive an EstimatedArrivalMessage from a CommunicationsOperative. Whether a boat already departed to his location or whether another emergency happened somewhere else is unknown to the Notifier, as long as none of those things are visible to him. The expected follow-up state can be represented and described in different ways to be suitable for stakeholders. While Figure 7 (*right*) illustrates it as a partial state in an object diagram, Figure 5 (*right*) presents a natural language representation that can easily be understood and modified: *[CommunicationsOperative] sends [EstimatedArrivalMessage] to [you (Notifier)].*

In Section III, a stalemate s_m occurred and the participating stakeholder representing a Notifier was asked to verbalize his expectations on how another stakeholder he interacted with might respond or, more generally, how his context might change. From his answers to the questions in Table I, a trigger for the other stakeholder can be generated. A *trigger* is a tuple (s_{sm}, $Role_T$, $Role_{Req}$, $Role_{Resp}$, $s'_{sm}|_{Role_T}$). It contains the *stalemate* state s_{sm} as it occurred during the simulation. Further, the role of the participant who defined the trigger ($Role_T$) is included. Additionally, to resolve the trigger, it is essential to know, which role is expected to continue the scenario and which role directly interacts with $Role_T$ next ($Role_{Req}$ and $Role_{Resp}$, respectively). Finally, the partial state $s'_{sm}|_{Role_T}$ that $Role_T$ expects to observe afterwards is included as well. Based on the answers of Notifier in Section III, the resulting

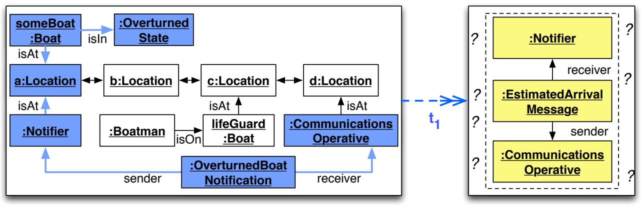
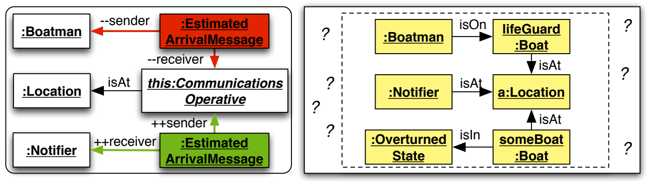


Figure 7. Based on s_m (*left*, Notifier's visibility is highlighted in *blue*), the Notifier can describe changes he expects as a partial state $s'_m|_{Notifier}$ (*right*) in $\mathcal{D}_W$'s vocabulary, leading to trigger t_1



(a) Story pattern x based on follow-up action f_1

(b) Eventually, the Notifier expects a lifeguard boat to arrive at his location

Figure 8. While the ComOp defines a follow-up action f_1, which leads to story pattern x (*a*), the Notifier expects to see a lifeguard boat (*b*).

trigger would be t_1 $=(s_m,\ Notifier,\ ComOp,\ ComOp,\ s'_m|_{Notifier})$, as illustrated in Figure 7.

A. Triggers and Alternatives

After the ComOp's *GoTo* message sending a Boatman to another location in response to the notification, ComOp expects that the Boatman responds with an *EstimatedArrivalTime* as prescribed by their protocol. Consequently, the ComOp defines a trigger $t_2 = (s_n, ComOp, Boatman, Boatman, s'_n|_{ComOp})$ expecting this message. While the ComOp might usually get an *EstimatedArrival* message, she might also be confronted with a *LowOnFuel* message, indicating that the boat needs to refuel first (Figure 9). Of course, the ComOp knows that all boats are fueled up at the beginning of each weekend. However, she does not know how much gas each boat may have left after several hours of service – an information only available to each respective Boatman. Thus, although a ComOp knows both possible outcomes, she cannot know, which one she will be confronted with, since she has no access to the information required for this decision.

From s_n, as for most stalemates, multiple continuations are possible from ComOp's point of view. Consequently, the *LowOnFuel* alternative can re-use s_n and is defined as trigger $t_3 = (s_n, ComOp, Boatman, Boatman, s''_n|_{ComOp})$ (cf. Figure 9).

B. Follow-Up Actions

A *follow-up action* f is an activity of a role, which is expected to apply if a specific precondition is fulfilled. It is characterized by a pair of states, the first being a precondition $(s_F|_{Role_T})$, which has to be fulfilled to execute the changes specified in the second $(s'_F|_{Role_T})$. As sketched in Table II, a follow-up action is the answer on how a role would continue after a stalemate has been overcome,

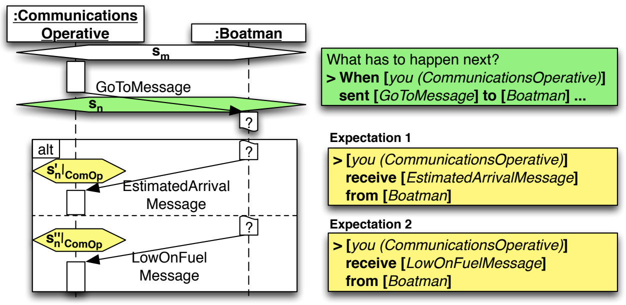

:Communications Operative
:Boatman
s_m
GoToMessage
s_n
?
alt
?
$s'_n|_{ComOp}$
EstimatedArrival Message
?
$s''_n|_{ComOp}$
LowOnFuel Message
What has to happen next?
> When [you (CommunicationsOperative)] sent [GoToMessage] to [Boatman] ...
Expectation 1
> [you (CommunicationsOperative)] receive [EstimatedArrivalMessage] from [Boatman]
Expectation 2
> [you (CommunicationsOperative)] receive [LowOnFuelMessage] from [Boatman]

Figure 9. Both expectations on how a Boatman may react to a *GoTo* message as experienced before and, thus, expected by a ComOp

TABLE II
BY ANSWERING THESE QUESTIONS, A STAKEHOLDER SPECIFIES A DISTINCT STATE AND HOW HE OR SHE FOLLOWS UP ON IT

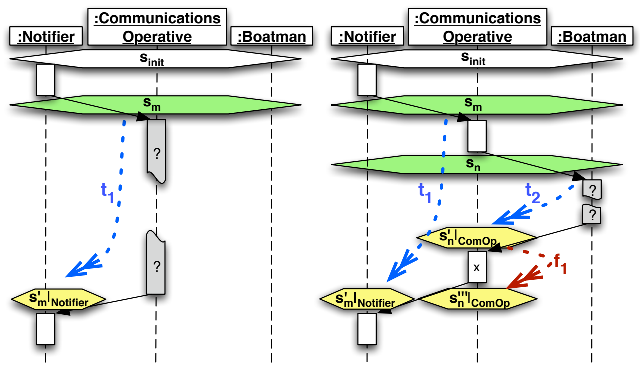
	Question for Stakeholder	*Answer of ComOp*	*Named*
Q_A	When do you become active?	"After the Boatman sent me an EstimatedArrivalMessage"	$s_F|_{Role_T}$
Q_B	How do you continue after $[s_F|_{Role_T}]$?	"I send the Notifier an EstimatedArrivalMessage"	$s'_F|_{Role_T}$

i.e., after the expectation has been fulfilled. After ComOp's expectations have been captured in t_2, the participating stakeholder can still describe how she as a ComOp would continue after her expectation $(s'_n|_{ComOp})$ is fulfilled. Consequently, $s'_n|_{ComOp}$ is presented to the stakeholder, either in an interactive visualization or in a textual representation. Based on this perspective, the stakeholder is able to specify the differences that her follow-up actions result in. In our example, the answer would be: *"I send the Notifier an EstimatedArrivalMessage"* $(s'''_n|_{ComOp}$, cf. Table II). Since the trigger has to be resolved for the stakeholder to follow up, the current state of the simulation needs to match the postcondition of the trigger. Thus, the postcondition of this trigger can be used as the precondition of the follow-up action. Combined with the follow-up state, this leads to the follow-up action $f_1 = (s'_n|_{ComOp}, s'''_n|_{ComOp})$. Similar to a pair of complete states, these two partial states can be used to derive a story pattern x (Figure 8*a*), which captures what the ComOp wants to achieve from her perspective.

In case of the ComOp, the precondition was already defined and was reused. If no trigger was defined beforehand, the stakeholder may still describe both situations, i.e., answer Q_A and Q_B, using natural language as sketched in Figure 5 (*right*). Of course, after having defined a follow-up action, additional follow-up actions can be defined.

V. RESOLVING TRIGGERS – SYNTHESIS

After the requirements engineer elicited an incomplete scenario ending in a stalemate s_{sm} and at least one trigger t_m $=(s_{sm},\ Role_T,\ Role_{Req},\ Role_{Resp},\ s'_{sm}|_{Role_T})$, the next stakeholder to talk to is already predetermined. To complete this scenario, a stakeholder of the corresponding role $Role_{Req}$ needs to participate to continue the interaction with $Role_T$. The requirements engineer starts a simula-



(a) Incomplete scenario after a Notifier session (b) Same scenario after a ComOp follows up

Figure 10. The expectation described in t_1 *(a)* might be fulfilled after t_2 has been resolved and follow-up action f_1 was executed *(b)*

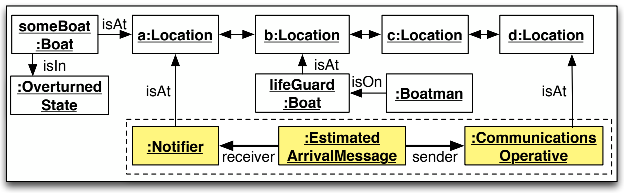


Figure 11. After ComOp is simulated using the story pattern x, the simulation is in state s_x with $s_x \supset s'_m|_{Notifier} = s'''_n|_{ComOp}$ (fulfilling the highlighted expectation in Figure 7)

tion session by loading the state s_{sm} for $Role_{Req}$. The visualization of s_{sm} corresponding to $Role_{Req}$'s perspective should visualize the last interaction with $Role_T$ in such a way, that the participating stakeholder is able to identify, which scenario the requirements engineers are currently interested in. By explicitly loading s_{sm}, inconsistencies between different triggers and their continuations can be avoided, since all follow-up activities are direct responses leading towards the expectation $s'_{sm}|_{Role_T}$.

In a simple case, $Role_{Req}$ is also the responding role $Role_{Resp}$, which can answer directly and fulfill $Role_T$'s expectation immediately with a suitable response. In case of t_1, however, ComOp cannot yet fulfill Notifier's expectation ($s'_m|_{Notifier}$). During the simulation for t_1, all activities of ComOp ($Role_{Req}$ and $Role_{Resp}$) and any other role involved (Boatman) within the interactive visualization of s_m lead to a state s_x, in which ComOp responded as expected.

An expectation is fulfilled, if its partial state $s'_{sm}|_{Role_T}$ expected by $Role_T$ can be found in s_x. Since $s'_{sm}|_{Role_T}$ and s_x are structural specifications conforming to $\mathcal{D}_W$, the simulator can try to match the expectation $s'_{sm}|_{Role_T}$ to a part of s_x. If such a match can be identified, the context of $Role_T$ in s_x is as expected: $s_x \supseteq s_x|_{Role_T} \supseteq s'_{sm}|_{Role_T}$. Consequently, if $Role_{Resp}$'s activities led to such a state, these observed activities are a suitable continuation for the scenario from the point of views of $Role_T$ and $Role_{Resp}$.

To resolve the trigger t_1, that Notifier created in a first session (1st row in Figure 12), its stalemate s_m (illustrated in Figure 7) is loaded and the participating ComOp stakeholder receives Notifier's notification of an overturned boat in the corresponding visualization of s_m. As always, the ComOp continues by starting an interaction with a Boatman by sending a *GoToMessage*, thereby bringing the simulation into the state s_n (2nd row). At this point, the ComOp cannot continue to play-in what needs to be done, since a stalemate s_n is reached in which she cannot deterministically predict how the Boatman will react. As outlined in Section IV-A, two different scenarios are possible. After the ComOp defined both corresponding expectations (t_2 and t_3), she also defines the follow-up action f_1 (cf. Section IV-B), which leads to the story pattern x (cf. Figure 8*a*) of how she continues after the expectation of t_2 has been fulfilled.

As specified in t_2 and t_3, the next role to talk to is Boatman, who needs to continue from s_n. After the stakeholder reviewed s_n in his visualization, he responds with an estimated arrival time (3rd row), thereby leading the simulation into state s_q in which Boatman fulfilled ComOp's expectation as defined in t_2. In s_q, ComOp's follow-up action f_1 is applicable since its precondition is identical to t_2's postcondition ($s'_n|_{ComOp}$). Consequently, by resolving t_2, the story pattern x is executed to simulate ComOp's follow-up action, leading the simulation into the follow-up state s_x (4th row). More importantly, the initial expectation of Notifier, i.e., the partial state $s'_m|_{Notifier}$, can be matched since the expected answer was provided through ComOp's follow-up action (cf. Figure 11). Thus, the requirements engineers end up with a completed scenario (5th row). Also, the story patterns necessary to reproduce and simulate it for other stakeholders can be derived from observations or follow-up actions. Now, the requirements engineers can continue by collecting alternatives, e.g., *what has to happen when a Boatman fulfills the expectation described in t_3?*

All triggers that are collected along the way are stored next to the story patterns, which are derived from observed scenarios. To resolve them, the simulator simply checks whether the expected outcome of an interaction ($s'_{sm}|_{Role_T}$) can be matched in the current state s_i of the simulation. Based on this algorithm, scenarios are completed step by step from stakeholder to stakeholder as illustrated for the notification example in Figure 12. The possibility remains, that no state $s_x \supseteq s'_m|_{Notifier}$ can be reached – even after multiple sessions of the role that is expected to reply. In this case, the requirements engineer has to be notified and two options are present: directly intervene and either talk to $Role_{Resp}$ to ask, e.g., *what needs to be true for the Boatman to not answer as expected* or talk to Notifier ($Role_T$) to check whether the expectation that was described is correct.

VI. Related Work

One of the main contributions of Harel and Marelly's Play-approach [11], [12], is the possibility not only replaying captured system behavior (play-out), but the possibility to capture additional system behavior (play-in) and, thus,

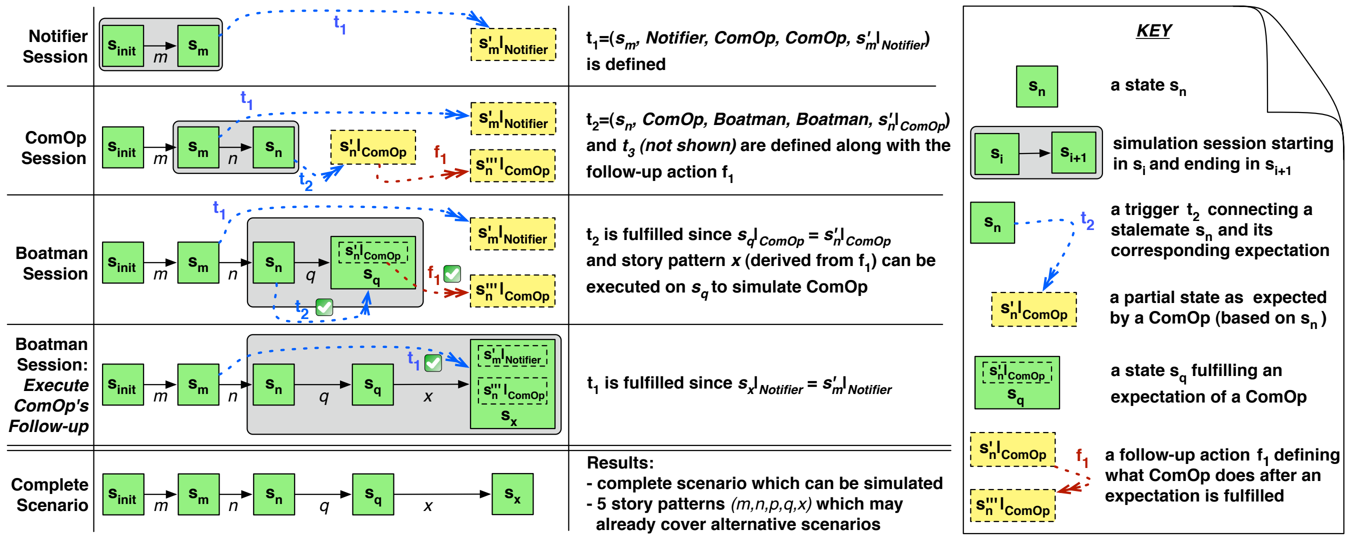

Notifier Session
ComOp Session
Boatman Session
Boatman Session: Execute ComOp's Follow-up
Complete Scenario
t1=(sm, Notifier, ComOp, ComOp, s'm|Notifier) is defined
t2=(sn, ComOp, Boatman, Boatman, s'n|ComOp) and t3 (not shown) are defined along with the follow-up action f1
t2 is fulfilled since sq|ComOp = s'n|ComOp and story pattern x (derived from f1) can be executed on sq to simulate ComOp
t1 is fulfilled since sx|Notifier = s'm|Notifier
Results:
- complete scenario which can be simulated
- 5 story patterns (m,n,p,q,x) which may already cover alternative scenarios
KEY
a state sn
simulation session starting in si and ending in si+1
a trigger t2 connecting a stalemate sn and its corresponding expectation
a partial state as expected by a ComOp (based on sn)
a state sq fulfilling an expectation of a ComOp
a follow-up action f1 defining what ComOp does after an expectation is fulfilled

Figure 12. After only three stakeholder sessions, the scenario has been completed with two triggers and one follow-up action

new scenarios while doing so. Their approach, however, is centered around user interfaces – for each input the user provides, she can play-in how the system should react. While this might suffice to capture the interaction between individual stakeholders and a software system, i.e., its user interface (UI), the elicitation of interactions between different stakeholders is more complex.

Still, the play-engine [12] can be used to enable stakeholders to perceive the complete state the system is in, reduced to what is presented in the UI. The similar visualization approach is used by Ponsard et al. for specific goals such as whether a door of a train is closed when its moving [21] or even to represent domain-specific UI elements and how they affect each other from the point of view of a specific stakeholder [16]. However, these approaches are limited to single stakeholders and their interaction with a software system only – collaborative processes with information asymmetry cannot be elicited or validated.

Scenario-based approaches have been researched quite broadly, most notably by Uchitel et al. [18][19][20]. Starting with sets of potentially incomplete, implied or negative scenarios, they are able to derive state machines for the involved components that are suitable for all of these scenarios, in case of Whittle et al. even hierarchical ones [23]. Still, these approaches are not suitable to elicit human interactions, which are limited by what stakeholders can perceive from their individual context. To obtain behavioral models of what the stakeholders do, it does not suffice to know, which messages arrived at a stakeholder in which sequence, but rather, which artifacts and information they can see or access. For instance, the Notifier's expectation, that a lifeguard boat arrives (Figure 8*b*) can only be fulfilled by the Boatman moving to the corresponding location (Figure 4*b*). Only by eliciting and validating them in a state-based manner, it becomes viable to visualize the current *state* a stakeholder is in based on what she has access to.

Similar to our approach, Ghezzi et al. [10] compare pairs of succeeding states to derive the behavior of Java classes as graph transformations. While their method of automatically eliciting behavior is identical, their approach is restricted to a single actor, i.e., an instance of a class, and cannot cope with information asymmetry. The same goes for van der Aalst's ProM approach [22], which is able to derive a process model of how people can achieve their goals collaboratively based on log files detailing different scenarios. However, ProM is only able to use the information that is available in these logs, provided a logging system is already in place and analog interactions cannot be captured.

VII. Conclusion and Future Work

In this paper, we presented an approach that reduces the effort necessary to elicit and validate new collaborative scenarios. Thereby, the number of sessions necessary to elicit a stakeholder's actions within a scenario can be reduced. In case of stalemates, stakeholders can express their expectations on interaction results in form of partial states. Based on these partial scenarios, stakeholders are then able to play-in continuing behavior decoupled from one another. Our simulator synthesizes the captured individual perspectives to obtain the complete scenarios, thereby overcoming the inherent fragmentation of different perspectives. We discussed how this technique extends our model-based validation approach to be applicable for model-based elicitation, too. The method was illustrated on a real life example of a lifeguard service.

Using the old approach, the requirements engineer would have to go back and forth between two or more stakeholders for each interaction. Thus, the total number of sessions required to elicit a complete scenario related to the number of stalemates the stakeholders ran into during the elicitation, since an additional session is necessary for each stalemate that occurred. Generally, the number of stalemates per role can vary strongly. Especially for the role ComOp in the lifeguard example, at least ten interactions need to be elicited for each scenario, which leads to the same number of sessions to resolve the implied stalemates. However, by being able to express her expectations and, thus, triggers intuitively, the ComOp is now able to defer the completion of all scenarios to other stakeholders later on. Consequently, instead of ten sessions to overcome ComOp's stalemates only, ComOp can express all her contributions to the collaborative scenarios in just one session. Decoupled from her, the other stakeholders can then complement the scenarios accordingly. By systematically completing these scenarios, the total number of elicitation sessions no longer depends on the number of interactions and stalemates, but on the number of roles and their ability to express their expectations. In this case, our approach can end up with only one elicitation session per role.

For future work, we want to evaluate whether stakeholders are able to describe all of their interactions in the proposed way and investigate, for which domains and scenarios this technique works best. Our approach currently focuses on completing one scenario at a time. We plan to investigate how the overall number of sessions can be further reduced by taking into account how stakeholders interact in multiple scenarios. For instance, the resolution of triggers from multiple scenarios might be handled in the same session.

REFERENCES

[1] A. Al-Rawas and S. Easterbrook. Communication Problems in Requirements Engineering: A Field Study. In *Proc. of the First Westminster Conference on Professional Awareness in Software Engineering*. Royal Society, London, 1996.

[2] I. Alexander and N. Maiden, editors. *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*. John Wiley, New York, 2004.

[3] B. Berenbach, D. Paulish, J. Kazmeier, and A. Rudorfer. *Software & Systems Requirements Engineering: In Practice*. McGraw-Hill, Inc., New York, NY, USA, 2009.

[4] A. Davis and K. Nori. Requirements, plato's cave, and perceptions of reality. *Computer Software and Applications Conference, Annual Int.*, 2:487–492, 2007.

[5] N. Desai, A. Mallya, A. Chopra, and M. Singh. Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering*, 31:1015–1027, 2005.

[6] G. Gabrysiak, H. Giese, and A. Seibel. Interactive Visualization for Elicitation and Validation of Requirements with Scenario-Based Prototyping. In *Proc. of the 4th International Workshop on Requirements Engineering Visualization*, pages 41–45, Los Alamitos, USA, 2009. IEEE Computer Society.

[7] G. Gabrysiak, H. Giese, and A. Seibel. Using Ontologies for Flexibly Specifying Multi-User Processes. In *Proc. of ICSE 2010 Workshop on Flexible Modeling Tools*, Cape Town, South Africa, 2010.

[8] G. Gabrysiak, H. Giese, and A. Seibel. Towards Next-Generation Design Thinking II: Virtual Multi-User Software Prototypes. In H. Plattner, C. Meinel, and L. Leifer, editors, *Design Thinking Research*, Understanding Innovation, pages 107–126. Springer, 2012.

[9] G. Gabrysiak, R. Hebig, and H. Giese. Simulation-assisted elicitation and validation of behavioral specifications for multiple stakeholders. In *Proc. of the 21st IEEE International Conference on Collaboration Technologies and Infrastructures*, Toulouse, France, June 2012.

[10] C. Ghezzi, A. Mocci, and M. Monga. Synthesizing intensional behavior models by graph transformation. In *Proc. of the IEEE International Conference on Software Engineering*, pages 430–440, Washington, USA, 2009. IEEE.

[11] D. Harel, H. Kugler, and A. Pnueli. Synthesis revisited: Generating statechart models from scenario-based requirements. In H.-J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, and G. Taentzer, editors, *Formal Methods in Software and Systems Modeling*, pages 309–324. Springer, 2005.

[12] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[13] R. Heckel. Graph transformation in a nutshell. *Electronic Notes in Theoretical Computer Science*, 148(1):187 – 198, 2006. Proc. of the School of SegraVis Research Training Network on Foundations of Visual Modelling Techniques.

[14] T. Kühne. Matters of (meta-) modeling. *Software and Systems Modeling*, 5:369–385, 2006.

[15] A. Luebbe and M. Weske. Tangible media in process modeling – a controlled experiment. In H. Mouratidis and C. Roland, editors, *23th Conference on Advanced Information Systems Engineering (CAiSE 2011)*, pages 283–298, 2011.

[16] C. Ponsard, N. Balych, P. Massonet, J. Vanderdonckt, and A. van Lamsweerde. Goal-Oriented Design of Domain Control Panels. In S. W. Gilroy and M. D. Harrison, editors, *DSV-IS*, volume 3941 of *LNCS*, pages 249–260. Springer, 2005.

[17] N. Seyff, N. Maiden, K. Karlsen, J. Lockerbie, P. Grünbacher, F. Graf, and C. Ncube. Exploring how to use scenarios to discover requirements. *Requirements Engineering*, 14(2):91–111, 2009.

[18] S. Uchitel, G. Brunet, and M. Chechik. Synthesis of Partial Behavior Models from Properties and Scenarios. *IEEE Transactions on Software Engineering*, 35(3):384–406, 2009.

[19] S. Uchitel, J. Kramer, and J. Magee. Detecting implied scenarios in message sequence chart specifications. *SIGSOFT Softw. Eng. Notes*, 26:74–82, September 2001.

[20] S. Uchitel, J. Kramer, and J. Magee. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Trans. Softw. Eng. Methodol.*, 13:37–85, 2004.

[21] H. T. Van, A. van Lamsweerde, P. Massonet, and C. Ponsard. Goal-oriented requirements animation. *Requirements Engineering, IEEE International Conference on*, 218–228, 2004.

[22] W.M.P. van der Aalst. Trends in business process analysis: From validation to process mining. In *International Conference on Enterprise Information Systems*, Funchal, Portugal, 2007.

[23] J. Whittle and P. Jayaraman. Synthesizing hierarchical state machines from expressive scenario descriptions. *ACM Trans. Softw. Eng. Methodol.*, 19:8:1–8:45, 2010.

A Formal High-level Modeling Approach to Develop Reliable Components in Vision-based Robotics

Andrea Luzzana, Mattia Rossetti, and Patrizia Scandurra
Università degli Studi di Bergamo, DIIMM, 24044 Dalmine (BG), Italy
andrea.luzzana@unibg.it mattia.rossetti@unibg.it patrizia.scandurra@unibg.it

***Abstract*—This paper proposes the use of the *control-state Abstract State Machines* for a rigorous foundation in high-level modeling and validating component-based applications in *Vision-Based Robotics*. In particular, an extension of the classical flowchart notation for control-state ASMs is proposed to support modularization and reuse in a direct way. The resulting ASM models are to be intended as "ground models" that can be used as basis or patterns to practically model and formally validate the behavior of typical robotic control tasks, and to link (via successive refinements) these high-level models of components to their implementation code by making their functional correctness mathematically controllable. The proposed flowchart extension and the availability of reusable and validated ground models allow a better system design and speed up the development of the system.**

***Keywords*-components; abstract state machines; ground modeling; robotic control tasks.**

I. Introduction

Vision-based robotics is a challenging research field [1]. One of the open and commonly stated problems in the field is the need for exchange of experiences, best practices, and high-level models of robust, reliable and flexible robot control applications with *visual servoing* functions.

Recently, we investigated [2] the use of the *Abstract State Machine* (ASM) formal method [3] for a systematic study and a rigorous foundation of modeling and validating *Visual Servoing* (VS) applications. The ASM method is a discipline for reliable system development, which allows to bridge the gap between informal requirements and executable code. The ASM formalism supports concurrency, heterogeneous state and modularity (compositional design and verification techniques). These features are essential to tailor *ground models* of control tasks definitions and associated synchronization/communication patterns of VS applications in rigorous, compositional and abstract terms. *Ground models* are blueprints of the to-be-implemented piece of "real world" that "ground the design in the reality" [4]. In particular, we exploit the notion of *control state ASMs* (a class of ASMs [3]) as a natural extension of Finite State Machines.

In this paper, we present an extension – called *pattern-oriented control-state ASMs* – of the classical flowchart notation for control-state ASMs to support *modularization* and *reuse* in a direct way. The proposed notation is useful to denote explicitly modeling elements to be further refined, to allow the definition/instantiation of recurring design solutions or patterns, to perform initial validation of separate high-level models and to improve model traceability between the flowchart diagrams and their concrete (textual) ASM specifications during the ground modeling and development process. In this context, the term "pattern" is to be intended to have its classical meaning, i.e., as a schema of a recurring solution, rather than the meaning of "design pattern" as in the book of the GoF (Gang of Four).

We here repeat our previous experience in the embedded system-on-a-chip domain [5][6] to shift the focus from implementation to design through high-level modeling. Our approach combines the expressive power and accuracy of control state ASMs with the intuition provided by visual flowchart descriptions to capture the behavioral view of task-level control of VS applications. As starting point, we manually extracted from the structure of existing (basically C/C++) code architectural descriptions (in terms of UML component diagrams) of high-level models of component control tasks and also recurring synchronization/communication patterns between tasks that could be used for the ground modeling and analysis in ASM. We then defined these ASM abstract models using the pattern-oriented control-state ASM notation, transformed such models into executable ASM models using the notation ASMETA/AsmetaL [7][8], and then validated them through basic formal analysis techniques (simulation and scenario-based simulation) [9]. The resulting and validated ASM models are used as basis or patterns for high-level modeling and validating typical control tasks of VS applications in a formal way, thus leading from the abstract models to executable (C/C++) code by making their functional correctness mathematically controllable.

This paper focuses on presenting the extended control-state ASM notation and it is organized as follows. Section II introduces the reader to the field of vision guided robotics by illustrating the synchronization/communication issues at control task-level covered in the paper. Section III provides background notions on the ASMs. Section IV presents the pattern-oriented control-state ASMs. Section V presents the ASM ground modeling of some control tasks synchronization/communication patterns and, as major case study, their instantiation to model a VS application. Section VI provides some details on implementation issues. Section VII presents some related works. Finally, Section VIII concludes the paper and sketches future directions.

II. Application Domain

This section discusses about modeling issues in visual assisted robotic control architectures, with emphasis on synchronization/communication of control tasks.

Robot control applications with VS features: The main task of a robot automatic control system is to drive robot actuators (typically electrical motors) in order to follow a trajectory passed to the controller by higher level applications of the control architecture, such as motion planners, production cell controllers, visual systems, etc.

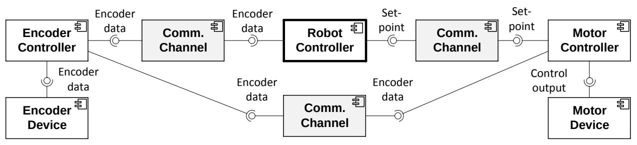


Figure 1. A Robot controller architecture.

The UML component diagram in Figure 1 shows a typical architecture of a robot control application. A robot controller sends set-point values to a motor controller. The encoder controller acquires data from encoders in order to send them to both the robot controller and the motor controller. This last task has to read also the data (motors set-point) produced by the robot controller in order to drive motors. The motor and the encoder controllers communicate directly with their respective devices through physical interconnections. This control scheme presents non-trivial issues related to the timing and the synchronization of the involved controllers since all the tasks have to exchange data periodically.

The need for strict periodic tasks makes it necessary to encapsulate the communication functionality in a separate component, the *communication channel*, thus decoupling the producer (or sender) from the consumer (or receiver). A communication channel can be implemented on top of several protocols. For data exchange among tasks, real-time programming guides (e.g., [10]) typically suggest the use of a *shared memory* (e.g., a FIFO, a ring buffer, a stack, etc.) in order to ensure lower memory usage and better performance. In this work, we will focus on the use of the *Swinging Buffer* [10] (described below) as shared memory.

A robot builds a representation of the surrounding environment by acquiring data from several sensors. *Visual servoing* [1] is a technique that uses feedback information extracted from a vision sensor to control the motion of a robot. A closed-loop control of a vision-based robotic system usually consists of two intertwined processes: tracking and control. This architecture (see Figure 2) performs the control of the robot in two separate stages: first, the vision system provides input to the robot controller by acquiring and elaborating images; then, the robot controller uses joint feedback to internally stabilize the robot. Optionally, set-point computation can require the acquisition of robot data via another communication channel.

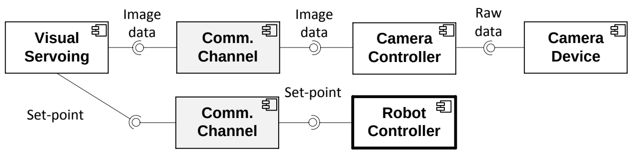
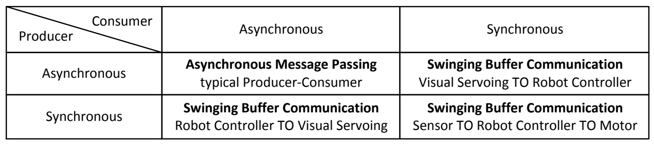


Figure 2. A Visual Servoing Robot architecture.

Using parallelism can improve the performance of the system, but it introduces new non-trivial issues due to VS functions. First, the visual servoing task can be exploited only at the presence of an image, so it is the only non-periodic (asynchronous) task in the control scheme. Second, the time required for elaboration is not constant because computer vision algorithms efficiency is strongly affected by aspects like quality of images, complexity of the environment and so on. Finally, the visual servoing task is both a producer and a consumer because it requires images from camera for its elaboration and then needs to transmit the set-point information to the robot.

Synchronization and Communication issues: Basically, control tasks can be classified in *asynchronous* and *synchronous*. Asynchronous tasks are data-driven, because their elaboration starts when there are data to be consumed and ends with data transfer. Synchronous tasks are, instead, time-driven, as they are periodic and have deadlines to respect. Figure 3 summarizes the possible communication types that we cover in our work, as collected from visual servoing and robot control applications (such as pick and place, object tracking and micro-assembly). The analyzed solutions involve the use of swinging buffers for the communication between tasks operating at different frequencies. A swinging buffer can be viewed as an advanced circular buffer using two or more shared memory arrays instead of the single array adopted by a circular buffer. While the producer task fills up one of the buffers, the consumer empties another one. When a task reaches the end of the buffer that it is using, it starts operating from the beginning of another unused array. Since tasks works on different memory locations, no lock for the mutual exclusion is needed to access to the data on the buffer, but only for updating the read/write indexes.



Consumer / Producer	Asynchronous	Synchronous
Asynchronous	**Asynchronous Message Passing** typical Producer-Consumer	**Swinging Buffer Communication** Visual Servoing TO Robot Controller
Synchronous	**Swinging Buffer Communication** Robot Controller TO Visual Servoing	**Swinging Buffer Communication** Sensor TO Robot Controller TO Motor

Figure 3. Task communication types.

III. Background on ASMs

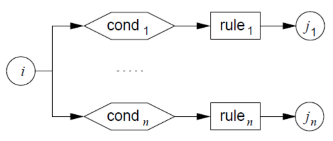
ASMs are an extension of Finite State Machines (FSMs) [3] where unstructured control states are replaced by states of arbitrary complex data. The *states* of an ASM are multi-sorted first-order structures, i.e., domains of objects with functions and predicates (boolean functions) defined on them. The *transition relation* is specified by named rules describing how functions change from one state to the next. A transition rule has the basic form of *guarded update* "**if** *Condition* **then** *Updates*" where *Updates* is a set of function updates of the form $f(t_1, \ldots, t_n) := t$, which are simultaneously executed when *Condition* is true. f is an arbitrary n-ary function and $t_1, \ldots, t_n, t$ are first-order terms. Essentially, to fire this rule in a state S_i, $i \geq 0$, evaluate all terms $t_1, \ldots, t_n, t$ at S_i and update f to t on parameters $t_1, \ldots, t_n$. This produces another state S_{i+1}, which differs from S_i only in the new interpretation of f. A set of *rule constructors* allows to express simultaneous parallel actions (`par`), sequential actions (`seq`), iterations (`iterate`, `while`, `recwhile`), and submachine invocations returning values. Non-determinism (existential quantification `choose`) and unrestricted synchronous parallelism (universal quantification `forall`) are also supported.

Control-state ASMs are a class of ASMs used to model some overall status or mode, guiding the execution of guarded synchronous parallel updates of the underlying state. Figure 4 (slightly adapted from [3]) shows on the left the conventional flowchart notation that include three basic symbols. It also shows on the right the corresponding ASM rule scheme (in textual notation) for control state ASMs. Circles denote phases (also called control states or internal states), hexagons (optional) denote test predicates (also called conditions or guards), and rectangles denote update actions (i.e., application of ASM rules, including rule invocations of submachines) and are also optional. The finitely many control states *ctl_state* $\in \{1 \ldots m\}$ are used to describe different system modes.

IV. Pattern-oriented Control State ASMs

We extended the flowchart notation of control state ASMs to better enhance some aspects and to capture new ones. Precisely, we defined a pattern-oriented extension of the classical flowchart notation for control state ASMs to denote explicitly parts of the models that have to be further refined, to denote the definition/instantiation of a pattern, and to improve model traceability by allocation links between the flowchart diagrams and their concrete implementations (i.e., the ASM specifications) in the textual language AsmetaL.

Revised symbols. We adopt (see Figure 5) dashed lines for guards and actions to indicate that these elements require further refinement, i.e., the test predicate for the guard and the rule for the action. For example, an action can be refined by introducing other action-state-condition blocks to model the intended activity. The optional text {`text`}



```
if $ctl\_state = i$ then
  if $cond_1$ then
    $rule_1$
    $ctl\_state := j_1$
      ...
  if $cond_n$ then
    $rule_n$
    $ctl\_state := j_n$
```

Figure 4. Control state ASMs.

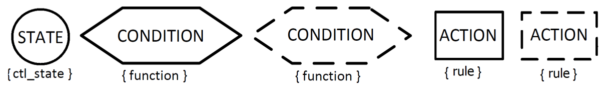


Figure 5. Extended control state ASM notation.

near a graphical symbol is used to link diagrams to their concrete ASM specification `spec` (in our case, AsmetaL specification). Specifically: for a state symbol it denotes the function name in the `spec` representing the underlying control state variable; for a guard symbol it denotes the test predicate name in the `spec`; for an action symbol it denotes the rule name in the `spec` implementing it.

Pattern machines. For modularization and reuse purposes, we introduce two new symbols (see Figure 6) denoting the concepts of *pattern* (pattern definition) and of *pattern instantiation* blocks, respectively. A pattern block is to be intended as a placeholder for a recurrent and complex action block, also referred to us as *pattern machine*. Figure 6 shows the shape of such a pattern machine that includes an entering arrow followed by (at least) an action-state-condition block closed with a floating exit arrow. A pattern machine consisting of an action block only is also admitted. The circles represent the internal states of the pattern machine and it usually requires a fresh control state variable `ctl_state`. It is a piece of reusable ASM model that can be validated and verified separately and then re-used in more complex ASM specifications. The entering arrow denotes always the evaluation of the guard `isUndef(_ctl_state)` that is the mandatory condition that enables the execution of the pattern machine. The floating exit arrow denotes the exit point and implies always the mandatory update `ctl_state := undef`.

A pattern machine is then specified in terms of a named rule and this rule will occur as subrule of the containing machine. Moreover, we assume that (otherwise specified) the pattern machine is composed with the other occurring rules (action blocks), if any, of the containing machine according

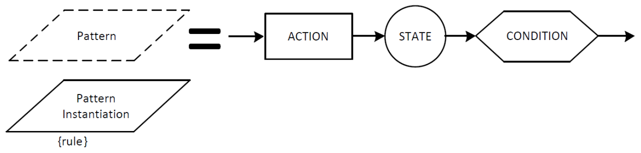


Figure 6. Pattern notation.

to the synchronous parallelism semantics of the **par**-rule. The rule name of the pattern machine can be specified (see Figure 6) near the pattern instantiation symbol.

V. Ground Modeling Robot Control Tasks

We use pattern-oriented control state ASMs for specifying ground models of robot control tasks.

A. Producer/Consumer models

Figure 7 shows the control state ASM of a simple data producer module. It contains a pattern machine for the writing operation on the swinging buffer, presented in Section V-B. The first action of a producer task is to wait for a trigger in order to start to acquire and elaborate data; lastly, data elaborated have to be written in a shared memory (in our case, a swinging buffer). This machine has blocks to be refined, because their specific behavior depends on the tasks nature: tasks can be asynchronous or synchronous, and in turn master or slave. Listing 1 reports, using the AsmetaL notation, a possible definition of the test predicates for the events *Trigger* and *Elaboration Time Elapsed* (their names are also denoted in the diagrams). The event *Trigger* may require further refinement. The listing reports a possible definition of this test predicate (in a particular phase of the specification development process) that captures two cases: it is the boolean OR of "waiting for a period of time" for a synchronous periodic task, and of an "always-true" condition for an asynchronous task. Both the test predicates take as parameter the scheduler `c`, since in our applications each component is associated to and managed by a scheduler. The current task scheduled on `c` is represented by the function `scheduler_MainThread(c)`.

The data consumer model is straightforward.

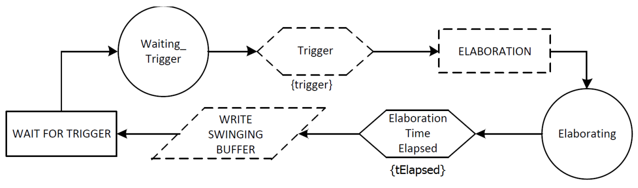


Figure 7. Data Producer.

Listing 1. Producer/consumer test predicates.

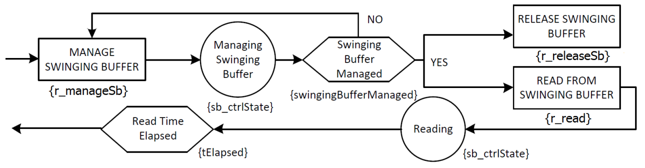
```
function trigger($c in Scheduler) = (task_kind(
    scheduler_mainThread($c)) = SYNCHRONOUS and
    task_elapsedTimeOfPeriod(scheduler_mainThread($c)) >
    time(scheduler_currentPhase($c)))
      or task_kind(scheduler_mainThread($c)) =
          ASYNCHRONOUS

function tElapsed($c in Scheduler) =
      scheduler_currentScheduleTime($c) > time(
          scheduler_currentPhase($c))
```

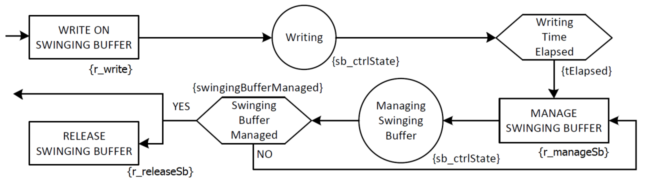
B. Swinging Buffer reading/writing models

The swinging buffer introduces the concept of multiple shared memory areas. Hence, the producer and consumer tasks do not share a memory, but only the read/write pointers to different memory areas. To avoid overwriting problems, only one task, the *master task* (either the producer or the consumer), can manage the indexes update, while the other tasks, the *slaves*, behave just classical producers/consumers. However, in real-time tasks, also the nature of the communicating tasks and their frequency must be considered. In the case of two asynchronous communicating tasks, a producer and a consumer, the master task can be either the producer or the consumer. In the case of two synchronous tasks, instead, a common practice is to set the slower task as the master one. In fact, as there is not a single shared memory area, the master task has to get the lock only when it has to update its data pointers. Moreover, during its elaboration, it does not need to get the lock because, as the slave tasks do not manage indexes, there is no possibility of an inconsistent update. Finally, in the case of a synchronous-to-asynchronous communication, the asynchronous task is the master slave because it does not carry out data elaboration periodically, but only when new data are available.

This behavioral variability is captured by the ASM pattern machines for reading/writing from/to swinging buffer shown in the Figures 8, 9 and 10. In particular, as an example, Listing 2 for an *Asynchronous-Master-Writing* operation (see Figure 8-b) reports the corresponding ASM specification using the AsmetaL notation. The AsmetaL implementation of the other pattern machines can be found in [11].



(a) Reading.



(b) Writing.

Figure 8. Swinging Buffer - Asynchr. Master Read/Write.

The *Asynchronous-Master-Writing operation* (see Figure 8-b) implies first to write data on the shared memory directly (state *Writing*), without acquiring the lock for the critical section. When the writing operation is terminated (after a certain period of time has passed), the swinging buffer pointers have to be updated in order to signal to the consumer that new data are ready. In the next state *Managing*

Swinging Buffer, the asynchronous task tries (by the iterative flowchart part) to get the lock to the critical section for updating the pointers by executing the action *MANAGE SWINGING BUFFER*. After the pointers are updated, the control exits by releasing the lock (*RELEASE SWINGING BUFFER*). As an example, Listing 3 reports the definition of the rule *MANAGE SWINGING BUFFER* using the AsmetaL notation; note again that this rule captures the behavior of both master and slave tasks by distinguishing clearly these roles, since it is reused also as action in the other pattern machines. The AsmetaL implementation for the other actions and test predicates can be found in [11].

The *Asynchronous-Master-Reading pattern* (see Figure 8-a) is similar to the writing one, despite of action order. In fact, in this case the *Managing Swinging Buffer* action has to be performed before reading.

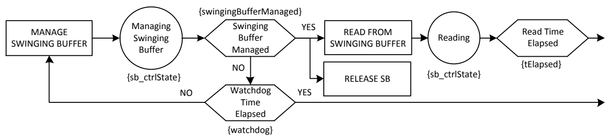
Listing 2. Asynchronous-Master-Writing pattern machine.

```
rule r_async_master_writing ($c in Scheduler) =
par
      //Entering into the pattern machine
 if (isUndef (sb_ctrlState(scheduler_currentPhase($c))))
 then par
   r_write [$c, scheduler_currentPhase($c)]
   sb_ctrlState(scheduler_currentPhase($c)):= WRITING
 endpar
 if (sb_ctrlState(scheduler_currentPhase($c)) = WRITING)
 then if (tElapsed ( $c))
         then par
         r_manageSb [$c, scheduler_currentPhase($c)]
         sb_ctrlState(scheduler_currentPhase($c)):=
              MANAGING_SB
             endpar
 if (sb_ctrlState(scheduler_currentPhase($c)) =
      MANAGING_SB) then
       if (swingingBufferManaged($c)) then //Index updated
             par
              r_releaseSb[$c]
              // Exit from pattern machine
              sb_ctrlState(scheduler_currentPhase($c)):=
                   undef
             endpar
       else //Index not updated
             par //Try again to acquire the lock
              r_manageSb [$c, scheduler_currentPhase($c)]
              sb_ctrlState(scheduler_currentPhase($c)):=
                   MANAGING_SB
             endpar
endpar
```

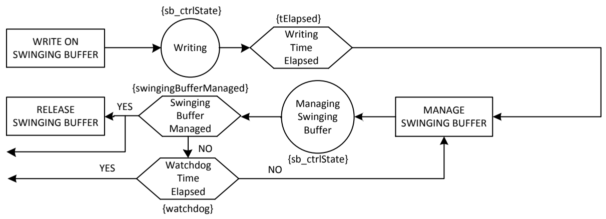
For the reading/writing operations of synchronous master tasks (see Figure 9), the transition for managing the swinging buffer indexes is slightly modified: if the lock has not been acquired, a further guard *watchdog Time elapsed* is evaluated: if false (there is no time), the task skips the index update phase and exits. Finally, for synchronous slave tasks (see Figure 10 for the reading operation) if the lock cannot be acquired, the task simply skips the reading/writing action. The writing operation is fully similar.

Listing 3. The MANAGE SWINGING BUFFER rule.

```
rule r_manageSb($c in Scheduler, $phase in SchedulePhase)=
par
       if (syncPriority($phase) = MASTER)
       then seq
             r_SwingingBuffer_getLock [sharedMemory($phase)
                   , scheduler_mainThread($c)]
             r_SwingingBuffer_updateIndexes [sharedMemory(
                  $phase) , scheduler_mainThread($c)]
       endseq
       if (syncPriority($phase) = SLAVE) then
             r_SwingingBuffer_getLock [sharedMemory($phase)
                   , scheduler_mainThread($c)]
endpar
```



(a) Reading.



(b) Writing.

Figure 9. Swinging Buffer - Synchr. Master Read/Write.

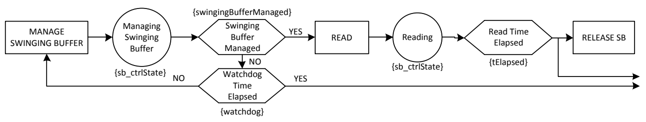


Figure 10. Swinging Buffer - Synchronous Slave Read.

C. Major case study: A VS application model

Let us consider the VS application case study described in Section II. Figure 11 shows the ASM control state for the visual servoing component shown in Figure 2. It elaborates images in order to produce the commands to send to the robot controller component. So, it is an asynchronous task because it can elaborate images only when they are available. Note that it plays the role of both an asynchronous producer and an asynchronous consumer: it is an asynchronous consumer of images coming from the synchronous camera controller component and a producer for the synchronous robot controller. It communicates with two synchronous tasks through two swinging buffers and, being asynchronous, it is the master.

The AsmetaL implementation of this major case study can be found at [11].

VI. Tool-support and Target Implementation

We adopt the ASM modeling and analysis toolset ASMETA [7], based on the Eclipse EMF. A graphical editor for the flowchart extension presented in this paper is being developed. For the execution of ASM models written in

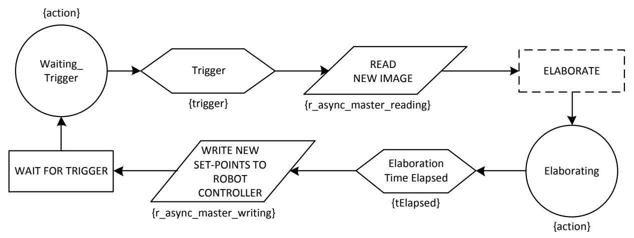


Figure 11. Visual Servoing component.

AsmetaL we use the ASM simulator *AsmetaS*. We also exploit the tool *AsmetaV* to run execution scenarios and report any violation of the expected behavior.

As target implementation platform, we have been working on generating automatically C/C++ code from the ASM models through a mapping towards common used VS libraries, including: *Posix Threads* library, *dc1394* library for managing the communication from/to a firewire (IEEE1394) [13] camera, and *OpenCv* for image elaboration.

VII. Related Work

The use of formal methods to describe the underlying data exchange mechanisms in distributed control systems is not a common practice. Within the ASM community, some few works exist related to the ASM modeling of multi-process synchronization problems and of inter-process communication problems [14][15][16]. We took inspiration from all of them. In particular, to compile ASMs into C/C++ code, we have been repeating the previous experience in the FALKO project (a tool for railway simulation) [12].

The work in [17] reports on the development of AsmL (an ASM notation from Microsoft Research) specifications of Synchronous Dataflow domain schedulers of Ptolemy II – simulation and code generation framework for heterogeneous, concurrent, real-time embedded systems [18]. Their goal is to give a precise semantics to the implementation. The use of the ASMs to design and verify low level communication and data exchange mechanisms, however, seems lacking.

VIII. Conclusion and Future Work

We proposed an ASM-based easy and scalable approach to design reliable and reusable ground models of control tasks and communication/synchronization mechanisms in VS robotic applications. The proposed flowchart extension for control-state ASMS and the availability of such reusable and validated ground models allow a major comprehension of the system design (even to non ASM experts) and speed up the development itself. Though our work is targeted to the VS domain, we believe the approach can be easily extended to a wide range of real-time applications.

As future work, we aim at defining an ASM model library for VS applications and using complex analysis tools in the ASMETA toolset for formal verification.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-231940-BRICS (Best Practice in Robotics).

References

[1] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, Vol. 12, pp. 651–670, 1996.

[2] A. Luzzana, M. Rossetti, P. Righettini, and P. Scandurra, Modeling Synchronization/Communication Patterns in Vision-Based Robot Control Applications Using ASMs. In *ABZ Conf.*, 2012, pp. 331-335.

[3] E. Börger and R. Stärk, *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer Verlag, 2003.

[4] E. Börger, Construction and analysis of ground models and their refinements as a foundation for validating computer-based systems. *Formal Asp. Comput.*, Vol. 19, pp. 225–241, 2007.

[5] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio, A SoC Design Methodology Involving a UML 2.0 Profile for SystemC. In *Proc. of Design, Automation and Test in Europe Conf.*, pp. 704-709, 2005.

[6] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio, A model-driven design environment for embedded systems. In *Proc. of the Design Automation Conf.*, pp. 915-918, IEEE Press, 2006.

[7] The ASMETA website. `http://asmeta.sf.net/`, [retrieved: October, 2012].

[8] P. Arcaini, A. Gargantini, E. Riccobene, and P. Scandurra, A model-driven process for engineering a toolset for a formal method. *J. Softw., Pract. Exper., Vol. 41, pp. 155-166, 2011.*

[9] A. Gargantini, E. Riccobene, and P. Scandurra, A Metamodel-based Language and a Simulation Engine for Abstract State Machines. *J. UCS, Vol. 14, pp. 1949–1983, 2008.*

[10] H. Bruyninckx, Real-time and embedded guide. *KU Leuven, Mechanical Engineering*, 2002.

[11] ASM RTpattern library. `http://asmeta.svn.sf.net/viewvc/asmeta/asm_examples/RTPatternLibrary/`, [retrieved: October, 2012].

[12] J. Schmid, Compiling abstract state machines to C++. *J. UCS*, Vol. 7, pp. 1068–1087, 2001.

[13] "IEEE Standard for a High-Performance Serial Bus". IEEE Std. pp. 1–906, 2008, doi:10.1109/IEEESTD.2008.4659233.

[14] E. Börger and I. Craig, Modeling an operating system kernel. In *Informatik als Dialog zwischen Theorie und Anwendung*, pp. 199–216. Vieweg+Teubner, Wiesbaden, 2009.

[15] R. F. Stärk and E. Börger, An ASM specification of C# threads and the .NET memory model. In *Abstract State Machines*, LNCS 3052, pp. 38–60. Springer, 2004.

[16] W. Müller, J. Ruf, and W. Rosenstiel, An ASM based SystemC simulation semantics. *SystemC: methodologies and applications*, pp. 97–126, 2003.

[17] D. L. Cuadrado, P. Koch, and A. P. Ravn, ASML specification of a Ptolemy II scheduler. In *Abstract State Machines*, LNCS 2589, page 417. Springer, 2003.

[18] PTOLEMY website. `http://ptolemy.eecs.berkeley.edu/`, [retrieved: October, 2012].

Aligning the Normalized Systems Theorems with Existing Heuristic Software Engineering Knowledge

Peter De Bruyn, Geert Dierckx, and Herwig Mannaert
Department of Management Information Systems
Normalized Systems Institute (NSI)
University of Antwerp
Antwerp, Belgium
{*peter.debruyn,herwig.mannaert*}*@ua.ac.be,* {*geert.dierckx*}*@student.ua.ac.be*

***Abstract*—Software applications used by contemporary organizations have to be expendable for incorporating additional functional requirements, as well as adaptable regarding ever changing user requirements. As this evolvability has frequently be noted to be lacking in most information systems, Normalized Systems (NS) theory has recently proposed a framework on evolvable modularity. Based on the concept of systems theoretic stability, NS formulates a number theorems, constituting a set of formally proven necessary conditions in order to obtain such evolvability in modular structures. These theorems were argued to strongly align with heuristic (often tacit) best-practice knowledge of experienced software developers. In order to further validate this claim, the present paper will investigate the degree in which the NS theorems align with best-practice software engineering guidelines based on the set of 22 "bad smells in code" as defined by Fowler and Beck. The analysis shows that the avoidance of the code smells indeed largely aligns with the Normalized Systems theorems. While 14 of the guidelines seem to be reflected by the NS theorems, 4 of them seem to be unrelated to the theorems and another set of 4 code smells seems to be contradicting with NS reasoning.**

***Keywords-Normalized Systems*; *Code Smells*; *Heuristic Knowledge*; *Knowledge Management*.**

I. Introduction

Software applications used by contemporary organizations have to be expendable for incorporating additional functional requirements, as well as adaptable regarding ever changing user requirements. While many best-practice principles exist in order to improve the evolvability of software programs, the knowledge management concerning these heuristics remains inadequate: most of the principles are often only known tacitly and are not applied consistently. In this regard, Normalized Systems (NS) theory has recently formulated a set of four (formally proven) theorems as necessary conditions to obtain evolvable modular structures in software systems [1]–[3]. While these theorems as such are not to be considered entirely new, their value should be seen in their unambiguous formulation and proof, as well as their unification based on a single postulate. In the present paper, the main focus will be aimed at the best-practice knowledge residing into these Normalized Systems theorems. Indeed, it has already been argued that the Normalized Systems theorems offer in fact a more specific and unambiguous way of representing some already existing (tacit) best-practices in the software engineering community [1]–[3]. Hence, we will try to further support this argument by analyzing how the Normalized Systems theorems seem to be largely supported by the guidelines of Fowler et al. [4] based on the prevention of bad code smells.

After briefly highlighting the essence of Normalized Systems theory, Section III will situate the bad code smells in software engineering literature and the relevance of comparing it with the NS theory. A mapping of both approaches will be proposed in Section IV, after which some conclusions will be presented in Section V.

II. Normalized Systems

Normalized Systems (NS) is a theory focusing on the evolvability of software architectures, based on the concept of stability from systems theory. For this purpose, it considers software systems as *modular* systems consisting of a set of instantiations of programming constructs (e.g., methods, data structures, etc.). In order to realize proven evolvability in these systems, first, an *unlimited systems evolution* is considered (meaning that in theory the number of instantiated constructs and their mutual dependencies eventually become unlimited in every software system). Next, the *stability* requirement demands that each bounded set of changes to the software system (e.g., adding a new data construct or adding a new version of an action construct) should have a bounded impact as well (i.e., the impact of a change should only be depended on the kind of change performed to the system and not dependent on the size of the system). Changes which do generate an impact dependent on the size of the system are regarded as instable (as their impact becomes unbounded under the unlimited systems evolution assumption) and are called *combinatorial effects*. In order to enable this stability, NS theory proposed the following four (formally proven) *theorems* as necessary conditions to prevent combinatorial effects [1]–[3]:

- *Separation of Concerns*, requiring that every concern (change driver) is separated from other concerns in its own construct;

- *Data Version Transparency*, requiring that data entities can be updated without impacting the entities using it as an input or producing it as an output;
- *Action Version Transparency*, requiring that an action entity can be updated without impacting its calling components;
- *Separation of States*, requiring that each step in a workflow is separated from the others in time by keeping state after every step.

In reality, building software applications in conformance with all four theorems seems to require a set of five encapsulated higher-level programming constructs (called *elements*) as building blocks for NS conform applications: data elements, action elements, workflow elements, connector elements and trigger elements [1], [3].

In [2, p. 94], it was already noted that the NS "*design theorems are not new, but relate to well-known heuristic design knowledge of software developers*". For instance, a limited set of manifestations of the above explained theorems were already mentioned in [1]–[3] as summarized in Table I. Therefore, it might be interesting to investigate the extent in which other best-practice heuristics in software engineering (e.g., the bad code smells as formulated by Fowler et al. [4]) conform with the NS theorems.

III. Bad smells in code

In the source code of a software program, *bad code smells* are typically considered as symptoms or indications regarding potentially troublesome or problematic code [4]. As such, the concept should be clearly distinguished from typical bugs: flaws or mistakes in the source code resulting in undesired effects (mostly at runtime), such as erroneous output values or security breaches. Consequently, code smells do not imply an erroneous execution of the software program at the time being. Rather, they point to parts in the code of which experience has shown that they have a high chance of causing real problems in the future, when the source code is adapted (e.g., due to highly complex code or its low evolvable structure). Based on this approach, Fowler et al. [4] presented a set of 22 bad smells in code, all being the expression of their experience-based heuristic programming knowledge build up over the years. In order to avoid the presence of these smells, the same authors propose a set of 72 refactorings further on in their book [4]. Here, *refactoring* is not to be considered as changing the delivered functionalities of a software program. Instead, the purpose is to redesign the structure of (a piece of) software code so that potentially troublesome parts (the "smells") are removed, while the program itself is still exhibiting the same behavior at runtime. A short overview of those 22 code smells from Fowler et al. [4] is presented in Table II by providing the name of each smell, a brief description of the potential problem anticipated to arise, as well as an explanation of the proposed remediation.

In general, the occurrence of code smells has become associated with the number and degree of difficulties programmers might be confronted with when trying to change existing code. The concept has become a generally known, accepted and established way for studying the maintainability of software. Indeed, after the publication of the book of Fowler et al. [4], researchers have been extending the repository of existing code smells or using them as a basis for empirical validation and software evaluation (see e.g., [5]–[8]).

Additionally, some limitations to the formulation of the code smells by Fowler et al. [4], can be derived from earlier related work as well, for instance based on Mäntylä et al. [5], [9]. In [9], the authors first argue that the 22 code smells as defined by Fowler et al. [4] are only presented in a single flat list without providing any classification. Hence, they might surpass the maximum number of guidelines which can be grasped and applied by a human being concurrently. Therefore, Mäntylä et al. [9] proposed a taxonomy of 6 bad smell categorizations, claiming to make the set of bad smells more understandable and clarifying the relationships between them. For example, the smells *Long Method*, *Large Class*, *Primitive Obsession*, *Long Parameter List* and *Data Clumps* were all characterized as "bloaters", representing situations in which a piece of code has grown so much that it becomes difficult to be handled effectively. Another taxonomy of the code smells can for instance be found in [10]. In some way, such taxonomies might already be seen as an early attempt to unify several of the code smells of Fowler et al. [4] while looking for some common causing grounds (i.e., what are the reasons for the smells to show up?). In this sense, the NS theorems (being formulated in a very widely applicable way) might show some analogy with these attempts as their aim was to identify and eliminate causes for barriers regarding evolvability (in terms of combinatorial effects) as well. Further, a second limitation suggested by Mäntylä et al. [5], [9] is the fact that the code smells are still somewhat ambiguous, implying that their most significant benefits are to be situated in the subjective evaluation of software evolvability (i.e., performed by individuals). This would limit their potential for a direct translation into software metrics allowing the full automatic detection of infringements by software tools. Indeed, while Fowler et al. [4, p. 63] claimed that their aim was to offer more specific refactoring clues than merely "*some vague ideas of programming aesthetics*", they specifically stated that they did not want to give very "*precise criteria for when a refactoring is overdue*" based on the argument that human intuition is intrinsically superior to a set of pure metrics. Also, both the studies of Mäntylä et al. [5] and Shneiderman [11] show that some disagreement between a set of such subjective software evaluations can arise (i.e., different peo-

Table I
SOME EXEMPLIFYING MANIFESTATIONS OF NORMALIZED SYSTEMS THEORY THEOREMS IN PRACTICE [1]–[3].

NS theorem	Exemplifying manifestations
Separation of Concerns	• Message or integration bus to separate the use of various messaging protocols • The separate use of external workflows by workflow management systems • Multi-tier architectures separating presentation logic, application or business logic, database logic, etcetera
Action version Transparency	• Polymorphism in object-orientation • Wrapper functions (e.g., in C) • Interface definition languages (IDL's) (e.g., used by frameworks such as CORBA and COM)
Data version Transparency	• XML-based technology (e.g., for web services) • Information hiding in object-orientation (e.g., getters and setters in the JavaBean component architecture) • Data structure passing via URL's, property files, tag-value pairs, etcetera
Separation of States	• Asynchronous communication systems • Asynchronous processing in general • Stateful workflow systems

ple might evaluate the same code differently). Therefore, the NS theorems might offer an interesting point of comparison in this situation as well, as their formulation was aimed at providing specific and formally proven guidelines for obtaining software evolvability and offering an unambiguous means of identifying violations against them. Hence, it might be interesting to analyze the extent in which we can map the bad code smells of Fowler et al. [4] on the NS theorems. Or, stated otherwise, to investigate the extent in which the bad code smells can be seen as manifestations or instantiations of (violations regarding) the NS theorems. This will be exactly our goal of the next section.

IV. ALIGNING NS THEOREMS WITH THE AVOIDANCE OF CODE SMELLS

In this section, our aim will be to examine the degree in which we can find conformance between the bad code smells of Fowler et al. [4] and the NS theorems. In order to do so, Table III tries to visualize and map each of the code smells with the NS theorems. The analysis reveals three kind of categories: (1) a set of 14 code smells in full, partial or indirect compliance or support of the NS theorems, (2) a set of 4 code smells not related to the NS theorems and (3) a set of 4 code smells contradicting with the NS theorems. We will now briefly discuss each of them.

A. Code smells in full, partial or indirect compliance or support of the NS theorems

Most of the code smells appear to be in full accordance with the Normalized Systems theorems. We will elaborate two examples here. First, the *Duplicate Code* smell straightforwardly follows from the Separation of Concerns theorem. Indeed, consider a situation in which a certain processing function A includes (amongst others) code chunk X, which is duplicated in another processing function B as well. This reveals that functions A and B contain at least two concerns (i.e., change drivers, tasks). In case X would then be changed (e.g., due to a new version or mandatory upgrade), both processing functions A and B should be adapted. Considering an unlimited systems evolution perspective, the eventual impact might become related to the overall system size and hence result in a combinatorial effect. Second, the *Long Parameter* smell is supported by, amongst others, the Action version Transparency theorem. Suppose that a processing function A has an interface w requiring a set of (primitive) input parameters $S_1, S_2, ..., S_a$ to perform its functionality. Suppose further that a set of L processing functions is calling A. A new version of A in order to incorporate some additional functionality might require additional primitive input parameter(s) and hence, the interface w might have to change (e.g., an additional input parameter S_b is added to the parameter list). In this case, all L processing functions should be adapted in order to keep calling A correctly (resulting in a combinatorial effect under to unlimited systems evolution assumption). The Action version Transparency theorem will prohibit the creation of such *Long Parameter* smell by requiring each processing function to exhibit version transparency. In practice, this is realized by avoiding the use of primitives (such as String, integer, etc.) in the interface of a processing function. Instead, objects as a whole are passed (i.e., encapsulated data structures as suggested by Fowler et al.). This would for example mean that instead of passing parameters amount (integer), beneficiary name (String), etcetera, the object Invoice will be passed to a processing function. Based on this reference to the Invoice object, the processing function can request all information it needs to perform its function. A new version of a processing function A (e.g., requiring information about the currency of the invoice) can now be implemented while keeping the same interface (i.e., the reference to the Invoice object) and not requiring any additional changes in the L calling processing functions.

Also, it is strikingly to note that the definition by Fowler et al. [4] concerning the two code smells which were mapped on all four NS theorems (i.e., *Divergent Change* and *Shotgun Surgery*) reflect the typical operationalization of evolvability in NS. For example, the definition of the *Divergent Change* smell almost fully corresponds with the notion of change drivers in NS as it demands for the identification of objects

Table II
AN OVERVIEW OF THE 22 CODE SMELLS AS DISCUSSED BY FOWLER ET AL. [4]

Code Smell	Summary
Duplicate Code	A code fragment occurring in two or more places in the code base. The code should be refactored in such way that all the duplicate fragments are grouped and become located at one place.
Long Method	A method containing a long sequence of code, often reflecting the incorporation of multiple functionalities. For ease of use and maintainability purposes, the code should be refactored in such way that each functionality becomes separated in its own method.
Large Class	A class containing too many (infrequently used) member variables and methods, often being an indication of duplicate code as well. The code should be refactored in such a way that duplicate code is eliminated and fixed clumbs of variables are separated in a distinct class.
Long Parameter	Methods having a long list of needed parameters for calling them, result in complexity and maintenance issues. The code should be refactored in such a way that instead of variables, objects are passed to the called methods. The needed information from those objects can than be requested by using for instance typical get-methods.
Divergent Change	The phenomenon that a class becomes frequently changed in different ways for different reasons. The code should be refactored in such a way that everything that changes for a particular cause becomes separated in its own class.
Shotgun Surgery	The phenomenon that when a (small) kind of change is aimed for, many (little) adaptations have to be made to a lot of different classes. This causes additional effort to perform changes and creates risks regarding internal consistency. The code should be refactored in such a way that changes remain contained in a single class resulting in a one-to-one link between common changes and classes.
Feature Envy	The case in which the method of an object tends to use more frequently variables (data) from other classes, than its own variables. This can occur as objects in object-orientation are typically a combination of both data (variables) and actions (methods). The code should be refactored in such a way that the method is replaced to the class from which it is intensively using the data.
Data Clumbs	Groups of data (variables, parameters) often occurring together in different objects. This hampers adaptability and increases complexity of method callings. The code should be refactored in such a way that the bunches of recurring data become separated in their own class.
Primitive Obsession	The (excessive) use of pure primitives or record types (i.e., a structure of data into a meaningful group) to pass on data in software. This is often a complex and inefficient way to deal with data. Rather, the code can often be refactored in such a way that small set of primitives is grouped into a (small) object such as a money class with variables for the number, currency, ranges, etcetera.
Switch Statements	Switch statements have the tendency to indicate duplicate code in the source code as often the same switch statement is scattered about a program in different places. In case a new clause is added, removed or changed within the statement, all statements have to be found and changed. As such, it is proposed to refactor the code by use of polymorphism.
Parallel Inheritance Hierarchies	The phenomenon in which a change in a subclass of one class implies a change in the subclass of another class. This can be seen as a special case of Shotgun Surgery smell. The code should be refactored in such a way that the instance of one hierarchy refer to the instances of the other.
Lazy Class	Each class created requires effort to create, maintain and understand. Hence, in case classes are present which are not performing enough functionality to justify these efforts,the code should be refactored in such a way that they are removed.
Speculative Generality	The presence of methods and classes incorporating future functionalities, but which do not always tend to be used in practice. The code should be refactored in such a way that this overhead is reduced in order to improve understandability and maintainability.
Temporary Field	The situation in which a class has an instance variable which is only set in certain circumstances. This works confusing and adds to complexity. As such, the code should be refactored in such a way that the temporary fields are replaced to a new class, in which each instantiation effectively uses the fields.
Message Chains	When a client asks for a certain object, the situation might occur that this object makes a request to another object, making at its turn a request to yet another object, and so on. Such method chain creates coupling and a dependency between the client and the calling stack. The code should be refactored in ways like adding a separate method handling the chain navigation.
Middle Man	The phenomenon in which delegation is taken to an extreme situation in which a class is nearly passing all of its incoming requests to other classes performing the actual functionality. The code should be refactored in such a way that the delegate (“middle man”) is eliminated from the hierarchy structure.
Inappropriate Intimacy	The case in which a class is too “intimately” tied to another class, often reflected in a low degree of cohesion of the considered, as well as a high degree of coupling between them. The code should be refactored in such a way that the coupling between the classes is lowered by for instance moving fields (variables), methods, rearranging directional links between classes, etcetera.
Alternative Class with Different Interface	The occurrence of a number of methods doing the same thing, but having several different interfaces. Frequently, this goes hand in hand with duplicate code. The code should be refactored in such way that the methods are renamed and adapted so they all have the same name and interface, and duplicate code becomes removed.
Incomplete Library class	When reusing external library classes when building your own code, these library classes may turn out to be incomplete for performing all required functionalities. Most often, adapting these library classes is very difficult or simply impossible.
Data Class	The occurrence of classes only having data fields with getter and setter methods. They form “dumb” data classes, often being manipulated in too much detail by other classes. The code should be refactored in such a way the data fields become grouped in the same class as the methods which mostly perform actions upon them.
Refused Bequest	The case in which a subclass does not need (many) of the methods its inherits from its base class. Sometimes, this is an indication of a wrong class hierarchy. In this situation, the code should be refactored in such a way that these inconsistencies become removed.
Comments	If many commentary notes are present in the source code, this often indicates bad quality of the considered code as apparently, many aspects need additional clarification. They are often a symptom of the above mentioned code smells. The code should be refactored in such a way that only a little or no extra comment is required in the source code.

Table III
MAPPING THE 22 CODE SMELLS OF FOWLER ET AL. [4] WITH THE NORMALIZED SYSTEMS THEORY THEOREMS [1]–[3].

Code Smell	SoC	AvT	DvT	SoS
Duplicate Code	●			
Long Method	○			
Large Class	○			
Long Parameter		●	●	
Divergent Change	●	●	●	●
Shotgun Surgery	●	●	●	●
Feature Envy		contradicting		
Data Clumbs			●	
Primitive Obsession			●	
Switch Statements	○			
Parallel Inheritance Hierarchies		not related		
Lazy Class		not related		
Speculative Generality		contradicting		
Temporary Field		contradicting		
Message Chains				●
Middle Man				●
Inappropriate Intimacy	●			
Alternative Class with Different Interface	●			
Incomplete Library class	○			
Data Class		contradicting		
Refused Bequest		not related		
Comments		not related		

●: Code smell guideline fully complying with a Normalized Systems theorem
○: Code smell guideline partly or indirectly complying with a Normalized Systems theorem

being subject to only one kind of change at a time. An instance of the *Shotgun Surgery* smell highly resembles the definition of a combinatorial effect and the notion of instability, entailing that a small change might have an impact located in multiple (and eventually an unbounded amount) of places. All four NS theorems precisely aim at avoiding these smells to show up. These examples further clearly illustrate that the code smells could be regarded as a kind of *symptoms* of low evolvable software architectures (i.e., one does not want to be confronted with *Shotgun Surgery*), whereas the NS theorems aim to focus on the *root causes* of these symptoms (i.e., how can one avoid the occurrence of *Shotgun Surgery* based on a set of proven theorems).

Finally, some code smells only seem to be partially or indirectly supported by the NS theorems. For instance, *Long Methods*, *Large Classes* or the use of *Switch Statements* are in themselves no strict violations of any of the NS theorems. However, as Fowler et al. [4] argue that they often tend to give rise to duplicate code and the combination of multiple change drivers (i.e., a violation of Separation of Concerns), they can be thought of as indirectly supporting the NS theorems.

B. Code smells contradicting with the NS theorems

A limited set of four code smells seems to contradict with the Normalized Systems theorems. For instance, the code smells *Feature Envy* and *Data Class* require and recommend programmers to incorporate both data and the actions that are most commonly performed on this data, in one single construct (class). However, in [3] it was argued to analyze the dynamic nature of programming constructs in a multi-dimensional way (i.e., considering different versions for a data structure, different versions for the interface of a processing function and different versions for each of the tasks a function consists of). As the dimensions of variability increase even further when both data and actions are combined into one construct (e.g., a class in typical object-oriented methodologies), the NS theorems imply the use of separate data elements (encapsulated with its get- and set-methods and supporting tasks for cross-cutting concerns such as remote access and persistence) and action elements (containing a single functional tasks and encapsulated with supporting tasks for cross-cutting concerns such as logging and access control). The arguments and parameters needed by an action element are thus to be encapsulated separately into their own data element. This reasoning contradicts with the guidelines based on the code smells from Fowler et al [4]. To the extent that the *Temporary Field* code smell is advocating the same combination of methods together with a set of variables (which all have to be used by those methods), this code smell seems contradictory to the Normalized Systems theorems as well. Indeed, in NS, the identification of separate data elements is prescribed even when not every action element will necessarily use all the fields in every instance.

The *Speculative Generality* smell stresses that the incorporation of future functionalities should only be implemented when there is a reasonable chance that the functionality will eventually be used: the implementation of less likely future functionalities would only add unnecessary complexity. To a certain extent, this might be considered as contradictory to the NS approach as NS would prescribe to isolate each change driver (i.e., each part of code which is anticipated to evolve independently) as soon as possible in its own construct as a way of anticipating all basic elementary future changes, irrespective of its frequency of occurrence. This might indeed introduce some additional design complexity initially, but avoids the occurrence of combinatorial effects later on. On the other hand, in parallel of the concerning code smell, NS would obviously also not prescribe to identify parts of code which are completely unlikely to change independently, as change drivers.

C. Code smells not related to the NS theorems

While most code smells seem to be supporting the NS theorems, some of them seem to be somewhat unrelated to NS as well. For instance, the *Lazy Class* smell deals with the deletion of code parts no longer used. This issue is not directly discussed in NS theory (which considers the deletion of unused parts to be an automatic process of garbage collection instead of a change to the information system) and therefore seems unrelated to the theorems. Equivalently, inheritance structures (cf. the *Parallel Inheritance Hierarchies* and *Refused Bequest* smells) are not really discussed in NS theory as it focuses on the very basic constructs of information systems in terms of data and actions. However, as inheritance typically suggests to use the typical combination in object-orientation of data and actions in one construct (i.e., a class) these smells do not seem to arise in NS systems (arguing for the use of separate data and action constructs). Finally, NS theory does not directly consider the use of *Comments* in the source code.

V. Conclusion and Future Work

In this paper, we presented Normalized Systems theory as an approach for building evolvable software systems based on a set of formally proven theorems, which seem to relate to existing (but often tacit) best-practice heuristic software engineering knowledge. With the aim of supporting the claim that the NS theorems correlate with this practitioners knowledge, we explored the relevance of the set of 22 bad code smells as formulated by Fowler et al. [4] in this regard. Each of the code smells was mapped onto the 4 NS theorems. The analysis showed that most of the bad code smells are reflected by NS reasoning, with the most prevalent impact apparently coming from the Separation of Concerns and Data version Transparency theorems. However, a set of 4 code smells seemed to be unrelated, while another set of 4 code smells even seemed to be contradicting with NS theorems. Besides relating both approaches to each other, this paper (1) supports the work of Fowler et al. [4] by offering a sound theoretical basis for most of their formulated heuristic design guidelines and (2) might offer practitioners more insights into how violations regarding NS theorems might manifest themselves in practice. To some extent, the code smells could then be regarded as a kind of symptoms of low evolvable software architectures, whereas the NS theorems aim to focusing on the root causes of these symptoms. Future research might then be aimed at relating other knowledge repositories regarding software engineering heuristics towards the NS theorems.

Acknowledgment

P.D.B. is supported by a Research Grant of the Agency for Innovation by Science and Technology in Flanders (IWT).

References

[1] H. Mannaert and J. Verelst, *Normalized systems: re-creating information technology based on laws for software evolvability*. Koppa, 2009.

[2] H. Mannaert, J. Verelst, and K. Ven, “The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability,” *Science of Computer Programming*, vol. 76, no. 12, pp. 1210 – 1222, 2011.

[3] ——, “Towards evolvable software architectures based on systems theoretic stability,” *Software: Practice and Experience*, vol. 42, pp. 89–116, 2012.

[4] M. Fowler, K. Beck, J. Brant, O. W., and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Addison Wesley Professional, 1999.

[5] M. Mäntylä and C. Lassenius, “Subjective evaluation of software evolvability using code smells: An empirical study,” *Empirical Software Engineering*, vol. 11, pp. 395–431, 2006.

[6] W. Li and R. Shatnawi, “An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution,” *Journal of Systems and Software*, vol. 80, no. 7, pp. 1120 – 1128, 2007.

[7] S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka, “The evolution and impact of code smells: A case study of two open source systems,” in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 390–400.

[8] N. Moha, Y.-G. Gueheneuc, L. Duchien, and A.-F. Le Meur, “Decor: A method for the specification and detection of code and design smells,” *Software Engineering, IEEE Transactions on*, vol. 36, no. 1, pp. 20–36, jan.-feb. 2010.

[9] M. Mäntylä, J. Vanhanen, and C. Lassenius, “A taxonomy and an initial empirical study of bad smells in code,” in *Proceedings of the International Conference on Software Maintenance*, 2003.

[10] W. C. Wake, *Refactoring Workbook*. Addison-Wesley Professional, 2003.

[11] B. Shneiderman, *Software psychology: human factors in computer and information systems*. Winthrop Publishers, 1980.

A Description Language for QoS Properties and a Framework for Service Composition Using QoS Properties

Chiaen Lin, Krishna Kavi, Sagarika Adepu
Department of Computer Science and Engineering
University of North Texas
Denton, TX 76203 USA
chiaen@unt.edu, Krishna.Kavi@unt.edu, sagarika121@gmail.com

***Abstract*—Web Services Description Language (WSDL) is an XML-based language for describing Web services and how to access them. There are established standards and frameworks for specifying and composing Web services based on the functional properties. A WSDL extension to specify non-functional or Quality of Service (QoS) properties is proposed in this paper. This enables the QoS-aware Web service composition. This paper introduces a framework that adapts publicly available tools for Web services, augmented by ontology management tools, along with tools for performance modeling to exemplify how the non-functional properties such as response time, throughput, and utilization of services can be addressed in the service acquisition and composition process. The framework provides support to achieve specified QoS goals by discovering services based on both functional and non-functional properties, and composing selected services such that the composed system satisfies the overall QoS requirements. The framework can be easily extended to automate the composition of services and update both functional and non-function properties of the combined services.**

***Keywords*-WSDL; Ontologies; Quality of Services; Non-functional Properties; Service Composition.**

I. Introduction

Service oriented architecture (SOA) offers a flexible methodology for the creation and management of software services. Software services are well-defined business functionalities situated in loosely-coupled and distributed computing settings such as Cloud and Web. Each service provides a specific and well defined functionality. Well defined interfaces permit for the discovery and invocation of services. Web service is a realization of the SOA concept. Available standards allow for the creation, registration, discovery and invocation of Web services. Web Services Description Language (WSDL) can be used to specify the functionality of a service along with its communication protocols. Service providers can register services with Universal Description Directory and Integration (UDDI) or other such registry services. Service repository can be queried by customers to discover needed services. The discovery of a service is based on searching through categories and by matching the specification given in WSDL.

The goal of our project is to discover services based not only on their functionality but also based on non-functional (or quality of service) properties. In addition, our goal includes service composition and specification of non-functional properties of composed services. These goals require the ability to specify non-functional (or QoS) properties with services, and the ability to compute non-functional measures of composed services. Ascertaining certain non-functional properties of composed service require models and tools that are appropriate for the specific property (e.g., stochastic models for performance measures). In this paper, we explore the development of the necessary framework for composing performance properties using queuing models.

WSDL can only be used for specifying functionality of services. Non-functional properties, including several quality of service (QoS) characteristics, are crucial to the success and wider adoption of Web services. Customers would like to use QoS characteristics of Web services for selecting from among several alternate implementations. Each of the potential service provider declares similar functionalities for the same purpose – thus the customer expects more information about services. Typical among QoS properties are security, reliability, and performance [1]. WSDL should be extended in order to provide QoS related information with services. Once non-functional properties of services are specified, it will be possible to develop or extend tools for the discovery of Web services based both on functionality and non-functional properties. Additional tools can be designed for service aggregation, integration and composition based on QoS characteristics.

As we proceed with the quality-aware extension to the specification of services, it will be necessary to define standard metrics for non-functional properties. The Cloud Council that is developing a practical guide to Service level Agreements [2], recommends using ISO definitions [3] for standard metrics. Service composition leading to the computation of QoS properties of the composed services present new challenges. Consider for example "response time" as a non-functional property, and consider the composition of two services with 3ms and 5ms response times. One cannot assume that the response time of the composed service is 8ms, since computation of service times are based on stochastic measures and it may become necessary to use appropriate models (e.g., queuing theory) for computing the response time of the composed service. The orchestration of services in a composed service plays an important role in modeling QoS properties of the composite service. In the case of performance, additional complexity results from the current workload at a processing node: a lightly loaded node leads to faster response times. This may necessitate specification of performance properties at different levels of workloads (e.g., at low, average and heavy loads). These complexities can be managed using ontologies for the specification of non-functional properties.

Since our motivation is not only the discovery of services meeting QoS requirements, but also to compose services leading to new services and ascertaining the QoS properties of composed services, we felt that available QoS extensions do not fully meet our needs. Hence we propose our own extensions to WSDL to specify QoS properties. To exemplify the utilization of these extensions, we propose a framework for service composition with the assistance of both ontological and performance modeling tools. QoS properties are modeled with the ontological engine that can be expanded in accordance with the service declaration. Properties that are subject to a chosen performance tool can also be noted in the ontology model for further semantic comparisons. In this paper, we will focus on the performance aspect of the service; in particular service response times, utilization, and throughput. As a proof of concept, we demonstrate the process of WSDL extension, along with its corresponding QoS ontology modeling, performance modeling, and service composition using an example.

The key contributions of our work are (a) WSDL extensions for specifying nonfunctional properties (b) ontology for classification of non-functional properties (c) framework for the discovery of services that meet both functional and non-functional requirements (d) a framework for computing performance (stochastic) measures of composed services.

The layout of the paper is as follows. Section 2 overviews research that is closely related to our work. Section 3 describes how we extended WSDL to include QoS properties. Section 4 introduces the creation of QoS ontology and performance modeling to be used in the Web services. Section 5 uses a case study to demonstrate QoS based Web service composition framework. Section 6 includes our conclusions about the study.

II. Related Works

The description of non-functional properties related to SOA operational management has been described in [4]. In addition to adding some QoS criteria, semantic interpretation to the extensions have been realized in various frameworks [5] [6] [7]. An approach to describing service lifecycle information and QoS guarantees offered by a service based on OWL-S can be found in [8]. Here, service profiles are appended with QoSCharacteristics to generate a corresponding service description repository. The OWL-S based repository can automatically cover the traditional UDDI registry by mapping its elements. In [9], WSDL is extended to X-WSDL where non-functional criteria are added in service definition. Following its predecessor X-UDDI [10], the Web service registration and publication can be queried on the basis of this criteria. In [11], a unified semantic Web services publication and discovery framework is proposed with a QoSMetrics extension to WSDL using PS-WSDL, USQL for service query, and UDDI mapping suites. In this paper, we focus on a proof of concept for WSDL extension and its correspondent non-functional semantic model engineering, but not on the service registration. With our framework, it should be straightforward to apply well-defined UDDI extension tools such as mentioned in [12], or other registry tools.

To enable semantic description of service extensions, several ontological languages have been proposed. An overview of some of these languages can be found in [13]. They focus on the semantic modeling and mapping ontology applied to service descriptions. Our framework focusses on the engineering of ontology model and its references to the performance modeling tools. With the help of ontology mapping, different service description and advertisement standards should be easy to adapt in our framework.

Service composition methods and their languages can be broadly categorized into different types: Orchestration, Choreography, Coordination, and Assembly [14]. While emphasizing from different aspects to approach the issue, composition methods use ontology to annotate QoS attributes that provide common ground for service synthesis, execution, and adaptation [15]. In QoS-aware service composition, services are selected based on inter and intra task constraints. They can also be grouped into deterministic and non-deterministic depending on when these attributes were made known [14]. Various researches are hoping to gain optimal results by using detailed descriptions of QoS values of services during composition [16] [17]. In [18], a quality-driven middleware serves as a composition manager that model multidimensional QoS attributes with utility functions, and optimizes them by local selection and global planning for different quality criteria.

In [19], requested and provided QoS properties are expressed as required specification documents and service specification documents respectively in the open dynamic execution environment. The framework serves as a broker for service compositions that utilizes QoS model in its own ontological language. Service selection algorithms and metrics based on the ontology are utilized by the service broker. Its objective is to support ad-hoc service collaborations, while ours is to facilitate the description of QoS properties of existing and new composite services. The work is similar to ours with the emphasis on using ontology model as the tool to reason QoS attributes semantically. When monitoring the execution condition, the ontology model can facilitate the selection of correct set of QoS values according to the execution environment. The QoS-aware ontology modeling framework we propose can serve the same purpose.

The advantage of our framework is in its facilitation of stochastic performance evaluation during service composition. The above mentioned related works do not consider the use of ontology and performance models working closely to address the evaluation of QoS properties of service composition. In addition, we consider the use of different performance tools with the model-related elements in the ontology, facilitating the usage of QoS attributes based on context and selecting appropriate models and tools for ascertaining properties during composition.

III. QoS-Aware WSDL

WSDL is the standard language suggested by World Wide Web Consortium (W3C) for service specification. It can be read as a conceptual model consisting of components with attached properties, which collectively describe the service [20]. A WSDL specification contains abstract and concrete descriptions of the service. At abstract level, it describes the interface to the service: operations with message exchange patterns (MEP) and parameter types. At the concrete level, a binding specifies the transport type that the interface uses. An endpoint then associates a real network address with the binding, which forms the service. The service is invoked by supplying the declared signature to the interface through its endpoints.

Although the syntactic specifications provide information about the structure of input and output messages, and the functional descriptions of the service, WSDL does not address non-functional properties. To fully utilize Web services, non-functional information, along with functionality, is needed in the service description. To augment any proposed extensions, backward compatibility and its extension level must be considered. Since WSDL description model addresses abstract and concrete components with services, the non-functional extensions to WSDL should be considered accordingly. It should be compatible with the original Web services mechanism in that the addition may be considered optional. Web service engines and operations should be able to freely ignore the QoS information as they choose to operate in the conventional environment. For applications that adapt our framework, the QoS-aware extensions are extracted easily. We decided that the extensions should be established at service level rather than at interface level, since the WSDL interfaces are bounded by the message exchange patterns and considered abstract models. At service level, an endpoint is where the abstract service binds to a concrete port type, where the overall service performance can be noted.

WSDL2.0 Core standard provides element-based extensibility that can be used to specify technology-specific binding. We create an element in WSDL to represent QoS property specification. Then we use the element as the extension element to the endpoint. The service with the endpoint is therefore being annotated by the extended properties. For a QoS property extension element, we use complex type in the XML schema to accommodate the data structure of the QoS. As depicted in Figure 1, the QoS-aware extension schema exemplifies a non-functional property of performance. Within the performance criteria, response time is noted with its value, unit, and category. The extension can also be further referenced by importing latest XML schema version which can be updated on-the-fly as revising the QoS ontology model, thus conforming to the latest XML standards.

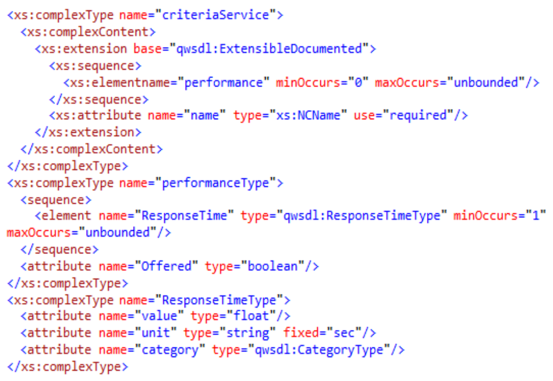

```
<xs:complexType name="criteriaService">
  <xs:complexContent>
    <xs:extension base="qwsdl:ExtensibleDocumented">
      <xs:sequence>
        <xs:elementname="performance" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:NCName" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="performanceType">
  <sequence>
    <element name="ResponseTime" type="qwsdl:ResponseTimeType" minOccurs="1"
maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Offered" type="boolean"/>
</xs:complexType>
<xs:complexType name="ResponseTimeType">
  <attribute name="value" type="float"/>
  <attribute name="unit" type="string" fixed="sec"/>
  <attribute name="category" type="qwsdl:CategoryType"/>
</xs:complexType>
```

Figure 1. QoS-Aware WSDL schema for Performance parameters

IV. Performance Service Composition

Service composition decisions have to be made from considerations of both functional and non-functional requirements. To manage the semantics of both aspects and facilitate the automatic selection of service components that meet the service level requirement, an ontology engine is proposed to efficiently and flexibly classify both functional and non-functional attributes. Services and their components can be further classified according to the application domain, using the category scheme, to facilitate the retrieval and management of corresponding services. The availability of desired service depends on the discovery resulting from querying the ontology model. In case of no services meet the requirements, existing services can be acquired and composed. The newly composed services can then be added to the ontology engine for future selections.

In some cases, computing the non-functional properties of composed services requires stochastic models. Consider performance properties of services such as response time. These performance attributes are used to filter and rank services so that service selections can be made. As new services are composed from its constituent service components, performance indexes can be generated by modeling the new composition through the stochastic model. The performance evaluation results along with the new composed service are added back to the ontological model for future reference.

The backend of the composition framework provides interfaces in utilizing ontology model and models for the evaluation of QoS properties for Web service composition. The two modules are independent and any potentially compatible models and tools can be plugged in. In this paper, we illustrate the process of creating the ontological model, and use a queuing model for composition of performance related properties of services.

A. *Ontological Property Model*

Ontologies offer more accurate and flexible cataloging of entities than taxonomies. While the latter uses hierarchical and branched static structures to group entities and manage information using structural organizations, ontological model annotates semantics with meta-data, relating properties and attributes with more complex organizations than branching or tree like structures. Ontology model therefore provides more flexible organization and semantic interpretation of data with entities.

The quality of service property of a Web service can be inferred by its performance attributes. Criterion of service selection can be formulated by the configuration of these attributes to indicate levels of service importance. Due to the dynamic nature provided by service oriented mechanism, even the meaning of performance metrics should be adapted to fit the context of the service domain. For example, a Web service component qualified for soft real-time application may be considered if they have reasonable response times; however they may not be suitable for hard real-time environments, unless the response times can be bounded. Different contexts impose different semantic interpretations on the same non-functional properties. However the ontology model is highly flexible and thus multiple semantic interpretations can be associated with properties associated with services.

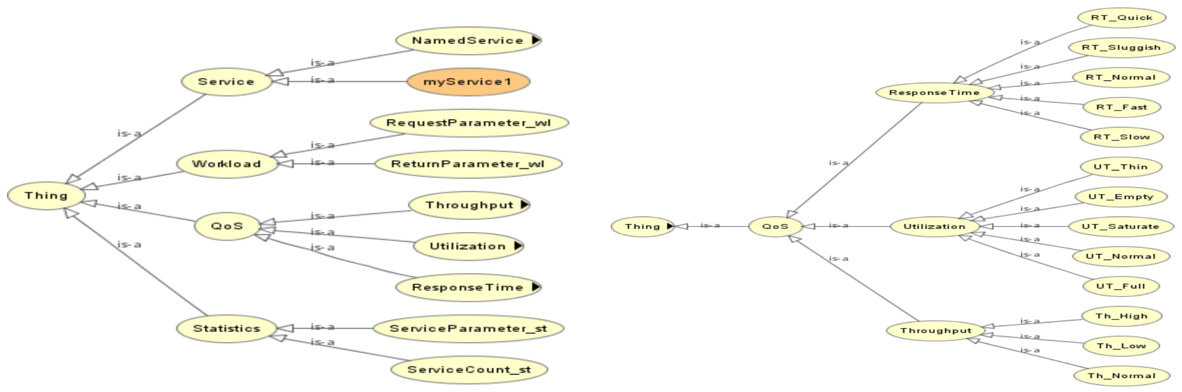
To create an ontological model for Web services, leading to service composition, we demonstrate the process of establishing performance as non-functional property of Web services, such as response time, server utilization, and throughput. Further context related performance indicators can also be easily added with similar considerations. In order to create, update, and query the performance properties during the Web service composition process, we need to establish records of each and every services. We adapt the Protege Editor [21] as the editing tool to help create an ontology model. Protege Editor has a GUI interface [22]. Users can specifically define Entity and Class as first-class elements in the schema along with their Object Properties. Instance of object can be initialized as an Individual and its Data Property can be appended. Visual tools are provided by the editor's plug-ins to facilitate various aspects views. There are also several reasoners available that can be invoked to check and infer the ontological derivations automatically.

We differentiate the first-class elements from base performance and model relevant ones. The model refers to the performance modeling used in the compostion. The base ones serve as the mandatory performance attributes that all the Web services are required to specify as performance indicators. The model-related properties serve as the supplement to the application-specific modeling approaches, thus can store additional attributes for use by specific methods and tools. In our example, the base performance classification is represented by Quality-of-Services (QoS). The QoS subclasses ResponseTime, Throughput, and Utilization are base performance indicators, to quantify performance property. Model-related attributes include Workload and Statistics. Workload here is used as an attribute to evaluate the significance of base QoS properties. The attributes allow for recording the criteria under which the performance properties were derived and thus allow for adjustments when new running environments differ from these values.

For each of the base and model-related first-class elements, classification can also be refined into detailed subclasses. For instance, a response time can be ranked into subcategory such as Fast, Quick, Normal, Slow and Sluggish. Each of the rank can also be noted with its values that represents the class, based on specific context. As the new service composition emerges, the new service can easily be accommodated in the ontological model, establish quantity and corresponding semantics, and is ready for further queries and reasoning. The example model described here includes base and model-related classes and depicted in Figure 2(a). The refined QoS rank subclasses example is depicted in Figure 2(b).

To be able to interface with Protege Editor so that we can update and query the ontology model for service composition processing, we further convert the ontological process into programming. Protege-OWL provides the capability to convert API to equivalent GUI functions and the mechanism for plugging reasoners [23]. We follow the process steps of creating the ontology model, and make the process programmable. The base QoS schema can serve as the building block for the extension of the ontology model. The automation enhances the flexibility to experiment on the first-class cataloging and their refined properties. It also provides a convenient facility to plugin a specific model-related ontology for performance modeling.

The automation process can be further extended with a reasoner to the ontology model that enhances the reasoning ability while interacting with the model. A reasoner implementing the reasoner plugin programming interface will be accessible in the same way that the built-in reasoners are. We choose the Jena Framework from Apache as the reasoner mechanism for our running example. It is well-known and an open source tool. Its query and storage architecture can



(a) Performance Ontology Model

(b) QoS Ontology Model with Refined Ranks

Figure 2. Ontology Models

enable more flexible online usage of the ontology engine.

We adapt Jena programming API [24] to read the ontology model built from Protege-OWL, and validate the model by the reasoner rules. For inference support, Jena provides a general purpose rule engine that the ontology model can be validated and the application specific rule can be applied to facilitate aspects of Web service composition management. The service composition can be fine-tuned by using the rules from domain experts or engineers that impose application related restrictions. For instance, assume the response time of a quick Web service is defined to be less then five milliseconds, the selection of the candidate Web service can be filtered by the rule :

[print-a-quick-WebService: (?x pre-ws:hasQoS ?a) (?a pre-ws:hasResponseTime ?b) (?b pre-ws:rt_value ?c) lessThan(?c, 5.0) → print(?x, 'has quick QoS:',?c)]

The print-a-quick-WebService rule prints out any service entity that has a QoS property with a ResponseTime value smaller than 5.0 msec. Similarly, other plug-in rules can be used to customize the model to meet the needs of an application, such as performance rank selection in a service category.It should be noted that it is possible to define a reasoner that uses context related information to define fast, slow response times subjectively, instead of using values.

The ontology model automation enables the Web service composition to be processed online. New classes and properties can be created on-the-fly to address the specific needs of applications. Web service processes can also benefit from adaptation to different service domains by interpreting the performance parameters. The online feedback from the analysis is the up-to-date data that enhances accuracy of the reasoning.

B. Performance Modeling for Service Composition

Different methodologies for evaluating performance of software services such as process algebra, queueing networks, and Petri nets come with different analysis tools for example, PEPA [25], LQN [26], and SPNP [27]. Stochastic performance models have been widely used in the performance evaluation community. In the Web services community, it also plays an important role in assuring that the service performance meets service level agreements.

The purpose of our framework is to provide a platform that enables the use of appropriate tools for performance evaluation in Web service composition. According to the approaches the process takes, developers can explore different tools fit the nature of the composition. Appropriateness can also be explored by comparing various tools for their usability. To demonstrate the usability of the framework, we explain the use of a queueing model with services containing mandatory performance attributes.

While composing services, the flow among the component services can be described using a workflow or business logic. Each of the services can be represented as service nodes, and the request flow can be modeled as waiting queues. In front of each service node, requests are waiting in line for the service to process them in order. The composition model is formed with the integration of the coordinated services network. The performance outcome of the queueing network is the performance result of the newly composed service. The mapping is seen as a close fit to both the performance evaluation mechanism and the Web service composition concept. In the example of our case study, layered queueing model [26] is adapted as the tool to demonstrate our framework. We will use an example to illustrate the framework (see Section 5).

Layered queueing model is a conventional queueing model embedded with the architecture of a software system and needed resources [26]. The first class elements are processor, task, entity, and activity. A task represents a resource that has processors and other entities to execute. Each of the entities in turn can invoke other entities on other tasks to fulfill the job. These invocations are modeled in layered fashion, and can be depicted as a directed graph. For further detailed modeling, each entity can be represented

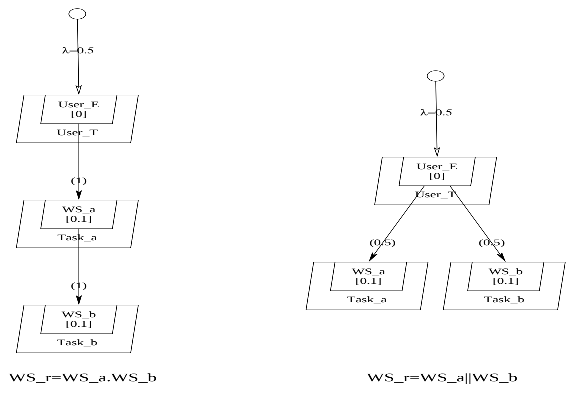


Figure 3. Web Services Layered Queueing Network Modeling

with more specific activities in its own data flow. For each task and activity, there will be resource requirements specified as service time that denotes a performance attribute. And, the mean number of calls represents the average of invocations from one entity to another. With the information for each task and their entities noted, a queuing network can be constructed to represent the integration of all the tasks, leading to workload model using either open or closed queuing models. The former can be modeled with mean arrival and service rates, while the latter can specified using mean value analysis (MVA). As soon as the model is developed, the layered queueing solver can generate reports on the performance indexes such as service time, throughput and utilization for both services and processors. It also generates average waiting times in open queuing model and mean delay in closed models.

The simplest form of a Web service composition involves two services, say WS_a and WS_b. The possible compositions of the two services can be sequential or parallel composition, say WS_rs and WS_rp. Borrowing the syntax from generic process algebra, the sequential composition can be represented as WS_rs=WS_a.WS_b, and the parallel composition can be represented as WS_rp=WS_a||WS_b. Assume WS_a and WS_b each represents an entity in different tasks say Task_a and Task_b. Each task is assigned to run on its own processor on different hosts say Proc_a and Proc_b. The sequential and parallel composition examples with an open arrival rate 0.5 are depicted in Figure 3. Note that the arrival rate is categorized as workload in the ontology model, and the example just serves as an instance. In the case of similar services encountered same workload but running on different platforms, the selection process have to compare the performance indices such as response time or throughput.

The service composition in both sequential and parallel topology can be scaled by accommodating multiple services at once. Resources can be exclusively owned or shared among services. Service composition can be based on either serial or parallel composition of the services involved. The final layout of the queueing network is the conceptual modeling of the Web service composition. The model can be solved by the analyzer and generate the performance indexes for the composed service.

C. Compositional Semantic Web Services

To export the ontological result that is acquired by the Web service composition mechanism, we use Axis [28] as the service publishing interface for demonstration purposes. The interface also enables the abstraction that Web service ontological engine (WSOE) and performance modeling engine provide.

The WSOE provides for composition of services in the context of Web services management. The utility of the composition services include basic service information maintenance and composition. Service management functions include insertion, update, and deletion. WSOE_Insertion creates a record in the performance ontology model with its name and associated performance properties. The performance properties in our running example is the response time of the service. Other non-functional or QoS properties can also be included within our framework. WSOE_Update and WSOE_deletion are used to update and remove the correspondent services.

New service composition information created by the WSOE can be obtained by the WSOE_Compose_Seq or WSOE_Compose_Par. The former will take the list of Web services in the order specified, and model them as a sequential network in the layered queueing model. The output will be the performance indexes for the composed service. For our simple example, the composition would return the predicted execution time. Likewise, WSOE_Compose_Par will take a list of Web service in the argument, and model them as parallel network in the queueing model. The sequential and parallel compositions can be combined to obtain any general compositions of services. The list of the service interfaces are listed in Table I.

V. Case Study

To demonstrate the web services composition framework, we will use a facial recognition service as an example. We chose this example because of our familiarity with it while working on a related project on service composition. The service collects image data from an attached camera and identifies the presence of human faces. The service consists of Facial Detection (FD), Image Converter (IC), and Facial Recognizer (FR), in that order. First each of the component services are described using our QoS-aware WSDL to denote both functional and non-functaional properties. For each of these services we keep their QoS records in the ontology repository. We will assume that the services are all registered so that search engine can match potential candidates

Table I
WEB SERVICE INTERFACES OF WSOE

Service Name	Parameters	Result
WSOE_Insert	Service_Name, Response_Time	Boolean (True/False)
WSOE_Update	Service_Name	Boolean (True/False)
WSOE_delete	Service_Name	Boolean (True/False)
WSOE_Compose_Seq	$SN_1 ... SN_n$	Service_Name, Response_Time
WSOE_Compose_Par	$SN_1 ... SN_n$	Service_Name, Response_Time

Table II
LAYERED QUEUEING MODEL FOR FACIAL RECOGNITION SERVICE COMPOSTION EXAMPLE

```
#General Section
G
"Web service modeling."
0.00001
100
1
0.9
-1

# Processor Information
P 0
p User_P f i
p FD_P f
p IC_P f
p FR_P f
-1

# Task Information
T 0
t User_T r User_E -1 User_P m
100
t FD_T n FD_E -1 FD_P
t IC_T n IC_E -1 IC_P
t FE_T n FR_E -1 FR_P
-1

#Entry Information
E 0
s User_E 0 -1
y User_E FD_E 1 -1
s FD_E 0.1 -1
y FD_E IC_E 1 -1
s IC_E 0.1 -1
y IC_E FR_E 1 -1
s FR_E 0.3 -1
-1
```

```
Service time:

Task Name Entry Name Phase 1
User_T User_E 50.4902
FD_T FD_E 0.5
IC_T IC_E 0.4
FE_T FR_E 0.3

Service time variance (per phase)
and squared coefficient of variation (over all phases):

Task Name Entry Name Phase 1 coeff of var **2
User_T User_E 7598.29 2.98059
FD_T FD_E 0.75 3
IC_T IC_E 0.34 2.125
FE_T FR_E 0.09 1


Throughputs and utilizations per phase:


Task Name Entry Name Throughput Phase 1 Total
User_T User_E 1.98058 100 100
FD_T FD_E 1.98058 0.990291 0.990291
IC_T IC_E 1.98058 0.792233 0.792233
FE_T FR_E 1.98058 0.594175 0.594175
```

Table III
LAYERED QUEUEING MODEL FOR FACIAL RECOGNITION SERVICE COMPOSTION EXAMPLE

```
#General Section
G
"Web service modeling."
0.00001
100
1
0.9
-1

# Processor Information
P 0
p User_P f i
p FD_P f
p IC_P f
p FR_P f
-1

# Task Information
T 0
t User_T r User_E -1 User_P m
100
t FD_T n FD_E -1 FD_P
t IC_T n IC_E1 IC_E2 -1 IC_P
t FE_T n FR_E -1 FR_P
-1

#Entry Information
E 0
s User_E 0 -1
y User_E FD_E 1 -1
s FD_E 0.1 -1
y FD_E IC_E1 1 -1
y FD_E IC_E2 1 -1
s IC_E1 0.06 -1
y IC_E1 FR_E 1 -1
s IC_E2 0.06 -1
y IC_E2 FR_E 1 -1
s FR_E 0.3 -1
-1
```

```
Service times:
Task Name Entry Name Phase 1
User_T User_E 82.342
FD_T FD_E 0.82
IC_T IC_E1 0.36
IC_E2 0.36
FE_T FR_E 0.3

Service time variance (per phase)
and squared coefficient of variation (over all phases):

Task Name Entry Name Phase 1 coeff of var **2
User_T User_E 20207.5 2.98037
FD_T FD_E 1.24138 1.84619
IC_T IC_E1 0.3096 2.38889
IC_E2 0.3096 2.38889
FE_T FR_E 0.09 1

Throughputs and utilizations per phase:
Task Name Entry Name Throughput Phase 1 Total

User_T User_E 1.21445 100 100
FD_T FD_E 1.21445 0.995846 0.995846
IC_T IC_E1 1.21445 0.437201 0.437201
IC_E2 1.21445 0.437201 0.437201
Total: 2.42889 0.874401 0.874401
FE_T FR_E 2.42889 0.728668 0.728668
```

meeting both functional and non-functional requirements of the customer.

To create the composed service, a list of qualified candidates of each component services are evaluated. Let us assume that our selection picked FD_E, IC_E, and FR_E as service components. We will now evaluate the non-functional values for response time of the composed service. We model the layered queueing network as follows. The service components are mapped as the entities in the layered queueing network with their correspondent tasks PD_T,IC_T, and FR_T, each of which uses processors FD_P,IC_P, and FR_P. The modeling script and the performance indexes of the example are shown in Table II.

Furthermore, let us assume that Image Converter (IC) service can be composed in parallel to improve performance. The composition engine can be configured to explore the service composition using parallel workflow among the services. The new composition would use two Image Converter (IC) services in parallel named IC_E1 and IC_E2. The modeling script of the example and the performance indexes result are shown in Table III.

Although we only used a simple example and a single property here, our framework is very general and flexible so that it can be easily extended for more complex service discovery based on many QoS properties, and can composed in very complex manner.

VI. CONCLUSION

In this paper, we described a framework for composing Web-services using both functional and QoS properties. We first extended WSDL descriptions of Web-services so that non-functional or quality of service parameters can be associated with the service. We also developed APIs for locating Web-services based on both functional and non-functional properties. We have developed ontologies that can be used to select and compose Web-services. For the purpose of composing non-functional properties of component services, new reasoning engines must be developed. Different non-functional properties may require different reasoning engines. In this paper, we outlined how performance properties can be composed using queuing engines. For the purpose of this paper we demonstrated how services can be composed either in series or in parallel, and used a queuing engine to derive the performance properties of the composed service.

We plan to extend the framework for composing Web-services using other types of QoS properties. While it is possible to use other available tools, in our study we will rely on open source tools.

VII. ACKNOWLEDGEMENTS

This research is supported in part by the Net-Centric Industry/University Cooperative Research Center (Net-Centric IUCRC) and a grant from NSF, #1128344.

REFERENCES

[1] S. Balasubramaniam, G. Lewis, E. Morris, S. Simanta, and D. Smith, "Challenges for assuring quality of service in a service-oriented environment," in *Principles of Engineering Service Oriented Systems, 2009. PESOS 2009. ICSE Workshop on*. IEEE, 2009, pp. 103–106.

[2] "Practical guide to cloud service level agreements," 2012, http://www.cloud-council.org/press-release/04-03-12.htm [retrieved: Oct,2012].

[3] "ISO/IEC 20926," 2009, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51717 [retrieved: Oct,2012].

[4] D. Edmond, J. O'Sullivan, and A. ter Hofstede, "What's in a service? towards accurate description of non-functional service properties," *Distributed and Parallel Databases Journal*, vol. 12, pp. 117–133, 2002.

[5] S. Chaari, Y. Badr, and F. Biennier, "Enhancing web service selection by qos-based ontology and ws-policy," in *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, 2008, pp. 2426–2431.

[6] H. Muñoz Frutos, I. Kotsiopoulos, L. Vaquero Gonzalez, and L. Rodero Merino, "Enhancing service selection by semantic qos," *The Semantic Web: Research and Applications*, pp. 565–577, 2009.

[7] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell, "Sawsdl: Semantic annotations for wsdl and xml schema," *Internet Computing, IEEE*, vol. 11, no. 6, pp. 60–67, 2007.

[8] C. Schröpfer, M. Schönherr, P. Offermann, and M. Ahrens, "A flexible approach to service management-related service description in soas," *Emerging Web Services Technology*, pp. 47–64, 2007.

[9] N. Parimala and A. Saini, "Web service with criteria: Extending wsdl," in *Digital Information Management (ICDIM), 2011 Sixth International Conference on*. IEEE, 2011, pp. 205–210.

[10] Parimala, N. and Saini, A., "Decision support web service," *Distributed Computing and Internet Technology*, pp. 221–231, 2011.

[11] T. Pilioura and A. Tsalgatidou, "Unified publication and discovery of semantic web services," *ACM Transactions on the Web (TWEB)*, vol. 3, no. 3, p. 11, 2009.

[12] C. Atkinson, P. Bostan, G. Deneva, and M. Schumacher, "Towards high integrity uddi systems," in *Business Information Systems Workshops*. Springer, 2009, pp. 350–361.

[13] C. Pedrinaci, M. Maleshkova, M. Zaremba, and M. Panahiazar, "Semantic web services approaches," *Handbook of Service Description*, pp. 159–183, 2012.

[14] G. Baryannis, O. Danylevych, D. Karastoyanova, K. Kritikos, P. Leitner, F. Rosenberg, and B. Wetzstein, "Service composition," *Service research challenges and solutions for the future internet*, pp. 55–84, 2010.

[15] G. Dobson and A. Sanchez-Macian, "Towards unified QoS/SLA ontologies," in *Services Computing Workshops, 2006. SCW'06. IEEE*. IEEE, 2006, pp. 169–174.

[16] G. Canfora, M. Di Penta, R. Esposito, and M. Villani, "Qos-aware replanning of composite web services," in *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005, pp. 121–129.

[17] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma, "A qos-aware selection model for semantic web services," *Service-Oriented Computing–ICSOC 2006*, pp. 390–401, 2006.

[18] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 311–327, 2004.

[19] A. Mukhija, A. Dingwall-Smith, and D. S. Rosenblum, "Qos-aware service composition in dino," in *Web Services, 2007. ECOWS'07. Fifth European Conference on*. IEEE, 2007, pp. 3–12.

[20] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana, "Web services description language (wsdl) version 2.0 part 1: Core language," *W3C Recommendation*, vol. 26, 2007.

[21] H. Knublauch, R. Fergerson, N. Noy, and M. Musen, "The protg owl plugin: An open development environment for semantic web applications," *The Semantic WebISWC 2004*, pp. 229–243, 2004.

[22] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe, "A practical guide to building owl ontologies using the protg-owl plugin and co-ode tools edition 1.0," *The University Of Manchester*, 2004.

[23] H. Knublauch, "Protg-owl api programmers guide," *2008-04-22].http://protege.stanford.edu/plugins/owl/api/guide.html*, 2006.

[24] A. Jena, "semantic web framework for java," *URL: http://jena.sourceforge.net*, 2007.

[25] S. Gilmore and J. Hillston, "The pepa workbench: a tool to support a process algebra-based approach to performance modelling," *Computer Performance Evaluation Modelling Techniques and Tools*, pp. 353–368, 1994.

[26] G. Franks, P. Maly, M. Woodside, D. C. Petriu, and A. Hubbard, "Layered queueing network solver and simulator user manual," *Dept.of Systems and Computer Engineering, Carleton University (December 2005)*, 2005.

[27] G. Ciardo, J. Muppala, and K. Trivedi, "Spnp: stochastic petri net package," in *Petri Nets and Performance Models, 1989. PNPM89., Proceedings of the Third International Workshop on*. IEEE, 1989, pp. 142–151.

[28] A. Axis, "Apache web services project," *Available HTTP: http://ws.apache.org/axis*, 2010.

Context Awareness in Learning Human Habits

Szymon Bobek
AGH University of Science and Technology
al. A. Mickiewicza 30
Krakow, Poland
Email: sbobek@agh.edu.pl

Weronika T. Adrian
AGH University of Science and Technology
al. A. Mickiewicza 30
Krakow, Poland
Email: wta@agh.edu.pl

***Abstract*—Mobile devices have gained steadily increasing popularity over the last few years. They collect and process various kinds of data which can be used as rich inputs to infer user preferences and habits. In this paper, we propose an architecture of a mobile application that will serve as an *intelligent assistant* capable of learning human habits and giving suggestions and recommendations. The system will learn patterns in its user's behavior with respect to his or her location, social preferences and activities that can be measured with a mobile phone sensors. Data gathered by the mobile device will be used to model daily, monthly or annual routines of the user. Based on that model, the mobile assistant will be able to find deviations in an organizational rhythm of the user and perform appropriate actions. The system will uses machine learning algorithms and ontologies to learn and model human behavior.**

***Keywords*-Context-awareness; machine learning; ontologies; mobile applications.**

I. INTRODUCTION

Various information and data sources can be nowadays reached from nearly every place in the world. Internet access is now possible through notebooks, tablets and more often – mobile phones, which are omnipresent in human daily routine. The last tend to play a role of *all–in–one* devices that serve as phones, calendars, web browsers, GPS navigations, and social media interfaces.

Modern mobile phones (or more commonly named: smartphones) can be themselves valuable data sources of human habits with respect to:

- one's usual location over time (for instance home, work, cinema, etc.),
- one's social preferences (who, where and how often the one meets),
- one's entertainment preferences (e.g., one more often goes to the opera than cinema),

and combinations of the above.

Monitoring human behavior can lead to development of a rich knowledge base, not only useful for sociologists, but also being a great input into systems that, based on the individual habits, will try to optimize user daily routines. Intelligent houses [1] are one of the most popular way of implementing ambient intelligence [2] solutions in real life. Tools monitoring human social behavior [3] with smartphones show that ubiquitous computing and machine learning techniques can be successfully implemented on mobile platforms as real-time hybrid systems.

One of the biggest advantage of using mobile platforms is that a system deployed on it can accompany its user almost everywhere. Hence, it can become a powerful source of information about the user behavior. This paper presents an idea of using smartphones to monitor and learn human habits from heterogeneous sources: phone sensors (accelerometer, Bluetooth, light sensor), GPS data or data available from functionalities like location sharing via Facebook. Information gathered from these sources can be treated as an input for a system that will learn human habits and act upon this *knowledge* as a intelligent mobile assistant.

The paper is organized as follows: In Section II, we present an overview of selected related work. The original contribution of our approach is discussed in Section III. Section IV describes techniques that can be used for knowledge acquisition about human behavior. The proposed method of modeling in a way that can be used for further inference is presented in Section V. Section VI describes challenges and main problems faced by our approach. The paper is summarized in Section VII.

II. RELATED WORK

Ambient intelligence applications and ubiquitous computing are nowadays widely used in intelligent systems. Various sorts of such systems cover different aspects of human living. One of the most popular automation systems are intelligent houses. Projects like CASAS [4], MavHome [5], or Intellidomo [1] build user profiles based one their activity at home during a day. Several techniques are incorporated within these projects that are crucial in discovering human habits. They include:

- temporal reasoning for describing time dependencies between the activities,
- methods of automatically constructing universal models by taking the output of sequential data mining algorithms and sequential prediction algorithms,
- methods of discovering sequences of user actions in a system based on speech recognition, and
- ontologies and production rules for describing model of human behavior.

However, intelligent houses build human profile and can adapt to it only within a limited space. They create human habits models using data from sensors installed within a house. Hence, the behavior profile is limited to describing user activities inside this building.

A different approach is represented in the SocialCircuits platform [3]. The platform uses mobile phones to measure social ties between individuals, and uses long- and short-term surveys to measure the shifts in individual habits, opinions, health, and friendships influenced by these ties.

Sociometric badge [6] has been designed to identify human activity patterns, analyze conversational prosody features and wirelessly communicate with radio base-stations and mobile phones. Sensor data from the badges has been used in various organizational contexts to automatically predict employee's self-assessment of job satisfaction and quality of interactions.

Reality Mining is a term coined by Eagle and Pentland [7]. The authors used mobile phone Bluetooth transceivers, phone communication logs and cellular tower identifiers to identify the social network structure, recognize social patterns in daily user activity, infer relationships, identify socially significant locations and model organizational rhythms.

III. Motivation

Most of the systems described in Section II monitor and build human profile based on data gathered from sources that are highly correlated. They are limited to one domain of user activity, like home daily routine or social activities. Moreover, they are more concerned with monitoring *human environment* than *a human within the environment*. Hence, it is not possible to create a comprehensive human habits profile based on a human daily routine. The project that is the closest to the idea that we want to incorporate in our work is described in [7]. However, the information presented by Eagle and Pentland was not incorporated into any real-life system. In addition to this, data gathered within the project was processed using statistical and machine learning tools, but no formal model of the human behavior was presented.

The original contribution of our approach consists in:

1) extending the idea presented in [7] by additional data sources: GPS location, accelerometer sensors, Wifi/GPRS activity, and RFID sensors,
2) learning and developing a model of human behavior with methods that allow for automated inference (formally grounded ontologies, see Section V), and
3) implementing a system that will work in real-time as a mobile assistant.

A system that has all the above mentioned information at its disposal can act as an intelligent assistant, monitoring regular behaviors of the users and recommending actions or signaling aberrations. Such an assistant can be a helpful tool optimizing daily routines of multi-tasking persons that interact with technology on a daily basis. This *optimization* will mainly consist in suggesting specific actions or decisions inferred from the model of the user habits and external knowledge. However, this is not an only possible area of application.

Equally important is enriching the quality of life for elderly people who wish to stay mobile and independent. Nowadays, technology services are available to and used by more and more adults, and the number of technology-aware raisins will increase over time. Helping them is not only an important goal, but also a trending research field, supported by international funding programs such as Ambient Assisted Living. The program is motivated by the demographic change and aging in Europe, which implies both challenges and opportunities for the citizens, the social and health care systems, the industry and the EU market. Our proposal aligns with the newest call for proposal entitled: "ICT-based Solutions for (Self-) Management of Daily Life Activities of Older Adults at Home". By encompassing other data sources into the profile modeling, we can enhance the solution to assist people also outside their houses.

IV. System proposal

This section is reworked to answer reviewers' comments We propose a mobile assistant system that will learn the human behavior patterns, build a semantic model of them and be able to reason over it. In particular, it will recognize aberrations and react to them or suggest decisions based on observations of human regular behavior. Example use cases may include: deciding on route based on the transportation habits and current traffic, adjusting daily meetings based on the user's location habits, or warning users if some variances are recorded (e.g., being late for work or forgetting shopping).

A. Proposed Architecture

The architecture of the proposed system can be observed in Figure 1. *Data Sources* module is responsible for gathering:

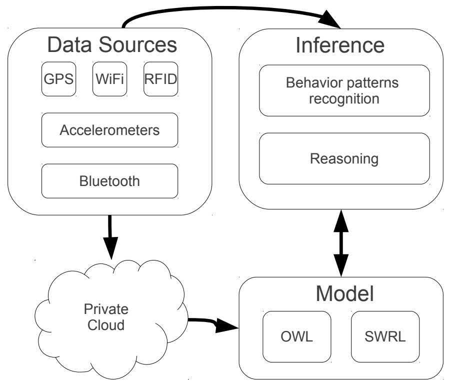


Fig. 1. Proposed Architecture of the Mobile Assistant System.

- GPS location over time,
- the location with respect to other devices (RFID and Bluetooth data),
- accelerometer data determining if the user is walking, running, or not moving at all.

Inference module performs the main learning and reasoning tasks. In the learning phase, it identifies human behavior patterns, by recognizing and clustering heterogenous information (location, time, speed etc.) The other function of the module is inferring relations between activities and suggesting actions to be taken. Some of the suggestions will be automatic, based on rules defined in the *Model* module, e.g., *If it is time that* *regularly the user is driving to work but now the user is not moving, then play a loud alarm to wake them up*. It is also planned that the user will be able to define their own rules, e.g., *If I run for 15 minutes at a speed above average, play a tune of... (my favourite song)*.

The *Model* module will store the model of the user profile. The model will comprise of an ontology and rules (see Section V for more details). It initially consists of a top-level ontology and rules defined for abstract concepts. During learning the human behavior patterns, the ontology will be specialized and adapted to the particular user, its common locations (e.g., home, work, park), typical intervals of time (e.g., morning, evening, workday) and objects (persons and devices). The abstract rules are then instantiated to work on specific locations and time intervals characteristic for the user.

B. Learning Human Habits

The system will work in two phases: learning and acting. In the learning phase, the data (in a form of vectors describing different dimensions of information) will be an input for machine learning algorithms [8] that will try discover patterns in human daily routine. The K-Means clustering algorithm will be implemented to identify users behavior patterns in daily routine. After the clusters are identified, they will be semantically annotated by a user.

The system will propose sets of initially classified data and ask to categorize them (identify certain ranges of values). For instance, locations can be flagged as *home, work, cinema* etc. Other devices recognized via Bluetooth or RFID sensors can represent other users or objects. Time intervals can be identified as mornings, evenings, workdays etc. Various speed values can be assigned by user as walking, running, or driving.

The above classification will lead to the development of a personal ontology based on a top-level ontology. The classification will allow to build *context predicates*, e.g., *<user, locatedIn, park>, <user, performs, running>*, based on which the *activities* (e.g., shopping, driving to work, meeting friends, relaxing) will be defined (see Section V).

Once the learning phase is done, the data gathered by the *Data Sources* module will be passed to the *Inference* module where on-line pattern identification will be performed and appropriate action taken. Since the on-line classification has to be fast, the artificial neural network implemented on the mobile device will be responsible for this.

The system, based on the real-life data, will predict if the user is realizing daily routine or if there are some derogations that should be reported. To allow this feature to work properly, a special model of human behavior and daily habits should be created. The proposal of this model is sketched in Section V.

V. Knowledge Representation

Data gathered by sensors can be uniformly represented using a graph model. Persons recognized via Bluetooth and objects identified with RFID can be uniquely represented as named nodes in the graph. This approach partially realizes the *Internet of Things* [9] idea in blending the borders between the real world and the virtual model. Accelerometer data will serve to calibrate the ranges for concepts such as running, walking or driving and GPS coordinates will be aggregated and named as locations (home, work, grocery store etc.). Multiple relations will combine into a semantic network representing the user behaviors and habits. The information will be represented with RDF [10], a flexible and universal Semantic Web [11] language that will allow the representation all of the information in a standardized way.

In order to enable automatic reasoning, formalization of the representation is needed. In our approach, we have adopted the top-level ontology for smart environments introduced by Ye et al. in [12]. The authors have identified common semantics for several domains (time, location, speed etc.) and their shared relations, such as: *finer-grained*, *equal to*, *conflicting* or *overlapping*. Consequently, this top-level ontology provides "generic rules to facilitate pervasive tasks including detecting inconsistency of information, generating new knowledge such as new activities for different applications and new relationships between concepts, contexts, and activities" [12].

The chosen ontology provides a framework for several dimensions of information. Therefore, it is suitable for modeling human habits profiles in various locations and time intervals. Each dimension of information contains a set of irreducible *grounded values* (e.g., GPS coordinates, second). Over the ranges of grounded values, *abstract values* are identified. These are mappings to a set of grounded values and are done by semantic tagging by user over preliminary classified data (see Section IV). Abstract values define the ranges of grounded values and allow to create user-specific subclasses (e.g., park, morning) of the top-level classes like *location* or *time*.

Abstract values serve to build the *context predicates* e.g., *<user, locatedIn, park>, <user, performs, running>*, which, in turn, are used to define *activities* (e.g., shopping, driving to work, meeting friends, relaxing). Activities are defined by a conjunction of context predicates, e.g. `<user, locatedIn, park>` $\wedge$ `<user, performs, running>` $\wedge$ `<sensor, time, 7:00>`$\Rightarrow$`<user, engagesIn, ''jogging''>` (see [12] for more details). The rules which govern the logic of the application will operate on the activities and timestamps and will activate certain actions or recommendations.

Using a formally-grounded ontology in the system allows for automated logical inference over the model, which is developed based on data gathered in the application. Intended modeling language is OWL [13]. The initial model has been prototyped in OWL and SWRL [14], because this is the modeling language of the Ontonym ontologies [15].

VI. Challenges and Problems

There are several challenges and problems to be considered. One of the main problems is the privacy issue. Users may not want to have information about their location stored on a global server. A separate system called *private cloud* has to be developed, that would allow users to store private data on their computers, and share it only with selected users. The idea of the *private cloud* combines two paradigms of storing data: 1) privacy and 2) distribution of data. The first aspect puts stress on the confidentiality of data: only the owner of the data and authorized users have access to the data. The second aspect touches the problem of accessability of data. If data has to be accessible for many mobile users, the most efficient way is to store it in the distributed environment called *cloud*. That comes with the privacy issues that leads to the conclusion that safety and distribution is very challenging combination that is out of the scope of this work. Another problem is battery cost. The software that would use data from several sensors and process it in a real time will be very power-consuming. Installed on a mobile phone, this may cause the battery to run out very quickly. Another type of challenges is related to users with high entropy of their daily routine. For example, a person who works as a taxi driver probably will not have an usual location that can be tagged as for instance *work*. Those problems and challenges will have to be addressed in later work in order to ensure the software works correctly and efficiently.

VII. Conclusion and Future Work

Mobile devices have gained steadily increasing popularity over the last few years. They collect and process various kinds of data from registering calls and storing messages to logging GPS locations and receiving data from other devices via wireless protocols. These collections of data can be used as rich inputs to infer user preferences and habits. In this paper, we presented a proposal of an architecture of a mobile application that will serve as an *intelligent assistant* capable of learning human habits and giving suggestions and recommendations. Several techniques are planned to be implemented in the project, including machine learning algorithms for patterns recognition in human daily routine and ontologies for modeling human behavior. The system can not only be dedicated to people who would like to optimize their daily routine, but also for those who need care and attention during a day. The system can work as an artificial assistant for older people, or as health care tool that will help monitoring and reporting patients behavior.

For future work, we plan to adapt the semantic knowledge-based wiki environment Loki to allow the user to monitor and configure their semantic profile [16]. Using visual methods for designing rules [17], the user will be able to define their own actions to be taken in case of a variance in their behavior. Enriching the functionality by allowing collaboration among users [18] will open up possibilities of e.g., organizing meetings and adjusting behavior profiles of several people.

Acknowledgment

The paper is supported by the AGH University of Science and Technology Grant 11.11.120.859.

References

[1] A. Lozano-Tello and V. Botn-Fernndez, “Analysis of sequential events for the recognition of human behavior patterns in home automation systems,” in *Distributed Computing and Artificial Intelligence*, ser. Advances in Intelligent and Soft Computing, S. Omatu, J. F. De Paz Santana, S. R. Gonzlez, J. M. Molina, A. M. Bernardos, and J. M. C. Rodrguez, Eds. Springer Berlin / Heidelberg, 2012, vol. 151, pp. 511–518.

[2] B. Epstein, “Ambient intelligence sources,” http://www.epstein.org/brian/ambient_intelligence.htm, [retrieved: September, 2012].

[3] I. Chronis, A. Madan, and A. S. Pentland, “Socialcircuits: the art of using mobile phones for modeling personal interactions,” in *Proceedings of the ICMI-MLMI ’09 Workshop on Multimodal Sensor-Based Systems and Mobile Phones for Social Computing*, ser. ICMI-MLMI ’09. New York, NY, USA: ACM, 2009, pp. 1:1–1:4.

[4] P. A. Valiente-Rocha and A. L. Tello, “Ontology and swrl-based learning model for home automation controlling.” in *ISAmI*, ser. Advances in Soft Computing, J. C. Augusto, J. M. Corchado, P. Novais, and C. Analide, Eds., vol. 72. Springer, 2010, pp. 79–86.

[5] G. M. Youngblood, D. J. Cook, and L. B. Holder, “Managing adaptive versatile environments,” *Pervasive and Mobile Computing*, vol. 1, no. 4, pp. 373–403, 2005.

[6] D. O. Olgun, B. N. Waber, T. Kim, A. Mohan, K. Ara, and A. Pentland, “Sensible organizations: Technology and methodology for automatically measuring organizational behavior,” *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B: CYBERNETICS*, pp. 43–55, 2009.

[7] N. Eagle and A. (Sandy) Pentland, “Reality mining: sensing complex social systems,” *Personal Ubiquitous Comput.*, vol. 10, no. 4, pp. 255–268, Mar. 2006.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[9] K. Ashton, “That ’Internet of Things’ Thing,” http://www.rfidjournal.com/article/view/4986, [retrieved: September, 2012].

[10] E. Miller and F. Manola, “RDF primer,” W3C, Tech. Rep., 2004.

[11] T. Berners-Lee, J. Hendler, and O. Lassila, “The Semantic Web,” *Scientific American*, May 2001.

[12] J. Ye, G. Stevenson, and S. Dobson, “A top-level ontology for smart environments,” *Pervasive and Mobile Computing*, vol. 7, no. 3, pp. 359 – 378, 2011, knowledge-Driven Activity Recognition in Intelligent Environments.

[13] F. van Harmelen and D. L. McGuinness, “OWL Web Ontology Language overview,” W3C, Tech. Rep., 2004.

[14] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, “SWRL: A semantic web rule language combining OWL and RuleML, W3C member submission 21 may 2004,” W3C, Tech. Rep., 2004.

[15] G. Stevenson, S. Knox, S. Dobson, and P. Nixon, “Ontonym: a collection of upper ontologies for developing pervasive systems,” in *Proceedings of the 1st Workshop on Context, Information and Ontologies*, ser. CIAO ’09. New York, NY, USA: ACM, 2009, pp. 9:1–9:8.

[16] W. T. Adrian, S. Bobek, G. J. Nalepa, K. Kaczor, and K. Kluza, “How to reason by HeaRT in a semantic knowledge-based wiki,” in *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2011*, Boca Raton, Florida, USA, November 2011, pp. 438–441.

[17] A. Ligeza and G. J. Nalepa, “A study of methodological issues in design and development of rule-based systems: proposal of a new approach,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 117–137, 2011. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/widm.11/pdf

[18] G. J. Nalepa, “Collective knowledge engineering with semantic wikis,” *Journal of Universal Computer Science*, vol. 16, no. 7, pp. 1006–1023, 2010. [Online]. Available: http://www.jucs.org/jucs_16_7/collective_knowledge_engineering_with

Automated Construction of Data Integration Solutions for Tool Chains

Matthias Biehl, Jiarui Hong, Frederic Loiret
Embedded Control Systems
Royal Institute of Technology
Stockholm, Sweden
{biehl,hong,floiret}@md.kth.se

***Abstract*—Modern software development relies increasingly on the orchestrated use of development tools in the form of seamless, automated tool chains. Tool chains are becoming complex software systems themselves, however, the efficient development of tool chains is a largely unsupported, manual engineering task. We propose both a domain specific modeling language for systematically specifying tool chains and generators for efficiently realizing the tool chain as software. Tool chain software consists of diverse components, such as service-oriented applications, models and model transformations, which we produce by different generative techniques. We study both the separate generative techniques and the dependencies between the generated artifacts to ensure that they can be integrated. We evaluate the approach both quantitatively and qualitatively, and show in a case study that the approach is practically applicable when building a tool chain for industrially relevant tools.**

***Keywords*-Domain Specific Modeling; Tool Integration; Prototyping; Higher-Order Model Transformation; Code Generation.**

I. Introduction

Since modern development relies more and more on sophisticated development tools, the integration of these tools becomes an important issue. The development tools may be modeling tools, simulation tools, verification tools etc., which are typically not designed with ease of integration in mind [7]. The integration of development tools thus requires a sizable engineering effort, including the extraction of data from the integrated tools, adherence to integration standards and mapping of the data between the formats of different tools. Manually implementing the tool chain is time-consuming and error-prone.

Realizing a model-based tool chain as software involves writing source code for two distinct parts. The first part deals with setting up the infrastructure for model-based tool integration, such as transformation engines or tracing tools. The second part realizes the actual exchange of tool data, such as the extraction of data from the tool and its transformation into a different format or representation.

Several approaches for tool integration are mentioned in the literature, for example model-based tool integration [1], weaving-based tool integration [6] or ontology-based tool integration [12]. Model-based tool integration assumes that data of different tools is available in the form of models, which adhere to metamodels, and model transformations, which describe the data conversion [1]. The focus of model-based tool integration is thus on describing tool data and its relations (expressed as models, metamodels and transformations).

The entire tool chain, however, is only an implicit concept. As a result of this lack of an overall picture of the tool chain, existing approaches do not uncover the potential for supporting the development of complete tool chains. Existing approaches typically assume that the source code for providing tool data and functionality is implemented manually. If the implementation needs to follow integration standards, such as OSLC (Open Services for Lifecycle Collaboration) [24], it can be tedious to implement this code. Existing approaches also assume that the data conversion rules, which are necessary for data exchange between tools, are implemented manually, e.g., in the form of weaving models or model transformations. Support to synthesize the conversion rules is missing.

The goal of this paper is to systematize and partly automate the development of tool chains. The central question we address is thus: *To what extent can the development of tool chains be automated through generative techniques?* Our approach is model-based, but differs from previous model based approaches, as not only the tool data is modeled, but also the architecture of the complete tool chain. For this purpose we use a domain specific modeling language for tool chains, which structures the tool chain into ToolAdapter components and connectors. A generative approach uses the structured information to synthesize implementations for the components, the connectors and the infrastructure of the tool chain. In previous work, we have described the modeling language [4], code generation of the components [5] and infrastructure [4]. The contribution of this paper is the automated synthesis of the data conversion rules in the connectors and the integration of the various generated parts into a complete, cohesive tool chain.

II. Approach

From our experience of developing tool chains in an industrial context, tool chains are often developed with an iterative prototyping approach. While the general goal for the tool chain might be clear to all stakeholders, the exact details of the execution and conversion in the tool chain might not be clear and different design options are explored with prototype implementations. The challenge lies in the large effort of creating one or potentially many prototype implementations of the tool chain. It involves creating a software adaptation layer – known as ToolAdapter – for each integrated tool and model transformations that convert the data between the proprietary data format of the tools. The method and automated techniques of the proposed approach aim to reduce the necessary development effort for tool chains.

In our approach, the user models the tool chain using abstractions from the domain of tool integration. A prototype implementation of a tool chain is produced by generating both source code and transformations from the model. In the following sections, we describe how we achieve the specification and the automated synthesis of the executable prototype. We divide the approach into several steps, as illustrated in Figure 1:

- Step 1 - Specification of a Tailored Tool Chain: The essential design decisions are described by modeling the tool chain in the Tool Integration Language (TIL) [4]. More details on modeling with TIL are presented in Section III.
- Step 2 - Synthesis of ToolAdapters: The synthesis automatically generates code based on the specification of the ToolAdapters. It is presented in Section IV.
- Step 3 - Synthesis of Channels: The synthesis automatically generates code and transformation rules, based on the specification of the Channels. It is presented in Section V.
- Step 4 - Integration of Generator Results: The generated parts need to be integrated into a tool chain, as presented in Section VI.
- Step 5 - From Prototype to Production Software: The prototype tool chain can be refined into a production tool chain, as presented in Section VII.

Steps 1 - 4 are typically iterated multiple times, until a satisfactory prototype is identified. Since only step 1 is manual and steps 2, 3 and 4 are automated, this is a viable iterative development approach for tool chains. We evaluate the approach qualitatively by a running example, which is embedded in Sections III-VII. We evaluate the approach quantitatively in Section VIII. In Section IX, we show the relation of this approach to other work in the field; in Section X, we mention future work and conclude.

A. Introduction to the Running Example

We illustrate all steps of the approach by stepwise constructing a tool chain, which serves as a running example. The intended use of the tool chain presented as running example is in the early design phase of automotive embedded system development [2]. An engineer creates a UML-conform model with behavioral and fault propagation

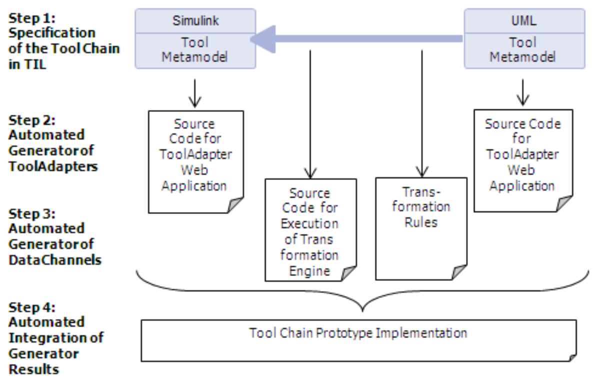

Step 1: Specification of the Tool Chain in TIL
Step 2: Automated Generator of ToolAdapters
Step 3: Automated Generator of DataChannels
Step 4: Automated Integration of Generator Results
Simulink
Tool Metamodel
UML
Tool Metamodel
Source Code for ToolAdapter Web Application
Source Code for Execution of Trans formation Engine
Trans- formation Rules
Source Code for ToolAdapter Web Application
Tool Chain Prototype Implementation

Figure 1. Overview of steps 1-4 of this approach

models using the GUI of the development tool and commits the model to the repository. Every time a new version of the UML model is committed, the tool chain executes a transformation of the UML model to the input format of the fault tree analysis tool HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies) [25] and executes HiP-HOPS. Another model transformation creates a MATLAB/Simulink model that mirrors the structure of the UML model. When the results of the fault tree analysis are satisfactory, i.e., there are no single points of failure, the engineer manually extends this Simulink model to perform simulations.

III. Step 1 - Specification of a Tailored Tool Chain

The prototyping process starts by specifying the big picture of the tool chain. We use TIL, a declarative, domain specific modeling language for tool chains. TIL models are concise, and expresses domain concepts, so users can relate to it. TIL allows us not only to describe a tool chain graphically and with well-defined semantics, but also to analyze it and generate code from it.

Here, we can only give a short overview of the language, for more detailed description of syntax and semantics, we refer to [4]. In the following, we introduce the language concepts and their concrete graphical syntax (compare ① .. ⑦ in Figure 2). TIL is a component-based language and consists of Components (ToolAdapter, Repository, Sequencer, User) and Connectors (ControlChannel, TraceChannel, DataChannel).

- A ToolAdapter ① exposes the data and functionality of a tool in a common technical format, making the data and functionality accessible for other ToolAdapters within the tool chain. A ToolAdapter is specified by means of a ToolAdapter metamodel. It describes the selection of data and functionality of the tool, which is exposed to the tool chain (see Figure 4, for an example). Creating the metamodel requires some engineering

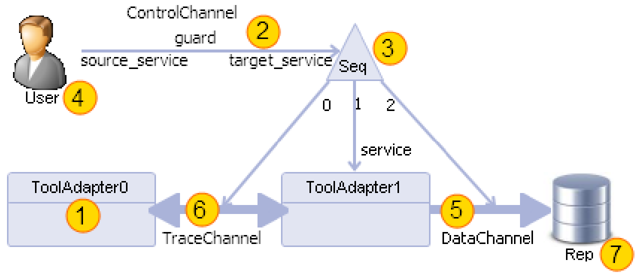


Figure 2. A simple TIL model illustrating the graphical syntax of the language concepts

effort, since each tool has its own metamodel, there is no common metamodel.

- A ControlChannel ② describes the control-flow between two Components by specifying a triggering event in the source, a called service in the target and a guard for conditional execution.
- A TraceChannels ⑥ connects two ToolAdapters and describes the possibility of creating traces between elements of certain data types of the ToolAdapter metamodels.
- A DataChannels ⑤ connects two ToolAdapters and describes the data-flow between them. The DataChannel preserves the semantics of the data. Since the ToolAdapter delivers all exposed data in a common technical format, only the data structure needs to be adapted. The data of the source ToolAdapter needs to be transformed into the structure expected by the target ToolAdapter. A model transformation can either be manually specified or it can be automatically synthesized, as described in Section V.
- A Sequencer ③ describes sequential control-flow; it executes a sequence of services in a specified order. The sequencer is used in combination with ControlChannels: it is activated by a ControlChannel and each of the sequentially called services is connected via a ControlChannel.
- A User ④ is a representative for a real tool chain user. This concept is used to describe the possible interactions of the real users with the tool chain. Outgoing ControlChannels from the User denote services invoked by the user, incoming ControlChannels to a User denote a notification sent to the user.
- A Repository ⑦ is a specific type of ToolAdapter that provides storage and version management of tool data.

A. *Running Example: Step 1*

We specify the previously introduced tool chain in TIL, resulting in the model displayed in Figure 3. An engineer, depicted by the user symbol, develops a new function of an embedded system as a UML component model. The engineer checks the model into the Subversion Repository, depicted by a ControlChannel, which activates the DataChannel uml2repository. Automatically, the model will be analyzed by a safety analysis tool to detect single points of failure in the embedded system. This is depicted by the triangular shape for the Sequencer `Seq0`, which is activated by a ControlChannel, whenever new UML models are checked into the repository. The Sequencer `Seq0` first triggers the DataChannel `uml2safety` to transfer the UML model to the safety analysis tool involving a model transformation. The Sequencer `Seq0` then calls the function to analyze single points of failure in the safety analysis tool. If no single points of failure have been found, which is expressed as a guard condition on the ControlChannel, a simulation of the behavior of the new model is started in Simulink. This is realized by another set of ControlChannels and the Sequencer `Seq1`. Finally, the engineer receives an email notification about the simulation results.

The TIL model presented in Figure 3 is linked to several ToolAdapter metamodels, which are illustrated in Figure 4. In addition, model instances of these metamodels are linked to each ToolAdapter. They serve as test data for the prototype implementation. Each of the metamodels describes the subset of the data of the tool that is exposed by the ToolAdapter towards the tool chain. The metamodel for the MATLAB/Simulink tool (A) describes a basic block diagram. The metamodel for the UML tool (B) comprises the elements of a basic UML component diagram. The metamodel for the safety analysis tool HiP-HOPS (C) is organized into systems and subsystems.

IV. Step 2 - Synthesis of ToolAdapters

ToolAdapters are realized as software components that have a web-based, RESTful architecture. This provides platform independence and allows for a distributed tool chain, where tools may reside on different network nodes. The input of the generator is the ToolAdapter metamodel and a model with test data, which conforms to the metamodel. The output is a Java source code and configuration files for the service infrastructure, such as a web server listening for requests. The generated Java source code provides a skeleton of the tool adapter implementation, including an implementation that operates on static test data and serves it conform to the format and protocols of the industrial initiative OSLC. The generated ToolAdapter is thus functional, but does not yet connect to the tool instance via APIs. The connection to the tools has to be added manually in step 5.

Since the details of generating service-oriented tool adapters are not the focus of this work, we refer to paper [5] for more details and examples.

V. Step 3 - Synthesis of Channels

The Channels between ToolAdapters can describe control-flow or data-flow. Control-flow is expressed by ControlChan-

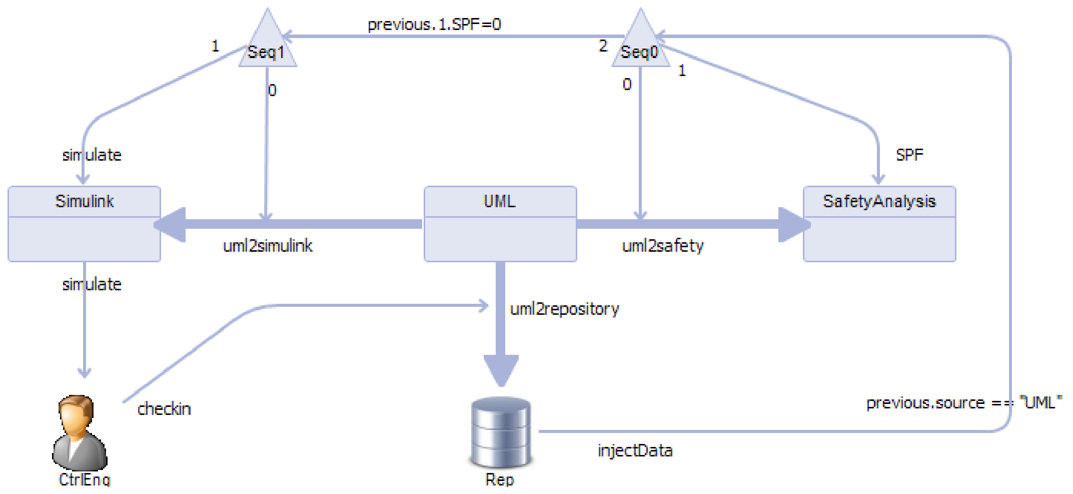


Figure 3. Specification of the tool chain as a TIL model

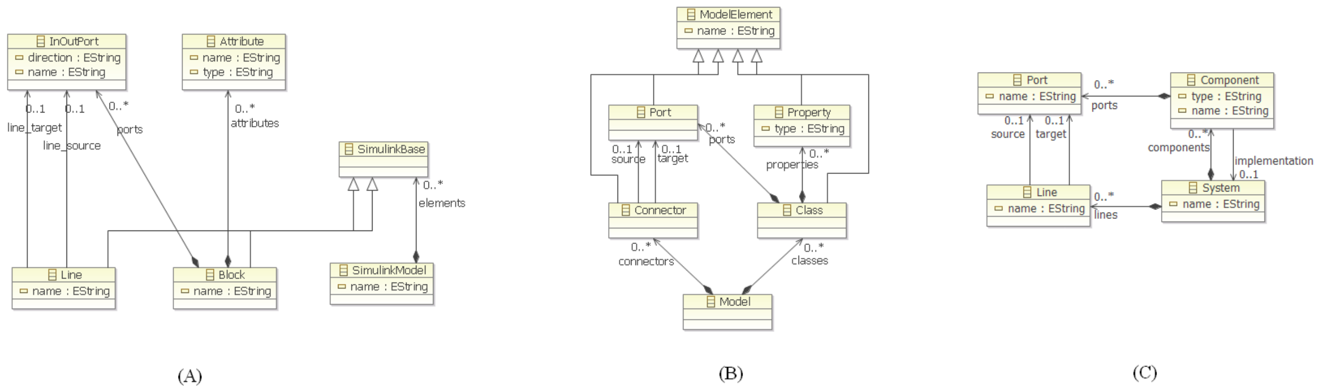


Figure 4. ToolAdapter metamodels for the Simulink tool (A), UML tool (B) and safety analysis tool (C)

nels, which are straightforward to synthesize as remote service calls to other ToolAdapters. Data-flow can be expressed by TraceChannels or DataChannels. TraceChannels provide the infrastructure for creating traces at runtime of the tool chain and this infrastructure is quite straightforward to generate.

DataChannels denote the transfer of data from a source ToolAdapter to a target ToolAdapter. The tool data is served by the ToolAdapter in the form of a model that conforms to the ToolAdapter metamodel. If the metamodels of source and target ToolAdapters are the same, the data can be simply copied between the ToolAdapters. In the more common case that the metamodels are different, the data needs to be transformed before it can be accepted by the target ToolAdapter. For this purpose, TIL offers the possibility to link a model transformation to each DataChannel. This transformation is a part of the implementation of the DataChannel and can be either manually specified or synthesized. If a transformation is required (due to different source and target ToolAdapter metamodels), but none is specified, a prototype transformation can be automatically synthesized. Another part of the implementation of DataChannels is the infrastructure for executing the transformation and transferring the data to another ToolAdapter.

The synthesis algorithm generates source code for the ControlChannels as well as source code for the infrastructure of DataChannels. This infrastructure accomplishes the following tasks at runtime of the tool chain: it gets the source model from the source ToolAdapter and provides it together with the transformation to the transformation engine. The target model, which is produced by the transformation engine, is sent to the target ToolAdapter.

In the following, we take a closer look at the automated generation of an appropriate prototype transformation. A transformation is appropriate if its source and target metamodel is identical to the metamodels of source and target ToolAdapters of the DataChannel and if it is semantics preserving. The TIL model contains some information for synthesizing the transformation, such as its execution direction and both its source and target metamodels. This information is not sufficient for an algorithmic approach, but a heuristic approach for prototyping model transformations can be realized. The intention is to use an automated approach to quickly create a first model transformation. In the following sections we explain each step of the generator for prototype transformations in more detail.

A. Step 3.1 - Finding Correspondences

We assume a certain level of similarity between the source and target metamodels; so we can find correspondences between them based on structural and naming similarities. To find correspondences, we use a matching algorithm that is based on the similarity flooding algorithm [20] and the Levensthein distance [16]. The similarity flooding algorithm is used to detect structural similarities, the Levensthein distance is used to identify naming similarities. The matching algorithm produces a matching table, consisting of a number of correspondences between metaclasses, metaattributes and metareferences of source and target metamodels. In more formal terms, the matching algorithm $\sigma(m_s, m_t) = \mu$ produces a matching table μ for a given tuple of source metamodel m_s and target metamodel m_t.

B. Step 3.2 - Refining the Matching Table into a Matching Model

To ensure that a valid target model can be produced by the synthesized transformation, we automatically refine the matching table μ into a matching model ν by adding information about the containment hierarchy of the target metamodel with the refinement function ρ, which is defined as $\rho(\mu, m_t) = \nu$. This refinement is necessary, so the synthesized transformation can produce target models with an adequate containment structure, which is specified in the target metamodel m_t. The containment hierarchy of a metamodel is a partial order over all metaclasses in the metamodel that have a direct parent-child relationship.

The metamodel of the matching model is depicted in Figure 5. It consists of a number of ordered matchings. A matching describes a correspondence and consists of a description of the source and target elements, a number of related matchings and a type. The type of the matching is based on the role that the target element of the matching takes in the target metamodel. We differentiate five types of matchings:

- Top: A top matching has a target element that is the root element of the containment hierarchy of the target metamodel, i.e., the element that is not contained anywhere else. A top matching specifies the names of classes and usually has a number of containment matchings.
- Containment: A containment matching represents a reference between two metaclasses in the target metamodel, where one class is the parent and the contained class is the child. A containment matching specifies the names of the reference and metaclasses.
- Reference: A reference matching represents a link between two arbitrary metaclasses in the target metamodel. A reference matching specifies the names of the reference and metaclasses.
- Class: In a class matching, the target element is a metaclass. A class matching specifies the names of the metaclasses and usually has a number of related matchings, which are of type containment, reference or attribute.
- Attribute: In an attribute matching, the target element is a metaattribute of a metaclass. An attribute matching specifies the names of the metaattribute and metaclass.

The automated refinement ρ adds a containment matching for a target metaclass if necessary and ensures that all target metaclasses are properly contained. It also checks that no target element is produced by more than one rule. Note, that this classification depends on the target element only, since the target element needs to have proper containment hierarchy to be produced.

C. Step 3.3 - Synthesis of Transformation Rules from the Matching Model

The model transformation τ_1 is automatically synthesized based on the matching model ν, which is produced in the previous step and is executed at runtime of the tool chain for the exchange of tool data. The synthesis of τ_1 is performed by a second model transformation τ_2, which is a higher-order model transformation, defined as $\tau_2(\nu) = \tau_1$. The transformation τ_2 produces τ_1 and is executed at design time of the tool chain. The transformation $\tau_1(n_s) = n_t$ maps the model n_s to the model n_t, where n_s corresponds to the source metamodel m_s, and n_t corresponds to the target metamodel m_t. The synthesized model transformation τ_1 is a model-to-model transformation, implemented with OMG QVT-R [23]. The synthesizing model transformation τ_2 is a model-to-text transformation, implemented with OMG MTL [22].

For each matching in the matching model ν, the transformation τ_2 produces one or several QVT relations. For each type of matching a different template for the relations is used. The template is instantiated with the values from the current matching. All customized QVT relations for all matchings together form the synthesized transformation τ_1. See listing 1 for an example.

D. Running Example: Step 3

For the DataChannels simulink2uml and simulink2hiphops, this transformation is synthesized by the generator. The metamodels for UML, Simulink and HiP-HOPS show a certain degree of structural and naming similarity, as they represent a hierarchical composition of

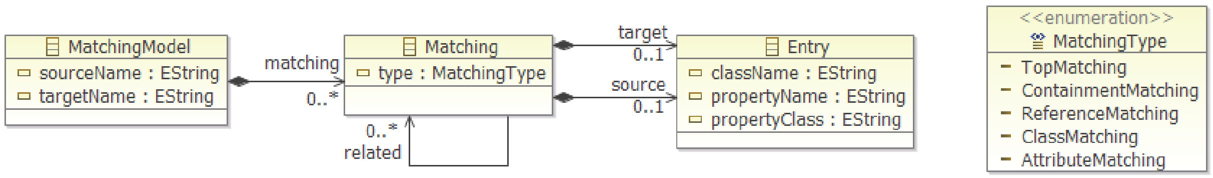


Figure 5. Metamodel of the matching model

components and components are linked by connectors via ports.

Table I
MATCHING TABLE FOR UML AND SIMULINK

UML Metaclass	Simulink Metaclass
Port	InOutPort
EString	EString
Class	Block
Property	Attribute
Connector	Line
Class.properties	Block.attributes
Property.type	Attribute.type
Class.ports	Block.ports
Connector.source	Line.line_source
Connector.target	Line.line_target

Table II
MATCHING TABLE FOR UML AND HIP-HOPS

Simulink Metaclass	HiP-HOPS Metaclass
Port	Port
Connector	Line
EString	EString
Class	Component
UMLModel	System
Property.type	Port.name
UMLModel.connectors	System.lines
Connector.target	Line.target
Connector.source	Line.source

As a first step in the automated synthesis of the transformation, a matching table is created. The automated metamodel matching algorithm is applied on the UML and the Simulink metamodels, yielding the matching table I. Applying the algorithm on the UML and the HiP-HOPS metamodels yields matching table II. Since the matching algorithm in step 3.1 is a heuristics, the automatically created matching table needs to be checked manually. All the mappings identified between Simulink and UML are correct, between UML and HiP-HOPS the matching algorithm correctly identified many mappings, however the heuristics introduced one error by mapping Property.type to Port.name. The subsequent refinement step 3.2 automatically corrects this error by replacing the matching with (Property.type,Component.type) through analysis of the containment hierarchy of the target metamodel, which is the Simulink metamodel.

Afterwards, the higher-order model transformation τ_2 of step 3.3 converts the matching model into a QVT-R transformation. A part of this synthesized transformation for the mapping between UML and Simulink is depicted in Listing 1. It shows the transformation code for different types of matchings, namely top, containment and class matchings. As the root element, the UML model is mapped to a Simulink model, contained UML Classes to Simulink Blocks, and the attributes of Classes to attributes of Blocks. In a similar manner – but not shown here due to space constraints – the value of each attribute is mapped, as well as Connectors and their attributes.

The computed mapping is 100% correct, but not complete, as not all elements are mapped. The missing mappings describe attributes, e.g., mapping the name attribute of the UML Class to the name attribute of Component or Block. Such missing attribute mappings concern only values and are relatively easy to add, since no other mappings depend on them. We evaluate the generated transformations with precision/recall metrics in Section VIII-B.

Listing 1. Synthesized QVT-R transformation from UML to Simulink

```
transformation uml22simulink2(source: uml2, target: simulink2) {

        --Top Matching
        top relation r_Model {
          checkonly domain source p : uml2::Model {
          };
          enforce domain target s : simulink2::SimulinkModel {
          };
          where{
                r_Model_classes(p,s);
                r_Model_connectors(p,s);
          }
        }

        --Containment Matching
        relation r_Model_classes {
          checkonly domain source p : uml2::Model {
                classes = co: uml2::Class{
                }
          };
          enforce domain target s : simulink2::SimulinkModel {
                elements = sb: simulink2::Block{
                }
          };
          where{
                r_Class(co,sb);
          }
        }

        --Class Matching
        relation r_Class {
          checkonly domain source co: uml2::Class{
          };
          enforce domain target sb: simulink2::Block{
          };
          where{
                r_Class_name(co,sb);
                r_Class_ports(co,sb);
                r_Class_properties(co,sb);
          }
        }
  [..]
}
```

VI. Step 4 - Integration of Generator Results

The generators in steps 2 and 3 produce ToolAdapters and DataChannels using different generative techniques. The generator for ToolAdapters uses code generation; the generator for DataChannels applies a matching algorithm to produce model transformations, which realize the conversion of tool data. Both generators use the ToolAdapter metamodels that are linked to the TIL model, but they use the metamodels in a different way.

The generator for ToolAdapters uses the tool metamodel as specification of the data managed by the ToolAdapter. The tool data is accessible as a model through the generated ToolAdapter and conforms to the tool metamodel.

The generator for transformations uses the metamodels of both ToolAdapters it connects to. Data-flow connections between ToolAdapters need to translate the tool data. The rules for the translation can be determined at designtime, since it is independent of the actual data, and only depends on the tool metamodels. As we have generated the ToolAdapters we know that they provide tool data that conforms to the tool metamodels.

The ToolAdapter metamodels are used as interface between the different generators to ensure compatibility between the generated artifacts.

VII. Step 5 - From Prototype to Production Software

The proposed approach promotes the iterative development of tool chains, where steps 1 - 4 can be repeated frequently to explore different what-if scenarios. This is supported by the generative approach, which produces executable source code and transformations automatically with only a small effort from the tool chain designer. The completely automatically generated source code works on test data and the automatically generated transformation might not be complete. While this level of accuracy might be sufficient for prototyping different what-if scenarios for tool chains, it needs to be improved for production software.

To create production software, the generated source code and transformation rules need to be manually adapted. The generated source code of the ToolAdapter needs to be extended to interact with the API of the integrated development tool, to extract and inject the data from the tool and to forward service calls to it. The generated model transformation needs to be refined, mainly by adding new transformation rules and less frequently by changing the generated transformation rules.

VIII. Evaluation

In this section, we intend to quantify to what extent a tool chain can be generated with the proposed approach. We separately evaluate the generator for tool adapters and for transformations.

A. Generator for ToolAdapters

The generator for ToolAdapters produces code skeletons and a complete prototype implementation, which serves test data. This prototype implementation needs to be manually replaced with code that serves the actual tool data using the API of the tool. The size of the code that needs to be manually added depends on the tool.

To show the effectiveness of the generator, it is not sufficient to compare the LOC of generated code with the manually added code. Instead, we create a baseline implementation completely manually. We now have two code bases realizing the same functionality. To quantify the generated and manually added code, we measure lines of code. This measurement has been criticized as a general measurement of software size for complete software, but here we apply it to fragments of code. We measure the lines of generated vs. manually implemented code for all ToolAdapters in the case study and present the measurements in Table III.

Table III
LOC of the ToolAdapters: generated and manually implemented code with the TIL approach vs. completely manually implemented baseline

ToolAdapter	TIL generated	TIL manually added	manual baseline
UML	1409	59	1313
Simulink	2030	1118	3077
Safety	3833	317	2359
Sum	7272	1494	6749
Percentage		22%	100%

The size of the generated code is not a significant indicator for the quality of the generated code. This is why we study the comparison of the manually added LOC (in step 5) with the LOC of the manual baseline. Both codes are manually created, realize the same functionality and their sizes can thus indirectly give clues about the quality of the generated code. On average, only 22% of the source code from the manual baseline needed to be implemented manually with the TIL approach.

B. Generator for Transformations

The generator for transformations introduced in Section V is a heuristics. It is the nature of heuristics to approximate the optimal solution and it cannot be guaranteed that the calculated result is the optimal solution. It is thus important to measure the quality of the results. In the following, we measure the quality of the results of applying our matching algorithm on a number of tool metamodels. The matching algorithm is based on two simplifying assumptions: (i) The transformation, which is part of the DataChannel between two ToolAdapters is intended to be semantics preserving. (ii) A semantics preserving transformation maps elements, which are similar regarding structure or naming. Assumption

Table IV
RELATIONSHIP BETWEEN SEMANTICS PRESERVATION AND STRUCTURAL/NAMING SIMILARITY

		semantics preservation	
		yes	no
structural/naming	yes	1	2
similarity	no	3	4

(i) is part of the semantics of the DataChannel in TIL (see Section III). In the following, we will evaluate assumption (ii). We analyze the relationship between semantics preservation and structural/naming similarity in Table IV and distinguish four situations.

Since assumption (ii) correlates semantics preservation with structural/naming similarity, the algorithm only distinguishes between situations 1 and 4 in Table IV. Situations 2 and 3 are in the “blind spot” of the algorithm, as semantics preservation may not be correlated with structural/naming similarity. The impact of situation 2 is measured by the precision metric, the impact of situation 3 is measured by the recall metric, which are defined as follows.

$$Precision = \frac{|\{correctmatches\} \cap \{foundmatches\}|}{|\{foundmatches\}|} \quad (1)$$

$$Recall = \frac{|\{correctmatches\} \cap \{foundmatches\}|}{|\{correctmatches\}|} \quad (2)$$

where $correctmatches$ is defined as the correct, semantics-preserving mapping, and $foundmatches$ is the mapping that was identified by the matching algorithm. We use the precision/recall measure and present statistics of the number of false positives and false negatives in the mappings.

We measure the quality of the calculated mappings of all six possible combinations between the three metamodels presented in Figure 4. The resulting precision/recall measurements are displayed in Table V. On average, the synthesis method returns mappings that have a high precision (93%), but only an average recall (56%).

Table V
PRECISION/RECALL METRIC FOR THE COMPUTED MAPPING OF UML, SIMULINK AND HIP-HOPS

Source	Target	Precision	Recall
UML	Simulink	1	0.56
UML	HiP-HOPS	0.89	0.53
Simulink	UML	1	0.67
Simulink	HiP-HOPS	0.9	0.6
HiP-HOPS	UML	0.89	0.53
HiP-HOPS	Simulink	0.9	0.5
Average		0.93	0.56

In situation 2, no mapping should be found since there is no semantic equivalent, but the algorithm finds a mapping due to structural similarity; this would result in a low precision. The measurements in Table V show a high precision metric. This means that the generated mappings are correct and only need to be changed seldomly. The mappings that need to be changed have a stable skeleton for manually added mappings.

In situation 3, there are semantically equivalent metamodels, for which no structural or naming similarity can be detected. If it is not possible to deduce clues about the semantic equivalence from the structural features of the metamodels, the automated algorithm does not have sufficient data to make mapping decisions. In this situation, either additional user data would need to be provided as input, e.g., via annotations, or the missing mappings need to be manually added after the algorithm is finished. Situation 3 is captured by the recall metric. The average recall metric in Table V is largely due to missing attribute mappings. Such attribute mappings concern only one value and are relatively easy to add manually, since no other mappings depend on them. An example for the transformation between UML and Simulink is the mapping (Class.name,Block.name).

Due to its high precision (93%) and average recall (56%) characteristic, the matching algorithm can be classified as a conservative method. The algorithm rather does not include a mapping into the result than produce a wrong mapping. The mappings that are included in the result are almost all correct, maximally one of the mappings is incorrect. The mappings that are not found automatically by the matching algorithm can be manually added to the result.

IX. RELATED WORK

The contribution of this paper is in the intersection of several fields, namely model-based tool integration, metamodel matching and rapid prototyping. Related work can be found in each of these fields. We list the approaches by fields and point out approaches that are in the intersection of two or more fields.

A. Tool Integration

Early work on tool integration focuses on identifying the scope of tool integration in form of aspects [29] and patterns [14]. A number of integration frameworks have been defined to support building tool chains, such as the one from Vanderbilt [13] and jETI [19]. Model-based integration frameworks focus on data integration, the other integration aspects (such as control, process, platform and presentation) defined by Wasserman [29] are excluded or a secondary issue. Examples are MOFLON [1] or ModelCVS [12]. These related approaches use metamodeling for describing the tool data. However, these approaches provide neither concepts to model a complete tool chain nor concepts to describe the architecture of the tool chain. The related approaches assume that tool data is available in the form of models and that the tool adapters are implemented manually. Only the MOFLON approach mentions code generation for tool adapters. The related approaches also use model transformations to translate between the metamodels of different tools, but the transformation usually has to be specified manually.

Tool integration platforms, such as ModelBus [11] or Jazz [9], mainly provide support for executing the tool chain, or generic building blocks, so constructing tailored, user-defined tool chains requires a lot of work. We automate the construction of such tailored tool chains.

B. Metamodel Matching

Matching metadata on data structures has been studied in the field of databases as schema matching [20], [26], [27]. These matching algorithms have been adopted in the modeling community, where metamodels are matched instead of schema definitions. Algorithms based on similarity flooding and naming similarity are described [6], [8]. Different metamodel matching algorithms are compared in [15] and formalized into a DSL for metamodel matching [10]. Model weaving approaches [6] can leverage metamodel matching to create weaving models that express the correspondence.

Del Fabro shows how metamodel matching can be applied for data migration between two bugtracking tools [6]. The approach assumes that the tool data is already available in a model format and focuses on the use of metamodel matching for weaving models. We use metamodel matching in an integration scenario and focus on a comprehensive approach for the creation of a complete tool chain.

C. Prototyping

Prototyping approaches focus on the early synthesis of an executable system from a high-level specification. Bernstein stresses the importance of prototyping [3] and lists advantages of the approach, among them he sees prototyping as a vehicle to better understanding the environment and the requirements, to validate requirements with the user and to study the dynamics of a system. We distinguish between throwaway and evolutionary prototyping [17]. In throwaway prototyping, the prototype is built to learn a specific thing and is discarded before a completely new prototype is built. In evolutionary prototyping, one prototype is refined over several iterations. The technology proposed in this paper can be used for either prototyping approach.

Many prototyping systems employ a prototyping language in combination with code generation techniques. These approaches are usually specialized to certain domains, such as CAPS and DCAPS [18] for embedded systems, information systems and user interfaces [30], component based systems [28] or data mining systems [21].

X. Future Work and Conclusion

The creation of tool chains is usually regarded as a completely manual implementation task. The presented approach shows that the tool chain implementation for a prototype can be automatically created with generative techniques. Different generative techniques need to be combined to produce the heterogeneous parts of the tool chain: code generation for ToolAdapters and a heuristic matching algorithm for transformations. The generated code for the ToolAdapters ensures compliance with standards and serves test data for prototyping.

This code is also the basis for production software as it provides a skeleton that needs to be refined with manually written code that interacts with the API of the integrated tool. In our case study the generated code for ToolAdapters makes up 78% of the total production software. The generated transformation code for the DataChannels provides a precise mapping for the data elements (93% precision), but does not cover all data elements (56% recall). Due to the conservative characteristic of the approach (high precision, average recall), the generated mapping can be be extended into a comprehensive mapping. The generated artifacts can serve as a starting point for manual extensions and refinements of the generated tool chain implementation.

The proposed approach for automated synthesis of both source code and transformations makes it possible to systematically and rapidly create an executable prototype of a tool chain. This allows the user to test and iteratively modify the tool chain prototype, before investing time to extend the prototype into the final production software.

An important next step is a further assessment of the practical applicability of this approach. We will apply our approach in additional case studies, which cover a broader set of development tools. This will allow us to further narrow down the conditions, in which the approach can achieve the best mapping, measured with the precision/recall metric. In addition, we will examine if the algorithm can also be applied to support the evolution of a tool chain, when tool A in the tool chain is exchanged against a similar tool B. The here presented algorithm might be applicable for realizing the migration of the data from tool A to tool B.

Acknowledgement

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement 100203.

References

[1] C. Amelunxen, F. Klar, A. Königs, T. Rötschke, and A. Schürr. Metamodel-based tool integration with MOFLON. In *ICSE '08*, pp. 807–810, 2008.

[2] E. Armengaud, M. Biehl, Q. Bourrouilh, M. Breunig, S. Farfeleder, C. Hein, M. Oertel, A. Wallner, and M.s Zoier. Integrated tool chain for improving traceability during the development of automotive systems. In *ERTS2 2012 — Embedded Real Time Software and Systems*, pp. 30–46, 2012.

[3] L. Bernstein. Importance of software prototyping. *Journal of Systems Integration*, 6(1), pp. 9–14, 1996.

[4] M. Biehl, J. El-Khoury, F. Loiret, and M. Törngren, "On the Modeling and Generation of Service-Oriented Tool Chains," *Journal of Software and Systems Modeling*, vol. 0275, 2012.

[5] M. Biehl, J. El-Khoury, and M. Törngren. High-Level Specification and Code Generation for Service-Oriented Tool Adapters. In *Proceedings of the International Conference on Computational Science (ICCSA2012)*, pp. 35–42, Jun. 2012 [Online]. Available: http://dx.doi.org/10.1007/s10270-012-0275-7

[6] M. Del Fabro, J. Bézivin, and P. Valduriez. Model-Driven Tool Interoperability: An Application in Bug Tracking. In*On the Move to Meaningful Internet Systems 2006*, LNCS, vol. 4275, pp. 863–881, 2006.

[7] J. El-khoury, O. Redell, and M. Törngren. A Tool Integration Platform for Multi-Disciplinary Development. In *31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 442–450, 2005.

[8] J.-R. Falleri, M. Huchard, M. Lafourcade, and C. Nebut. Metamodel Matching for Automatic Model Transformation Generation. In *Model Driven Engineering Languages and Systems*, LNCS, vol. 5301, pp. 326–340, 2008.

[9] R. Frost. Jazz and the Eclipse way of collaboration. *IEEE Software*, vol. 24, no. 6, pp. 114–117, 2007.

[10] K. Garcés, F. Jouault, Pi. Cointe, and J. Bézivin. A Domain Specific Language for Expressing Model Matching. In *Proceedings of the 5ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM09)*, pp. 30–45, 2009.

[11] C. Hein, T. Ritter, and M. Wagner. Model-Driven Tool Integration with ModelBus. In *Workshop Future Trends of Model-Driven Development*, pp. 1–12, 2009.

[12] E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, and M. Wimmer. On Models and Ontologies - A Layered Approach for Model-based Tool Integration. In *MOD2006*, pp. 11–27, 2006.

[13] G. Karsai and J. Gray. Component generation technology for semantic tool integration. vol. 4, pp. 491–499, 2000.

[14] G. Karsai, A. Lang, and S. Neema. Design patterns for open tool integration. *Software and Systems Modeling*, vol. 4, no. 2, pp. 157–170, 2005.

[15] L. Lafi, S. Issam, S. Hammoudi, and J. Feki. Comparison of two metamodel matching techniques. In *4th Workshop on Model-Driven Tool & Process Integration (MDTPI2011)*, pp. 54–65, 2011.

[16] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Doklady Akademii Nauk SSSR*, 163(4), pp. 845–848, 1965.

[17] V. Luqi, V. Berzins, M. Shing, R. Riehle, and J. Nogueira. Evolutionary Computer Aided Prototyping System (CAPS). In *Technology of Object-Oriented Languages and Systems*, pp. 363, 2000.

[18] V. Luqi, J. Berzins, J. Ge, M. Shing, M. Auguston, B. Bryant, and B. Kin. DCAPS-architecture for distributed computer aided prototyping system. In *Rapid System Prototyping, 12th International Workshop on*, pp. 103–108, 2001.

[19] T. Margaria, R. Nagel, and B. Steffen. jETI: A Tool for Remote Tool Integration Tools and Algorithms for the Construction and Analysis of Systems. In *TACAS*, LNCS, vol. 3440, pp. 557–562, 2005.

[20] S. Melnik, H. Garcia-molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm, 2002.

[21] I. Mierswa, M. Scholz, R. Klinkenberg, M. Wurst, and T. Euler. YALE: Rapid Prototyping for Complex Data Mining Tasks. In *In Proceedings of the 12th ACM SIGKDD*, pp. 935–940, 2006.

[22] OMG. MOF Model to Text Language (MTL). Technical report, OMG, 2008.

[23] OMG. MOF 2.0 Query / View / Transformation. Technical report, OMG, 2009.

[24] OSLC Core Specification Workgroup. OSLC core specification version 2.0. Technical report, Open Services for Lifecycle Collaboration, 2010.

[25] Y. Papadopoulos and J. McDermid. Hierarchically Performed Hazard Origin and Propagation Studies. In *SAFECOMP*, LNCS, vol. 1698, pages 139–152, 1999.

[26] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[27] P. Shvaiko, J. Euzenat. A Survey of Schema-Based Matching Approaches Journal on Data Semantics IV. In *Journal on Data Semantics IV*, LNCS vol. 3730, pp. 146–171, 2005.

[28] M. Tkachuk, A. Zemlyanoy, and R. Gamzayev. Towards Prototyping-Based Technology for Adaptive Software Development Information Systems and e-Business Technologies. In LNBIP, vol. 5 , pp. 508–518, 2008.

[29] Anthony I. Wasserman. Tool Integration in Software Engineering Environments. In *Software Engineering Environments, International Workshop on Environments Proceedings*, LNCS, pp. 137–149, 1989.

[30] W. Zhou. A Rapid Prototyping System for Distributed Information System Applications. *Journal of Systems and Software*, vol. 24, no. 1, pp. 3–29, 1994.

A Neurolinguistic Method for Identifying OSS Developers' Context-Specific Preferred Representational Systems

Methanias Colaço Júnior [a,b], Manoel Mendonça, Mario André de F. Farias
[a] Software Engineering Laboratory - LES
UFBA – Federal University of Bahia
Salvador/BA – Brasil
mjrse@hotmail.com, manoel.g.mendonca@gmail.com, mario-fa@infonet.com.br

Paulo Henrique, Daniela Corumba
[b] Competitive Intelligence Research and Practice Group - NUPIC
Information Systems Department - DSI
UFS - Federal University of Sergipe
Itababaiana/SE – Brasil
diretor.ph@gmail.com, danielacorumba@gmail.com

***Abstract* — Open Source Software (OSS) projects use mailing lists as the primary tool for collaboration and coordination. Mailing lists can be an important source for extracting behavioral patterns in the OSS development. A new approach for that is the use of NeuroLinguistic theory to determine what is the Preferred Representational cognitive System (PRS) of software engineers in that specific context. Different resources and cognitive channels are used by developers in order to achieve software understanding. An important question on this matter is: What types of representational systems are preferred by software engineers? This paper presents a psychometrically-based neurolinguistic method to identify the PRS of software developers. Experimental evaluation of the approach is carried out in an experiment to assess the Preferred Representational System of top developers at Apache server and Postgresql mailing lists. The results showed that the PRS scores of the top-committers clearly differ from the general population of the projects. Qualitative analysis also indicated that the PRS scores obtained are aligned with the top committer's profiles.**

***Keywords:* open source; text mining; neurolinguistic; mental imagery; experimental software engineering.**

I. Introduction

Developing and maintaining software systems is an arduous task. Large systems are complex and difficult to understand. In order to understand them, the developer must construct a mental model of the software works and structure [1].

In the comprehension process, developers use different resources and representational systems, such as: (1) examples, analogies, and code execution; (2) visual descriptions, diagrams and graphic models of the system; and (3) textual descriptions and source code analyses. Clearly, these resources are complementary and may be combined. However, is there a Context-Specific Preferred Representational System (PRS)? Or, is there a preferred order or combination of the representational systems in the understanding process?

Visual resources, like diagrams and non-conventional visualization metaphors, are being increasingly used in software engineering [2]. Studies show that the way software engineers process those resources impacts on the success of that processing [3], for both text [4] and diagrams [5]. However, we do not know complete studies that evaluate what types of representational systems are preferred by software engineers.

This is a broad question in the sense that different people may have different preferences in different contexts. Actually, the conception that different representational ways for cognition exist is well accepted in psychology area [6, 7, 8]. However, this statement has raised new theories such as Neuro-linguistic, which proposes the use of a PRS in specific contexts [9]. Internal mental processes such as problem solving, memory, and language consist of visual, auditory and kinesthetic representations that are engaged when people think about or engage in problems, tasks, or activities. Internal sensory representations are constantly being formed and activated. Whether making conversation, writing about a problem or reading a book, internal representations have an impact on a one's performance. The *Preferred Representational System is* the one that the person tends to use more than the others to create his/her internal representation.

Bandler and Grinder, Neuro-linguistic Programming (NLP) champions, claim that people say sensory-based words and phrases, or verbal cues, which indicate a context-specific visual, kinesthetic or auditory processing [9, 10]. These affirmations divide researchers of cognitive psychology area. Some have not found evidences for the declarations [11] hence they were criticized by the lack of concept understanding [12], meanwhile others have shown empirical scientific evidences and the need to expand researches [13, 14].

Thus, motivated by the psychometric text analysis presented by Rigby and Hassan [15], we developed a psychometrically-based neurolinguistic analysis tool. Our tool, NEUROMINER, uses Linguistic Inquiry and Word Count (LIWC) to classify developers' Preferred Representational Systems (PRS). NEUROMINER combines text mining and statistic analysis techniques with NLP sensory-based words in order to classify programmers.

NEUROMINER was used in an experiment which analyzed top committers and subjects of two large-scale OSS projects: Apache Server and Postgresql. The results showed that the measured PRS scores can indeed differentiate top committers from the general population. Qualitative analysis also indicated that the PRS scores obtained are aligned with the top committers profiles.

The rest of this paper is organized as follows. The next section introduces NLP. Section 3 reports text mining definitions used throughout the article. Section 4 describes our approach to LIWC and to mining software development mailing lists. In the Section 5, we detail an experimental validation of our approach. Section 6 discusses related works. Finally, Section 7 closes the paper with a discussion of future research.

II. Neuro-Linguistic Programming

A. *History and Some Concepts*

Neuro-Linguistic Programming (NLP), created in the 70's, consists of a set of techniques in which the neurological processes, behavioral patterns and a person's language are used and organized to achieve better communication and personal development. The term NLP is broadly adopted in education, management and training fields. However, although evidences of NLP have been published as model for comprehension and learning [16], few academic works exist on the subject.

NLP claims that people are intrinsically creative and capable, acting according to how they understand and represent the world, instead of how the world is. Literature constantly cites Korzybski's statement [17] "the map is not the territory", a reference to individual understanding that everyone has – mental model -, according to his/her experience, beliefs, culture, knowledge and values.

In [13], an article written by NLP scientific research group, NLP is presented as an epistemological perspective, with scientific principles which are not usually presented. The first works published by Bandler and Grinder [9, 10] were based on Fritz Perl's models, Gestalt founder, Virginia Satir, researcher in family therapy, and Milton Erickson, doctor in medicine, master in psychology and hypnotherapist recognized worldwide. As a consequence, the epistemological view of NLP presents a roadmap to develop the necessary scientific basis to support its beliefs. The research reported in this paper explores this path by scientifically characterizing the use of preferred representational systems for cognition.

This representational system (or internal representation) is highly dependent on context (i.e. it varies with the situation) [12]. This way, some people, in specific contexts, may prefer to use one or more basic systems to communicate and learn [6, 7, 8]. Most authors in the area recognize the following basic systems:

(1) Visual, that involves internal images creation and the use of seen or observed things, including pictures, diagrams, demonstrations, displays, handouts, films, and flip-chart;

(2) Auditory, that involves sounds reminders and information transferred through listening; and

(3) Kinesthetic, that involves internal feelings of touch, emotions and physical experience: holding and doing practical hands-on experiences.

We use all of our senses all of the time and ,depending on the circumstances, we may focus on one or more of them – for instance, when listening to a favorite piece of music, we may close our eyes to more fully listen and to experience certain feelings. In order to see things more clearly, we might need to close our eyes and visualize the situation, person or place.

So, we all use each of the senses and each of us also has a Preferred Representational System (PRS), one that we use most when we speak, learn or communicate in any way. For example, when learning something new, some of us may prefer to see it or imagine it performed, others need to hear how to do it, others need to get a feeling for it, and yet others have to make sense of it. In general, one system is not better than another and sometimes it depends on the situation or task that we are learning or doing as to which one or more representational systems might be more effective than another.

Supporters of NLP believe that word predicates let us know what is consciousness state of a person. They believe that specific, sensory-based, word predicates are chosen when a person is using a specific representational system. The predicates indicate what portion - of internal representations - they bring into awareness [10]. Such predicates may be identified and used to improve communication among the analyzed subjects, for example.

One of the major problems in communication, be it informal or technical, is the difficulty to arouse interest on the receiving end, the person who is reading or listening to your message. Many times, the person who receives the message does not assimilate what is being transmitted, be it a simple message or a technical diagram. NLP can then be one approach to improve communication. The challenge lies in identifying the representational system that is being used by the subject and match the same system for empathy construction. The matching consists of identifying the predicates that indicate a representational system and use them, or other predicates that belong to the same system, for communication [10].

In order to exemplify this matching process, consider the following question "have you seen the logic of the algorithms that I showed you?", and the following answer "not yet, I am going to examine them carefully, once I get a clear picture of the whole system." This is a coherent answer to the question from the sensory system matching perspective. The sensory-based words "seen" and "showed" in the first phrase indicate a visual processing, and the response used the same system through the visual sensory words "examine them" and "clear picture".

In this context, detecting the developers' representational preferences may enhance the empathy in the team communication, i.e, each member may be more stimulated in his/her Preferred Representational System, enhancing the effectivity of communication, software comprehension and the solution of activities of development and maintenance.

Allocating a person in a task, considering his/her technical abilities as well as his/her personality, is essential for the success of any software project. Productivity secret is to adjust the project needs with its members' personalities. Detecting, for instance, that a system analyst barely uses his/her visual representational system may help solve his/her difficulties with project diagrams or estimulate his/her reallocation to another activity. Many times a member is lost because of wrong job allocation. A good programmer may become a not so good analyst. In other situations, a person's preferential cognitive system may not match his/her colleagues' profile, or the way the organization works.

Our research deals with the identification of sensory-based words used by developers in OSS discussion lists. We then use these words to characterize the preferred representational systems of the developers and analyze these against their profile and role in the projects.

B. *Neurolinguistic Criticism*

NLP is dismissed as theoretically impossible or implausible, especially in websites where one cannot fully trust.

The literature in academic journals is minimal, and the reference [47] is a good example. There has been virtually no published investigation into how NLP is used in practice. The experimental research consists largely of laboratory-based studies from the 1980’s and 1990’s, which investigated two particular notions from within NLP, the ‘eye movement’ model, and the notion of PRS.

Heap [48], in particular, has argued that, on the basis of the existent studies, these particular claims of NLP cannot be accepted. Heap conducted a meta-analysis of these and appears entirely justified in criticising the unequivocal claims made in NLP literature. It is notable, however, that Heap’s meta-analysis included many postgraduate dissertations. His bibliography refers only to sources of abstracts of those dissertation studies, not to the dissertations themselves. Thus, his meta-analysis appears based on the reported outcomes of these studies, not on critical appraisal of their methodology or validity.

Einspruch and Forman [12], and Bostic St.Clair and Grinder [49] have also argued that the types of study reviewed by Heap are characterised by problems affecting their reliability, including inaccurate understanding of NLP claims and invalid procedures due to (for example) the inadequate training of interviewers, who therefore may not have been competent at the NLP techniques being tested. Heap himself offers only an ‘interim verdict’ and acknowledges Einspruch and Forman’s view that ‘the effectiveness of NLP therapy undertaken in authentic clinical contexts of trained practitioners has not yet been properly investigated’ [48].

Given these concerns, in [13], for example, Tosey and Mathison suggest that the existing body of experimental research cannot support definitive conclusions about NLP. It seems clear that there is no substantive support for NLP in this body of experimental research, yet it also seems insufficient to dismiss NLP.

Our study does not test NLP techniques, but rather it shows an association between NLP based-measures and developers’ roles and profiles.

III. TEXT MINING BASIS

Our work is based on Text mining (TM), a technology for analysis of large collections of unstructured documents, aiming to extract patterns or interesting and non trivial knowledge from text [18].

A. *Preprocessing*

Similar to conventional data mining, text mining consists of phases that are inherent to knowledge discovery process [19]. Classification of knowledge discovery phases may vary for different authors, but most comprises at least data selection, preprocessing, mining and assimilation. Text mining pays special attention to preprocessing, because its data is unstructured for computer analysis. In other words, after setting the base with texts to be mined, it is necessary to convert each document to a format suitable for a computational algorithm.

One may use three different ways – boolean, probabilistic or vector-based models – to structure the information of a text document for computational analysis. The vector model utilizes geometry in order to represent documents. Introduced by [20], this model was developed to be used in a retrieval system called SMART. According to the vector model approach, each document is represented as a term vector and each term receives a weight that indicates its importance in the document [20].

In more formal terms, each document is then represented as a vector, which is composed of elements organized as a tuple of values: $d_j = \{w_{1j} ,.. , w_{ij}\}$, where d_j represents a document and w_{ij} represents a weight associated to each indexed term of a set of t terms of the document. For each element of the term vector, a dimensional coordinate is considered. This way, the documents can be placed in a Euclidian space of n dimensions (where n is the number of terms) and the position of the document in each dimension is given by the term weight in this dimension.

In this model, the consultations are also represented by vectors. This way, the document vectors can be compared with the consultation vector and the similarity between them can be easily computed. The most similar documents (those that show the closest vectors to the consultation vector) are relevant, and returned as a response to the user. Besides, documents that show the nearest vectors can be considered similar to the target document.

A term vector is built by the following steps.

B. *Term Extraction*

Researchers from the information retrieval field claim that the main difference between data and information retrieval is exactly the relevance of the information obtained [21].

In general, not all terms that compose a document are relevant when one intends to extract high level information. So, in order to compose a term vector for a text, it is necessary to identify words with high semantic content, selecting only those that are meaningful for the objective at hand.

The task of term extraction from a document consists of various steps, all of them contributing for the final purpose of producing a vector with high semantic content [22]. They are described as follows:

1. Lexical analysis: the original document is not always represented in a purely textual format. Therefore, it is necessary to convert it to a standardized format, eliminating any attributes of presentation formatting.

2. Characters conversion to uppercase or lowercase: such procedure enables equal words written with a character in a different format in uppercase or downcase – for example, neuro and Neuro may be interpreted as the same term.

3. The use of a word list to be ignored: commonly called stopwords. This list consists of a relation of words that have no significative semantic content (e.g., prepositions, conjunctions, articles, numerals etc.) and consequently are not relevant for text analysis.

4. Morphological normalization: aiming to cluster terms with the same conceptual meaning, e.g., the words compute and computation. A conversion algorithm of terms to radicals may be applied in this case. In the example, the words "compute and computation" have the same radical "comput", so they can be reduced to this term.

5. Selection of simple or compound words: in some cases, during the preprocessing of a document, several joint words (phrases) may be managed as a single term. This selection can be done using predefined word lists or statistical and syntactic techniques.

6. Normalization of synonyms: words with the same meaning can be reduced to a specific term, for example, the acronym SEL and the composition Software Engineering Lab, both have the same meaning.

7. Structural analysis: this step consists of associating information to each term regarding its positioning in the document structure, in order to distinguish it from a homonym term situated in another position.

C. Assigning weights

The process of associating numeric values to each term previously extracted is known as assigning weights. In general, the settlement of the term weight in a document can be resolved with two paradigms [23]:

1. the more a term appears in the document, the more relevant the term is to the document subject;

2. the more a term occurs among all documents of a collection, the less important the term is to distinguish between documents.

This calculation can be done in two ways:

1. Binary or Boolean – The values 0 and 1 are used to represent, respectively, the absence or presence of a term in the document.

2. Numeric – It is based on statistical techniques regarding the term frequency in the document.

The numeric weights can be represented by measures such as:

- Term Frequency (tf): Simple method which consists of the number of times that a term w_i occurs in a document d. This method is based on the premise that the term frequency in the document provides useful information about the relevance of this term for the document.
- Document Frequency (DF): it is the number of documents in which the term w_i occurs at least once.
- Inverse Document Frequency (idf): it defines the relevance of a term in a set of documents. The bigger this index is, more important the term is to the document in which it occurs. The formula to calculate idf is:

$$idf_i = \log \left(|D| \,/\, |\{d: t_i \varepsilon d\}| \right)$$

Where $|D|$ represents the total of documents and $|\{d: t_i \varepsilon d\}|$ represents the number of documents where the term t_i appears.

tf-idf: it combines the term frequency with its inverse frequency in the document, in order to obtain a higher index of its representativeness. The formula to calculate tf-idf weight is:

$$(tf\text{-}idf)_{i,j} = tf_{i,j} \times idf_i$$

D. Grammatical classes and noun phrases

To further strengthen the semantic meaning of the structured data, our work uses word composition. Words that have similar semantic and syntactic behaviors can be clustered in the same class, creating syntactic or grammatical categories, more commonly named parts of speech (POS). The three main ones are noun, verb and adjective. The nouns refer to people, animals, concepts and things. The verb is used to express action in a sentence, whereas the adjectives express noun properties.

The POS detection is important, because in specific contexts two or more words with different grammatical categories may have one unique meaning. The semantical composition of words is known as a Noun Phrase [24]. Noun phrases (NPs) cluster words in a context and its detection can improve the search accuracy in texts. Usually a noun is the central element (head part) which determines the syntactical character of a NP, and a verb or an adjective modifies this noun (mod part).

In order to implement NP detection, it is necessary that a dictionary specifies which words can appear together. In general, it is not necessary to store words in a compound way, because this process demands time and does not enhance the system efficiency significantly. What can be done is to store information about the distance between words, and the consultation technique is responsible for evaluating whether words are adjacent or not.

NEUROMINER, the tool discussed in this article, uses the vector spatial model, transforming the developer's emails into vectors, classifying the words grammatically and identifying NPs, as well as assigning weights to the extracted terms.

IV. LIWC FOR NEUROLINGUISTIC

A. Motivation

We identified works that try to pinpoint people's preferred representational systems, but those researches are only in psychology, and in domains like sports and education [25]. We also found some software engineering papers that use text mining to identify developers' general emotional content. However, these papers do not try to relate the developer's personality, or other psychological aspect, to the software engineering activities themselves [26, 15]. This gap of knowledge stimulated us to use text mining to investigate the association between a psychological concept – PRS – and software development roles and activities.

Our tool, NEUROMINER, uses Linguistic Inquiry and Word Count (LIWC) to classify the Preferred Representational Systems (PRS) of developers in a given context. We could not find any tools that make automated neurolinguistic text analysis and, as discussed later, our LIWC approach can be adapted to other domains.

Finally, due to the scarcity of scientific researches about NLP itself, this paper generates the opportunity to show empirical results of applying one of its principles to our, human-intensive, domain.

B. Neurominer

NEUROMINER combines statistic and text mining techniques with sensory predicates of NLP, aiming to classify programmers's PRS.

The basic characteristics of NEUROMINER are:

- Use of a neurolinguistic dictionary;
- Use of ANOVA for PRS classification. An ANOVA is an Analysis of the Variation present in an experiment. It is a test of the hypothesis that the variation in an experiment is no greater than that due to normal variation of individuals' characteristics and error in their measurement;
- Use of an ontology to identify Software Engineering and neurolinguistic terms combined in noun phrases;
- Use of synonym normalization resources with dictionaries for Brazilian Portuguese [50] [27], and for English [51] [28].

This paper will not focus on Neurominer internal architectural, but rather in its NLP and PRS classification approach.

Building and Using a NLP Dictionary

According to NLP, the words a person chooses to describe a situation – when they are specific to representational system (i.e., sensory-based) – let us know what his/her consciousness is. This predicate indicates what portion of internal representations the person brings into awareness [10].

The goal of our work is to identify the most used RS and the percentage of use of the others. For this, we have adopted a LIWC approach similar to the one presented by [15]. As shown in Table I, it uses a NLP dictionary with four basic dimensions composed of sensory-based words or phrases [10, 14].

TABLE I. NEUROLINGUISTIC DIMENSIONS

DIMENSION	EXAMPLE WORD	TAG
Visual	'brilliant'	Mod
Auditory	'dissonant'	Mod
Kinaesthtic	'concrete'	Mod
Concepts	'algorithm'	Head

The Concept dimension was created to increase contextual classification power. A noun phrase (NP) such as 'brilliant algorithm' indicates a visual PRS cue used in the context of software engineering. The tag column of Table I indicates that the dimension is part of a modifier (PRS) or head (SE context) of the NP. In this very simple way, NPs formed with SE ontological concepts have a bonus multiplied to the score in our text mining approach.

The concepts were extracted from software document ontology discussed in [29] and described in [30], which is based on various programming domains, including programming languages, algorithms, data structures and design decisions such as design patterns and software architectures. Our goal is to verify the direct relation of sensory-based words with Software Engineering context. This way, we can find noun phrases formed with ontological concepts and sensory-based words or phrases, our first innovation.

Email Mining with Neurominer

Figure 1 summarizes the text mining main steps. The approach is summarized only briefly, since details about preprocessing [29], and clean messages [15, 31] have already been published.

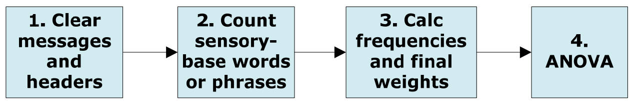
Step 1 includes steps such as stemming, part-of-speech tagging and noun-phrase detection. For example, in the latter step cited we use the MuNPEx approach (Multi-Lingual Noun Phrase Extractor, [52]).

After downloading the email archives, the system parses each email for meta-data as discussed in [31], and places its relevant information into a data mart [32]. This data mart was designed based on a software engineering data warehousing architecture proposed by us in previous papers [33, 34].

The process only uses the text actually written by the sender and its timestamp. It removes all diffs, attachments, quoted replies, signatures, code and HTML that is not part of a diff.

We adopted a daily frequency-based cumulative approach. In step 2, the system finds and counts the senders' sensory-based words and phrases by month, considering the NLP dimensions in the dictionary.

In step 3, the system uses a text mining approach for the NLP classification of individuals, instead of the traditional document classification, our second innovation. In it, the set of all emails written by a developer is treated as a 'big text' to be classified. A simple approach for that is to count all the words found in all emails of a developer and verify the percentage of each representational system. However, aiming more detailed analyses of evolution, the system considers the daily frequencies of the words.


1. Clear messages and headers
2. Count sensory-base words or phrases
3. Calc frequencies and final weights
4. ANOVA
Figure 1. Text mining process chain

Our alternative to the basic tf-idf formulation (see Text Mining section) computes weights or scores for sensory-based words. The values are positive numbers so that it captures the presence or absence of the word in a month. Equation (1) indicates that *neuro* weight assigned to a word j is the term frequency (i.e., the ratio between word count and the sum of number of occurrences of all words) modified by scale factor for the importance of the word. The scale factor, for our approach, is called daily frequency df(j), which is the ratio between the number of days containing word j and the number of loaded days. Thus, when a word appears in many days, it is considered more important and scale is increased.

$$neuro(j) = (tf(j) + df(j)) \times b \qquad (1)$$

In addition, a bonus b is also multiplied to the measure. The bonus can be 1 or 2, where b will be equal to 2 if term is a NP or phrase, and 1 if term is a simple word.

At the end of each month, the term weights are recalculated and a general total of weights (final weight) are stored for each representational system. Lastly, each representational system monthly mean is computed.

In the step 4, we use ANOVA (analysis of variance) to determine if the means are statistically different.

V. Experiment

The rest of this paper describes an experimental validation of our approach. The presented experimental process follows the guidelines by [35]. This section will focus on the experiment definition and planning. The following section will present the obtained experimental results.

A. Goal definition

The main goal of our study is to evaluate if OSS top committers have a PRS. This goal is formalized using the GQM Goal template proposed by [36] and presented in [37]:

Analyze Project top committers
with the purpose of evaluation
with respect to NLP context-specific Preferred Representational Systems
from the point of view of software engineering researchers
in the context of development mailing lists of OSS projects

B. Planning

Context selection: The experiment will target OSS projects.

Hypothesis formulation:

The issues we are trying to explore are as follows.

1. We are interested in verifying if OSS top committers have a PRS.

2. Besides that, we believe top committers are more kinesthetic than auditory and visual. Our belief is that experienced programmers of the OSS community rely heavily on their experiences, and are less dependent on visual and auditory artifacts than the general population of OSS software engineers.

Considering the arduous manual work of searching for valid emails used by top committers and, as a consequence, the small sample size due to the low number of top committers, a formal statistical test will not be performed for the second issue. This hypothesis is:

Null hypothesis H_0: OSS top committers have the same frequency for the three profiles (Visual, Auditory and Kinesthetic).

Alternative hypothesis H_1: The frequency of OSS Kinesthetic top committers is higher than Visual and Auditory.

However, considering the large number of emails that will be mined, the test of the existence of a PRS top committer for each selected will have large power. We will also do a detailed qualitative analysis of the top committers' profiles in order to sanity check NEUROMINER measures.

NEUROMINER will be used to calculate the final weights for each representational system, as well as representational systems monthly means (see Email Mining section).

Formally, the hypothesis we are trying to confirm is:

Null hypothesis H_0: OSS top committers have the same representational system monthly mean.

H_0^{PRS}: μ(Visual Final Weight) = μ(Auditory Final Weight) = μ(Kinesthetic Final Weight)

Alternative hypothesis H_1: at least one of the representational systems' monthly means is different from the others.

Participant and artifact selection:

To answer our research questions, we extracted email messages from the Apache [53] and Postgresql Projects [54] mailing lists. For the Apache, we analyzed the body of all email messages between 1996 and 2005 (35,483 messages), and selected the four developers who had the greatest number of commits. Those are the same developers studied by [15]. For Postgresql, we analyzed the body of all email messages between 1997 and 2006 (57,159 messages), and also selected the four developers who had the greatest number of commits. In both projects, two top committers still contribute to the project and others have already left.

We also created clusters of all other developers for both projects. During data reporting we will refer to this general population measures as the *cluster*.

The analysis is completely non-intrusive to developers as the data was drawn directly from the project mailing lists. For each developer and cluster, once a month, we calculated the PRS using the method described in Section 4.2.2 (email mining). At the end, we had one data point of mined e-mails per month for each subject. Clusters were mined for 3 years (36 months). Top-committers were mined for the last 10 years, but data points were produced only for those months in which they posted at least one e-mail at the project discussion list. NEUROMINER then tested the population distribution and calculated the analysis of variance of the monthly PRS scores for each participant (all calculation was double checked using SPSS). The population distribution for each sample is normal.

C. Results

Tables II and III summarize our results. The column Totals represents the number of months (data points for each participant), days and emails. For each representational system the final weight is shown for the set of all sensory-based words found and the monthly average of this weight. The column ANOVA p-value reports P values for the null hypothesis.

D. Analysis and Interpretation

For the statistical testing, we established an apriority significance level (α) of 0.05. Tables II and III show that our first hypothesis is accepted as we obtained the p-value of 0.000

for all means but one, developer G. The results for the clusters and developers A-F and H are significantly lower than 0.05, strongly rejecting the null hypotheses.

We observed that Developers B, D, E, F and H did not have a higher value for the Kinesthetic RS. This contradicts our initial hypothesis that top committers are more Kinesthetic than Visual and Auditory. Moreover, this is also the PRS of the general population (see Cluster Row in Tables II and III).

With respect to the first point, we found out that there are four visual, two kinesthetic and one auditory top-committers. Looking at their profiles, we realized that most of them are quite concerned with following procedures and documenting information, contradicting our initial stereotype of a hardcore OSS developer.

The second point, the other developers being kinesthetic on average, leads us to believe that most people that post in the list are indeed involved with practical activities in the project, and counters our initial belief that many posters were by newbies or people that were simply curious – wanted to hear – about the project.

TABLE II. APACHE TOP COMMITTERS RESULTS

Participant	Left the Project?	Totals (1996 - 2005)			Visual		Auditive		Kinesthetic		ANOVA p-value
		Months	Days	Emails	Final Weight	Monthly Mean	Final Weight	Monthly Mean	Final Weight	Monthly Mean	
A	Yes	53	773	4357	2.458847752	2.7274	2.222439202	2.4645	2.579237886	2.9774	.000
B	Yes	33	320	1082	2.258647848	2.6680	1.557743226	1.6667	1.900853069	2.2514	.000
C	No	72	1213	4279	1.904784203	2.3577	1.684352265	2.0152	2.17853237	2.6210	.000
D	No	42	366	644	0.557085631	0.6013	0.526795847	0.6581	0.441884768	0.4684	.000
Cluster	-	36	1091	25121	6.906216384	7.7567	6.456228874	7.3756	8.849529521	9.5515	.000

TABLE III. POSTGRESQL TOP COMMITTERS RESULTS

Participant	Left the Project ?	Totals (1997 - 2006)			Visual		Auditive		Kinesthetic		ANOVA p-value
		Months	Days	Emails	Final Weight	Monthly Mean	Final Weight	Monthly Mean	Final Weight	Monthly Mean	
E	Yes	62	899	2854	0.928310296	0.929891595	0.546308965	0.530733142	0.637687699	0.638343065	.000
F	Yes	53	478	1284	0.97883767	0.958569136	0.419031696	0.421918608	0.615530516	0.631261987	.000
G	No	55	536	1176	0.845112965	0.718432684	0.672360338	0.629946198	0.561439549	0.725690981	0.085
H	No	121	2728	17712	0.901184217	0.855757104	0.648274323	0.644466038	0.596461266	0.600263054	.000
Cluster	-	36	731	34133	0.745095331	0.717685413	0.623314426	0.617197033	0.727219871	0.752834067	.000

Even where there is dominance of the Kinesthetic RS, the results show that OSS developers also have significant visual and auditory RS. This may indicate an opportunity to introduce better visualization tools and better support for cooperative work, increasing direct developer interaction, in OSS development.

Digging a bit deeper into the top committers' profiles [55] and [56], we found out that Developer B had a strong involvement with the project architecture and the work to hybridize Apache. This seems to support his/her Visual PRS (see Table II and Figure 2).

Developer D – the most singular subject among the top committers – has an Auditory PRS and also a strong Visual RS. His/her profile indicates that he/she contributes heavily with the project documentation and his/her predominant working language is XML. This possibly matches the mined profile, as one would expect strong listening and reading capabilities from people involved in OSS documentation.

These insights are quite aligned with the results presented in [15]. This paper reports that the measures collected for Developer D were the least associated with the other subjects in the study. Our study, however, went further and indicated a classification that directly matched the subject profile and project role.

Regarding the Postgresql top-committers, the first thing that catches the eyes (see Figure 3), is that three of them are highly visual. Moreover, the visual PRS is high even for Developer G and the project cluster itself. Top-committers E, F and G are highly involved with both documentation and implementation. Top-committer G, the only one who is not classified in any category, p-value 0.085, also works on performance testing and tuning, which may be related to his/her relatively high kinesthetic score. He/She also works with user groups and on providing general direction for the project advocacy, which may be related to his/her relatively high auditory score. Top-committer H, by far the most active top-committer of them all, is visual but also has a high auditory score, even higher than his/her kinesthetic score. His/her scores may be explained by the fact that he/she is highly involved with development, but also does training and maintains the project FAQ and TODO list.

E. Threats to Validity

In spite of the fact that Apache and Postgresql are a mature, real world, large projects, and our results seem to be quite consistent with the obtained top-committer profiles, the PRS measures still need further investigation to assure external validity.

A new study is being run in an industrial setting. The completely different setup and higher control over the study environment will help to increase the generalization power of the results.

We obtained the top committer profiles through the project sites. Better analysis would be possible with more extensive information. Gathering more profiling data would help us improve our analysis. Aiming at this, we developed a questionnaire to characterize and assess the PRS of software engineers. This questionnaire is publicly available at [57].

We contacted the top-committers by e-mail and asked them to fill it out. Unfortunately, they could not find the time to fill it out.

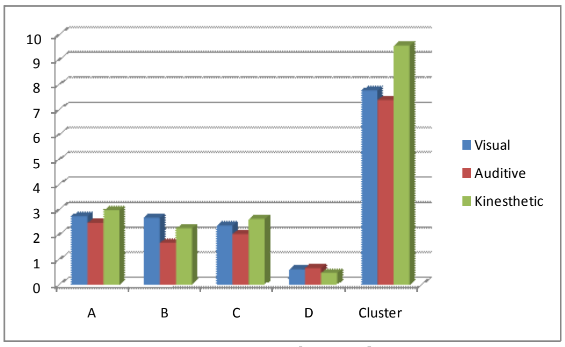


Figure 2 Apache Results

VI. Related Work

Regarding NLP, there are some scientific articles showing evidences of its assertions. In addition, there are several publications about preferences for some specific representational systems in the cognitive and learning processes, even in computing [38].

The basis for models and techniques presented by NLP can be found in psychological studies that involve the so-called "Chameleon Effect", which concerns non-matching and matching stimuli to the empathy increase in communication. Reference [39] did an experiment at a restaurant in the south of Netherlands in which half of the studied waitresses used the "Chameleon Effect" to serve customers. Results showed that the average value of the tips almost doubled for the waitresses who used matching language and behavior. The reference [40] analysed subjects who interacted with artificial intelligence based software – an agent which simulates a subject giving an explanation. The agent that imitated subject's movements was more convincing, receiving more positive evaluations. It was the first virtual reality study that showed the effects of a non verbal automatic imitator in order to gain empathy.

Reference [14] tested NLP hypothesis about matching processes which enhance empathy in communication. The relation between matching and empathy increase were significant. Education was also related to the empathy increase, however, even when it was controlled, the relation between matching and empathy remained significant.

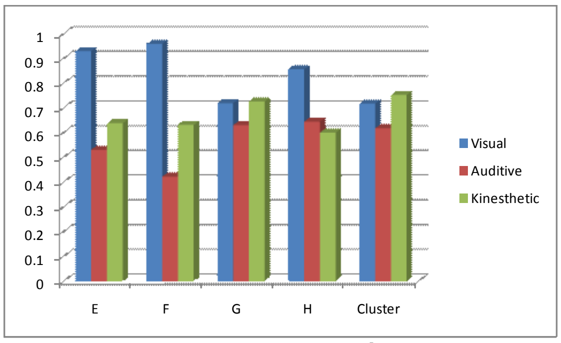


Figure 3 Postgres Results

Paolo et al. [38], presupposing some students' preferences for the kinesthetic processing in certain contexts, developed and tested a set of kinesthetic activities for a distributed systems course, with graduation and post graduation students. The article presents detailed descriptions of the exercises and discusses the factors that contributed for their success and failure.

Fleming presented a questionnaire developed and used at Lincoln University to identify the preferences of students for particular modes of information representation [25]. Named the VARK model, the questionnaire is now the basis of a commercial service for educational planning (http://www.vark-learn.com/english/page.asp?p=questionnaire). The acronym originates from questionnaire classification of the learning styles. "V" is for visual learners, "A" is for auditory learners, "R" is for reader/writer learners, people that best learn through seeing printed words. And, "K" is for tactile/kinesthetic learners.

The VARK classification differs from the NLP classic classification, because it includes the readers-writers category on top of the usual the visual, kinesthetic and aural categories. According to Fleming, results show that students with preferences for R and V information use their eyes to "take in the world" but they have preferences within that sensory mode; some like text and others like diagrammatic or iconic material - information that is symbolically displayed [25].

Another point raised by the VARK data is that the same subject may have different profiles in different areas (martial arts, music, languages, etc) for different time periods, i.e., a subject may be Visual (V) to learn martial arts for a period of time and become Kinesthetic (K) after that.

These evidences support some NLP techniques and establish an empirical basis for further studies.

Considering text mining in Software Engineering, independent from the database, linguistic analyses have been used to comprehend the development of OSS softwares. Witte at al. [29] considered the semantic importance of the documents written in natural language in the process of maintenance and reengineering. The result of the research consisted of creating a text mining system capable of filling software ontology with information extracted from these documents.

Other works have already considered email specific analysis to study OSS development process [41, 31]. Pattison et al. [42] studied the relation between the several software entities mentioned in emails and the number of times these entities are included in the changes made.

Two works are closest to the research presented here. In the first, Scialdone et al. [26] used emails to evaluate the social presence in maintenance groups of OSS projects. Social presence theory classifies different communication media along a one-dimensional continuum of social presence, where the degree of social presence is equated to the degree of awareness of the other person in a communication interaction. According to social presence theory, communication is effective if the communication medium has the appropriate social presence required for the level of interpersonal involvement required for a task. On a continuum of social presence, the face-to-face medium is considered to have the most social presence, whereas written, text-based communication, the least. It is assumed in social presence theory that in any interaction involving two parties, both parties are concerned both with acting out certain roles and with developing or maintaining some sort of personal relationship [43, 44].

Core and Peripheral members were compared, and the results showed that respect behavior to another one's autonomy may contribute to the survival of the group and continuity of the project. The work does not raise alternatives to social presence or solutions to increase empathy. It is based solely on psychological and social measures. It establishes no relation between these aspects and software engineering roles and profiles.

The second work is [15], which analysed the content of Apache discussion list to find developer's personality and general emotional content. Like ours, this work uses a LIWC tool (Linguistic Inquiry and Word Count) [45] to help ratings. However, the work uses a general purpose psychological analysis tool. It was neither developed to explore emails nor to preprocess text mining and score terms.

In [46], we presented an initial report for the use of neurolinguistic ratings by mining development discussion lists. This work motivated and guided the need for extended studies and details about innovations and technologies involved, which are now presented in this article.

VII. CONCLUSION AND FUTURE WORK

We presented a text Neurolinguistic mining tool that is capable of extracting sensory-based words from software mailing lists. The system is novel in four important aspects: (1) it automates parts of NLP practices; (2) it combines a SE taxonomy with sensory-based words; (3) it adapts traditional text mining process to NLP practices; and (4) it uses specific Text Mining Data Mart in a software engineering data warehouse. The approach itself is novel in its use of NLP concepts in the software engineering area.

The results are encouraging. In spite of being contrary to our expectation, the PRS scores clearly differentiate the top-committers from the general population of the projects. Moreover, the scores are aligned with the participant profiles, indicating that they indeed can be used to profile people to software engineering tasks and, possibly, better communication. It is worth noting that the classifications presented in this work are not fixed, ie, they initially represent only the greater use of one or other system within the context analyzed.

Thus, in specific contexts, a particular sensory system may take dominance (for example, (a) being primarily aware of external kinesthetic representations - bodily movements and sensations - while training. (b) Concentrating preferentially on auditory comparisons while analyzing client requirements), representational system preferences thus tend to be a contextual artifact in that when an individual considers specific contexts, his/her language can reflect how he/she processes the information relating to the process of considering that context. In certain cases a person may find himself/herself with certain rigid representations and strategies which preclude behavioural choice. In such a case, one representational system may predominate and important for enhancing emphaty.

Our future work will address three key issues: (1) examine the empathy of exchanged messages to assess communication success over PRS alignment; and (2) better profile PRS scores with usage of software engineering artifacts and the roles that a person plays in a project; and (3) devise new ways to measure PRS.

ACKNOWLEDGEMENTS

This work is partially (for Methanias and Manoel) supported by INES - National Institute for Software Engineering (www.ines.org.br) funded by CNPq (grant 573964/2008-4).

REFERENCES

[1] Klemola, T. and Rilling, J. Modeling Comprehension Processes in Software Development. Proceedings: First IEEE International Conference on Cognitive Informatics, pp. 329-336, 2002.

[2] Diehl, S. Software Visualization -Visualizing the Structure, Behavior and Evolution of Software. New York: Springer Verlag, 2007.

[3] Hungerford, B. C., Hevner, A. R and Collins, R. W. Reviewing Software Diagrams: A Cognitive Study. IEEE Transactions on Software Engineering, vol. 30, no. 2, pp. 84-95, 2004.

[4] Maldonado, J. C., , Carver, J., Shull, F., Fabbri, S. C. P. F., Doria, E. S., Martimiano, L., Mendonca, M., and Basili, V. Perspective-based reading: A replicated experiment focused on individual reviewer effectiveness. Empirical Software Engineering An International Journal, 11(1): pp. 119–142, 2006.

[5] Travassos, G., Shull, F., Fredericks, M., Basili, V.R.: Detecting defects in object-oriented designs: using reading techniques to increase software quality. In: Proceedings of the 14th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'99), pp. 47–56. ACM, New York, 1999.

[6] Dent, K. A. Cognitive Styles: Essence and Origins: Herman A. Witkin and Donald R. Goodenough. Journal of the American Academy of Psychoanalysis, 11: pp. 635-636, 1983.

[7] Matthews, D. B. Learning Styles Research: Implications for Increasing Students in Teacher Education Programs. Journal of Instructional Psychology, 18 pp. 228-236, 1991.

[8] Peters, Derek, Gareth, Jones, and Peters, John. Preferred 'learning styles' in students studying sports-related programme in higher education in the United Kingdom. Studies in Higher Education. Vol 33, pp. 155 – 166, 2008.

[9] Bandler, R. and Grinder, J. Frogs into Princes: Neuro-linguistic Programming, Moab, Utah: Real People Press, 1979.

[10] Dilts, R., Grinder, J., Bandler, R. and DeLozier, J. Neuro-linguistic Programming: Volume 1 California: Meta Publications, 1980.

[11] Elich, M., Thompson, R. W., and Miller, L. Mental imagery as revealed by eye movements and spoken predicates: A test of neurolinguistic programming. Journal of Counseling Psychology, 32(4), pp. 622-625, 1985.

[12] Einspruch, Eric L. and Forman, Bruce D. Observations concerning Research Literature on Neuro-Linguistic Programming. Journal of Counseling Psychology, vol. 32, no. 4, pp. 589-596, 1985.

[13] Tosey, Paul and Mathison, Jane. Fabulous Creatures of HRD: A Critical Natural History Of Neuro-Linguistic Programming. 8th International Conference on Human Resource Development Research and Practice across Europe, Oxford Brookes Business School. Available in http://www.nlpresearch.org/, last access in Dez/2009, 2007.

[14] Turan, Bulent and Stemberger, Ruth M. The effectiveness of matching language to enhance perceived empathy. Communication and Cognition, Vol 33(3-4), pp. 287-300, 2000.

[15] P. Rigby and A. Hassan. What Can OSS Mailing Lists Tell Us? A Preliminary Psychometric Text Analysis of the Apache Developer Mailing List. Proceedings of the Fourth International Workshop on Mining Software Repositories, 2007.

[16] Tosey, P. and Mathison, J. Neuro-linguistic Programming and Learning Theory: a response. The Curriculum Journal Vol. 14 no.3 pp. 361 – 378, 2003.

[17] Korzybski, A. Science and Sanity. The International Non-Aristotelian Library Publishing Company. Available online at http://www.esgs.org/uk/art/sands.htm, accessed 14.01.2009, 1941.

[18] Visa. A. Technology of Text Mining. Proceedings of Machine Learning and Data Mining in Pattern Recognition, Second International Workshop, MLDM 2001, Leipzig, 2001.

[19] Fayyad, Usama; Piatetski-Shapiro, Gregory; Smyth, Padhraic. The KDD Process for Extracting Useful Knowledge from Volumes of Data. In: Communications of the ACM, pp.27-34, Nov.1996

[20] Salton, G., Wong, A., andYang C.S. A Vector Space Model for Automatic Indexing. Communications of the ACM, New York, v.18, 1975.

[21] C. J. V. Rijsbergen. Information Retrieval, 2nd edition. Butterworths, London, 1979.

[22] Hiemstra, D. and Jong, F. Statistical Language Models and Information Retrieval: Natural language processing really meets retrieval, Glot International 5(8), Blackwell Publishers, 2001.

[23] Sholom, M. Weiss et al. Text Mining: Predictive Methods for Analyzing Unstructured Information. Springer, 2005.

[24] Cunningham, H., Maynard, D., Bontcheva, K., and Tablan, V. GATE. In Proceedings of the 40th Meeting ACL, 2002.

[25] Fleming, N. D. I'm different; not dumb. Modes of presentation (VARK) in the tertiary classroom, in Zelmer,A., (ed.) Research and Development in Higher Education, Proceedings of the 1995 Annual Conference of the Higher Education and Research Development Society of Australasia (HERDSA), HERDSA, Vol 18, pp. 308 – 313, 1995.

[26] Scialdone, Michael J. Group Maintenance Behaviors of Core and Peripherial Members of Free/Libre Open Source Software Teams. In Proceedings of the OSS 2009, pp. 298-309, 2009.

[27] Maziero, E.G. Electronic Thesaurus for the Brazilian Portuguese. VI Workshop on Information Technology and Human Language, pp. 390-392, 2008.

[28] Cognitive Science Laboratory. WordNet - a lexical database for the English language, Princeton University, 2006.

[29] Witte, R., Li, Q., Zhang, Y., and Rilling, J. Text mining and software engineering. IET Softw – Special Section on Natural Language in Software Engineering, 2, (1), pp. 3–16, 2008.

[30] Wongthongtham, P., E. Chang, T. S. Dillon, and I. Sommerville. Development of a software engineering ontology for multi-site software development. IEEE Transactions on Knowledge and Data Engineering, volume 21, Issue 8, pp. 1205-1217, 2009.

[31] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. Proceedings MSR 2006, pp. 137–143, 2006.

[32] Colaço Jr., Methanias. Implementing Decision Support Systems and Data Warehouses. Rio de Janeiro, Brazil: Axel Books, 2004.

[33] Colaço Jr. , Methanias, Mendonça, M. G. and Rodrigues, F. Data Warehousing in an Industrial Software Development Environment. In: 33rd Annual IEEE/NASA Software Engineering Workshop, 2009A.

[34] Colaço Jr., Methanias, Mendonça, M. G. and Rodrigues, F. Mining Software Change History in an Industrial Environment. In: XXIII Brazilian Symposium on Software Engineering, Fortaleza, Brazil, 2009B.

[35] Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslén. Experimentation in software engineering: an introduction. Kluwer Academic Publishers, 2000.

[36] V. Basili and D. Weiss, “A Methodology for Collecting Valid Software Engineering Data,” IEEE Transactions on Software Engineering, vol.10(3): pp. 728-738, November 1984.

[37] R. van Solingen and E. Berghout. The Goal/Question/Metric Method: A practical guide for quality improvement of software development. McGraw-Hill, 1999.

[38] Paolo A. G. Sivilotti and Scott M. Pike. A collection of kinesthetic learning activities for a course on distributed computing. ACM SIGACT news distributed computing column 26. SIGACT News 38(2): pp. 56-74, 2007.

[39] Van Baaren, R. B., Holland, R. W., Steenaerts, B., and van Knippenberg, A. Mimicry for money: Behavioral consequences of imitation. Journal of Experimental Social Psychology, 39, pp. 393-398, 2003.

[40] Bailenson and Yee. Digital chameleons: automatic assimilation of nonverbal gestures in immersive virtual environments. Psychological Science. v16. Pp. 814-819, 2005.

[41] A.Mockus, R. T. Fielding, and J. Herbsleb. Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology, 11(3):pp. 1–38, 2002.

[42] David S. Pattison , Christian A. Bird and Premkumar T. Devanbu. Talk and work: a preliminary report. Proceedings of the 2008 international working conference on Mining software repositories, May, pp. 10-11, 2008.

[43] Sallnas, E.L., Rassmus-Grohn, K., and Sjostrom, C. Supporting presence in collaborative environments by haptic force feedback. ACM Transactions on Computer-Human Interaction, 7(4), pp. 461–476, 2000.

[44] Stein, D.. Creating shared understanding through chats in a community of inquity. The Internet and Higher Education 10(2), pp. 103-115, 2007.

[45] J. Pennebaker, M. Francis, and R. Booth. Linguistic inquiry and word count: Liwc 2001. Mahwah, NJ: Erlbaum, 2001.

[46] Colaço Jr, Methanias, Mendonça Neto, M. G., Farias, M. A. and Henrique, Paulo. OSS Developers Context-Specific Preferred Representational Systems: An Initial Neurolinguistic Text Analysis of the Apache Mailing List. In: 7th IEEE Working Conference on Mining Software Repositories, Cape Town, 2010.

[47] Thompson, J. E., Courtney, L., and Dickson, D. The effect of neurolinguistic programming on organizational and individual performance: a case study, Journal of European Industrial Training, vol. 26, no. 6, pp. 292-298, 2002.

[48] Heap, M. "Neurolinguistic programming - an interim verdict," in Hypnosis: current clinical, experimental and forensic practices, M. Heap, ed., Croom Helm, London, pp. 268-280, 1988.

[49] Bostic St.Clair, C. and Grinder, J. Whispering in the Wind J & C Enterprises, Scotts Valley, CA, 2001.

WEB REFERENCES

[50] http://www.nilc.icmc.usp.br/tep2/index.htm accessed in October, 2011.

[51] http://wordnet.princeton.edu/wordnet/download/ accessed in October, 2011.

[52] http://www.ipd.uka.de/~durm/tm/munpex/ accessed in July, 2009.

[53] www.apache.org accessed in January, 2011.

[54] www.postgresql.org accessed in July, 2011.

[55] http://httpd.apache.org/contributors/ accessed in January, 2011.

[56] http://www.postgresql.org/community/contributors/ accessed in July, 2011.

[57] www.neurominer.com/survey accessed in October, 2011.

Architecture Centric Tradeoff

A decision support method for COTS selection and life cycle management

Subhankar Sarkar
Senior Manager, Public Sector ERP
IBM USA
ssarkar@us.ibm.com

***Abstract*— Current methods of COTS selection have not been widely accepted in industry, and have been found to lack architectural orientation and a Cost of Ownership perspective. This paper reviews the current methods, and proposes a new method - Architecture Centric Tradeoff (ACT) – for COTS decision support. ACT prescribes a 3-layer Metamodel, Heuristics for Cost of Ownership computations, and a Processor that iterates through candidate solutions to find the optimal tradeoff. In ACT, COTS selection is not driven solely by functional features, but also by architectural characteristics. ACT also takes into account IT portfolio convergence and various COTS delivery methods such as SaaS and Cloud services.**

Keywords- COTS; ERP; Composition based systems; Component evaluation; Cost of Ownership; Tradeoff Analysis

I. INTRODUCTION

Commercial Off the Shelf (COTS) products nowadays comprise a significant proportion of most IT portfolios. In-house software development, following traditional waterfall methodologies, started giving way to *composition based systems* in the late 1990s, and the trend accelerated in the 2000s. Lower costs and shorter implementation cycles were an obvious driver. COTS products provided a viable means to replace outdated systems [1] or integrate disparate portfolios [2]. Also, in the face of the technology revolution, many CEOs were content to leave product development to COTS providers. Around the same time, generally accepted practices and well-formed standards started to emerge in many domains, such as Accounting, Supply Chain and Human Resources. COTS vendors such as SAP, Oracle and PeopleSoft created products in these domains, using design patterns that allowed the same product to be adapted for many businesses. Many organizations adopted COTS as a platform for Business Process Engineering (BPR), and as a means of gaining strategic advantage [3].

Several COTS selection methods exist in literature. One of the first, and the one that gave shape to the generally accepted COTS selection process, was the Off the Shelf Option (OTSO, 1995). This method employed progressive filtering, based on evaluation criteria that included functionality, non-functional properties, strategic considerations and architecture compatibility [4]. Procurement Oriented Requirements Engineering (PORE, 1998), stressed the use of knowledge discovery techniques for progressive elaboration of requirements, and decision support techniques for product ranking [5]. COTS-based Requirements Engineering (CRE, 2002) added a Non-Functional Requirements (NFR) framework to the selection process [6]. COTS-Aware Requirements Engineering (CARE, 2004) intertwined requirements engineering with component evaluation [7]. Mismatch-Handling Aware COTS Selection (MiHOS, 2005) introduced processes for handling mismatches between requirements and COTS, and suggested optimization techniques, such as linear programming [8]. And then, of course, there is the ubiquitous fit/gap spreadsheet.

II. ANALYSIS OF CURRENT METHODS

Most COTS selection methods fit into a general pattern, referred to as General COTS Selection (GCS) [9]:

1. Define evaluation criteria based on stakeholder.
2. Search for candidate COTS.
3. Filter search results based on “must-haves”.
4. Evaluate candidates using decision support techniques.
5. Select COTS, and tailor as needed.

Data suggests that none of these methods have found wide adoption in industry. In a study of Small-and-Medium-Enterprises (SME) in Norway and Italy, it was found that none of them used any of the formal methods for COTS selection [10]. Current criticism for the GCS is summarized below:

- Although most of the proposed approaches were developed for general use, there is no commonly accepted approach for COTS selection [11]. Also, these approaches were proposed without a clear explanation of how they can be adapted to different domains and projects.
- Current approaches suggest using decision making techniques such as weighted score method (WSM) or analytic hierarchy process (AHP) [12]. However, there are several limitations to these techniques [13]. For example, these techniques estimate the fitness of COTS candidates based on ‘one’ total fitness score. This is sometimes misleading due to the fact that high performance in one COTS aspect might hide poor performance in another.
- …what is needed is a more robust negotiation component through which COTS can be progressively selected based on functional and non-functional requirements, architecture, and at the same time resolving conflicts between stakeholders [9].

In this paper, we take a holistic look at the challenges in COTS selection, and discover several problems that have not been adequately addressed in current literature or practice.

Current methods lack a "Cost of Ownership" perspective.

- They look at product features at face value, and select the product with the highest (weighted) feature score. The focus is on the number of mismatches, and on negotiating that to a low value. *The predicate is that the product will not be customized, or that the cost of customization is a function of the number of mismatches alone*. The first predicate is not true in most implementations; usually, the persistent goal in the COTS life cycle is to arrive at the optimal level of customization, not to eliminate customization as a possibility. The second predicate is even less true – the cost of customization is not a function of the number of mismatches, but of the *mismatch type*, and more importantly, of the underlying product architecture and the extensibility mechanisms.
- Development cost is only one component of the customization cost (or the "*Cost of Repair*"), not even the larger part. The life cycle impact of customizations – the potential regressive impact, and resulting increase in the cost of sustenance – is by far the greater cost. That cost, too, is driven not so much by the number of missing features, but by the *mismatch type* and the underlying product architecture.
- Current methods fail to capture the true business impact of accepting a set of mismatches, or the "*Cost of Acceptance*". This cost is not simply the (weighted) mismatch score; it also depends on the level of the requirements hierarchy where those mismatches occur. Mismatches at a higher level, involving foundational requirements, will have a larger cost. The Cost of Acceptance also depends on the mitigation thereof. In the simplest case, the customer organization will stop doing something; then the Cost of Acceptance is simply the value of the lost function. In most cases, the organization will add a manual process, expand another function, or distribute work to another segment of the enterprise.

Current methods lack architectural orientation.

- While many methods mention "architectural reconciliation", there is insufficient detail on how such reconciliation may be pursued. Most of the current methods focus on requirements negotiation, and treat architectural characteristics or non-functional requirements (NFRs), as simply another group of requirements. But architectural characteristics *enable* multiple functions; architectural gaps, unlike functional ones, have a *multiplier* effect on the Cost of Ownership. Current methods do not treat architectural characteristics as enablers, and fail to account for this multiplier effect.
- Current methods do not have a *portfolio perspective*. Architectural characteristics influence IT portfolio convergence, and *ROI* of the organization's IT *portfolio*. For example, if the organization has invested substantially in LDAP services, absorption of a product that does not support LDAP integration will lead to portfolio divergence and diminished ROI. While COTS functionality is best viewed from a Line of Business (LoB) perspective, COTS architecture is best viewed from a portfolio (i.e. CIO) perspective. Then again, if the COTS is delivered as Software as a Service (SaaS) or as a service from a shared community Cloud, the customer organization need not have an equal interest in the underlying architecture, and IT portfolio convergence need not be an issue. Therefore, the COTS *delivery method*, of which there are several in industry today, becomes a factor in the selection process.

III. PROPOSED METHOD

In this paper, we propose a new method for COTS selection and life cycle management – **Architecture Centric Tradeoff (ACT)**. ACT is a decision support method for the entire COTS lifecycle, starting from COTS selection, and persisting through the Design, Build, Deploy and Maintain phases. The fundamentals of the ACT method remain unchanged through the life cycle, while the underlying model data is progressively refined. The salient features of ACT are:

1. ACT explicitly recognizes that COTS based system development is an *optimization, not a construction*, problem. The central object in ACT is the *Tradeoff. Matrix*, not the Requirement Traceability Matrix (RTM).
2. ACT supports a holistic *Cost of Ownership* perspective. In ACT, the Cost of Ownership is a function of the business-product mismatch. The mismatch can be assessed from multiple viewpoints, each resulting in one component of the Cost of Ownership. (Function and Technology are the fundamental viewpoints.) ACT seeks the minima for the Cost of Ownership function, i.e. the collection of Accept and Repair decisions that result in the lowest Cost of Ownership. *In ACT, the COTS product with the lowest minimum Cost of Ownership is selected, which may not necessarily be the product with the highest (weighted) feature score.*
3. ACT is an *architecture-centric process*. It goes beyond features, and explores structural aspects of businesses and products. ACT recognizes the multiplier effect of architectural characteristics such as extensibility. Through the technology viewpoint, the method supports IT portfolio convergence and ROI of IT investments. ACT explicitly recognizes the need for new COTS to leverage IT investments already made.

A. Model Organization and Relationships

ACT comprises of 3 parts:

- **Metamodel**: ACT uses a 3-layer metamodel, constructed in the Unified Modeling Language (UML). The first layer describes the conceptual model, the second the logical, and the third the physical.
- **Heuristics**: These process the model data to calculate the various components of the Cost of Ownership. The heuristics can be adjusted based on organizational assessments and COTS architecture reviews.
- **Processor**: The processor iterates through various candidate solutions, defined by the analyst, to find the optimal tradeoff.

ACT is founded on the *Evolutionary Process for Integrating COTS (EPIC)* [14], which itself is an extension of the Rational Unified Process (RUP). EPIC was developed by the Software Engineering Institute (SEI) at Carnegie Mellon.

- Like RUP, EPIC is *incremental, iterative and architecture-centric*. EPIC uses the well-formed artifacts (e.g. Use Cases) and the modeling language (Unified Modeling Language (UML)) of RUP. But while the constructs are the same, the focus is different. *RUP focuses on the progressive realization of a fixed set of requirements, while EPIC focuses on the systematic tradeoff of requirements and COTS capabilities.*
- EPIC represents a paradigm shift in COTS based system integration. In EPIC, business needs and COTS capabilities converge across multiple iterations. EPIC allows the understanding of requirements and COTS capabilities to evolve along the life cycle. Because it is tradeoff oriented rather than requirements oriented, EPIC reduces risk, decreases cost and facilitates use of delivered capabilities. *Importantly, it also transforms system integration into an optimization problem, which, in the traditional approach, it is not.*

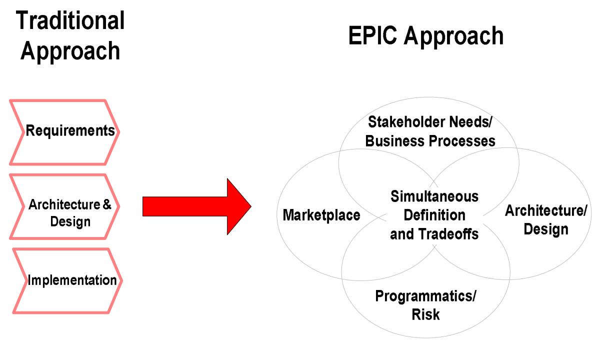


Figure 1. Evolutionary Process for Integrating COTS

ACT builds on the EPIC process framework. ACT quantifies the tradeoffs in EPIC, and facilitates the iterative convergence of function, technology and COTS. EPIC is a broad process framework, and does not say how tradeoffs should be calculated and managed. *ACT takes EPIC from theory to practice; it enables tradeoff oriented COTS program management, governance and tool development.*

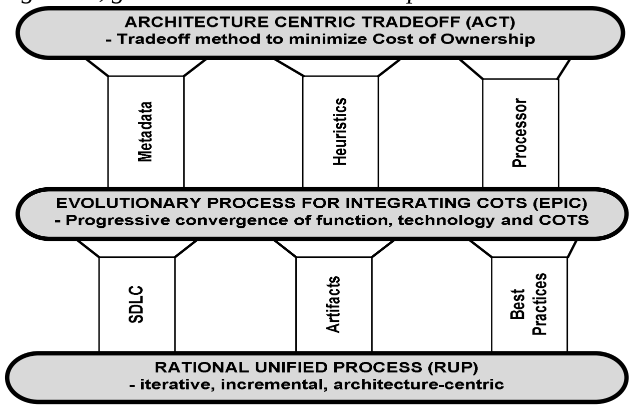

ARCHITECTURE CENTRIC TRADEOFF (ACT)
- Tradeoff method to minimize Cost of Ownership
Metadata
Heuristics
Processor
EVOLUTIONARY PROCESS FOR INTEGRATING COTS (EPIC)
- Progressive convergence of function, technology and COTS
SDLC
Artifacts
Best Practices
RATIONAL UNIFIED PROCESS (RUP)
- iterative, incremental, architecture-centric

Figure 2. ACT Model Organization

B. *Model Metadata*

ACT is a repository based method. The repository metadata describes the entities in the model and the relationships between them. Inferentially, it defines the boundaries of the model and the subset of problems it can solve. The metadata is in 3 layers - conceptual, logical and physical. The inheritance hierarchy of ACT, as shown the figure below, allows it to work with multiple products, businesses, SDLCs and EA frameworks, while maintaining the same core metadata.

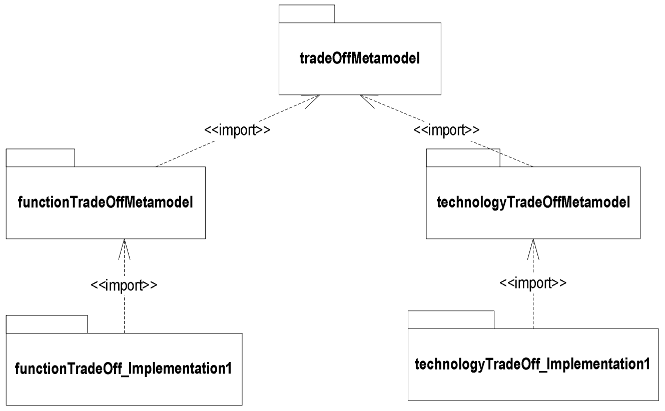

tradeOffMetamodel
<<import>>
<<import>>
functionTradeOffMetamodel
technologyTradeOffMetamodel
<<import>>
<<import>>
functionTradeOff_Implementation1
technologyTradeOff_Implementation1

Figure 3. ACT 3-Layer Metamodel

Layer 1 Metadata describes the core concepts.

- "**Enterprise**" is a unit (company, agency, department…) that does, provides or supports "things of value". The enterprise is structured as a hierarchy, with the "relative importance" at each node distributed amongst lower nodes. Function and technology are the two fundamental hierarchies.
- "**Mismatch**" is where the Enterprise is not fully supported by (or does not have) a Product Context. Mismatch can be full or partial. Each mismatch is traced to a specific node in the Enterprise hierarchy, with preference for the lowest possible node, and is fully distributed to "Accept" or "Repair".
- "**Accept**" is where the enterprise needs to do something differently, or stop doing something. "**Cost of Acceptance**" measures the impact to the Enterprise. "Cost of Acceptance" derives from the size and type of the acceptance, the nodes in the Enterprise which it affects, and organizational factors, which may, in turn, inherit from the business domain. Note that a mismatch in one node may have to be resolved by doing things differently at other nodes. This situation is common when consolidating enterprises on a single COTS.
- "**Repair**" is where the product needs to be changed to support the enterprise. "**Cost of Repair**" measures the impact to the life cycle cost of ownership. "Cost of Repair" derives from the size and type of the repair, the nodes in the product context which it affects, and product technology factors, which may, in turn, inherit from the technology domain.

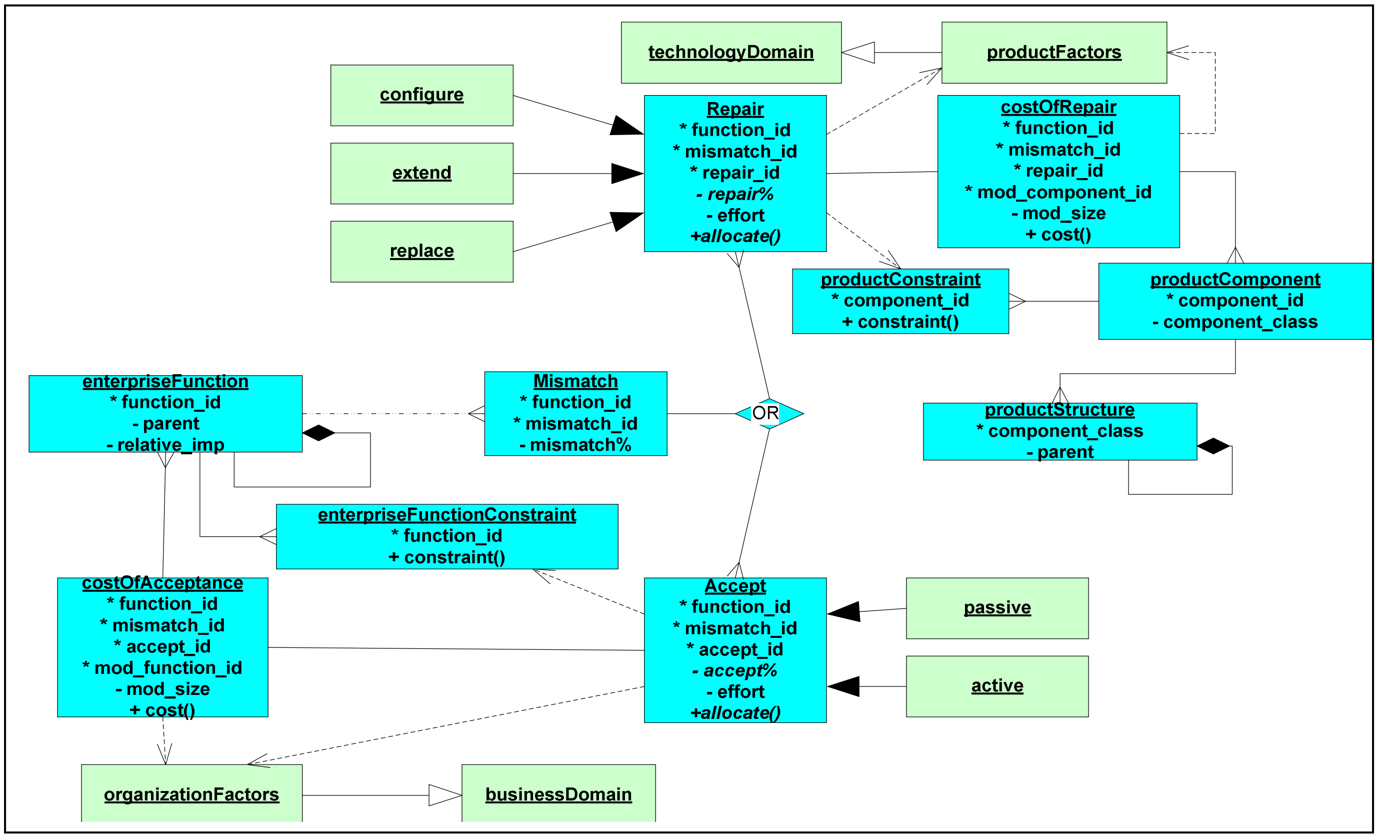


Figure 4. ACT Layer 2 Metadata

Layer 2 Metadata implements the layer 1 concepts as logical constructs.

- "**Enterprise Function**" in Layer 2 implements "Enterprise" from Layer 1. The Enterprise Function hierarchy contains the functional decomposition for the Enterprise. The "relative importance" at each node is a product of the "relative importance"s along the path to that node.
- **Product Structure** is a function-oriented decomposition of the relevant section of the COTS product. (There is no need to model the entire COTS.) Product Component relates specific repair candidates to the Product Structure.
- "**Accept**" is extended into its subtypes – "Active" and "Passive". Passive is where the enterprise stops performing a function e.g. stops selling a product, because the COTS does not support it. Active is where the enterprise reorganizes work, adds manual processes, trains employees or adds compensating controls, to resolve a mismatch. The available subtypes for a function/mismatch depend on the Enterprise Constraints. For example, Passive will not be available for mandatory functions. "**Cost of Acceptance**" is influenced by the type of the Acceptance, and the nodes in the function hierarchy that are impacted by the Acceptance. Cost of Acceptance is also influenced by *organizational factors*.
- "**Repair**" is extended into its subtypes – Configure, Extend and Replace. *Configure* is where only certain literals that drive product behavior (i.e. configuration data or settings) need to be changed. *Extend* is where components may be extended to provide new functionality without modifications to the delivered COTS metadata, such that there is no potential for regressive impact to adjunct components. *Replace* is where there are modifications to the COTS metadata, and thereby potential regressive impact or loss of upgradeability. The subtypes available depend on the *Product Constraints*. Where only certain components are exposed through APIs, for example, the Extend subtype is available only for those components. "**Cost of Repair**" is influenced by the type of the Repair, and the level in the component hierarchy where the changes are taking place. For example, changes at the structural layer (e.g. database schema) will have a greater Cost of Repair than that of changes at the presentation layer (e.g. JSP pages).

Layer 3 describes specific implementations; the logical constructs of layer 2 are implemented for a specific business and candidate COTS. Key activities include formation of the function hierarchy and product structure. In addition to Function, Technology is another fundamental viewpoint. The technology hierarchy, also known as the *Technology Reference Model*, shows the relevant technology platforms and services within the organization. The relative importance at each node indicates the technology investment at that node. The principle is that ROI targets can be achieved by acquiring COTS that leverage IT investments already made. However, if the COTS is provided as SaaS or Cloud service, Technology ceases to be a viewpoint, and the choice of COTS may be made based on the Function viewpoint alone.

C. Model Heuristics

ACT is a quantitative model for COTS decision support, and the ACT metadata is structured to support computation and optimization. Much of the computations that follow are simple SQL operations on the ACT database. The model has to be calibrated, and the coefficients established, for each organization and COTS implementation program. (The costs shown are notional, and are meant to support relative comparisons, not absolute dollar estimates.)

Cost of Ownership

$$\mathbf{C\_own = C\_accq + {}_{\rho}\sum W_{\rho}*(C\_acpt_{\rho} + C\_repr_{\rho})}$$

• C_own	: Cost of ownership
• C_accq	: Cost of acquisition (or licensing cost)
• ρ	: Viewpoint (Function/Technology)
• W	: Weight assigned to viewpoint ρ
• C_acpt	: Cost of acceptance (from viewpoint ρ)
• C_repr	: Cost of repair (from viewpoint ρ)

Cost of Acceptance

The Cost of Acceptance has passive and active constituents. Passive is where the enterprise stops performing a function because of a mismatch. Active is where the enterprise reorganizes work, adds manual processes, trains employees or adds compensating controls, to resolve a mismatch.

$$\mathbf{C\_acpt = C\_acpt\_passive + C\_acpt\_active}$$

$$\mathbf{C\_acpt\_passive = \prod_i EM\_flex_i * {}_m\sum F_m*FM_m*MA_m*A_m*AP_m}$$

- EM_flex: Effort multipliers related to the (lack of) organizational flexibility
- m: Mismatch pointer
- F : Relative importance of the node where there’s a mismatch
- FM: The percentage of mismatch
- MA: The percentage of the Mismatch that is Accept
- A: The size of the acceptance.
- AP: The percentage of Accept that is Passive.

$$\mathbf{C\_acpt\_active=\prod_i EM\_adapt_i*{}_m\sum A_m*(1-AP_m)*\{{}_n\sum F\_mod_n*mod\_size_n\}}$$

- EM_adapt: Effort multipliers related to the (lack of) organizational adaptibility
- m: Mismatch pointer
- n: Modified function pointer
- F_mod: Relative importance of the node modified
- mod_size: Size of the modification

Effort Multipliers

Organizational flexibility: These multipliers quantify the constraints that impede an organization from dropping a function in the event of a mismatch. Where these constraints are severe, e.g. where the organization is bound to fulfill certain functions by law, the EM_flex will be high, and Passive acceptance will lead to an unsustainable Cost of Acceptance.

Organizational adaptability: These multipliers quantify constraints that impede an organization from reorganizing work or workforce to resolve a mismatch. Where these constraints are severe, e.g. where there are restrictions on hiring, the EM_adapt will be high.

Cost of Repair

The Cost of Repair comprises of the *Cost of Configuration, Cost of Extension* and *Cost of Replacement*. The product architecture, and the technology domain from which it inherits, will place limits on what options are available for a given repair.

$$\mathbf{C\_repr = C\_cfg + C\_extn + C\_repl}$$

$$\mathbf{C\_cfg = \prod_i EM\_cfg_i* {}_r\sum \{R*RC_r *(1 + {}_n\sum mod\_size_n^{1/\lambda})\}}$$
$$\mathbf{C\_extn=\prod_i EM\_extn_i*{}_r\sum\{R*RE_r *(1+ {}_n\sum mod\_size_n^{1/\lambda})\}}$$
$$\mathbf{C\_repl=\prod_i EM\_repl_i*{}_r\sum\{R*RR_r *(1 + \eta*{}_n\sum mod\_size_n^{1/\lambda})\}}$$

- EM_cfg: Effort multipliers related to configurability
- EM_extn: Effort multipliers related to extensibility
- EM_repl: Effort multipliers related to overwrite
- r: Repair pointer
- n: Modified component pointer
- R: Size of the repair
- RC, RE, RR: Percentages of Repair that are Configuration, Extension or Replace
- mod_size: Size of the modification to the component, measured as Source Lines of Code (SLOC), etc.
- λ: The level of the modified component in the hierarchy. More foundational the level (lower λ), higher the effective size of the modification.
- η: The number of upgrades expected during the product life cycle, e.g. for a product life cycle of 10 years, where the vendor releases 2 upgrades a year, η will be 20.

Effort Multipliers

These effort multipliers reflect the product’s architectural characteristics.

Configurability:

- (Lack of) Configuration wizards or utilities
- (Lack of) Configurable rules engine and field edits
- (Lack of) Modularization or distinct inter-module interfaces

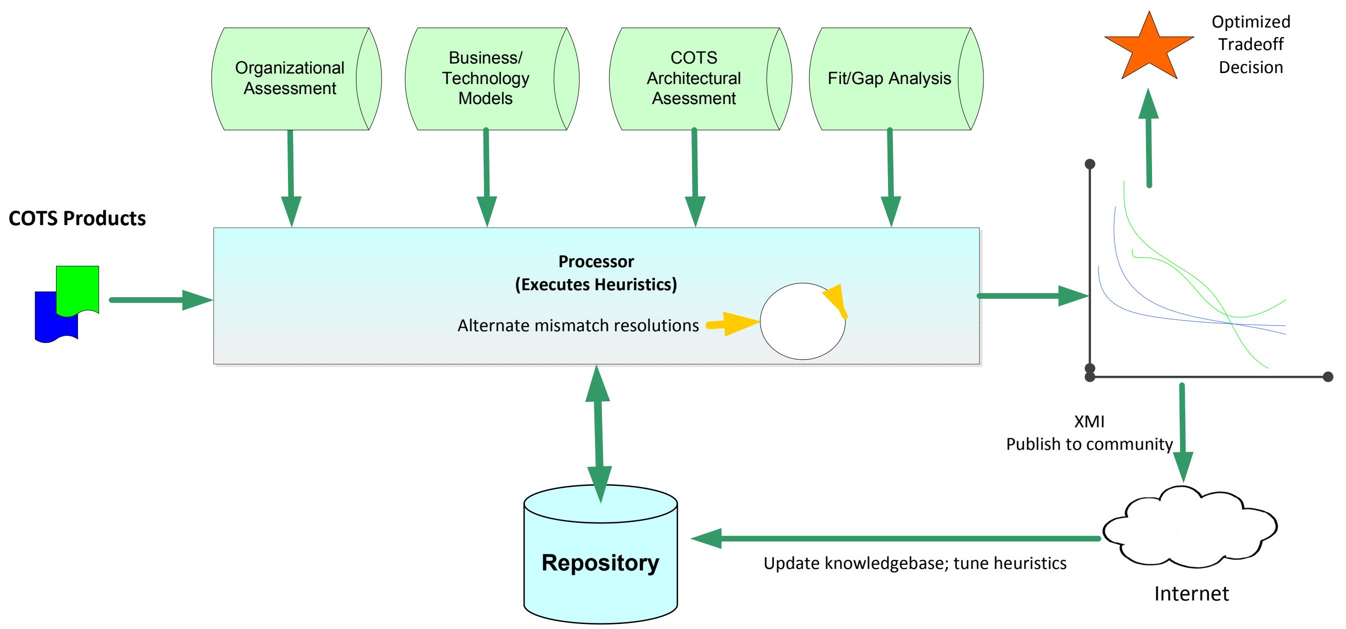

Organizational Assessment
Business/ Technology Models
COTS Architectural Assessment
Fit/Gap Analysis
Optimized Tradeoff Decision
COTS Products
Processor (Executes Heuristics)
Alternate mismatch resolutions
XMI Publish to community
Repository
Update knowledgebase; tune heuristics
Internet

Figure 5. The ACT Processor

Extensibility:

- (Lack of) Integrated Development Environment (IDE), or support for common IDEs (e.g. Eclipse).
- Proprietary programming languages; (Lack of) Support for Java, C++, etc; (Lack of) Object orientation
- (Lack of) Published APIs, user exits and extensible abstract classes

Replacebility:

- (Lack of) Integrated Development Environment (IDE), or support for common IDEs (e.g. Eclipse)

D. *Model Processor*

Several iterations take place in the COTS life cycle, some before COTS selection, and some after. The processing sequence for a given iteration during the COTS evaluation phase is as follows.

1. The metadata is populated with organizational assessment results, business and technology hierarchies, and model scaling factors and coefficients.
2. COTS products are fed into the processor. For each COTS, the metadata is populated with results of fit/gap and architectural assessments.
3. The Processor is run with alternate sets of mismatch resolutions, and Cost of Acceptance and Cost of Repair is calculated for each set. The Processor outputs performance profiles for each COTS, which graphs the Cost of Ownership against the level of repair.
4. Finally, an Optimized Tradeoff Decision is reached, i.e. a COTS product at *a specific repair level* is selected.

Figure 6 shows 2 COTS – B has a better feature match, while A is more extensible. The minima of the Cost of Ownership function is lower for A than for B. Most current COTS methods will select B; ACT will select A.

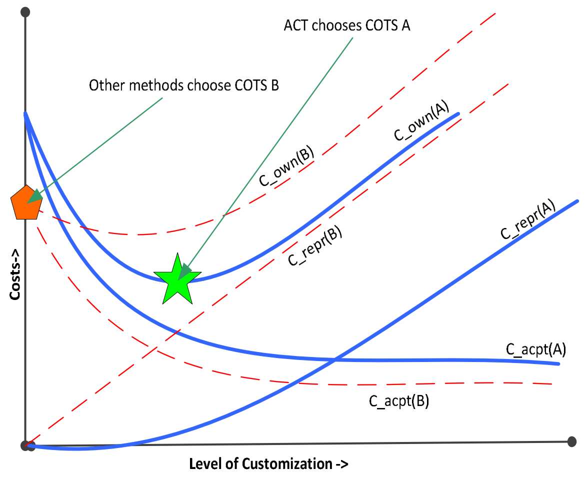


Figure 6. Example of ACT Processor output

IV. CONCLUSION AND FUTURE WORK

ACT provides a comprehensive framework for COTS selection, implementation and governance.

Benefits to the customer organization include:

- Enables a true Cost of Ownership perspective.
- Enables COTS selection at optimal customization, rather than as delivered.
- Costs of Acceptance and Repair are adjusted to organization and product.
- Emphasizes architectural capabilities rather than features; exposes what matters in the long run.
- Explicit shift from requirements to tradeoff collapses project schedules, cost and risk.
- Enables IT portfolio convergence and ROI.

Benefits to the system integrator include:

- Standard analytical model for COTS/business convergence reduces project risk.
- Quantitative model facilitates client communications and Business Process Engineering (BPR) negotiation.
- Common metadata and process model can be reused across engagements. Continual refinement of models, based on engagement feedback.

Future work in this area will include the formation of an adjunct framework to assess configurability and extensibility of COTS products. Future work will also include establishing UML profiles for the metamodel, and finally, the calibration of the model over multiple iterations in the field.

V. REFERENCES

[1] R. O'Callaghan,*“Technology Diffusion and Organizational Transformation: An Integrative Framework”*, Idea Group Publishing, 1998.

[2] M.L. Markus, *“Paradigm Shifts - E-Business and Business/Systems Integration”*, Communications of the AIS, 4 (10), 2000.

[3] T.H. Davenport, *“Mission Critical: Realizing the Promise of Enterprise Systems”*, Harvard Business School Press, 2000.

[4] J. Kontio, *“OTSO: A Systematic Process for Reusable Software Component Selection”*, Univ. of Maryland report CS-TR-3478, 1995.

[5] C. Ncube and N. A. Maiden, *“PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm”*, Second International Workshop on Component-Based Software Engineering, Los Angeles, 1999.

[6] C. Alves and J. Castro, *“CRE: a systematic method for COTS components Selection”*, XV Brazilian Symposium on Software Engineering (SBES), Rio de Janeiro, 2001.

[7] L. Chung and K. Cooper, *“Defining Goals in a COTS-Aware Requirements Engineering Approach”*, Systems Engineering, 7(1), Wiley, 2004.

[8] A. Mohamed, G. Ruhe and A. Eberlein, *"Decision Support for Handling Mismatches between COTS Products and System Requirements”*, ICCBSS'07, Banff, 2007.

[9] A. Mohamed, G. Ruhe and A. Eberlein, *"COTS Selection: Past, Present, and Future", ECBS'07,*Tucson, Arizona, 2007.

[10] M. Torchiano and M. Morisio, *“Overlooked Aspects of COTS-Based Development”*, IEEE Software, 2004.

[11] G. Ruhe, *"Intelligent Support for Selection of COTS Products"*, LNCS, Springer, vol. 2593, 2003.

[12] J. Kontio, G. Caldiera, and V. R. Basili, *"Defining factors, goals and criteria for reusable component evaluation", CASCON'96,* Toronto, 1996.

[13] C. Ncube and J. C. Dean, *"The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Components”*, *ICCBSS*, 2002.

[14] C. Albert and L. Brownsword, “*Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview*”, (CMU/SEI-2002-TR-009), Software Engineering Institute, Carnegie Mellon University, 2002.

Implementation of Business Processes in Service Oriented Architecture

Krzysztof Sacha and Andrzej Ratkowski
Warsaw University of Technology
Warszawa, Poland
{k.sacha, a.ratkowski}@ia.pw.edu.pl

***Abstract*—The paper develops a method for transformational implementation of business processes in a service oriented architecture. The method promotes separation of concerns and allows making business decisions by business people and technical decisions by technical people. To achieve this goal, a description of a business process designed by business people is automatically translated into a program in Business Process Execution Language, which is then subject to a series of transformations developed by technical people. The transformations are selected manually and executed by an automatic tool. Each transformation changes the process structure to improve the quality characteristics. The method applies a correct-by-construction approach and defines a set of transformations, which do not change the process behavior. The quality of the process implementation is assessed using a set of metrics.**

Keywords-business process; BPEL language; service oriented architecture; SOA; transformational implementation.

I. INTRODUCTION

A business process is a set of logically related activities performed to achieve a defined business outcome [1]. The structure of a business process and the ordering of activities reflect business decisions made by business people and, when defined, can be visualized using an appropriate notation, e.g., Business Process Model and Notation [2] or the notation of ARIS [3]. The implementation of a business process on a computer system is expected to exhibit the defined behavior at a satisfactory level of quality. Reaching the required level of quality may need decisions, made by technical people and aimed at restructuring of the initial process in order to benefit from the characteristics offered by an execution environment. The structure of the implementation can be described using another notation of, e.g., Business Process Executable Language [4] or UML activity diagrams [5].

This paper describes a transformational method for implementing business processes in a service oriented architecture (SOA). The method begins with an initial definition of a business process, written by business people using Business Process Modeling Notation (BPMN). The business process is automatically translated into a program in Business Process Executable Language (BPEL), called a reference process. The program is subject to a series of transformations, each of which preserves the behavior of the reference process, but changes the order of activities, as means to improve the quality of the process implementation, e.g., by benefiting from the parallel structure of services. Transformations applied to the reference process are selected manually by human designers (technical people) and performed automatically, by a software tool. When the design goals have been reached, the iteration stops and the result is a transformed BPEL process, which can be executed on a target SOA environment.

Such an approach promotes separation of concerns and allows making business decisions by business people and technical decisions by technical people.

A critical part of the method is providing assurance on the correctness of the transformation process. In this paper we apply a correct-by-construction approach, and define a set of safe transformations, which do not change the process behavior. If each transformation is safe, the resulting program will also be correct, i.e., semantically equivalent to the original reference process.

The rest of this paper is organized as follows. Related work is briefly surveyed in Section II. The semantics of a BPEL process and its behavior are defined in Section III. An illustrative case study is provided in Sections IV and VI. Safe transformations are introduced in Section V. Quality metrics to assess transformation results are described in Section VII. Conclusions and plans for future research are given in Section VIII.

II. RELATED WORK

Transformational implementation of software is not a new idea. The approach was developed many years ago within the context of monolithic systems, with the use of several executable specification techniques. The formal foundation was based on problem decomposition into a set of concurrent processes, use of functional languages [6] and formal modeling by means of Petri nets [7].

An approach for transformational implementation of business processes was developed in [8]. This four-phase approach is very general and not tied to any particular technology. Our method, which can be placed in the fourth phase (implementation), is much more specific and focused on the implementation of runnable processes described in BPMN and BPEL.

BPMN defines a model and a graphical notation for describing business processes, standardized by OMG [2]. The reference model of SOA [9,10] and the specification of BPEL [4] are standardized by OASIS. An informal mapping of BPMN to BPEL was defined in [2] and a comprehensive discussion of the translation between BPMN and BPEL can be found in [11,12]. An open-source tool is available for download at [20].

The techniques of building program dependence graph and program slicing, which we adopted for proving safeness of transformations, were developed in [13,14] and applied to BPEL programs in [15].

Quality metrics to measure parallel programs have been studied for many years. A traditional tool for measuring performance of a parallel application is Program Activity Graph, which describes parallel flow of control within the application [16]. We do not use this graph, nevertheless, our metrics Length of thread and Response time can be viewed as an approximation of Critical path metric described in [16]. Similarly, our Number of threads metric is similar to Available concurrency defined in [17].

Our work on the implementation of business processes in a service oriented architecture is to the best of our knowledge, original. An early version of our approach was published in [18]. A definition of safeness, an extended set of transformations, the proofs of transformation safeness, a revised algorithm for building program dependence graph and performance metrics are introduced in this paper.

III. The Semantics of a Business Process

A business process is a collection of logically related activities, performed in a specific order to produce a service or product for a customer. The activities can be implemented on-site, by local data processing tasks, or externally, by services offered by a service-oriented environment. The services can be viewed from the process perspective as the main business data processing functions.

A specification of a business process can be defined textually, e.g., using a natural language, or graphically, using Business Process Modeling Notation. An example BPMN process, which shows a simplified processing of a bank transfer order is shown in Fig. 1. The process begins, and waits for an external invocation from a remote client (another process). When the invocation is received, the process extracts the source and the target account numbers from the message, checks the availability of funds at source and splits into two alternative branches. If the funds are missing, the process prepares a negative acknowledgement message, replies to the invoker and ends. Otherwise, the alternative branch is empty. Then, the process invokes the withdraw service at source account, invokes the deposit service at target account, packs the results returned by the two services into a single reply message, replies to the invoker and ends. This way, the process implements a service, which is composed of another services.

BPMN specification of a business process can be automatically translated into a BPEL program, which can be used for a semi-automatic implementation.

BPEL syntax is composed of a set of instructions, called activities, which are XML elements indicated in the document by explicit markup. The set of BPEL activities is rich. However, in this paper, we focus on a limited subset of activities for defining control flow, service invocation and basic data handling.

The body of a BPEL process consists of simple activities, which are elementary pieces of computation, and structured elements, which are composed of other simple or structured activities, nested in each other to an arbitrary depth. Simple activities are <assign>, which implements substitution, <invoke>, which invokes an external service, and <receive>, <reply> pair, which receives and replies to an invocation. Structured activities are <sequence> element to describe sequential execution, <flow> element to describe parallel execution and <if> alternative branching. An example BPEL program, which implements the business process in Fig. 1, is shown in Fig. 2. Name attribute will be used to refer to particular activities of the program in the subsequent figures.

The first executable activity of the program is <receive>, which waits for a message that invokes the process execution and conveys a value of the input argument. The last activity of the process is <reply>, which responds to the invocation and sends a message that returns the result. The activities between <receive> and <reply> execute a business process, which invokes other services and transforms the input into the output. This is a typical construction of a BPEL process, which can be viewed as a service invoked by other services.

SOA services are assumed stateless [19], which means that the result of a service execution depends only on values of data passed to the service at the invocation, and manifests to the outside world as values of data sent by the service in response to the invocation. Therefore, we assume that the observable behavior of a process in a SOA environment consists of data values, which the process passes as arguments when it invokes external services, and data values, which it sends in reply to the invoker.

To capture the influence of a process structure into the process behavior, we use a technique called program slicing [13,14], which allows finding all the instructions in a program, which influence the value of a variable in a specific

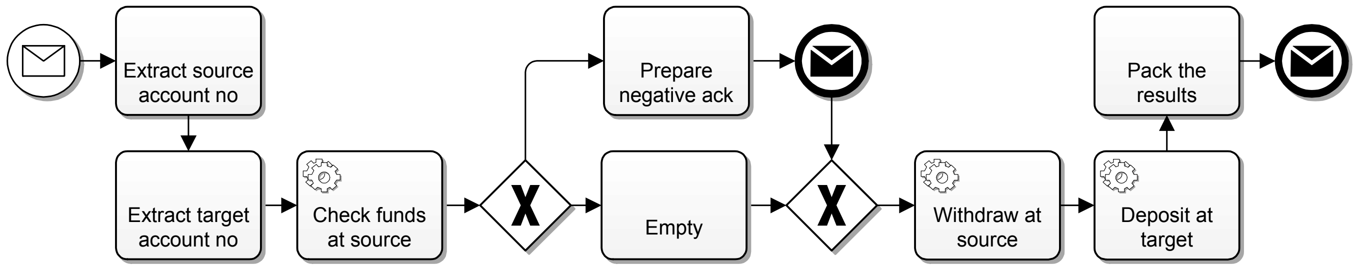


Figure 1. BPMN specification of a business process

point of the program. For example, finding the instructions, which influence the value of a variable that is used as an argument by a service invocation activity or by a reply activity of the process.

The conceptual tool for the analysis is Program Dependence Graph (PDG), which nodes are activities of a BPEL program, and edges reflect dependencies between the activities. An algorithm for constructing PDG of a BPEL program consists of the following steps:

1. Define nodes of the graph, which are activities at all layers of nesting.
2. Define control edges (solid lines in Fig. 3), which follow the nested structure of the program, e.g., an edge from <sequence> to <if> shows that <if> activity is nested within the <sequence> element. Output edges of <if> node are labeled "Yes" or "No", respectively.
3. Define data edges (dashed lines in Fig. 3), which reflect dataflow dependencies between the activities, e.g., an edge from activity "rcv" to activity "src" shows that an output variable of "rcv" is used as input variable to "src".
4. Convert "Yes" and "No" edges that output <if> activities into data edges (Fig. 3).
5. Add data edges from <receive> activity, which is nested within a <sequence> element, to each subsequent activity of this <sequence> such that no paths from <receive> to this activity exists (there are no such items in Fig. 3).

Data edges within a program dependence graph reflect the dataflow dependencies between activities, which determine values of the program variables. Data edges added in step 5 reflect the semantics of the process as a service, which starts after receiving an invocation message. The flow of control within a BPEL program complies with data edges of its program dependence graph.

In the rest of this paper we adopt a definition that a transformation preserves the process behavior, if it keeps the set of messages sent by the process as well as the data values carried by these messages unchanged. Such a definition neglects the timing aspects of the process execution. This is justified, given that it does not change the business requirements. There are many delays in a SOA system and the correctness of software must not relay on a specific order of activities, unless they are explicitly synchronized.

A transformation, which preserves the process behavior will be called safe.

Definition (Safeness of a transformation)

A transformation is safe, if the set of messages sent by the activities of a program remains unchanged and the flow of control within the transformed program complies with the direction of data edges within the program dependence graph of the reference process. □

The set of activities executed within a program may vary, depending on decisions made when passing through decision points of <if> activities. To fulfill the above definition, the set of messages must remain unchanged, for each particular combination of these decisions.

A path composed of data edges in a program dependence graph reflects the data flow relationships between the activities, and implies that the result of the activity at the end of the path depends only on the preceding activities on this path. If the succession of activities executed within a program complies with the data edges, then the values of variables computed by the program remain the same, regardless of the ordering of other activities of this program.

Safeness of a transformation guarantees that the transformation preserves the behavior of the transformed program as observed by other services in a SOA environment. Safeness is transitive and a sequence of safe transformations is also safe. Therefore, a process resulting from a series of safe transformations applied to a reference process preserves the behavior of the reference process.

IV. CASE STUDY

Consider a process of transferring funds between two different bank accounts, shown in Fig. 1, implemented by a BPEL process.

```
<sequence>
   <receive name="rcv" variable="transfer"/>
   <assign name="src">
      <copy> <from variable="transfer" part="srcAccNo"/>
      <to variable="source" part="account"/> </copy>
      <copy> <from variable="transfer" part="srcAmount"/>
      <to variable="source" part="amount"/> </copy>
   </assign>
   <assign name="dst">
      <copy> <from variable="transfer" part="dstAccNo"/>
      <to variable="target" part="account"/> </copy>
      <copy> <from variable="transfer" part="dstAmount"/>
      <to variable="target" part="amount"/> </copy>
   </assign>
   <invoke name="verify" inputVariable="source"
      outputVariable="fundsAvailable"/>
   <if> <condition> $fundsAvailable.res </condition>
      <empty name="empty"/>
   <else> <sequence>
      <assign name="fail">
         <copy> <from> 'lack of funds' </from>
         <to variable="response" part="fault"/> </copy>
      </assign>
      <reply name="nak" variable="response"/>
      <exit name="exit"/>
   </sequence> </else> </if>
   <invoke name="withdraw" inputVariable="source"
         outputVariable="wResult"/>
   <invoke name="deposit" inputVariable="target"
         outputVariable="dResult"/>
   <assign name="success">
      <copy> <from variable="wResult" part="res"/>
      <to variable="result" part="withdraw"/> </copy>
      <copy> <from variable="dResult" part="res"/>
      <to variable="result" part="deposit"/> </copy>
   </assign>
   <reply name="ack" variable="result"/>
</sequence>
```

Figure 2. A skeleton of a BPEL program of a bank transfer (Fig. 1)

The process body is a sequence of activities, which starts at <receive>. Then, it proceeds through a series of steps to process the received bank transfer order and to invoke services offered by the banking systems to verify availability of funds at source account, to withdraw funds and to deposit the funds at the destination account. Finally, it ends after replying positively, if the transfer has successfully been done, or negatively, if the required amount of funds was not available at source. A skeleton of the simplified BPEL program of this process is shown in Fig. 2.

PDG of this program is shown in Fig. 3. The first two <*assign*> activities process the contents of the received message in order to extract the source and destination account numbers and the amount of money to transfer. Therefore, there are data edges from "rcv" to "src" and to "dst" nodes in PDG. The next consecutive <invoke> activity uses the extracted source account number and the amount of money to invoke the verification service, and the response of the invocation is checked by <if> activity. Therefore, two data edges from *src* to *verify* and from *verify* to <if> exist in the graph. Similarly, the <invoke> activities named "withdraw" and "deposit" use the account numbers calculated by "src" and "dst", respectively. Two data edges from "withdraw" and "deposit" nodes to "success" node, and then an edge from "success" to "ack", reflect the path of preparing the acknowledgement message that is sent to the invoker when the transfer is finished.

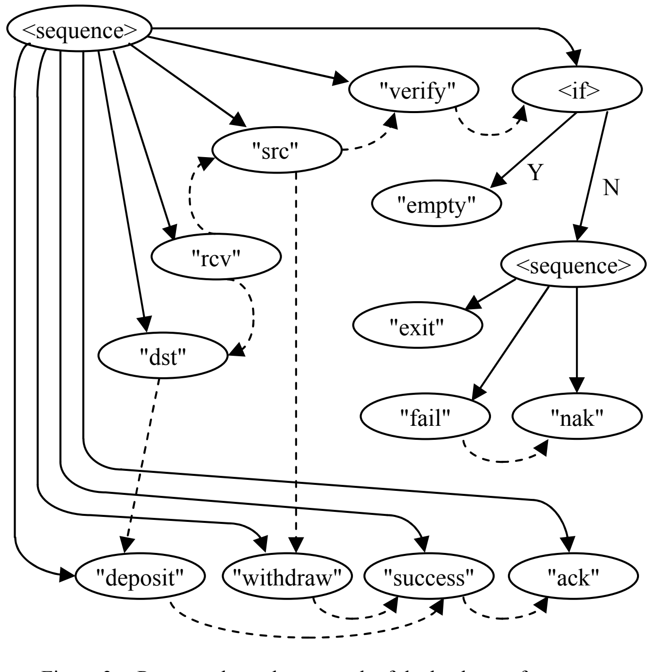


Figure 3. Program dependence graph of the bank transfer process

V. Transformations

The body of a BPEL process consists of simple activities, e.g., <assign>, which define elementary pieces of computation, and structured elements, e.g., <flow>, which is composed of other simple or structured activities. The behavior of the process results from the order of execution of activities, which stem from the type of structured elements and the positioning of activities within these elements. A transformation applies to a structured element and consists in replacing one element, e.g., <flow>, by another element, e.g., <sequence>, or in relocation of activities within the structured element. If the behavior of the transformed element before and after the transformation is the same, then the behavior of the process stands also unchanged.

Several transformations have been defined. The basic ones: Simple and alternative displacement, parallelization and serialization of the process operations, and process partitioning are described in detail below. All the transformations are safe, according to definition of safeness given in Section III. As pointed out in Section III, a safe transformation does not change the behavior of a process, which is composed of stateless services. A problem may arise, if the services invoked by a process have an impact on the real world. If this is the case, a specific ordering of these services may be required. In our approach, a designer can express the necessary ordering conditions adding supplementary edges to the program dependence graph.

Transformation 1: Simple displacement

Consider a <sequence> element, which contains n arbitrary activities executed in a strictly sequential order. Transformation 1 moves a selected activity A from its original position i, into position j within the sequence.

***Theorem* 1.** Transformation 1 is safe, if no paths between activity A and the activities placed on positions i+1, … j in the sequence existed in the program dependence graph of the transformed program.

Proof: Assume that $i < j$ (move forward). The transformation has no influence on activities placed on positions lower than i or higher than j. However, moving activity A from position i to j reverts the direction of the flow of control between A and the activities that are in-between. This could be dangerous if a data flow from A to those activities existed. However, if no data paths from A to the activities placed on positions i+1, … j existed in the program dependence graph, then no inconsistency between the control and data flow can exist.

If $j < i$ (move backward), the proof is analogous. The lack of data path guarantees lack of inconsistency between the data and control flows within the program. □

Transformation 2: Pre-embracing

Consider a <sequence> element, which includes an <if> element preceded by an <assign> activity, among others. Branches of <if> element are <sequence> elements. Transformation 2 moves <assign> activity from its original position in the outer <sequence>, into the first position within one branch of <if> element.

***Theorem* 2.** Transformation 2 is safe, if neither a path from the moved <assign> to an activity placed in the other branch of <if>, nor a path from the moved <assign> to the activities positioned after <if> in the outer sequence, existed in the program dependence graph of the transformed program.

Proof: The transformation has no influence on activities placed prior to <if> element in the outer <*sequence*>. Moving <assign> activity to one branch of <if> removes the flow of control from <assign> to activities in the other branch of <if> and – possibly – to activities placed after

<if>. But according to the assumption of this theorem, there is no data flow between these activities. Therefore, no inconsistency between the control and data flow can exist. □

Transformation 3: Post-embracing

Consider a <sequence> element, which includes an <if> activity followed by a number of another activities. Branches of <if> element are <sequence> elements, one of which contains <exit> activity. Transformation 3 moves the activities, which follow <if>, from its original position in the outer <sequence> into the end of the second <sequence> of <if> element.

***Theorem* 3.** Transformation 3 is safe.

Proof: Activities, which are placed after an <if> element in the reference process, are executed only after the execution of <if> is finished. The existence of <exit> in one branch of <if> prevents execution of these activities when this branch is selected. The activities are executed only in case the other branch is selected. Therefore, neither the flow of control nor the flow of data is changed in the program, when the activities are moved to the other branch of <if>, i.e., the branch without <exit> activity. □

Transformation 4: Parallelization

Consider a <sequence> element, which contains *n* arbitrary activities executed in a strictly sequential order. Transformation 4 parallelizes the execution of activities by replacing <sequence> element by <flow> element composed of the same activities, which – according to the semantics of <flow> – are executed in parallel.

***Theorem* 4.** Transformation 4 is safe, if for each pair of activities A_i, A_j neither a path from A_i to A_j nor a path from A_j to A_i existed in the program dependence graph of the transformed program.

Proof: The transformation changes the flow of control between the activities of the transformed element. The lack of data paths between these activities means that no inconsistency between the control and data flow can exist. □

Transformation 5: Serialization

Consider a <flow> element, which contains *n* arbitrary activities executed in parallel. Transformation 5 serializes the execution of activities by replacing <flow> element by <sequence> element, composed of the same activities, which are now executed sequentially.

***Theorem* 5.** Transformation 5 is safe.

Proof: The proof is obvious. Parallel commands can be executed in any order, also sequentially.

Transformation 6: Asynchronization

Consider a two-way <invoke> activity, which sends a message to invoke an external service and then waits for a response (Fig. 4a). Transformation 6 replaces the two-way <invoke> activity with a sequence of a one-way <invoke> activity followed by a <receive> (Fig. 4b). This way, a synchronous invocation of a service is converted into an asynchronous one.

Transformation 6 can be proved safe, if we add a data edge from <invoke> node to <receive> node in the program dependence graph of each program, which includes an asynchronous service invocation shown in Fig. 4b.

***Theorem* 6.** Transformation 6 is safe.

```
<invoke name="xxx"                                        (a)
    inputVariable="source"  outputVariable="target"
/>

<sequence>                                                (b)
    <invoke name="xxx_i"  inputVariable="source"/>
    <receive name="xxx_r"  variable="target"/>
</sequence>
```

Figure 4. Synchronous (a) and asynchronous service invocation (b)

Proof: The transformation has no influence on activities executed prior to <invoke> activity. Data edges from these activities to <invoke> remain unchanged. The transformation has no influence on activities executed after <invoke>, as well. Data edges to these activities from <invoke> are redirected to begin at node <receive>. Hence, there is a one-to-one mapping between the sets of data paths, which exist in program dependence graph of a program before and after the transformation. Therefore, no inconsistency between the control and data flow can exist.

Transformations 1 through 6 can be composed in any order, resulting in a complex transformation of the process structure. Transformations 7 and 8 play an auxiliary role and facilitate such a composition. These transformations are safe, because they do not change the order of execution of any activities within a BPEL program. □

Transformation 7: Sequential partitioning

Transformation 7 divides a single <sequence> element into a nested structure of <sequence> elements (Fig. 5a).

Transformation 8: Parallel partitioning

Transformation 8 divides a single <flow> element into a nested structure of <flow> elements (Fig. 5b).

```
<sequence>          (a)          <flow>               (b)
    <sequence>                       <flow>
        <C1> </C1>                       <C1> </C1>
        ......                           ......
        <Ck> </Ck>                       <Ck> </Ck>
    </sequence>                      </flow>
    <sequence>                       <flow>
        <Ck+1> </Ck+1>                   <Ck+1> </Ck+1>
        ......                           ......
        <Cn> </Cn>                       <Cn> </Cn>
    </sequence>                      </flow>
</sequence>                      </flow>
```

Figure 5. Sequential (a) and parallel (b) partitioning of commands

VI. CASE STUDY (CONTINUED)

Consider BPEL program of a bank transfer process described in Section IV. The analysis of the program dependence graph in Fig. 3 reveals that no data flow path between activity named "dst" and the next two activities "src" and "verify" exists in the graph. Therefore, these activities can be executed in parallel. Similarly, there is no data flow path between two consecutive <invoke> activities "withdraw" and "deposit". These two activities can also be executed in parallel.

To perform these changes, we can partition the outer <sequence> element using transformation 6 three times, and then parallelize the program structure using transformation 4 twice. A skeleton of the resulting BPEL program is shown in Fig. 6. Only names of the activities are shown in Fig. 6. The variables used by the activities are omitted for brevity.

```
<sequence>
   <receive name="rcv">              - receive transfer order
   <flow>
      <assign name="dst">            - extract destination no
      <sequence>
         <assign name="src">         - extract source no
         <invoke name="verify">      - verify funds at source
      </sequence>
   </flow>
   <if>
      <condition> ... </condition>   - check availability
         <empty name="empty">        - do nothing if available
      <else> <sequence>
         <assign name="fail">        - set response
         <reply name="nak">          - reply negatively
         <exit name="exit">          - end of execution
      </sequence> </else>
   </if>
   <flow>
      <invoke name="withdraw">       - withdraw funds
      <invoke name="deposit">        - deposit funds
   </flow>
   <assign name="success">
   <reply name="ack">                - reply positively
</sequence>
```

Figure 6. A skeleton of the transformed bank transfer process – variant I

However, this is not the only way of transformation. Alternatively, the designer can displace "dst" forward, just before <if> activity, and then use transformation 2 in order to enter "dst" to the inside of <if> in place of <empty> activity. Next, transformation 3 can be used to embrace the last three activities of the outer <sequence> element into the first branch of <if> element, consecutively following "dst". Then, the designer can move "dst" forward, adjacent to "deposit", partition the inner sequence of <if> using transformation 6, and parallelize the program structure using transformation 4. A skeleton of the resulting BPEL program is shown in Fig. 7. We removed "exit" activity from the final program, because it is obviously redundant at the end of the program.

The main advantage of the transformed process over the original one is higher level of parallelism, which can lead to better performance of the program execution. If we compare the two alternative designs, then intuition suggests that the structure of the second process is better than of the first one. In order to verify this impression, the reference process and the transformed processes can be compared to each other, with respect to a set of quality metrics. Depending on the results, the design phase can stop, or a selected candidate (a transformed process) can be substituted as the reference process for the next iteration of the design phase.

```
<sequence>
   <receive name="rcv">                   - receive order
   <assign name="src">                    - extract source no
   <invoke name="verify">                 - verify funds
   <if>
      <condition> ... </condition>        - check availability
      <sequence>
         <flow>
            <invoke name="withdraw">      - withdraw funds
            <sequence>
               <assign name="dst">        - extract dst. no
               <invoke name="deposit">    - deposit funds
            </sequence>
         </flow>
            <assign name="success">
            <reply name="ack">            - reply positively
      </sequence>
      <else> <sequence>
         <assign name="fail">             - set response
         <reply name="nak">               - reply negatively
      </sequence> </else>
   </if>
</sequence>
```

Figure 7. A skeleton of the transformed bank transfer process – variant II

VII. Quality Metrics

Many metrics to measure various characteristics of software have been proposed in literature [16,17]. In this research we use simple metrics that characterize the size of a BPEL process, the complexity and the degree of parallel execution. The value of each metric can be calculated using a program dependence graph.

The size of a process is measured as the number of simple activities in a BPEL program. More precisely, the value of this metric equals the number of leaf nodes in the program dependence graph of a BPEL process. For example, the size of the processes shown in Fig. 2 and 6 is 12, while the size of the process in Fig. 7 equals 10.

Leaf nodes are simple activities, which perform the processing of data. Therefore, the value of the process size metric could be considered a measure of the amount of work, which can be provided by the process. However, smaller number of this metric may result from removing excessive, unstructured activities, like <empty> and <exit>. This is the case of program in Fig. 7.

The complexity of a process is measured as the total number of nodes in PDG. For example, size of the process structure of the program shown in Fig. 2 is 15, size of the process structure of the program in Fig. 6 is 18, and size of the process structure of the program in Fig. 7 is 16.

The number of nodes in PDG, compared to the size of the process, describes the amount of excess in the graph, which can be considered a measure of the process complexity.

The number of threads is measured as the number of items within <flow> elements of a BPEL program, at all levels of nesting. A problem with this metric is such that the number of executed items can vary, depending on values of conditions within <if> elements. Therefore, the metric is a

TABLE I. NUMBER OF THREADS METRIC

if - condition	Process in Fig. 2	Process in Fig. 6	Process in Fig. 7
YES	1	2	2
NO	1	2	1

TABLE III. RESPONSE TIME METRIC

if - condition	Process in Fig. 2	Process in Fig. 6	Process in Fig. 7
YES	36	25	25
NO	16	15	14

vector of values, computed for all combinations of values of these conditions. The algorithm of computation assigns weights to nodes of the program dependence graph of the process, starting from the leaves up to the root, according to the following rules:

- the weight of a simple BPEL activity is 1,
- the weight of a <flow> element is the sum of weights assigned to the descending nodes (i.e., nodes directly nested within the <flow> element),
- the weight of a <sequence> element is the maximum of weights assigned to the descending nodes (i.e., nodes directly nested within the <sequence> element),
- the weight of an <if> element is the weight assigned to the activity in this branch of <if>, which is executed according to a given value of condition within the <if> element.

The number of executed items can be influenced also by the presence of <exit> activity, which ends the process execution. Therefore, the nodes directly nested within a <sequence> element are ordered in compliance with the order of execution. Nodes subsequent to a node, which is, or which comprises, <exit> activity, are not taken into account in the computation.

The metric value equals the weight assigned to the top <sequence> node of PDG. Values of the metric for the processes in Fig. 2, 6 and 7 are shown in Table I. Program dependence graph and calculation of the metric for the program in Fig. 6 is shown in Fig. 8 (grey numbers right to the nodes).

The length of thread is measured as the number of sequentially executed activities within a BPEL program. Because the number of executed items can vary, depending on values of conditions within <if> elements, the metric is a vector of values, computed for all combinations of values of these conditions. The algorithm of computation assigns weights to nodes of the program dependence graph of the process, starting from the leaves up to the root, according to the following rules:

- the weight of a simple BPEL activity is 1,
- the weight of a <flow> element is the maximum of weights assigned to the descending nodes (i.e., nodes directly nested within the <flow> element),
- the weight of a <sequence> element is the sum of weights assigned to the descending nodes (i.e., nodes directly nested within the <sequence> element),

TABLE II. LENGTH OF THREAD METRIC

if - condition	Process in Fig. 2	Process in Fig. 6	Process in Fig. 7
YES	9	7	7
NO	7	6	5

- the weight of an <if> element is the weight assigned to the activity in this branch of <if>, which is executed according to a given value of condition within the <if> element.

Nodes directly nested within a <sequence> element are ordered in compliance with the order of execution. Nodes subsequent to a node, which is, or which comprises, <exit> activity, are not taken into account in the computation.

The metric value equals the weight assigned to the top <sequence> node of PDG. Values of the metric for the processes in Fig. 2, 6 and 7 are shown in Table II.

The response time is measured as the sum of estimated execution times of activities, which are sequentially executed within a BPEL program. Because the number of executed items can vary, depending on values of conditions within <if> elements, the metric is a vector of values, computed for all combinations of values of these conditions The algorithm of computation is identical to the algorithm of computation of the length of thread metric, except of the first point, which now reads:

- the weight of a simple activity is the estimated execution time of this activity,

In the simplest case, the estimated execution time can just differentiate between local data manipulation activity

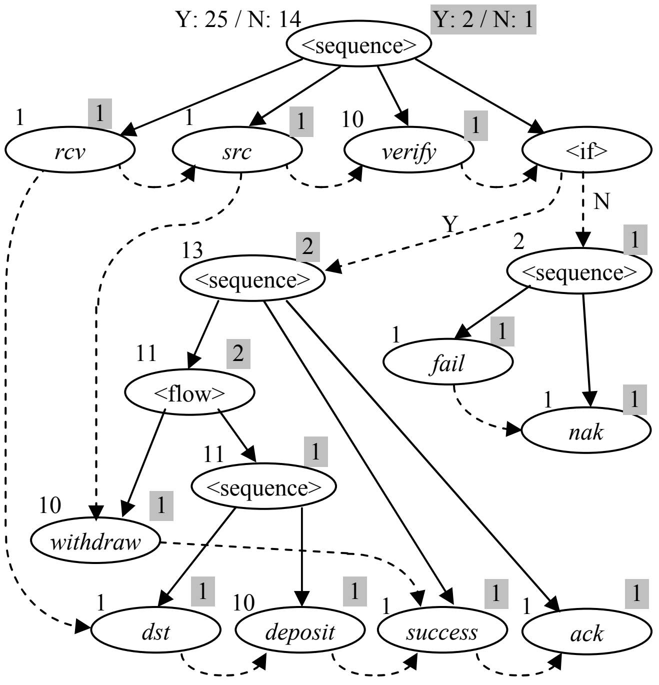


Figure 8. Program dependence graph of the program in Fig. 6 and calculation of metrics: Number of threads (grey numbers right to the nodes) and length of execution (left to the nodes)

and a service invocation. Values of the metric for the processes in Fig. 2, 6 and 7, calculated under an assumption that a local data manipulation time equals 1, while a service execution time equals 10, are shown in Table III. Program dependence graph and calculation of the metric for the program in Fig. 7 is shown in Fig. 8 (numbers left to the nodes).

Comparing the values of metrics calculated for the processes considered in the case study in Sections IV and VI, one can note that both transformed processes are faster than the original reference process (smaller value of the response time metric). Speeding up the process execution is a benefit from parallel invocation of services in a SOA environment. Comparing the variants of the transformed bank transfer process (Fig. 6 and Fig. 7), one can note that the second variant is a bit faster and simpler (smaller values of the size metrics). This variant can be accepted by the customer or used as a new reference process in the next transformation cycle.

VIII. CONCLUSION AND FUTURE WORK

Defining the behavior of a business process is a business decision. Defining the implementation of a business process on a computer system is a technical decision. The transformational method for implementing business processes in a service oriented architecture, described in this paper, promotes separation of concerns and allows making business decisions by business people and technical decisions by technical people.

The transformations described in this paper are correct by construction in that they do not change the behavior of a transformed process. However, the transformations change the process structure in order to improve efficiency and benefit from the parallel execution of services in a SOA environment. The quality characteristics of the process implementation are measured by means of quality metrics, which account for the process size, complexity and the response time of the process as a service. Other quality features, such as modifiability or reliability, are not covered in this paper.

The correct-by-construction approach is appealing for the implementation designer because it can open the way towards automatic process optimization. However, the approach has also some practical limitations. If the external services invoked by a process have an impact on the real world, as is usually the case, a specific ordering of these services may be required, regardless of the dataflow dependencies between the service invocation activities within a program. In our approach, a designer can express the necessary ordering conditions adding supplementary edges to the program dependence graph. Therefore, the approach cannot be fully automated and a manual supervision over the transformation process is needed.

It is also possible that small changes to a process behavior can be acceptable within the application context. Therefore, part of our research is aimed at finding a verification method, capable not only of verifying the process behavior, but also of showing the designer all the potential changes, if they exist. The results of this research are not covered in this paper.

ACKNOWLEDGMENTS

This research was supported in part by the Ministry of Science and Higher Education under the grant number 5321/B/T02/2010/39.

REFERENCES

[1] T. H. Davenport and J. E. Short, The New Industrial Engineering: Information Technology and Business Process Redesign, Sloan Management Review, pp. 11-27 (1990)

[2] OMG, Business Process Model and Notation (BPMN), Version 2.0, (2011) http://www.omg.org/spec/BPMN/2.0/PDF/ 20.09.2012

[3] A. W. Scheer, ARIS - Business Process Modeling, Springer, Berlin Heidelberg (2007).

[4] D. Jordan and J. Evdemon, Web Services Business Process Execution Language Version 2.0. OASIS Standard (2007).

[5] OMG, Unified Modeling Language (UML): Superstructure, V2.1.2, http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF (2007).

[6] P. Zave, An Insider's Evaluation of Paisley. IEEE Trans. Software Eng., vol. 17 (3), pp. 212-225 (1991)

[7] K. Sacha, Real-Time Software Specification and Validation with Transnet. Real-Time Systems J., vol. 6, pp. 153-172 (1994)

[8] F. J. Duarte, R. J. Machado, and J. M. Fernandes, BIM: A methodology to transform business processes into software systems, SWQD 2012, LNBIP 94, pp. 39-58 (2012)

[9] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz, Reference Model for Service Oriented Architecture 1.0. Technical report, OASIS (2006)

[10] J. A. Estefan, K. Laskey, F. G. McCabe, and D. Thornton, Reference Architecture for Service Oriented Architecture Version 0.3. Working-draft, OASIS (2008)

[11] S. A. White, Using BPMN to Model a BPEL Process, BPTrends 3, pp. 1-18 (2005) www.bptrends.com

[12] J. Recker and J. Mendling, On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. In: T. Latour, M. Petit (Eds.): Proc. 18th International Conference on Advanced Information Systems Engineering, pp. 521-532 (2006)

[13] M. Weiser, Program slicing. IEEE Trans. Software Eng., 10 (4), pp. 352-357 (1984)

[14] D. Binkley and K. B. Gallagher, Program slicing, Advances in Computers, 43, pp. 1-50 (1996)

[15] C. Mao, Slicing web service-based software. IEEE International Conference on Service-Oriented Computing and Applications, IEEE, pp. 1-8 (2009)

[16] J. K. Hollingsworth and B. P. Miller, Parallel program performance metrics: A comparison and validation, Proc. ACM/IEEE Conference on Supercomputing, pp. 4 - 13, IEEE Computer Society Press (1992)

[17] A. S. Van Amesfoort, A. L. Varbanescu, and H. J. Sips, Proc. 15th Workshop on Compilers for Parallel Computing, pp 1-13 (2010)

[18] A. Ratkowski and K. Sacha, Business Process Design In Service Oriented Architecture. In: A. Grzech, L. Borzemski, J. Świątek, Z. Wilimowska (Eds.): Information Systems Architecture and Technology, pp. 15-24. Wroclaw University of Technology (2011)

[19] T. Erl, Service-oriented Architecture: Concepts, Technology, and Design. Prentice Hall, Englewood Cliffs (2005)

[20] Bpmn2bpel Project Home, A tool for translating BPMN models into BPEL processes, http://code.google.com/p/bpmn2bpel/, 22.10.2012

On Re-Architecting Legacy Software Systems: The Case of Systems at Umm Al-Qura University

Basem Y. Alkazemi
Department of Computer Science
Umm Al-Qura University, UQU
Makkah, Saudi Arabia
bykazemi@uqu.edu.sa

***Abstract*—This short paper describes our proposed architecture for the software systems at Umm Al-Qura University (UQU). We adopted the notion of SOA to derive the building block of the new architecture. The proposed solution is the first step towards migrating the legacy systems at UQU to new architecture that can respond seamlessly to the emerging e-government requirements.**

Keywords- legacy systems; SOA; e-government.

I. INTRODUCTION

Responding to rapidly changing IT markets - including expanding e-government applications - requires adopting a reliable, versatile and fully flexible system capable of accommodating recent and upcoming changes and modifications efficiently and smoothly while keeping old business needs intact [1-2]. In this day and age, such a system can be described as a mandatory rather than an optional when responding to ever increasing business needs.

Adaptable systems nevertheless are not always available to many large private companies, public organizations, government agencies, hospitals, municipalities, and universities in the Saudi Arabian context. These institutes in reality usually maintain their respective legacy systems as far as the systems provide the basic necessary functionality. However, these organizations are aware of the rapidly changing IT market and are duly planning to replace their old systems at some point in order to accommodate the growing new business requirements should finical resources become available. They may also consider the more cost-effective option of modernizing or re-architecting [3].

Many challenges are attributed to the nature of legacy systems which cannot be easily modified. Systems are usually treated as black boxes not because they lack documentations or because the source code is not available. Instead, the systems are poorly architected in the sense they can no longer cope with new business needs. Hence, become one of the key barriers to adopt any potential e-government business models.

Poorly architected legacy systems can encourage CEOs to replace them with entirely new ones. However, such a decision should be informed and well researched as it still has consequences. Legacy systems usually provide highly customized functionalities that none of the available solutions in the market may provide if purchased as is. For example, setting up new systems may require making huge modifications that can take up to several years to comply with old business needs within organizations while accommodating newer ones.

This research aims at investigating an architectural model to analyze the feasibility of re-architecting legacy systems in order to satisfy e-government business needs. The paper is organized in six sections. Section II presents the background work that set the context of our work. Section III introduces a conceptual system architecture model of SOA. Our proposed model is given in section IV. Section V describes some potential advantages of applying the notion of re-architecting as compared to purchasing new products. Finally, the conclusion and future works is given in section VI.

II. BACKGROUND WORK

We chose Umm Al - Qura University (UQU) [11] as a typical Saudi organization running a legacy system in need of urgent updating to act as our case study for applying and evaluating our proposed alternative model. The goal is to establish a fully integrated environment that supports e-government business. In other words, the institution needs a rigorous solution that promotes changes without interrupting their daily working activities. However, funding seems to be a major constraint that constantly influences the decision for adopting any major new development plans.

Umm Al-Qura University launched its information systems in early 2001 to serve around 3600 employees and just over 40,000 students at the time. It owns old fashioned systems based on Oracle 6i for forms and reports those are built entirely on client-server pattern [7]. The major systems include an in house built *Enterprise Resource Planning* (ERP), *Student Information System* (SIS), *Library Information System* (LIS), and *Healthcare Information System* (HIS). These systems are still used today to serve around 75000 students and more than 7000 employees, a much higher figure than in 2001, with minor improvement to the original functionality.

However, software systems at UQU still lack many capabilities that become core-requirement nowadays in terms of compatibility with different environments (e.g., Mobile devices) and the services provided to students and faculty members in the University. Moreover, with the pioneering of e-government movements in Saudi Arabia, organizations may need to apply major changes to their systems in order to accommodate these new requirements; one of which is process automation that solely requires splitting functional aspects of an application from the business aspects.

Current practices for the modifications to add features to any of the systems are done in an ad-hoc manner by which an application's code is modified to satisfy requirements. Specifically, business processes are implemented directly into the forms confusing the functional aspects of an application with the non-functional ones. As a result, the complexity of UQU systems is building up rapidly in a manner that will become very hard to manage in the near future.

III. Conceptual System Architecture

One key driver for establishing our framework is the representation of workflow within a software system. Currently many systems develop their business processes hardcoded into the source code. So, whenever new business processes are required to be implemented the overall code must be modified. Moreover, applications are integrated in a one-to-one manner by writing glue code to establish the integration. This glue code is usually written as a mediator between two applications. Although this approach might look simple to some developers, it causes process design to become totally confused and mixed. In some cases glue code is injected into one of the applications themselves. This is the worst scenario as it will result in very tangled code that cannot be managed over the years especially when developers are dealing with an enterprise solution.

The described framework considers SOA [8] as an integration facilitator mechanism and not as a service delivery mechanism. The framework is composed of different layers that, based on our previous work for analyzing a number of systems [9], any enterprise solution in the market must satisfy in order to ensure flexibility and extensibility of their systems. Figure 1 presents our proposed architecture for an enterprise solution.

Each layer is independent of the other surrounding layers in terms of their main functionality. The description of these layers is as follows:

- Data Access Layer: this layer is responsible for managing the interaction between application and database and hiding the databases used in the organization. So, if different database technologies are used (e.g., MYSQL, Oracle), this layer will manage the connectivity with the corresponding source.
- Application Layer: this layer is responsible for executing the basic functionality that represents an organization’s business needs. In the context of an ERP solution, this layer represents the fundamental modules offered by the solution such as HR, Finance, Projects, and Sales. Every one of these modules must be a standalone application that is not aware of any other modules.
- Packaging Layer: this layer is responsible for wrapping the available applications from the application layers into standard component model [12]. All applications are therefore decoupled from their underlying environment and made available through request-response interaction mode.

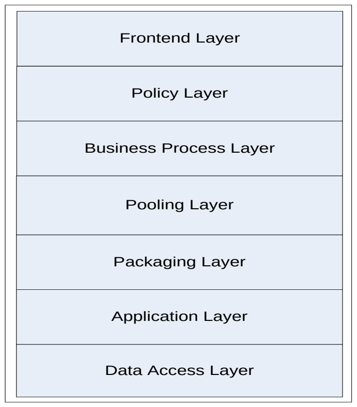


Figure 1. Architectural Layers for Enterprise Solutions

- Pooling Layer: this layer is responsible for hosting the different packaged components and make them ready to be used in a business. In addition, the layer is responsible for establishing the communication pattern and routing protocols that enable service discovery and interaction. It defines the policies that comply with the standards adopted by vendors. For example, web services interact by exchanging SOAP messages over HTTP protocol. Any interaction between services must be accomplished through this layer. This is usually referred to as the Enterprise Service Bus (ESB) layer.
- Business Process Layer: this layer defines the workflows that are employed by an organization. It is responsible for establishing the sequence by which services are going to be invoked to satisfy business requirements. For example, an attendance service might need to issue a request to a finance service to deduct a certain amount from an employee salary.
- Policy Layer: this layer is responsible for defining the privileges for accessing services. A different level of access rights can therefore be granted at this layer according to the defined policy.
- Frontend Layer: this layer is responsible for exposing services for different types of devices and technologies (e.g., web applications, mobile application, cloud computing).

This layered architecture is technology neutral and designed partially utilizing the concept of the SOA pattern. The identified layers are not interchangeable as they must build up in a bottom-up manner. So, for example, a database can be established and tables created for an ERP system. Then, a number of standalone applications are developed on top of these tables to utilize the data in the tables. These applications must then be exposed in a standard manner in order to facilitate their integration with other applications to achieve new business needs. So, the new exposed interfaces are pooled and made ready for requests. Workflows can then be defined on top of the available pool of services in order to integrate different applications seamlessly without affecting each application's concern. In fact, a workflow defines the design of a system where different components can be executed in a pre-defined sequence. Once all the business requirements are established (i.e., all functionality is implemented), there should be privileges assigned to personnel who are authorized to execute certain processes in the system.

IV. PROPOSED ARCHITECTURE FOR UQU SYSTEMS

UQU is moving steadily and progressively toward providing e-government services which goes in line with the university's technological ambitions. A main objective from the university's website indicates fully automating all its internal processes and establish rigorous infrastructure capable of supporting internal and external exchange of data. In fact, the organization dedicates huge resources and funds in order to achieve this objective.

This requires a comprehensive architecture to be established in order to facilitate harmonious integration of different systems. UQU systems currently operate in three different environments, SharePoint 2010, PHP (codeigniter), and Oracle 6i. Our proposed architecture is meant to integrate all systems regardless of technology in a rather neutral manner. The proposed architecture model is given below.

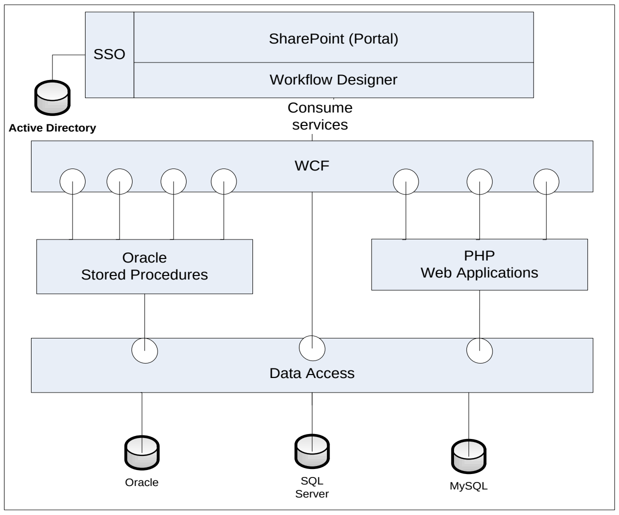


Figure 2. UQU Proposed System Architecture

Figure 2 illustrates the proposed architecture for facilitating the adoption of the emerging business need of UQU based on the resources that are currently available to the university. The main objective of this solution is to promote fully integrated environment that facilitates internal and external data exchange, in addition to promote scalability for future development. UQU currently own full package of SharePoint 2010, in-house built Oracle ERP solution, website and a number of services in PHP, and an Internet Information Server (IIS).

In our proposed solution, SharePoint is utilized to play two main roles; the web presence and the service orchestration layer where business processes are defined through windows workflow foundations (WWF) provided by the SharePoint workflow engine. Services are exposed to SharePoint through the Microsoft-IIS layer where web services are defined. As a result, every application must be wrapped and exposed as a standalone web service that can be consumed by SharePoint. This capability simulates the basic functionality of the well-known Enterprise Service Buss (ESB) pattern for service integration and management which represent the communication layer for integrating the various applications in an organization. SharePoint 2010 must work only on SQL server, hence, in this solution; we propose to use the SQL server for document flow management purposes without interfering with the university database by any means.

This architecture proposes a flexible solution for the ERP system within the UQU and also establishes rigorous platform for any potential ECM functionality required by the university. SharePoint 2010 together with Microsoft-IIS provides the necessary pool and management of services. They facilitate services orchestration in order to enable the interaction between the different services of the system. Any new service can be exposed into this layer and then composed with other services by defining a workflow that corresponds to a predefined business process model.

Ideally, the resulted architecture should promote high degree of extensibility and flexibility where different business processes within or between departments become composable and fully automated. The first step toward that ultimate goal, as far as system structure is concerned, is re-architecting of the legacy system in order to increase the flexibility of IT within UQU. Re-architecting UQU legacy systems with SOA concepts in mind allows for quick response to changing market needs, can implement IT systems that can quickly adapt to changing markets, shifting customer requirements, and new business opportunities.

V. POTENTIAL BENEFITS OF RE-ARCHITECTING

Legacy systems' re-architecting is a cost-effective modernization alternative to the creation of software systems in an organization when a new market or business need arises. Given that purchasing new software has huge cost implications to the organization, it is better for organizations to improve their own proven legacy systems to address their specific emerging business needs. The huge costs of purchasing new software from the market come from the actual price of the software, rollout cost, and training costs. In fact, being new, it might cost higher to maintain in the initial days because it might involve vendor intervention from time to time [4]. More so, this approach promotes low operating costs, with the software built on existing hardware and other systems [5].

The re-architecting approach is low risk modernization option in the sense that existing software is already tested and proven to work in addressing existing business needs. This is more favorable as opposed to software systems an organization purchases and have pilot tests done before establishing if it actually addresses the organization's needs [4]. It allows an organization to transform its existing legacy applications to meet the current market demands without overhauling the entire system. This, in turn, minimizes the loss of existing IT systems' investments in which the legacy systems hold crucial information and data that is required in the daily operations of the organization.

Another advantage of this approach is that re-architecting encourages the development of a custom software system architecture that is based on the organization's demands and capabilities. This is because this process allows the re-architecting team to survey and understand not only the requirements of the new system, but also the overall capabilities of the organization in managing the new system [6]. Consequently, this enables the development of a system that the organization will use and manage comfortably. During the re-architecting of legacy systems, highlights that re-architecting legacy systems gives the development team an opportunity to transform the current system user interface to the popular web-based user interface if it is not in place already. This helps the users interact with the new system in a friendly manner and, thus, enable the usual operations of the organizations to run with minimum delays.

Software system re-architecting approach permits customization in the training of the system users and maintenance teams. This is realized through re-architecting experts who train the users at each stage of the development process, thus, enabling them to understand the new system with much ease. This is achievable since re-architecting targets certain system enhancements concerning the central part of the solution which aims at handling given business needs.

Improving legacy systems helps sustain an organization's reputation because it principally helps minimize any interruption to routine business operations. This means that customers, too, will feel minimal negative impact, if any. This is critical in businesses where reputation is very important, particularly due to competition. Consider a situation where rolling out a new system takes days to realize. Business operations would have to stop until the system is working as expected but clients may not be that patient and, therefore, consider the organization insensitive to their needs.

Finally, this approach grants an organization the opportunity to employ modern technical architecture such as Service Oriented Architecture and Cloud Computing Architecture. These have tested and proven levels of flexibility to accommodate future technological changes. For instance, when SOA is implemented to support business intelligence (BI), it allows a flawless technology integration to form a consistent BI environment that makes the delivery of data straightforward while simplifying low latency diagnostics at the same time.

VI. CONCLUSION AND FUTURE WORK

This paper accounts for a major obstacle that challenges the decision to adopt e-business solution in any given organization which is the lack of standard architecture of the legacy systems. The paper proposed that re-architecting legacy system can be beneficial to some organizations in improving the architecture of their systems without affecting their underlying business logics. We summarized the main advantages of re-architecting in the following points:

- Re-architecting connects legacy business logic with modern technologies and concepts.
- Re-architecting can evolve legacy applications into SOA-based deployments.
- The new system will require less time spent coding when modifying or developing logic.
- By being based on SOA concepts and built on an advanced framework, the new system will be flexible, transparent, and reliable.
- The new system will be expandable without the danger of a 'spaghetti architecture' emerging.

The next step in this work is to utilize one of the tools available in the market such as the BAZ [10] tool performs the conversion of 6i forms into ADF [13] compatible components. Then, components will be exposed as web services and deployed into the IIS for business process utilizations. Also, we will apply this model to some other universities within the region in order to evaluate its applicability to a wider range of cases.

REFERENCES

[1] C. Holland, and B. Light, "A Critical Success Factors Model for ERP Implementation", Software IEEE, vol. 16, no. 3, 1999, pp. 30 – 36.

[2] K. Bennett; M. Ramage, and M. Munro, M. Decision "Model for Legacy Systems", Software, IEE Proceedings, vol. 146, no. 3, 1999, pp. 153 – 159.

[3] R. C, Seacord, D. Plakosh & G. A. Lewis, Modernnising Legacy Systems: Software Technologies, Engineering Processes, and Business Practices. 2003, Boston: Pearson Education.

[4] D. Reeves, "Legacy systems re-engineering: leveraging your existing assets", Revenue Solutions, Inc. 2009, Available from: http://www.taxadmin.org/fta/meet/09tech/Tech%2009%20papers/Reeves-Legacy.pdf [retrieved: March 2012].

[5] A. Umar and A. Zordan, "Reengineering for service oriented architectures: a strategic decision model for integration versus migration", Journal of Systems and Software, vol. 82 vo. 3, 2008, pp. 56 – 64.

[6] D. Quah, 2005. Thesis on 'Case Study on Re-Architecting of Established Enterprise Software Product: Major Challenges Encountered and SDM Prescriptions from Lessons Learned.' Massachusetts Institute of Technology, pp. 1-122.

[7] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture Volume 1: a System of Patterns, 1996.

[8] L. Grace, M. Edwin, S. Dennis, and S. Soumya. SMART: Analyzing the Reuse Potential of Legacy Components in a Service-Oriented Architecture Environment, In CMU/SEI-2008-TN-008. Software Engineering Institute, Carnegie Mellon University, 2008.

[9] B. Y. Alkazemi, ,"A Conceptual Framework to Analyze Enterprise Business Solutions from a Software Architecture Perspective ", in the IJCSI, vol. 9, no. 3, 2012.

[10] SmartDeveloper Co., http://www.sd4it.com/Baz.html, [retrieved: Jan 2012]

[11] Umm Al-Qura University, http://www.uqu.edu.sa, [retrieved: Nov 2012].

[12] K. Lau, Z. Wang, "Software Component Models", IEEE Transaction on Software Engineering, vol. 33, no. 10, 2007.

[13] Oracle Co, Oracle Application Development Framework, Available from:http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html [retrieved: June 2012]

Business Process Modeling in Object-Oriented Declarative Workflow

Marcin Dąbrowski, Michał Drabik, Mariusz Trzaska, Kazimierz Subieta
Polish-Japanese Institute of Information Technology
Warsaw, Poland
{mdabrowski, mdrabik, mtrzaska, subieta}@pjwstk.edu.pl

Abstract—**The paper presents motivations, the idea and design of an object-oriented declarative workflow management system. The main features that differ this system from many similar systems are: inherent parallelism of all workflow instances and tasks, the possibility of dynamic changes of running process instances and integration of workflow instances with an object-oriented database. Workflow instances, tasks, subtasks, etc., are implemented as so-called active objects, which are persistent data structures that can be queried and managed according to the syntax and semantics of a query language. and also possess active parts that are executable. The prototype has been implemented on the basis of ODRA, an object-oriented distributed database management system. As the workflow programming language we use SBQL, an object-oriented database query and programming language developed for ODRA.**

Keywords-workflow; object-oriented; declarative; query language; active object, dynamic workflow change; ODRA; SBQL

I. Introduction

Current workflow technologies, developed mainly by commercial companies and standardization bodies, see for instance [1, 2, 3], present a considerably well recognized domain, with a lot of commercial successes. The core of the current approaches to workflows is the control flow graph, which determines the order of tasks performed by a particular process instance. Other issues related to workflows, such as resource management, workflow participant assignments, database structure and organization, transaction processing, synchronization of parallel activities, exception handling, tracking and monitoring of workflow processes, are frequently treated with attention, but are seen as secondary with respect to the work control flow. The model based on a control flow graph is defined formally as a Petri net [4].

There are problems that undermine applications of workflow management systems in important business domains. Below, we list the following features that are frequently required in complex business applications, but are absent or poorly supported by workflow systems:

1. Mass parallelism of tasks within workflow processes.
2. Dynamic changes of workflow instances during their run.
3. Reactions to unexpected events or exceptions and aborting running processes or their parts.
4. Resource management as a main workflow driving factor.

Below, we discuss the above aspects.

Ad.1. Currently workflow systems enable parallel sub-processes through splits and joins (AND, OR, XOR) that are explicitly determined by the programmer. Such a form of parallelism is insufficient for many business cases. During the run, a process instance could be split into many parallel sub-processes, but their number is large and unknown during development of the process definition. For example, processing an invoice requires splitting it into as many sub-processes as the items that the invoice consists of. This typical situation cannot be covered by explicit splits and joins. Moreover, as noted by Reichert and Dadam [5], it is often not convenient and not efficient to determine task sequences in advance.

Ad2. Although there is a valuable research (e.g., [5, 6, 7, 8, 9, 10, 11]) aiming at dynamic changes of process instances, especially workflow patterns [12, 13], it can be anticipated that the scope of the changes must be limited. There are several problems with modifications of a currently executed process instance graph:

- Current workflow programming languages are not prepared to deal with dynamic changes of a running code.
- Parts of a flow control graph have no identity, they cannot be separated from other parts and they are not described by some metamodel (like a database schema).
- Process instance graph elements are tightly interconnected. If one would try to alter the code (e.g. by removing some its part) the problem is how to fix other elements to create a consistent whole.
- Changes can violate the consistency of process instances, hence some discipline of changes is required.
- If many possible actors are allowed to alter a process instance graph, then elements of the graph should follow ACID transactions.

Ad3. Usually, programming languages have programming means to define and process exceptions (events). However, this concerns only situations when exceptions are known during developing a process definition. The behavior of business processes is frequently unpredictable. There are exceptional situations that are known only at the time when process instances are already running. For instance, a new type of malicious attack on a banking workflow system is discovered in situations when thousands of process instances are already executed. The workflow system administrator has practically the only option: abort processes, change the process definition and start processes from the beginning. From the business point of view this could be unacceptable and might generate a huge additional cost. Aborting already running processes is a problem, because they engage entities and resources from the business environment (clients, personnel, documents, contracts, etc.). They may require a lot of compensating actions, which must be done manually, with no help from the workflow system.

Ad4. In currently developed workflows, the work control flow (a la Petri net) is on the primary plan and the resources (people, money, time, work power, equipment, infrastructure, offices, vehicles, etc.) are secondary and sometimes not taken into account at all. This is unnatural for business processes because just availability, unavailability of resources and their monitoring, planning and anticipating are the main factors that determine a process control flow. Just availability of resources should trigger some tasks. Because information on resources is usually a property of a database supporting the workflow system, conditions within a workflow control flow graph should include accesses to a database. This is usually impossible in typical workflow programming languages or burdened by an infamous effect known as impedance mismatch [14] between querying and programming.

The above issues were the reasons that we started the research on a new workflow management system that will be able to overcome limitations of the current systems concerning mass parallelism and dynamic changes of running workflow instances. The assumptions of our design is that an element of a workflow instance should have a double nature. On the one hand, it should be perceived as a data structure (an object) that can be addressed by a database query and programming language. The structure is to be stored in a database and should be the subject of database transactions. On the other hand, the element should contain executable parts, i.e., the code of a workflow process or sub-process.

This way, we have come to the concept of active object. An active object is a database object that contains some static parts (attributes) and some active parts (codes). We distinguish four such parts: *firecondition, executioncode, endcondition* and *endcode*. An active object waits for execution until the time when its *firecondition* becomes true. After that, the object's *executioncode* is executed. The execution is terminated when either all the actions are completed (including actions of active sub-objects) or its *endcondition* becomes true. After fulfillment of the *endcondition* some terminating actions can be executed through *endcode*. This may be required to terminating some actions, e.g. closing connections, aborting transactions, setting a new object state, etc. Active objects belong to their classes, follow the principle of encapsulation and are typechecked according to the strong typing system. They can be updated as regular database objects. Preventing undesired updates can be accomplished by well-known database capabilities such as user rights, integrity constraints, triggers and active (business) rules. Unexpected events can be served by inserting new active sub-objects into running active objects and/or by altering active objects.

Active objects accomplish an important feature: mass parallelism of executed tasks. In principle, all active objects act in parallel. In life, tasks performed by people can be done in parallel with no conceptual limitations. Some tasks, however, must wait for completing other tasks and this model can be expressed as a PERT (Program Evaluation and Review Technique) graph. Active objects act as PERT graphs: if object A has to wait for object B, then the *firecondition* of A tests the state of B, which should be set to "completed" when B is terminated. This way, one can determine the sequence of processes, but this does not constraints one from using parallelism whenever possible. Because the sequence of tasks is not determined explicitly, we describe this workflow model as "declarative". Note that this idea of declarative workflow processes is considerably different from the idea of the DECLARE model presented in [15], which is a logic-oriented formalism for specification of various dependencies between (sequences of) events. In our case, "declarative" means that the programmer specifies a workflow code as collections of (nested) active objects, with *fireconditions* and *endconditions* specified by means of the declarative query language SBQL[16].

In our idea, workflow resources, as any database properties, can be used to form *fireconditions* and *endconditions*. In this way the resource management is properly shifted on the first plan. Both active objects and resource description objects are integrated in the same database thus can be addressed by the same integrated query and programming language. Hence, any form of impedance mismatch is avoided.

The widely recognized paper devoted to dynamic workflow changes is [5]. It presents some framework for formalizing process graphs and updating operations addressing such a graph. There are valuable observations concerning the necessity of dynamic workflow changes for real business processes and the necessity of strong discipline within the changes to avoid violation the consistency of the processes. Numerous authors follow the ideas of this paper. The fundamental difference of our approach is that the process control flow graph is not explicitly determined. It can be different from one run to next run, depending on the state of the workflow environment, database, computer environment, *fireconditions* and *endconditions*. The problem of the necessity of various control flow graphs for the same business process is one of the motivations for the research presented in [5], but it is not easy to see how such a feature can be achieved within the proposed formal workflow model. In our idea the feature is an inherent property.

Some disadvantage of our concept concerns the performance. Decreasing performance can be caused by late binding and the necessity of cyclic checking of *fireconditions* and *endconditions*. We believe that performance problems of our idea can also be overcome by new optimization methods and new computer architectures.

The prototype is implemented under the ODRA system [17]. As a workflow programming language, we use SBQL, an object-oriented query and programming language developed for ODRA.

The paper is organized as follows. Section 2 presents the concept of an active object. Section 3 describes the implemented prototype. Section 4 presents a comprehensive example of a declarative workflow. Section 5 concludes the paper.

II. Active Objects

In the following, we use the term *active object* as a generalization of process instance and task instance. Because of the relativity of objects assumed in SBQL, components of active objects are active objects too. Due to this, there is usually no need to distinguish between process instances and task instances – all are represented as active objects. To represent process and task instances, active objects are specialized, and belong to a special class named *ActiveObjectClass*, which contains basic typing information, basic methods and other necessary invariants.

An active object is a nested object with the following main properties:

- Unique internal object identifier.
- External (business) name that can be used in source programs.
- Certain number of public and private attributes.
- One distinguished attribute (sub-object) containing an SBQL procedural workflow *executioncode* of a process or a task; it may contain an empty instruction only.
- One distinguished attribute (sub-object) containing an SBQL code with a *firecondition* (a condition for starting the run of the given active object).
- One distinguished attribute (sub-object) containing an SBQL code with an *endcondition* (a condition for terminating the run of the given active object). An *endcondition* may be absent. In this case the action of an active object is terminated when its *executioncode* is terminated and/or when all its active subobjects are terminated.
- One distinguished attribute (sub-object) containing an SBQL code with an *endcode* (a code executed to consistently terminate the run of the given active object). An *endcode* can be absent.
- Any number of named pointer links (binary relationship instances) to other (active or passive) objects.
- Any number of inheritance links connecting the given object to its classes (multiple inheritance is supported).
- Any number of nested active objects. The construction of a nested active object is identical to that of a regular active object (the object relativism is supported). The number of nesting levels for active objects is unlimited.

When an active object consists of active sub-objects, the *endcondition* determines whether the process or task is completed. An *endcondition* can accomplish all kinds of joins (AND, XOR) of parallel processes, and much more.

For example, if within an active object *Invoice* there are many (unknown number of) active sub-objects *TestingAnItem*, then we can impose the *endcondition* of *Invoice* (the end of the invoice checking process) in the form of a query involving an universal quantifier:

```
forall TestingAnItem as x (x.State =
"completed" or x.State = "cancelled")
```

We can also impose more complex conditions. For instance, let the cost of an invoice item will be stored within *TestingAnItem* as a *Cost* attribute, and assume that the entire invoice is checked if more than 95% of its total cost is checked. In this case, the endcondition will have the form:

```
sum((TestingAnItem where State =
"completed").Cost) / Invoice.TotalCost > 0.95
```

Because active objects are regular objects in the SBQL terms, they can be manipulated without limitations. For instance, active objects can be altered and deleted. Their state can be changed, including the code of active parts. New active sub-objects can be inserted into an active objects. This feature makes it possible to split the process (represented by the active object) into any number of subprocesses (inserted active subobjects). Proper construction of the object's *endcondition* (e.g., with the use of quantifiers) makes it possible to do any join of them, as illustrated in examples.

Active objects are specified by their classes and follow the strong typechecking. The definition of a workflow process determined by its class can be instantiated by creating an active object of this class. The object creation follows the standard routine of SBQL. The only difference concerns executable parts: during instantiation their codes are created as strings and then compiled to bytecodes.

III. Design and Implementation of the Prototype

The implemented prototype makes it possible:

- Creating and modifying workflow definitions;
- Instantiating them (creating workflow instances);
- Running a workflow monitor which processes workflow instances.

The prototype has been implemented as a web application using several technologies. The web part utilizes Groovy [18], Grails [19] and JavaScript [20]. The ODRA DBMS, the ODRA wrapper and the process monitor are written in Java.

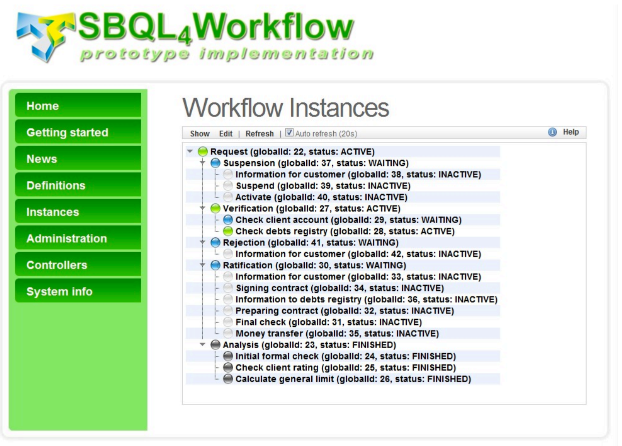
The main part of the system resides on the Tomcat [21] servlet container hosting most of the application logic. The most important parts are the following:

- The module for generating GUI. It is based on the core Grails framework technology called GSP (Groovy Server Pages). It is similar to well-known JSP (Java Server Pages);
- The application logic which manages the workflow model on the functional level. It provides an interface to administrative tasks in a workflow system for all applications. It is suitable not only for our custom built GUI interface, but also for any Java-based application.
- The ODRA wrapper simplifies all tasks related to the ODRA DBMS. ODRA is responsible for storing workflow related information (definitions, instances) and executing SBQL codes within active objects.

- The process monitor accomplishes time sharing among active parts. It periodically switches the control flow among all currently executed parts of active objects and their subobjects.
- The cache memory speeds up the access to commonly utilized DB objects.

The client component is executed in the standard-compliant web browser and consists of the following features:

- Regular web forms which are used for creating and updating instances and definitions.
- An AJAX part written in JavaScript using the jQuery library. Such an approach makes it possible to use powerful widgets like definitions/instances trees (Fig. 1) or SBQL code editor with syntax highlighting. Another advantage was lack of reloading a web page (post/get) in some cases, i.e., auto refreshing of instances' status in the tree. As a result overall user experience was greatly enhanced.



Workflow Instances

```
Request (globalId: 22, status: ACTIVE)
  Suspension (globalId: 37, status: WAITING)
    Information for customer (globalId: 38, status: INACTIVE)
    Suspend (globalId: 39, status: INACTIVE)
    Activate (globalId: 40, status: INACTIVE)
  Verification (globalId: 27, status: ACTIVE)
    Check client account (globalId: 29, status: WAITING)
    Check debts registry (globalId: 28, status: ACTIVE)
  Rejection (globalId: 41, status: WAITING)
    Information for customer (globalId: 42, status: INACTIVE)
  Ratification (globalId: 30, status: WAITING)
    Information for customer (globalId: 33, status: INACTIVE)
    Signing contract (globalId: 34, status: INACTIVE)
    Information to debts registry (globalId: 36, status: INACTIVE)
    Preparing contract (globalId: 32, status: INACTIVE)
    Final check (globalId: 31, status: INACTIVE)
    Money transfer (globalId: 35, status: INACTIVE)
  Analysis (globalId: 23, status: FINISHED)
    Initial formal check (globalId: 24, status: FINISHED)
    Check client rating (globalId: 25, status: FINISHED)
    Calculate general limit (globalId: 26, status: FINISHED)
```

Figure 1. A workflow instances tree

The last two remaining architecture's items are: the ODRA system and a mail server. The last one is utilized for sending progress messages to parties involved in a workflow.

The schema of a database used to store workflow data, is presented in the Fig. 2. The process objects represent structures created by the workflow programmer before it is actually ran. Once a process is initiated, all data, including the data of sub-processes, is copied to the corresponding ProcessInstance objects. The parent-child bidirectional pointer, combined with SBQL query operators, gives a great flexibility in expressing conditions and codes. For instance:

- Find all my children (the code is written with regard to one particular *ProcessDefinition*).
- Find my parent.
- Find a process with a given status.
- Find a process with a given name.

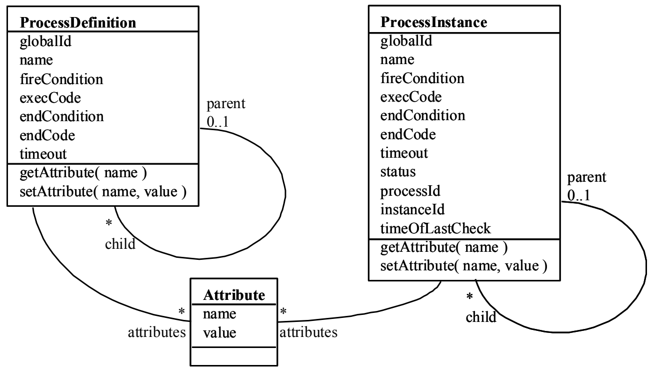


Figure 2. ODRA database schema

These constructs can be easily combined for more complex search, for instance:

- Find a child that has a certain name and status.
- Check if all my children have the status 'Finished'.
- Find my "brother" (using parent.children).
- Find all my "nephews" (using parent.children.children).

To allow processes to store "ad-hoc" some additional data we have provided the Attribute class with a set of methods in the *ProcessDefinition* and *ProcessInstance* classes. Attributes can be easily used to control the flow (when the conditions are based on them) and enable the communication between processes (as one process can query other process attributes and can change their values).

The *ProcessMonitor* is a Java based application, that can be run as a separate thread on a separate machine. Its duty is to periodically check (basing on timeouts) each *ProcessInstance*. Then, according to the values retrieved from condition codes, the *ProcessMonitor* executes the inner code of the process and pushes it forward through the workflow.

IV. SAMPLE DECLARATIVE WORKFLOW DEFINITION

As an illustration, we have created a sample workflow, which utilizes basic concepts of our idea. The workflow application supports processing of a credit request within a bank. It is a complex structure of active objects representing various tasks. The structure is presented in Fig. 3. Apart from objects representing processes, there are resource objects that are available through names such as *Customer*, *ApplicationForm*, *Account* and *Contract*. A rough scenario for the *Request* process is described below.

1. A customer submits an application for a credit in the form of an *ApplicationForm* object.
2. After checking that all of necessary resources are available the *Analysis* sub-process is activated.
3. The data is checked formally by the analyst for formal and business correctness (Initial formal check).
4. If the data is incorrect, the customer is informed about that and further processing of the application is suspended (Suspension) until reaction of the customer is received. If there is no reaction the application is rejected, and the customer is informed about that by an appropriate e-mail message (Rejection).

5. If the data is correct the client rating is calculated (Calculate Client Rating).
6. After successful calculating of the client rating, a check is performed if the amount of the credit does not exceed the general bank limit (Calculate general limit).
7. A positive result of the Analysis sub-process activates the Verification sub-process.
8. Verification consists of two stages:
    a. Checking if the customer is not present in the government registry of persons having debts;
    b. Checking if the customer has an account within the bank; if not, creating such an account.
9. If this sub-process is successfully completed, the sub-process Ratification is triggered.
10. The sub-process Ratification is split into sub-processes:
    a. Checking if the customer's current income is sufficient for the requested credit (Final check).
    b. Preparing contract for the customer (Preparing contract);
    c. Sending information to the customer (Information to customer);
    d. Signing the contract with the customer (Signing contract);
    e. Transfer of the money to the customer's account (Money transfer);
    f. Checking and sending information to the government registry of customers that apply for credits (to avoid many applications of the same customer to different banks submitted at the same time).
11. If these tasks are completed (successfully for the customer or not), the process instance is terminated.
12. If at any stage the application is rejected the appropriate information is sent to the customer.

Let us consider the *Ratification* sub-process in more detail. It consists of six sub-processes: *Final check, Preparing contract, Information for customer, Signing contract, Money transfer* and *Information* to debts registry. In order to start the *Ratification* process the fire condition should check if the parent's attribute 'state' is empty and the Verification process has got finished status. This condition means that the application has not been rejected yet and the Verification sub-process is finished. After satisfying the fire condition, *Ratification* process changes its status to *Active* and all of its children changes status to *Waiting*. A *Ratification* process is ended when all its children are completed or when the application is rejected.

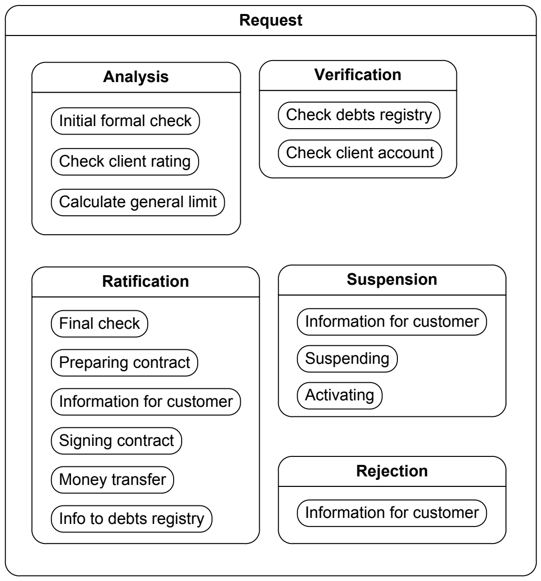


Figure 3. Structure of the Request process

When the *Ratification* is active, the fire condition of the child with the name *Final check* is checked. It fires as soon as its parent has got active status. When the process activates, the code of this task is executed. The purpose of this code is to check if the customer can afford such a credit and according to that sets the proper value to the attribute "state" of the *Request* object. The *endcode* of the *Final check* is absent, hence the process ends immediately after completing the execution code.

The next process in order is *Preparing contract*. It fires as soon as *Final check* is finished and the state attribute of a *Request* process has the "accepted" value. The purpose of this process is to create a new *Contract* object assigned to an application form filled by the customer with start date equal to the current date and with an attachment being a reference to the application form of the customer. If the contract has been successfully created the process ends.

Finishing of *Preparing contract* process activates *Information for customer*. The main task of this process is to send an e-mail to the customer with the information that the contract is ready to sign up. Depending on the result of this operation the attribute *mailSent* is set with a proper value. If sending does not succeed, the status of the process is changed to *Waiting*, so the next process monitor check will trigger its run again. When the e-mail is sent the process ends. After informing the customer on the contract, the processing waits for the signature. The next process *Signing contract* provides information if a contract has been already signed or not. It is started after finishing *Preparing contract* and is active till a *contractSigned* attribute is false.

When the contract is signed, the bank transfers the money into the customer account. The *Money transfer* process is responsible for this action. It is activated when the *Signing contract* process is finished.

The execution code for this process updates the amount attribute from the customer's *Account* object with the value of the *creditAmount* attribute from the specific *ApplicationForm* object. The process ends immediately after completing this action (no *endcondition*).

The last action in the ratification procedure is sending an info about a customer to a debts registry. After completing all the sub-processes the *Ratification* process is finished.

The manager of workflow processes can do any changes to process instances, including currently running instances by simple database updates. For instance, for any reason he/she can delete active object *Check client rating* from active object *Analysis* for the given customer *Request*. It is possible that in such a case the *endcondition* of the *Analysis* object should be changed too.

V. Conclusion and Future Work

The perception of workflow processes as autonomous objects can be very useful in terms of maintaining and managing process definitions and execution. Controlling the process execution with fire and end conditions gives a workflow creator a powerful tool to create very flexible and advanced control structures. Moreover every process attribute such as conditions, execution code etc. can be accessed in every moment of the process lifetime, which gives the opportunity to apply a changes to an already working workflow instance if needed. The mentioned features had been successfully implemented in working prototype. It gives a foundation to achieve important features like mass parallelism and flexible resource management.

The idea is very new, hence it presents a lot of opportunities for future research. One of the research lines concerns mass parallelism of processes and tasks executed on many (thousands of) servers. This require developing and implementing a process monitor and a task balancing tool. Another research concerns a user-friendly API for dynamic process changes. Proper modifications of notations such as BPMN (Business Process Modeling Notation) [22] and execution languages such as XPDL (XML Process Definition Language) [23] and BPEL (Business Process Execution Language) [24] could also be the subject of research. There is also a need for preventing running processes from undesired changes using such means as user rights, semi-strong type checking, triggers and business rules.

References

[1] IBM developer works: Business Process Execution Language for Web Services, ver. 1.1, May 2003.

[2] OMG. Business Process Modeling Notation (BPMN) specification. Final Adopted Specification. Technical Report, 2006

[3] WfMC, WorkFlow process definition interface – XML Process Definition Language. WfMC TC 1025 (2.1); October, 10, 2008

[4] Petri nets: http://www.petrinets.info/

[5] M. Reichert and P. Dadam. ADAPTflex: Supporting dynamic changes of workflow without loosing control. Journal of Intelligent Information Systems, 10(2), pp. 93-129, 1998

[6] C.A.Ellis. K.Keddara, GRozenberg. Dynamic change within workflow systems. Proc. ACM Conf. on Organisational Computing Systems (COOCS 95)

[7] C.A.Ellis. K.Keddara, and J.Wainer. Modelling workflow dynamic changes using time hybrid flow. In Workflow Management: Net based Concepts, Models, Techniques and Tools (WFM'98), 98(7), Computing Science Reports, pp. 109-128. Eindhoven University of Technology, 1998

[8] D.C.Ma, J.Y.-C.Lin, M.E.Orlowska. Automatic merging of work items in business process management systems. Proc. 10th Intl. Conf. on Business Information Systems (BIS2007), Poznań, Poland, 2007

[9] W.M.P. van der Aalst. Generic workflow models: How to handle dynamic change and capture management information? Proc. 4th Intl. Conf. on Cooperative Information Systems (CoopIS'99), Los Alamitos, CA, 1999

[10] S.Sadiq, O.Marjanovic, M.E.Orlowska. Managing change and time in dynamic workflow processes. Intl. Journal of Cooperative Information Systems (IJCIS), 9(1-2), 2000

[11] S.Sadiq, M.E.Orlowska. Architectural considerations in systems supporting dynamic workflow modification. Proc. 11th Conf. on Advanced Information Systems Engineering, CAiSE99, Heidelberg, Germany, 1999

[12] W.M.P. van der Aalst, A.H.M.Hofstede, B.Kiepuszewski, A.P.Barros. Workflow patterns. Distributed and Parallel Databases, 14(3), pp. 5-51, 2003

[13] G. Vossen, M. Weske: The WASA Approach to Workflow Management for Scientific Applications . In: Workflow Management Systems and Interoperability. ASI NATO Series, Series F: Computer and Systems Sciences, Vol. 164, pp. 145-164. Berlin: Springer 1998

[14] Impedance mismatch: http://www.sbql.pl/Topics/ImpedanceMismatch.html

[15] F. M. Maggi, A. J. Mooij, and W. M. P. van der Aalst, User-Guided Discovery of Declarative Process Models , 2011 IEEE Symposium on Computational Intelligence and Data Mining, 2011

[16] SBQL: Stack-Based Query Language: http://www.sbql.pl/

[17] ODRA: Description and Programmer Manual. http://www.sbql.pl/various/ODRA/ODRA_manual.html, 2008

[18] Groovy: A dynamic language for the Java Platform. http://groovy.codehaus.org/

[19] Grails: http://grails.org/

[20] Javascript: http://www.w3schools.com/js/default.asp

[21] Apache Tomcat: http://tomcat.apache.org/

[22] BPMN: Business Process Modeling Notation: http://www.bpmn.org/

[23] XPDL: XML Process Definition Language: http://www.wfmc.org/xpdl.html

[24] BPEL: Business Process Execution Language: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

Data Transformations using QVT between Industrial Workflows and Business Models in BPMN2

Corina Abdelahad, Daniel Riesco
Departamento de Informática
Universidad Nacional de San Luis
San Luis, Argentina
{cabdelah, driesco}@unsl.edu.ar

Alessandro Carrara, Carlo Comin, Carlos Kavka
Research and Development Department
ESTECO SpA
Trieste, Italy
{carrara, comin, kavka}@esteco.com

***Abstract*— Automation of business processes (workflows) has become today a key component to support the growth of organizations. Many standards in the field of business processes, however, are not directly applicable to the field of process engineering due to specific engineering workflow requirements. The Business Process Model and Notation (BPMN), a widely used notation to model business process which has been recently enhanced, allows to extend the use of workflows from the field of business process to the field of engineering. However, the large base of existing engineering workflows used currently by industry represents one of the main obstacles for the adoption of the new standards. Transformations play a pivotal role in the implementation of Model-Driven Architecture by providing a mechanism to express model refinement. Query/View/Transformation (QVT) is an OMG standard established to create queries, views and transformations of models. The QVT Relations, a component of QVT, allows to formalize model-driven transformations. ESTECO is a company which has developed a proprietary workflow modeler and an associated workflow engine. Even if its proprietary model has proven to be useful in the context of engineering processes, a standard-complaint industrial process flow will enable the building of unified and standardized models across all sectors of the business. This paper presents a model-to-model partial transformation using QVT between the ESTECO metamodel and the BPMN2 metamodel, where input data of ESTECO generates DataInputs, InputSets and IoSpecification for use in BPMN2. This transformation allows the conversion of most ESTECO industrial workflow to BPMN2, assuring portability between tools that support the BPMN2 standard.**

Keywords - BPMN2; business workflow; ESTECO; industrial workflow; metamodel.

I. INTRODUCTION

An industrial workflow is an automated business process usually used to execute complex processing tasks that requires many features that most business process models do not currently support [7]. These kinds of workflows are widely used in natural science, computational simulations, chemistry, medicine, environmental sciences, engineering, geology, astronomy, automotive industry and aerospace among other fields.

BPMI (Business Process Modeling Initiative) together with the OMG have developed the widely used BPMN notation for modeling business processes [2][5]. BPMN defines a formal notation for developing platform-independent business processes, opposed to specific definitions of business processes such as XPDL (XML Process Definition Language) [9] or BPEL4WS (Business Process Execution Language for Services Web) [8]. BPMN defines an abstract representation for the specification of executable business processes within a company, which can include human intervention, or not. BPMN also allows collaboration between business processes of different organizations. The last definition of the BPMN standard (the release 2.0) has been developed by taking as one of its objectives the overcoming of the limitations that prevented its use in scientific and engineering applications [1][2].

The definition of this new standard allows, for the very first time, to extend the use of workflows from the field of business process to of the field of engineering. Engineering workflows, which share many properties with well-known scientific workflows, are heavily used in industry today. Although they are widely used, there is currently no standard accepted for the definition of engineering workflows, despite the efforts of standard organizations in the field of business processes. The large base of existing engineering workflows used currently by industry, which will need to be transformed between proprietary legacy formats to the new standard in order to be executed, represents one of the main obstacles for the adoption of the new standards.

QVT is a standard relation language for model transformation defined by the OMG with a specification based on MOF and OCL[17]. The language consents to express a declarative specification of the relationships between MOF models and metamodels supporting complex object pattern matching. A QVT transformation defines the rules by which a set of models can be transformed into a different set [4]. Furthermore, it specifies a set of relations that the elements of the implicated models in the transformation must fulfill. The model types are represented by their corresponding metamodels. A relation in QVT specification consists in a set of transformation rules where a rule contains a source domain and a target domain [6]. A domain is a set of variables to be matched in a typed model, with each domain defining a candidate model and also having its own set of patterns [4].

The paper is structured in sections. Section II presents related works while section III describes the motivations of current research. The ESTECO metamodel used as the source model for transformations is described in section IV. Sections V and VI present the transformation architecture and the transformation between models respectively. The paper ends with conclusions in section VII.

II. Related work

Several works in the field of software engineering are related to the concept of transformation between models, and many of them use BPMN to model business process. To the best of our knowledge, no other research work has considered BPMN2 as the target model for transformation in the context of industrial workflows.

Marcel van Amstel et al. [12], investigate what factors have an impact on the execution performance of model transformation. This research estimates the performance of a transformation and allows to choose among alternative implementations to obtain the best performance.

In this same line, a model-to-model transformation between PICTURE and BPMN2 is presented in [11]. PICTURE is a domain-specific process modeling language for the public administration sector. The transformation allows to model administrative processes in PICTURE and to get BPMN2 models for these processes automatically, helping electronic government by making possible the implementation of supporting processes.

In [13], three sets of QVT relations are presented as a mean of implementing transformations in a model-driven method for web development. One of them transforms a high-level input model to an abstract web-specific model. The other two transform the abstract web model to specific web platform models.

An example application is presented in [14] to demonstrate an automated transformation of a business process model into a parameterized performance model.

III. Motivations

As mentioned before, engineering workflows are heavily used in industry today to execute complex processing tasks like simulation or optimization [3]. Current examples include the areas of automotive, aerospace, turbines and other industries where the development of complex products is modeled as an engineering process defined in terms of the collaboration of various engineering services with usually large exchange of information between them.

Both scientific and engineering workflows differ from business workflows in many aspects. For example, business workflows usually deal with discrete transactions, but engineering and scientific workflows in most cases deal with many interconnected software tools, large quantities of data with multiple data sources and in multiple formats. Also, engineering services have usually a very long execution duration and depend on the execution environment.

Even if engineering workflows have been used successfully since many years, most of the tools used to define and execute them are not based on standard technologies. Until now, a single standard could not be used to represent both the abstract view (used by the engineer to represent the process at the scientific domain) and the workflow representation used for execution (at workflow engine level). The use of standards like BPMN for abstract representation and BPEL for execution were proposed in the past, but never went too far in industry due to the need to support two different standards for the same workflow. BPMN2, however, defines a standard with support for both levels, with many different graphic editors and workflow engines available, making a business standard accessible for the very first time to industry to completely support engineering workflows.

With BPMN2, many companies will be tempted to support a standard workflow for engineering applications. However, it must be considered that there exists a large base of engineering workflows already designed and used currently by industry which cannot be just thrown away. In order to provide legacy workflow support, we propose a methodology for the transformation of legacy proprietary workflows into BPMN2 standard workflows. This approach will provide an extra incentive for companies to abandon proprietary workflows and move to standard technologies coming from the field of business processes.

However, the transformation is not without pain. The extra data and process requirements in engineering workflows need to be handled properly. Fortunately, BPMN2 has been defined with an extension facility which allows to add required constructions without breaking standard compliance.

As part of the methodology, this work presents a partial transformation for selected constructions of a widely used industrial engineering workflow to BPMN2 in order to present a valid path to perform legacy workflow conversion to a well-defined standard. The next section presents a short introduction to the legacy metamodel.

IV. ESTECO metamodel

The metamodel selected as an example is the workflow model used by ESTECO for modeling simulation workflows in the context of industrial multi-objective optimization. This workflow, which is typical in this kind of environments, includes one task node for each activity and data nodes used to represent input, output and temporary data objects. Data objects can represent simple data like integer, doubles, vectors, matrix or more complex data like files or databases. Activities correspond to the execution of simulators, scripts and other applications in local or remote locations. Usually each activity has associated a set of configuration files, which can be large (many gigabytes being common), and a set of inputs and outputs (which can also be very large files or databases). Distributed execution is required, meaning that the activities specified in the workflow can be executed in different nodes (on the grid or the cloud system), requiring data to be passed between them.

The next section provides a description of the framework used for the transformation by applying it to a subset of ESTECO's workflow.

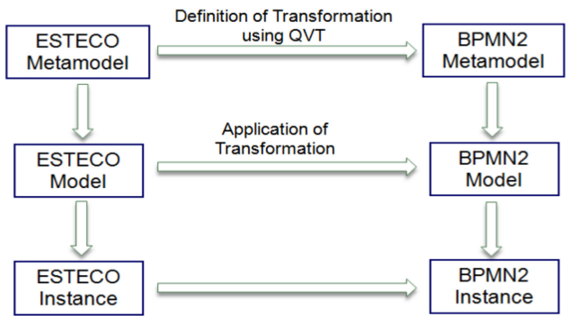


Figure 1. Transformation architecture.

V. TRANSFORMATION ARCHITECTURE

Our proposal aims to apply the most recent concepts of business processes to the field of engineering workflows in industrial fields. The use of standards in industry is important since it guarantees portability between tools that support BPMN2.

The industrial legacy workflow selected has an XML representation, allowing the use of tools like QVT for transformation. There is no one-to-one correspondence between the different components of ESTECO's workflow and BPMN2 constructions, since control nodes and data nodes are very differently handled in both models. Also, files and database handling put extra requirements which can only be handled properly with BPMN2 extensions.

The QVT transformations describe relations between the source metamodel and the target metamodel, both specified in MOF. The transformation defined is then applied to a source model, which is a part of ESTECO source metamodel, to obtain a target model, which is part of the BPMN2 target metamodel, as can be seen in Fig. 1.

The metamodels used in the definition of the transformation are shown at the top level. The specific models to which the transformation defined in the metamodel level is applied in order to obtain BPMN2 models is shown at the middle level. The lower level represents the instances of the models which can be executed in the corresponding workflow engines.

As mentioned before, activities and processes need data in order to be executed, and in addition, they can produce data during or as a result of their execution. In BPMN2, data requirements are captured as *DataInputs* and *InputSets*. The produced data is captured using *DataOutputs* and *OutputSets*. These elements are aggregated in an *InputOutputSpecification* class [2] as can be seen from Fig. 2. The *DataInputs* and *DataOutputs* are additional attributes of the *InputOutputSpecification* element; these elements are optional references to the *DataInputs* and *DataOutputs* respectively. A *DataInput* is a declaration that a particular kind of data will be used as input of the *InputOutputSpecification*. A *DataOutput* is a declaration that a particular kind of data can be produced as output of the *InputOutputSpecification*. *DataInputs* and *DataOutputs* are *ItemAware* elements. If the *InputOutputSpecification* defines no *DataInput*, it means no data is required to start an Activity. If the *InputOutputSpecification* defines no *DataOutput*, it means no data is required to finish an Activity [2].

A partial view of the ESTECO metamodel with the metaclasses involved in the relations described in this work is shown in Fig. 3. The *TInputDataNode* and

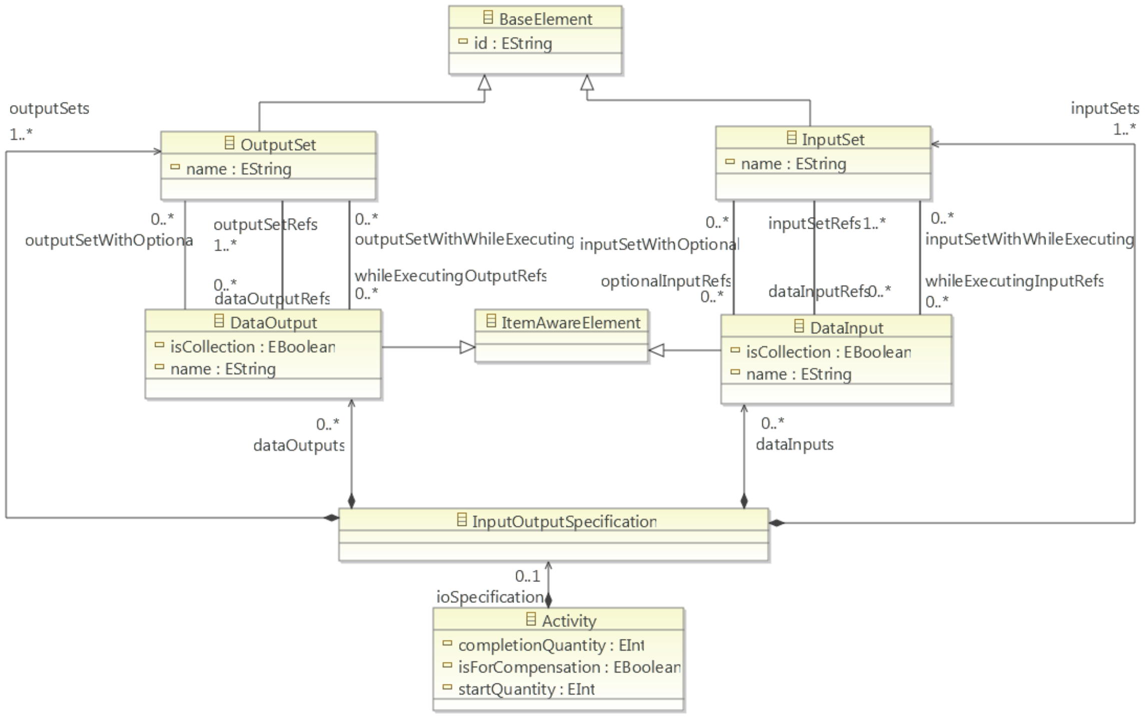


Figure 2. Partial view of the BPMN2 metamodel.

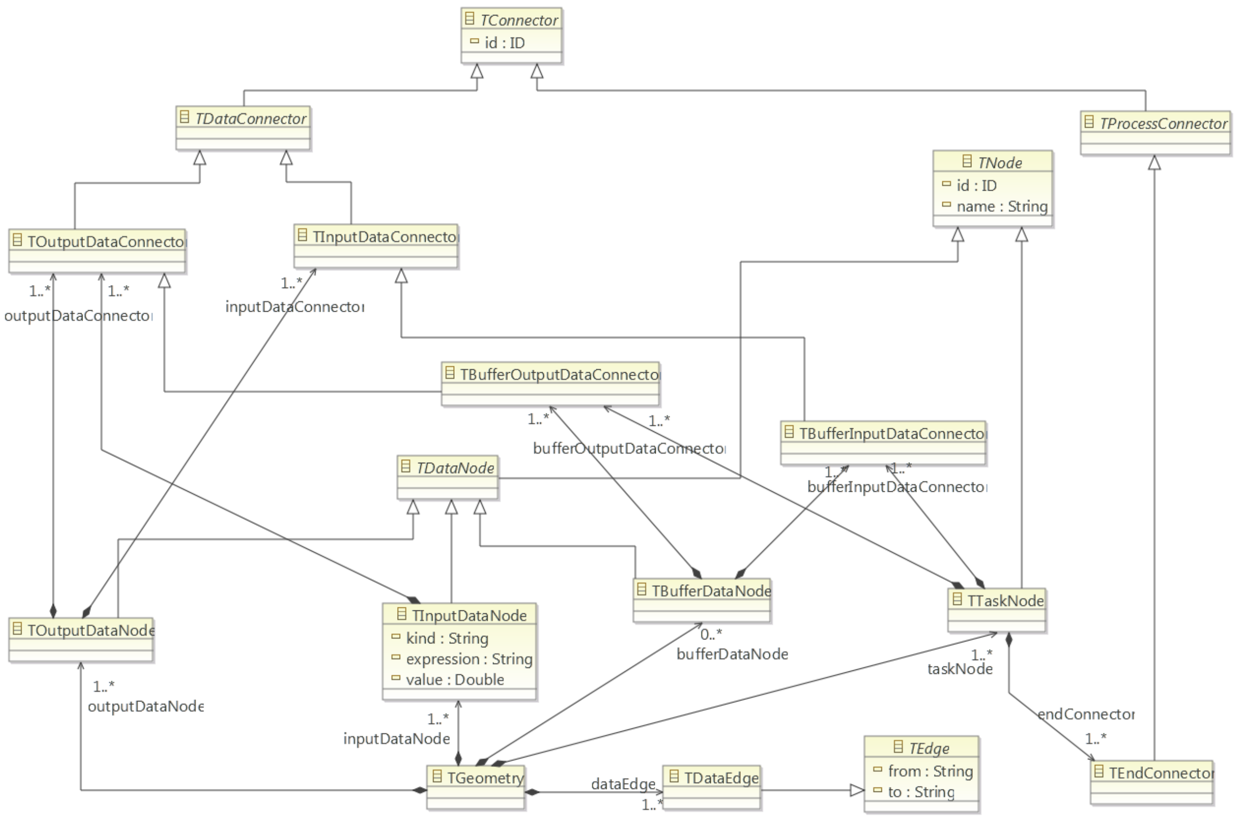


Figure 3. Partial view of ESTECO metamodel.

TOutputDataNode elements inherit the attributes and model associations of *TDataNode*, which in turn, inherits from *TNode*. The *TGeometry* class is the outermost object for all ESTECO elements, i.e., all these elements are contained in a *TGeometry*. The *TInputDataNode* element is a particular kind of *TDataNode* that will be used as input of *TGeometry* to a *Task*. The *TOutputDataNode* element is a particular kind of *TDataNode* which can be produced as output of a *Task* contained in *TGeometry*. A *TTaskNode* class represents the task that is performed within an industrial workflow.

VI. TRANSFORMATION BETWEEN MODELS

A transformation specifies a group of relations that the elements of the involved models must fulfill. A transformation may have any number of input or output parameters known as domains. For each output parameter, a new model instance is created according to the metamodel of the output metamodel (in this case, the metamodel BPMN2).

Each domain identifies a corresponding set of elements defined by means of patterns. A domain pattern can be considered an object template. A relation in QVT defines the transformation rules. A relation implies the existence of classes for each one of its domains. In a relation, a domain is a type that may be the root of a template pattern. A domain implies the existence of a property of the same type in a class. A pattern can be viewed as a set of variables and a set of constraints that model elements must satisfy. A template pattern is a combination of a literal that can match against instances of a class and values for its properties. A domain can be marked as *checkonly* or *enforced*. A *checkonly* domain simply verifies if the model contains a valid correspondence that satisfies the relation. When a domain is *enforced*, if checking fails, the elements of target model can be created, deleted or modified so as to satisfy the relationship.

A relation can contain two sets of predicates identified by a *when* or a *where* clause. The *when* clause specifies the condition that must be satisfied to execute the transformation. The *where* clause specifies the condition that must be satisfied by all model elements involved in the relation, and it may contain any variable involved in the relation and its domains [4]. In the context of transformation, a model type represents the type of the model. A model type is defined by a metamodel and an optional set of constraint expressions.

The transformation between ESTECO metamodel and BPMN2 metamodel is defined as follows:

```
transformation ESTECOToBPMN2(source : esteco_m,
                             target : bpmn2)
```

This transformation takes as input an ESTECO model, which is an instance of the ESTECO metamodel, and produces a BPMN2 model, that will be an instance of the BPMN2 metamodel.

Below, the relations which define the mapping between ESTECO metamodel classes and BPMN2 metamodel classes are shown. This correspondence is not straightforward. As we mentioned in the previous section, the *DataInput*s are captured in *InputSet*s and both are added into an *InputOutputSpecification*.

The same happens with the *DataOutput*s. So, in the transformation it is necessary to generate an *IoSpecification* object to aggregate *DataInput*s, *DataOutput*s, *InputSet*s and *OutputSet*s.

The relation used to create an *IoSpecification* object is shown below:

```
relation createIOSpecificationTask {
   checkonly domain source g:esteco_m::TGeometry  { };
   enforce domain target t:bpmn2::Task  {
   ioSpecification= ioSpecif :
                              bpmn2::InputOutputSpecification {}
  };
  primitive domain id_task:String;
  where {
   getDataInputTask(g,ioSpecif, id_task);
   createInputSetsTask(ioSpecif,ioSpecif);
   getDataOutputTask(g, ioSpecif, id_task);
   createOutputSetsTask(ioSpecif, ioSpecif);
   }
}
```

The relations used to create *InputSet*s and *OutputSet*s is presented below.

```
relation createInputSetsTask {
  checkonly domain target ioSpecif:
                             bpmn2::InputOutputSpecification {
  };
  enforce domain target ioSpecif :
                             bpmn2::InputOutputSpecification {
        inputSets = input_set :bpmn2::InputSet{
        dataInputRefs= ioSpecif.dataInputs
     }
  };
}
relation createOutputSetsTask {
  checkonly domain target ioSpecif:
                             bpmn2::InputOutputSpecification{
  };
  enforce domain target ioSpecif :
                             bpmn2::InputOutputSpecification{
       outputSets = output_set :bpmn2::OutputSet{
       dataOutputRefs= ioSpecif.dataOutputs
     }
   };
}
```

Note that an *InputSet* is a collection of *DataInput* elements that together define a valid set of data inputs associated to an *InputOutputSpecification*. An *InputOutputSpecification* must define at least one *InputSet* element. An *OutputSet* is a collection of *DataOutput*s elements that together can be produced as output from an Activity. An InputOutputSpecification element must have at least OutputSet element [2].

The relations used to obtain the *DataInputs* of the ESTECO model and the generation of *DataInputs* in BPMN2 is presented below.

```
relation getDataInputTask{
   id_input, name_input : String;
   value_input : Real;
   checkonly domain source g:esteco_m::TGeometry{
     taskNode = t:esteco_m::TTaskNode{
     bufferInputDataConnector = buffer_input :
                       esteco_m::TBufferInputDataConnector {}
     },
     inputDataNode = input : esteco_m::TInputDataNode {
       id = id_input,
       name = name_input,
       value = value_input,
       outputDataConnector = output_data :
                         esteco_m::TOutputDataConnector {}
     },
     dataEdge = data_edge : esteco_m::TDataEdge {}
   };
   enforce domain target ioSpecif :
                            bpmn2::InputOutputSpecification  {
         dataInputs = data_input : bpmn2::DataInput {
         id= id_input + '_T',
         name = name_input,
         itemSubjectRef = item : bpmn2::ItemDefinition {
             id = 'DoubleItemDefinition'
         }
       }
   };
   primitive domain id_task:String;
   when {
     if (data_edge.from = output_data.id) and
       (data_edge.to = buffer_input.id ) and
       (id_task=t.id) then true else false
  endif;
  }
}
```

Each data input of ESTECO must be transformed into a data input of BPMN2. This transformation is straightforward; the QVT code presented before shows the procedure by which the *id*, *name*, *value* and *connectors* are obtained.

The relation used to obtain the *DataOutput*s of ESTECO model and the generation of *DataOutput*s in BPMN2 is shown below.

```
relation getDataOutputTask{
   id_output, name_output : String;
   checkonly domain source g:esteco_m::TGeometry {
      taskNode = t:esteco_m::TTaskNode{
      bufferOutputDataConnector = buffer_output :
                  esteco_m::TBufferOutputDataConnector {}
      },
      outputDataNode = output :
                        esteco_m::TOutputDataNode {
         id = id_output, name = name_output,
         inputDataConnector = input_data :
                        esteco_m::TInputDataConnector {}
      },
      dataEdge = data_edge : esteco_m::TDataEdge {}
    };
   enforce domain target ioSpecif :
                       bpmn2::InputOutputSpecification  {
      dataOutputs = data_output : bpmn2::DataOutput {
      id= id_output + '_T',
      name = name_output,
      itemSubjectRef = item : bpmn2::ItemDefinition {
         id = 'DoubleItemDefinition'      }
      }
    };
    primitive domain id_task:String;
    when {
       if (data_edge.from = buffer_output.id) and
          (data_edge.to = input_data.id )  and
          (id_task=t.id) then  true else false
       endif;
   }
}
```

VII. CONCLUSION

In last years, business processes have gained popularity and have been subject to a large number of studies.

In the context of engineering, the execution of many parallel activities with complex interdependencies is required. At the same time, configuration control of the models should be maintained in order to ensure the traceability of the experiments, a requirement that is not necessarily considered in the typical business models. The efficient integration with platforms such Service Oriented Architecture (SOA) and Cloud Computing is also essential in the context of industrial workflows, a feature that is not considered in typical business workflows [15][16].

The objective of this work has been to apply the latest concepts of business processes to the industrial field. Furthermore, it intended to show the importance of the use of standards in the industrial field to guarantee portability between tools that support BPMN2.

In order to validate experimentally the methodology, the approach has been applied to the engineering environment supported by a company specialized in multi-objective optimization. Even if the company is currently working to fully support the standard for future workflows, the approach presented in this paper will allow to guarantee the support for legacy workflows by performing a transformation between the old metamodel to the BPMN2 standard metamodel. It is important to stress that this transformation allows the conversion of most ESTECO industrial workflows to BPMN2 consenting their execution in BPMN2 workflow engines with adequate extensions support.

ACKNOWLEDGMENT

The authors thank the reviewers for the very useful comments that have contributed to enhance the paper.

REFERENCES

[1] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau, and Jim Myers, “Examining the challenges of scientific workflows”. IEEE Computer vol. 40, no. 12, 2007, pp. 24-32.

[2] Object Management Group (OMG), “Business process model and notation”, http://www.omg.org/spec/BPMN/2.0 [retrieved: October, 2012]

[3] ESTECO S.p.A., “modeFRONTIER applications across industrial sectors involving advanced CAD/CAE packages”, http://www.esteco.com/home/mode_frontier/by_industry, [re-trieved: October, 2012]

[4] Object Management Group (OMG), “Meta object facility (MOF) 2.0 query/view/transformation, V1.1”, http://www.omg.org/spec/QVT/1.1 [retrieved: October, 2012]

[5] The Business Process Management Initiative (BPMI.org), http://www.bpmi.org/ [retrieved: October, 2012]

[6] Li Dan, “QVT based model transformation from sequence diagram to CSP”, 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), 2010, pp. 349-354.

[7] Li Hongbiao, Li Feng, and Yu Wanjun, “The research of scientific workflow engine”, IEEE International Conference on Software Engineering and Service Sciences (ICSESS), 2010, pp. 412-414.

[8] Oasis, “Web services business process execution language version 2.0”, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, [retrieved: October, 2012]

[9] Workflow Management Coallition (WfMC), “XML Process Definition Language”, http://www.xpdl.org, [retrieved: October 2012]

[10] Narayan Debnath, Fabio Zorzan, German Montejano, and Daniel Riesco. “Transformation of BPMN subprocesses based in SPEM using QVT”, IEEE International Conference on Electro/Information Technology, 2007, pp. 146-15.

[11] Henning Heitkoetter, “Transforming PICTURE to BPMN 2.0 as part of the model-driven development of electronic government systems”, 44th Hawaii International Conference on System Sciences (HICSS), 2011, pp. 1-10.

[12] Marcel van Amstel, Steven Bosems, Ivan Kurtev, and Luís Ferreira Pires, “Performance in model transformations:

experiments with ATL and QVT”, Lecture Notes in Computer Science, Volume 6707, Theory and Practice of Model Transformations, Springer, 2011, pp. 198-212.

[13] Ali Fatolahi, Stéphane Somé, and TimothyLethbridge, “Automated generation of abstract web models using QVT relations”, Technical Report TR-2010-06, School of Information Technology and Engineering, University of Ottawa, September 2010.

[14] Paolo Bocciarelli and Andrea D'Ambrogio, “A BPMN extension for modeling non functional properties of business processes”, TMS-DEVS'11 Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, Springer-Verlag, 2011, pp. 160-168.

[15] Cui Lin, Shiyong Lu, Xubo Fei, Artem Chebotko, Darshan Pai, Zhaoqiang Lai, Farshad Fotouhi, and Jing Hua, “A reference architecture for scientific workflow management systems and the VIEW SOA solution”, IEEE Transactions on Service Computing, vol. 2, no. 1, 2009, pp. 79-92.

[16] Gideon Juve and Ewa Deelman, Scientific workflows and clouds, ACM Crossroads, vol. 16, no. 3, 2010, pp. 14-18.

[17] Object Management Group (OMG), “Modeling and metadata specifications”, http://www.omg.org/spec, [retrieved: October 2012]

A Data-driven Workflow Based on Structured Tokens Petri Net

Nahla HADDAR
MIRACL/FSEGS laboratory :Computer Department
FSEGS, Airport avenue Km 4, BP 1013 Sfax 3018, Tunisia
Email :nhaddar@ymail.com

Mohamed TMAR, Faiez Gargouri
MIRACL/ISIMS laboratory :Computer Department
ISIMS, B.P. Technology Center:
242 Sfax 3021, Tunisia
Emails :mohamed.tmar@isimsf.rnu.tn, faiez.gargouri@gmail.com

***Abstract*—Business processes design and implementation within a company are mainly based on the specification of actors and their different tasks. Data and general information are transmitted in a very specific organization among actors, applications and the information system, which constitute a workflow. In this paper, we present an approach for workflow process modeling. The model is in charge of representing both control flow and shared data in the workflow process, and it can be analysed to verify its correctness before implementation. This workflow modeling approach has been implemented into *Opus* system that provides a set of graphical interfaces to model and execute the business process tasks. The system also provides a workflow engine that grants automatic workflow processing by interpreting the workflow process.**

***Keywords*—Workflow modeling; Workflow management system; Petri Nets; Data-driven approach; Structured token.**

I. Introduction

At the beginning of this century, workflow management concentrated on the design and documentation of business process [1]. Therefore, it focused on the dependencies between tasks and their sequencing, while data and resources played a very minor role. Many new approaches have been introduced, e.g, Petri Nets [2], Business Process Modeling Notation (BPMN) [3], Business Process Execution Language (BPEL) [4], etc.; but only a few of them are of ongoing interest in modeling the exchanged data flow in the business process. Moreover, the importance of data in business processes has increased progressively in recent years with the appearance of the data-driven approaches.

As execution and expressiveness have got more attention, also validation of the workflow model has needed to get attention. One big standard in this attribute is Petri Nets. Petri Nets are currently among the best known techniques for workflows specification [5].

In this paper, we present a formal approach inspired from the data-driven approach and the Petri Net formalism to model workflow processes. The resulting model can be analyzed for validation and automatically generated by the workflow engine for process execution.

The rest of the paper is organized as follows :we illustrate the related work in Section II and then, we elucidate our approach for workflow modeling in Section III. We illustrate in Section IV the possible information flows routing. Then, we demonstrate our approach by an example of workflow model in Section V. In Section VI, we explain how our workflow model can be analyzed and verified and we present our workflow management system *Opus* in Section VII. Section VIII concludes the paper.

II. Related Work

Many new approaches have emerged, which shifted their focus to combination of data flow and control flow. An emerging approach uses artifacts, that combine data and process by using atrifacts and Petri Nets model, is the Business Artifacts (BA) [6].

The BA approach focuses on solving decision problems, related to reachability, avoiding dead-ends and redundancy, but it does not provide a graphical notation for process modeling. Despite it was formally defined, the BA does not provide a formal mechanism for process verification. Process verification has been widely studied in workflow research, with states machines in Petri Nets [7], [8], graphs [9], data dependencies [10], etc.

Another formal approach based on Petri Nets model is the CorePro Framework [1]. The CorePro enables to model the data-driven specification and then, to create automatically the process structures based on given data structures in the model level. As well, CorePro provides some simple rules to verify the soundness properties of the data-driven process structures. However, it has skipped to retain the object states which have already been activated before the execution.

Many extensions of Petri nets in which tokens carry data have been defined in the literature, in order to improve expressiveness of workflow models. Data Nets (DN) [11] are an extension of Petri nets in which tokens are taken from a linearly ordered and dense domain, and transitions can perform whole place operations like transfers, resets or broadcasts. Although, a data net can be viewed as a constrained multiset rewriting systems (CMRS) enriched with whole-place operations. And, according to researches developped in [12], whole-place operations augment the expressive power of Petri nets only in the case of black indistinguishable tokens, but not for models in which tokens carry data taken from an ordered domain. Weakness refers here to the fact that the CMRS encoding simulates a lossy version of data nets, e.g., data nets in which tokens may get lost.

All the approaches mentioned above focus on the data routing and data managed by the process, but they consider activities as black-boxes in which application data is managed by invoked application components. Some of them, like DN, can apply transitions that read from or write to some data element, but with limited power to manage all the handled data element. This is why processes have to be modeled at a higher-level of abstraction to reflect the preferred work practice.

III. Workflow Modeling Using Structural Petri Net Tokens

We have inspired from Petri Nets to propose a new workflow modeling approach leading to a workflow process model. To manage all the data handled by the work procedures, we use the notion of data-driven process structures.

So, we describe the process by respective data structures, and we define a data structure as a pair $s = (C,\ D)$, where C is a list of attributes and D is a list of tuples, each tuple is an ordered set of attribute values. Formally, $\forall n,\ m \in \mathbb{N}$:

$$\begin{aligned} C &= (c_1, c_2 \dots c_n) \\ D &= \{(d_{1_1}, d_{1_2} \dots d_{1_n}), (d_{2_1}, d_{2_2} \dots d_{2_n}) \dots (d_{m_1}, d_{m_2} \dots \\ &\quad d_{m_n})\} \end{aligned}$$

Each attribute c_i is an ordered pair of attribute name n_i and type name t_i, such as :

$\forall i, t_i \in \{SmallInt,\ Int,\ BigInt,\ Float,\ Double,\ Real,\ Decimal,\ Char,\ Varchar,\ Text,\ Date,\ Year,\ Boolean\}$

$\forall i, j,\ d_{i_j} \equiv t_j$:an attribute value is a specific valid value for the type of the attribute.

The workflow process is defined as a Petri Net representing the work, where a place corresponds to a data structure that contains structured tokens (tuples) and a transition corresponds to a task. A workflow is then a quadruplet $WF = (S,\ T,\ Pre,\ Post)$ where :

- S is a finite set of data structures,
- T is a finite set of tasks,
- $Pre\ : S \times T \to \mathbb{N}$ is the pre-incidence matrix,
- $Post\ : T \times S \to \mathbb{N}$ is the post-incidence matrix.

A workflow process is defined by an oriented net with two node types representing *data structures* and *tasks* manipulating the tuples of these structures. A task consumes data structure tuples to produce others, which can then be consumed by other tasks.

A task t is said to be enabled if each input data structure $s \in S$ is marked with at least x_i tuples (refers to $Pre(s,\ t)$, which defines the weight of the edge from s to t). A firing of an enabled task t consumes x_i tuples from each input data structure s, and produces x_j tuples (refers to $Post(t,\ s)$) to each output data structure of t. $Post(t,\ s)$ is the weight of the edge from t to s.

We have to clarify that in our case we cannot be limited to a simple post-incidence matrix. In fact, each transition consumes an undefined number of tuples and produces a number belonging to a well determined range, depending on its processing (See Table I, in Appendix). For example, if a transition is a tuples union operation of two data structure s_1 and s_2 containing respectively x_1 and x_2 number of tuples, it will produce a number of tuples belonging to the interval :max($x_1,\ x_2$) and $x_1 + x_2$ (because the union operation eliminates the duplicated tuples).

So, we define two post-incidence matrices :$Post_{Min}$ and $Post_{Max}$, as a values interval which limits all possible post-incidence matrices. Formally :

$\forall t \in T$ and $s \in S$, $Post_{Min}(t,\ s)$:is the edge going from transition t to place s minimal weight.

$\forall t \in T$ and $s \in S$, $Post_{Max}(t,\ s)$:is the edge going from transition t to place s maximal weight.

$\forall t \in T$ and $s \in S$, $Post(t,\ s) \in [Post_{Min}(t,\ s),\ Post_{Max}(t,\ s)]$.

We explain this idea in details through the example in Figure 1.

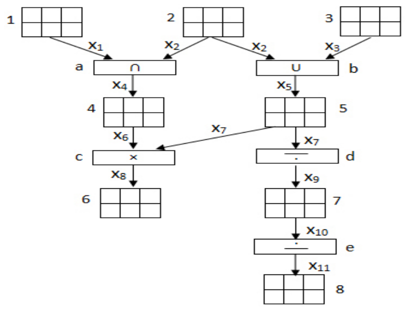


Figure 1. Example of workflow model

The example illustrated by Figure 1 contains eight places $(s_1,\ s_2 \dots s_8)$ and five transitions $(t_a,\ t_b \dots t_e)$. Each edge is associated with a weight $(x_i > 0)$.

We define its Pre matrix by :

$$Pre = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{ccccc} t_a & t_b & t_c & t_d & t_e \end{array} \\ \begin{pmatrix} x_1 & 0 & 0 & 0 & 0 \\ x_2 & x_2 & 0 & 0 & 0 \\ 0 & x_3 & 0 & 0 & 0 \\ 0 & 0 & x_6 & 0 & 0 \\ 0 & 0 & x_7 & x_7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_{10} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Following the example illustrated in Figure 1, and the definition of its transitions in Table I, we can determine the values range of output tokens for each fired transition as follows :

- $x_4 \in [0,\ min(x_1, x_2)]$
- $x_5 \in [max(x_2, x_3), x_2 + x_3]$
- $x_8 = x_6 \times x_7 \in [x_6 \times x_7,\ x_6 \times x_7]$
- $x_9 = x_7 \in [x_7,\ x_7]$
- $x_{11} = x_{10} \in [x_{10},\ x_{10}]$

So, we can deduce the matrices :

$Post_{Min}$:

$$\begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{ccccc} t_a & t_b & t_c & t_d & t_e \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & max(x_2, x_3) & 0 & 0 & 0 \\ 0 & 0 & x_6 \times x_7 & 0 & 0 \\ 0 & 0 & 0 & x_7 & 0 \\ 0 & 0 & 0 & 0 & x_{10} \end{pmatrix} \end{array}$$

$Post_{Max}$:

$$\begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{ccccc} t_a & t_b & t_c & t_d & t_e \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ min(x_1, x_2) & 0 & 0 & 0 & 0 \\ 0 & x_2 + x_3 & 0 & 0 & 0 \\ 0 & 0 & x_6 \times x_7 & 0 & 0 \\ 0 & 0 & 0 & x_7 & 0 \\ 0 & 0 & 0 & 0 & x_{10} \end{pmatrix} \end{array}$$

Using these three matrices (Pre, $Post_{Min}$ and $Post_{Max}$) we can derive several properties of the designed workflow model to be verified. We detail this idea in Section VI.

To reach the lowest level of abstraction, we need algebra over data structures. So, we have inspired from the relational algebra to define the tasks needed to produce data structures from others. As illustrated in Table I, we redefine the relational algebra operations in a formal way in order to suit the Petri Net formality. To keep equivalence between the attributes of data structures assigned to operations as Union, Difference, Intersection, and Division, we define the Permutation and Substitution operations.

Furthermore,we suggest an Extension operation to add attributes in a structure scheme, where its values are generated through applying a function. And finally, to insert data structure tuples in a data structure, we define the Alimentation operation.

As for example, we explain the Projection operation illustrated in Table I by the following example :

Whether the structure $Products = (C_j,\ D_j)$, where :

$C_j = (Id,\ designation,\ price,\ Stock)$,

$D_j = \{(1,\ aa,\ 20.5,\ 1000),\ (2,\ ab,\ 25.0,\ 2500),\ (3,\ ac,\ 22.75,\ 1500)$.

$Prod\ =\ (C_i,\ \ D_i)\ =\ \dot{-}(Products,\ \ b)$ such as $b = (1,\ 0,\ 0,\ 1)$.

So, q is defined as following :

$$q = \sum_{k=1}^{5} b_k = 2 \quad \Rightarrow C_i = (c_{j_{j'_1}},\ c_{j_{j'_2}})$$

$$j'_1 = \min_{\substack{l = \{1,2\ldots5\} \\ \sum_{p=1}^{l} b_p = 1}} l \ = 1 \quad \Rightarrow c_{i_1} = c_{j_1} = Id$$

$$j'_2 = \min_{\substack{l = \{1,2\ldots5\} \\ \sum_{p=1}^{l} b_p = 2}} l \ = 4 \quad \Rightarrow c_{i_2} = c_{j_4} = Stock$$

$$\Rightarrow Prod = ((Id,\ Stock), \{(1,\ 1000), (2,\ 2500), (3,\ 1500)\})$$

IV. Information Flows Routing

Our workflow model can express sequential, conditional and parallel routing flow.

$Sequential$ routing is used to deal with causal relationships between tasks [8]. Figure 2 shows that sequential routing can be modeled by our operations graph.

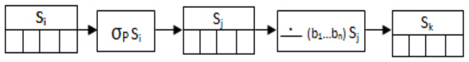


Figure 2. Sequential routing

$Parallel$ routing is used where two tasks B and C have to be executed at the same time. To model parallel routing, two building blocks are identified :The AND-Split and the AND-Join [8]. Figure 3 shows that both building blocks can be modeled by our operations graph.

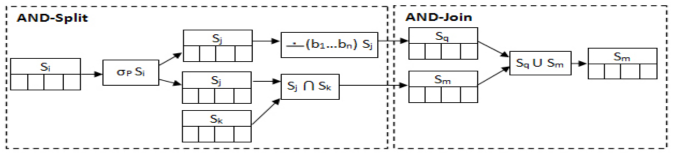


Figure 3. Parallel routing

$Conditional$ routing is used when there is a mutual execution between two tasks according to a condition. We can express conditional routing by a simple network using *control* operations.

Indeed, the control operation decides to continue, or not, the information flows routing according to the controlled data structure content. Whether s_i is the controlled data structure, s_j is the data structure expected by the next transition if the condition is verified, so, s_i will be controlled by one of the *control* operations which are defined as follows :

Control operation 1, noted $\pm$:

$$s_i \ \pm s_j = \begin{cases} s_i \ if \ s_j = \phi \\ \phi \ otherwise \end{cases}$$

Control operation 2, noted $\mp$:

$$s_i \ \mp s_j = \begin{cases} s_i \ if \ s_j \neq \phi \\ \phi \ otherwise \end{cases}$$

An example of control flow is illustrated in Figure 5 in Appendix, where the structure s_6, which contains all the unpaid bills of the current customer, is used by task t_5 to decide the customer solvency. So, if s_6 contains one or more tokens, t_5 will decide that the customer is not solvent, and it will finish the order management process. Otherwise, t_5 will reproduce s_2 tokens in s_7 in order to be sent to Inventory Check Role.

V. Example Modeled Using our Approach

Consider an office procedure for order processing within a company. When a customer sends his order by email, the job is sent to the customer solvency check, and then to the inventory check. After the evaluation, either a rejection mail is sent to the customer, or the order is sent to shipping and billing. In this paper, we restrict our example to the solvency check and the inventory check processes.

To simplify the representation of the model, we group the tasks related to the same function in the company according to roles. So, each role work is presented by a sub-process belonging to the whole workflow process definition.

As shown in Figure 5, when a customer mail arrives, the workflow will launch. S_1 tokens (present customers data) are to be consumed by t_1 in order to select the current customer (CC) information by his first name and his last name (the selection condition is to be seized by the Solvency Check Role (SCRole) during the execution of the workflow).

The resulted structure s_2 token (s_2 contains only one token presenting the CC information), and the s_3 tokens (present bills data of customers) are to be used by t_2 to produce a single data structure containing bills data, and the CC information. Resulted structure s_4 tokens are to be used by t_3 to create an inner join between bills data and the CC, in order to select only the CC bills. So, s_5 tokens present the CC history on bill payment. To check customer solvency, t_4 selects only s_5 tokens which have a paid attribute value equals to false. The resulted structure s_6 is to be then used to decide the customer solvency. Task t_5 is a control operation, which verifies s_6 content. If s_6 contains one or more tokens, t_5 will decide that the customer is not solvent (because he has unpaid bills), and it will finish the order management process. Otherwise, t_5 will reproduce s_2 tokens in s_7 in order to be sent to Inventory Check Role (ICRole).

To select the ordered products, t_6 extends s_8 (contains all products data) by the *ord_qtity* attribute (accepts only integer values), in order to allow the ICRole to seize the ordered quantities relatively to the ordered products. Then, t_7 selects from the resulted structure s_{10} only tokens having an ordered quantity value higher than zero and lower than the stocked product quantity. The resulted tokens are stocked in s_{11}.

In parallel, t_8 applies a projection operation on s_7, to get the structure s_9 having as a token, the CC identifier. If there are available ordered products, the ICRole has to create a new order. To verify availability, we define the control operation in t_9. If s_{11} contains one or more tokens (there is, at least, one available product), t_9 will reproduce s_9 token in s_{12}, then, t_{10} will add a new order in s_{13}. It remains to create the new order lines. So, the ICRole has to seize the new order identifier, t_{13} will save his seizure in s_{17}. Then, t_{14} will create the new order lines by applying a simple inner product between s_{17} token and s_{15} tokens (present identifiers of the ordered products and their relative ordered quantities).

VI. The Workflow Verification

We provide techniques based on Pre and Post matrices, to ensure that WF satisfies the minimum requirements for correctness.

First of all, we verify that each data structure is the result of at most a single transition. Formally, consider n places and m transitions in the workflow model :$\forall\ i \in \{1,\ 2 \ldots n\}$,

$$\forall\ i \in \{1,\ 2 \ldots n\},\ |j \in \{1,\ 2 \ldots m\},\ Pre(s_i,\ t_j) \neq 0| \leq 1. \tag{1}$$

To explain Equation 1, we resume the example in Figure 1, and we verify s_4. So, for $i = 4$:

$|j \in \{a \ldots e\},\ Pre(s_4,\ t_j) \neq 0| = |x_6| = 1 \ \Rightarrow\ s_4$ verifies the condition.

In the rest of this section we focus on the verification of liveness property of the model. For us, to verify this property, we have to begin with defining the initial and the final marking of WF.

Formally, the initial marking i is defined as : $i = \begin{pmatrix} i_1 \\ i_2 \\ \ldots \\ i_n \end{pmatrix}$,

such as :$\forall j \in \{1,\ 2, \ldots n\}$

$$i_j = \begin{cases} \max\limits_{k \in \{1,\ 2, \ldots m\}} Pre(s_j,\ t_k), \\ \quad if\ \forall l \in \{1,\ 2, \ldots m\} Post_{Max}(s_j,\ t_l) = 0. \\ 0, \text{otherwise.} \end{cases} \tag{2}$$

We explain Equation 2 using the example in Figure 1 : $\forall j \in \{1 \ldots 8\}$, the condition $\forall l \in \{a \ldots e\} Post_{Max}(s_j,\ t_l) = 0$ return $true$ only for $j = 1$, $j = 2$ and $j = 3$. So, $\forall j \in \{4 \ldots 8\}$, $i_j = 0$.

For $j = 1$: $\max\limits_{k \in \{a \ldots e\}} Pre(s_1,\ t_k)$ return $Pre(s_1,\ t_a) = x_1$.

$\Rightarrow\ i_1 = x_1$

For $j = 2$: $\max\limits_{k \in \{a \ldots e\}} Pre(s_2,\ t_k)$ return $Pre(s_2,\ t_a) = x_2$ (or $Pre(s_2,\ t_b) = x_2$).

$\Rightarrow\ i_2 = x_2$

For $j = 3$: $\max\limits_{k \in \{a \ldots e\}} Pre(s_3,\ t_k)$ return $Pre(s_3,\ t_b) = x_3$.

$\Rightarrow\ i_3 = x_3$

As we define an interval for Post matrices, we define an interval for final possible markings. Formally, $\forall j \in \{1,\ 2, \ldots n\}$: A minimal final marking o^- is defined as :$o^- = \begin{pmatrix} o_1^- \\ o_2^- \\ \ldots \\ o_n^- \end{pmatrix}$, where :

$$o_j^- = \begin{cases} \max\limits_{k \in \{1,\ 2, \ldots m\}} Post_{Min}(s_j,\ t_k), \\ \quad if\ \forall l \in \{1,\ 2, \ldots m\} Pre(s_j,\ t_l) = 0. \\ 0,\ otherwise. \end{cases} \tag{3}$$

Let us calculate o^- of the model in Figure 1 :$\forall j \in \{1 \ldots 8\}$, the condition $\forall l \in \{a \ldots e\} Pre(j, l) = 0$ return $true$ only for $j = 6$, $j = 8$. So, $\forall j \in \{1 \ldots 8\} \backslash \{6,8\}$, $o_j^- = 0$.

For $j = 6$:

$\max\limits_{k \in \{a \ldots e\}} Post_{Min}(s_6,\ t_k)$ return $Post_{Min}(s_6,\ t_c) = x_6 \times x_7$.

$\Rightarrow\ o_6^- = x_6 \times x_7$

For $j = 8$:

$\max\limits_{k \in \{a \ldots e\}} Post_{Min}(s_8,\ t_k)$ return $Post_{Min}(s_8,\ t_e) = x_{10}$.

$\Rightarrow\ o_8^- = x_{10}$

The maximal final marking o^+ is defined as o^- but with using the $Post_{Max}$ matrix instead of the $Post_{Min}$:

$o^+ = \begin{pmatrix} o_1^+ \\ o_2^+ \\ \ldots \\ o_n^+ \end{pmatrix}$, such as :

$$o_j^+ = \begin{cases} \max\limits_{k \in \{1,\ 2, \ldots m\}} Post_{Max}(s_j,\ t_k), \\ \quad if \forall l \in \{1,\ 2, \ldots m\} Pre(s_j,\ t_l) = 0. \\ 0,\ otherwise. \end{cases} \tag{4}$$

Assuming i as the initial state, o as the final state of a process, the workflow model is live if and only if :

- For every state M reachable from state i, there exists a firing sequence leading from state M to state o [8]. We adopt this rule to WF by applying the following rule : Whether R^+ (resp. R^-) is the net presenting the maximal (resp. minimal) output function, such as :
  $R^+ = (S,\ T,\ Pre,\ Post_{Max})$,
  $R^- = (S,\ T,\ Pre,\ Post_{Min})$,
  $$\begin{aligned}\forall_M(i \overset{*}{\to} R^+(M)) \Rightarrow (R^+(M) \overset{*}{\to} o^+),\\ (i \overset{*}{\to} R^-(M)) \Rightarrow (R^-(M) \overset{*}{\to} o^-).\end{aligned} \qquad (5)$$
- There are no dead transition in the workflow model [8]. We also adopt this rule to WF by applying the following rule :
  $$\forall_{t \in T} \exists_{M,M'}, (i \overset{*}{\to} R^+(M) \overset{t}{\to} M'). \qquad (6)$$

We define the simple algorithm below to ensure the verification of Equations 5.

Algorithm 1 Verification

```
/* $t$ is a task to verify */
Procedure VerificationT(t)
VerificationT(t)= $\bigwedge_{i \in \{1,\ 2 \dots n\},\ Pre(s_i,t) \neq 0}$ VerificationS($s_i$);
Procedure VerificationS(s)
/* $s$ is a root node */
if $\forall j \in \{1,\ 2 \dots m\},\ Post_{Max}(s,\ t_j) = 0$ then
    VerificationS(s)=true;
else
    /* $t_i$ is the task which has $Post_{Max}(s,\ t_i) \neq 0$ */
    VerificationS(s)=VerificationT($t_i$) where $Post_{Max}(s,\ t_i) \neq 0$ ;
end if
```

We apply Algorithm 1 on the example illustrated in Figure 1, and we choose to verify task t_e since its output data structure is a final state in the model; so, its verification generates the verification of all firing sequences leading from a state M to this final state.

VerificationT(t_e)= $\bigwedge_{i \in \{1,\ 2 \dots 8\},\ Pre(s_i,t_e) \neq 0}$ VerificationS(s_i)
= VerificationS(s_7) = VerificationT(t_d)
= VerificationS(s_5) = VerificationT(t_b)
= VerificationS(s_2) $\wedge$ VerificationS(s_3)
= true $\wedge$ true = true.

To verify Equation 6, we have to verify that the model is without structural conflicts. we assume that WF has a structural conflict if it contains at least two tasks t_i and t_j having the same input data structure s. As the case in Figure 1, the model has a structural conflict caused by t_b and t_c which share s_2. To avoid these cases, we extend the model by adding extra tasks $T^* = \{t_{copy_1}, t_{copy_2} \dots t_{copy_k}\}$ such as k is the number of data structures which cause conflicts, and t_{copy} is a $Copy$ operation (See Table I), which allow to create copies from a shared data structure to satisfy the need of tasks in a conflict.

The extended model $WF_+ = (S_+,\ T_+,\ Pre_+,\ Post_+)$ is defined as follows :$S_+ = S$, $T_+ = T \cup T^*$, $Pre_+ = S \times T_+$ and $Post_+ = T_+ \times S$.

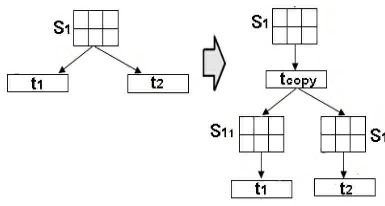


Figure 4. Removing the conflict

So, to resume, Algorithm 1 can verify that WF_+ is live.

VII. Implementation of the Workflow Management System *Opus*

The *Opus* workflow system consists of a number of components including a workflow engine and a Petri Net editor. Workflow specifications can be designed using the *Opus* editor and deployed in the *Opus* engine for execution.

The *Opus* engine follows the workflow model definition and interprets automatically the code executing the workflow. Then it invites each role to perform its tasks according to its feasibility and urgency. The verification of the conceived model is automatically ensured as follows in Algorithm 1 and Equation 1. To integrate workflows with the Information System (IS), we developed some tools, e.g., the Import tool (it imports a table tuples to a definite data structure belonging to the workflow process), the ImportId tool (it imports the tuple identifier of the last tuple inserted in a definite table), the Insert tool (it inserts data structure tuples in a definite IS table) and the Update tool (it updates a table in the IS with a data structure tuples). To perform these operations, and operations which requires two identical data structure schemes, *Opus* system is equipped with a matching tool, which uses the Substitution and the Permutation operations.

VIII. Conclusion and Future Work

The proposed approach is modular in a sense that the workflow process is to be decomposed on sub-processes which facilitates any eventual updates on the workflow process model. In fact, the changes related to the evolution in the role work, causes the change of its sub-process without damaging other sub-processes. In particular, the detailed formal definition of tasks and data structures is useful for the *Opus* engine, to extract all the process specifications. However, this approach must be completed by many functionalities. In fact, we plan to provide techniques to verify others Petri Nets property, like boundness, soundness, etc. We also plan to implement a simulation tool to decision-makers, in order to improve the business process, and a module for documents generation (invoice, purchase order, etc.) :the system can manage the content but not the container.

References

[1] D. Müller, M. Reichert, and J. Herbst, "Data-driven modeling and coordination of large process structures," in *OTM Conferences (1)*, 2007, pp. 131–149.

[2] C. Petri, "Communications with automata," Ph.D. dissertation, Institut für instrumentelle Mathematik, Bonn, 1962.

[3] Object Management Group (OMG), "Business process model and notation (bpmn)(version 2.0)," Tech. Rep., 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0/

[4] OASIS WSBPEL Technical Committee, "Web services business process execution language version 2.0," 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[5] W.V.D. Aalst, J.M. Colom, F. Kordon, G. Kotsis, and D. Moldt, *Petri Net Approaches for Modelling and Validation*, ser. Lincom Studies in Computer Science, 2003, vol. 1.

[6] A. Nigam and N.S. Caswell, "Business artifacts: An approach to operational specification," *IBM Syst. J.*, vol. 42, no. 3, pp. 428–445, July 2003.

[7] W.V.D Aalst, "Verification of workflow nets," in *Application and Theory of Petri Nets 1997*, P. Azéma and G. Balbo, Eds., vol. 1248, 1997, pp. 407–426.

[8] W.V.D Aalst, "The application of petri nets to workflow management," *Journal of Circuits, Systems, and Computers*, vol. 8, no. 1, pp. 21–66, 1998.

[9] W. Sadiq and M.E. Orlowska, "Analyzing process models using graph reduction techniques," *Information Systems*, vol. 25, pp. 117–134, 2000.

[10] S.X. Sun, J.L. Zhao, J.F. Nunamaker, and O.R.L. Sheng, "Formulating the data-flow perspective for business process management," *Information Systems Research*, vol. 17, no. 4, pp. 374–391, 2006.

[11] R. Lazic, T.C. Newcomb, J. Ouaknine, A.W. Roscoe, and J. Worrell, "Nets with tokens which carry data," *Fund. Informaticae*, vol. 88, no. 3, pp. 251–274, 2008.

[12] P.A. Abdulla, G. Delzanno, and L. Van Begin, "A language-based comparison of extensions of petri nets with and without whole-place operations," in *Proceedings of the 3rd International Conference on Language and Automata Theory and Applications*, ser. LATA '09. Springer-Verlag, 2009, pp. 71–82.

APPENDIX

Table I
OPERATIONS DEFINITION

Operation	Formal definition
Named :Inner Product Description :Performs the combination of all structure tuples with those of another structure. Noted:×	$\forall\ s_j = (C_j,\ D_j),\ s_k = (C_k,\ D_k)$ $C_j\ = (c_{j_1},\ c_{j_2} \dots c_{j_{n_j}}),$ $C_k\ = (c_{k_1},\ c_{k_2} \dots c_{k_{n_k}})$ $s_i\ =\ s_j\ \times\ s_k$ $\Rightarrow s_i = ((c_{j_1} \dots c_{j_{n_j}}, c_{k_1} \dots c_{k_{n_k}}), D_i)$ Where : $D_i = \bigcup_{\substack{l \in \{1 \dots n_j\} \\ p \in \{1 \dots n_k\}}} \{(d_{{j_l}1} \dots d_{{j_l}n_j},\ d_{{k_p}1} \dots d_{{k_p}n_k})\}$ Resulted tokens number :$x_i = x_j \times x_k$
Named :Selection Description :Selects only the structure tuples that meet the desired criteria. Noted :σ	Whether P is the selection property, $\forall\ s_j = (C_j,\ D_j), s_i = \sigma_P s_j$ $\Leftrightarrow s_i = (C_j, \bigcup_{\substack{e \in D_j \\ P(e)}} \{e\})$ Resulted tokens number :$x_i \in [0,\ x_j]$
Named :Difference Description :Subtracts the tuples of a data structure from another one. Noted :−	$\forall\ s_j = (C,\ D_j),\ s_k = (C,\ D_k)$ $\Rightarrow\ s_j\ -\ s_k = (C,\ D_j\ -\ D_k)$ Resulted tokens number : $x_i \in [x_j - x_k,\ x_j]$
Named :Projection Description :Selects only the structure columns (attributes) that we are interested in. Noted :$\dot{-}$	$s_j = (C_j, D_j), \forall (b_1 \dots b_n) \in \{0,1\}^n$ $s_i = (C_i, D_i) = \dot{-}_{(b_1 \dots b_n)} s_j$ Where c_i is a selected (resp. not selected) attribute, if $b_i = 1$ (resp $b_i = 0$). $\Leftrightarrow \begin{cases} C_i = (c_{j_{j'_1}},\ c_{j_{j'_2}} \dots c_{j_{j'_q}}) \\ D_i = \{(d_{j_{1_{j'_1}}},\ d_{j_{1_{j'_2}}} \dots d_{j_{1_{j'_q}}}), \\ (d_{j_{2_{j'_1}}},\ d_{j_{2_{j'_2}}} \dots d_{j_{2_{j'_q}}}) \dots (d_{j_{m_{j\,j'_1}}}, \\ d_{j_{m_{j\,j'_2}}} \dots d_{j_{m_{j\,j'_q}}})\} \end{cases}$ Such as : $q = \sum_{k=1}^{n} b_k$:is the number of attributes in the structure result. And : $j'_k = \min_{\substack{l = \{1,2 \dots n\} \\ \sum_{p=1}^{l} b_p = k}} l$:refers to the projection attributes indices. Resulted tokens number : $\begin{cases} x_i = 0,\ if\ x_j = 0 \\ x_i \in [1,\ x_j],\ otherwise \end{cases}$
Named :Union Description :Groups the tuples of two structures into a single one. Noted :$\cup$	$\forall\ s_j = (C,\ D_j),\ s_k = (C,\ D_k)$ $\Rightarrow\ s_j\ \cup\ s_k = (C,\ D_j\ \cup\ D_k)$ Resulted tokens number : $x_i \in [Max(x_j, x_k),\ x_j + x_k]$
Named :Intersection Description :Retrieves the common tuples of two structures. Noted :$\cap$	$\forall s_j = (C,\ D_j),\ s_k = (C,\ D_k)$ $s_j\ \cap\ s_k = (C,\ D_j \cap D_k)$ Resulted tokens number : $x_i \in [0,\ Min(x_j, x_k)]$
Named :Division Description :Allows to get a data structure tuples that are associated with all tuples of another structure. Noted :÷	$\forall s_j\ =\ (C_j,\ D_j),\ s_k\ =\ (C_k,\ D_k)$ Where :$C_j\ =\ (c_1 \dots c_{n_j}),$ $C_k\ =\ (c_1, \dots c_{m_j})$ If $n_j > m_j$ then : $\begin{cases} s_i\ = s_j\ \div\ s_k\ =\ (C_i,\ D_i) \\ C_i\ =\ (c_{m_j+1},\ c_{m_j+2} \dots c_{n_j}) \\ \forall\ q\ \in\ D_i,\ D_k\ \times\ q\ \in\ D_j \end{cases}$ Resulted tokens number : $x_i \in [0,\ E(x_j/x_k)]$
Named :Substitution Description :Changes a structure attribute name. Noted :$\boxslash$	$\forall s_j = (C_j,\ D_j),\ s_i = \boxslash(c_{j_k},\ c,\ s_j)$ $\Leftrightarrow s_i = ((c_{j_1} \dots c_{j_k-1}, c, c_{j_k+1}$ $\dots c_{j_n}),\ D_j)$ Resulted tokens number :$x_i = x_j$
Named :Permutation Description :Allows to permute two columns in a data structure. Noted : $\circlearrowright$	$\forall s_i = (C_i,\ D_i), s_j\ =\ \circlearrowright\ (s_i,\ k,\ l)$ $\Leftrightarrow \begin{cases} k,\ l\ \in\ \{1,2 \dots n\} \\ k\ <\ l \\ C_j = (c_{i_1} \dots c_{i_{k-1}}, c_{i_l}, c_{i_{k+1}} \\ \dots c_{i_{l-1}}, c_{i_k}, c_{i_{l+1}} \dots c_{i_n}) \\ D_j = \{(d_{1\,i_1} \dots d_{1\,i_{k-1}}, d_{1\,i_l}, \\ d_{1\,i_{k+1}} \dots d_{1\,i_{l-1}}, d_{1\,i_k}, d_{1\,i_{l+1}} \\ \dots d_{1\,i_n}), (d_{m\,i_1} \dots d_{m\,i_{k-1}}, \\ d_{m\,i_l}, d_{m\,i_{k+1}} \dots d_{m\,i_{l-1}}, d_{m\,i_k}, \\ d_{m\,i_{l+1}} \dots d_{m\,i_n})\} \end{cases}$ Resulted tokens number :$x_i = x_j$
Named :Extension Description :Extends a structure scheme by adding a attribute c =(n,t) and applying a function f. Noted :$\underset{\cdot}{-}$	$\forall s_j = (C_j,\ D_j),\ s_i = \underset{\cdot}{-}(s_j,\ c,\ f)$ $\Leftrightarrow s_i = ((c_{j_1}, c_{j_2} \dots c_{j_n}, c), \{(d_{j_{1_1}},$ $d_{j_{1_2}} \dots d_{j_{1_n}},\ f(d_{j_{1_1}}, d_{j_{1_2}} \dots d_{j_{1_n}},$ $D_j)) \dots (d_{j_{m_1}}, d_{j_{m_2}} \dots d_{j_{m_n}}, f(d_{j_{m_1}},$ $d_{j_{m_2}} \dots d_{j_{m_n}},\ D_j))\})$ Resulted tokens number :$x_i = x_j$
Named :Add_Tuple Description :Add a tuple of data d in a data structure. Noted :+	$\forall s_j = (C_j,\ D_j),\ D_k = (d_{k\,1}, d_{k\,2} \dots d_{k\,n}),$ $s_i = +(s_j,\ D_k)$ $\Leftrightarrow s_i = ((c_{j_1} \dots c_{j_n}), \{(d_{j_{1_1}}, d_{j_{1_2}} \dots$ $d_{j_{1_n}}) \dots (d_{j_{m_1}}, d_{j_{m_2}} \dots d_{j_{m_n}}), (d_{k\,1},$ $d_{k\,2} \dots d_{k\,n})\})$ Resulted tokens number :$x_i = x_j + 1$
Named :Copy Description :Makes n copies of a data structure. Noted :t_{copy}	$\forall s_i = (C_j,\ D_j),$ $t_{copy}(s_i,\ n) = \{S_{j_1}, S_{j_2} \dots S_{j_k}\},$ where $k \in \{1,2 \dots n\}$ and $s_j = s_i$. Resulted tokens number :$x_j = x_i$

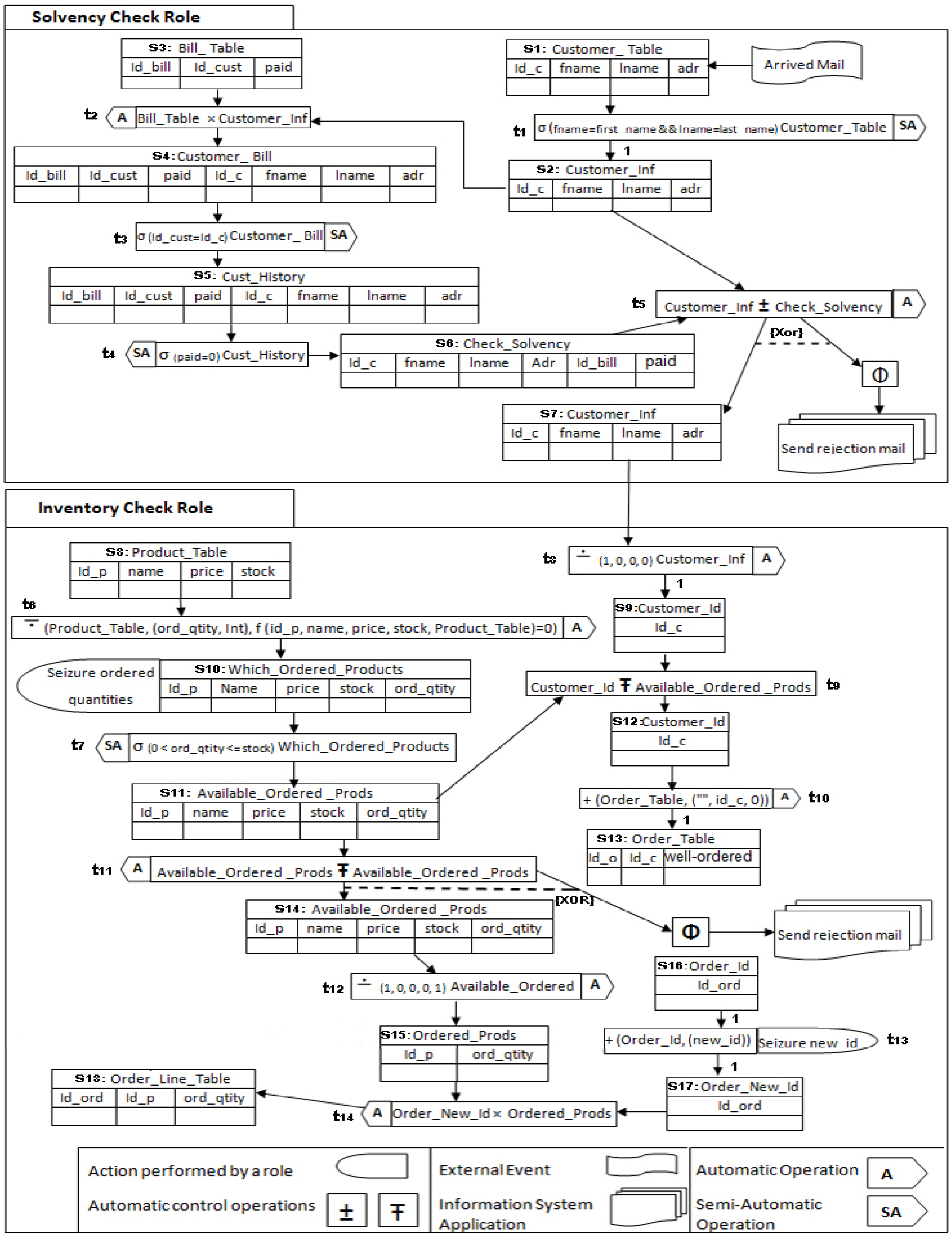

Solvency Check Role
S3: Bill_ Table
Id_bill Id_cust paid
S1: Customer_ Table
Id_c fname lname adr
Arrived Mail
t2 A Bill_Table × Customer_Inf
t1 σ (fname=first name && lname=last name) Customer_Table SA
S4: Customer_ Bill
Id_bill Id_cust paid Id_c fname lname adr
S2: Customer_Inf
Id_c fname lname adr
t3 σ (Id_cust=Id_c) Customer_ Bill SA
S5: Cust_History
Id_bill Id_cust paid Id_c fname lname adr
t5 Customer_Inf ± Check_Solvency A
{Xor}
t4 SA σ (paid=0) Cust_History
S6: Check_Solvency
Id_c fname lname Adr Id_bill paid
Φ
S7: Customer_Inf
Id_c fname lname adr
Send rejection mail
Inventory Check Role
S8: Product_Table
Id_p name price stock
t8 ∸ (1, 0, 0, 0) Customer_Inf A
t6 ∸ (Product_Table, (ord_qtity, Int), f (id_p, name, price, stock, Product_Table)=0) A
S9: Customer_Id
Id_c
Seizure ordered quantities
S10: Which_Ordered_Products
Id_p Name price stock ord_qtity
Customer_Id ∓ Available_Ordered _Prods t9
S12 Customer_Id
Id_c
t7 SA σ (0 < ord_qtity <= stock) Which_Ordered_Products
S11: Available_Ordered _Prods
Id_p name price stock ord_qtity
+ (Order_Table, ("", id_c, 0)) A t10
S13: Order_Table
Id_o Id_c well-ordered
t11 A Available_Ordered _Prods ∓ Available_Ordered _Prods
{XOR}
S14: Available_Ordered _Prods
Id_p name price stock ord_qtity
Φ
Send rejection mail
S16: Order_Id
Id_ord
t12 ∸ (1, 0, 0, 0, 1) Available_Ordered A
+ (Order_Id, (new_id)) Seizure new id t13
S15: Ordered_Prods
Id_p ord_qtity
S18: Order_Line_Table
Id_ord Id_p ord_qtity
S17: Order_New_Id
Id_ord
t14 A Order_New_Id × Ordered_Prods
Action performed by a role
Automatic control operations ± ∓
External Event
Information System Application
Automatic Operation A
Semi-Automatic Operation SA

Figure 5. Orders management workflow

Bankruptcy and Financial Standing Models Application for SMEs

David Plandor, Lenka Landryová
Department of Control Systems and Instrumentation
VSB – Technical University Ostrava
Ostrava, Czech Republic
e-mails: david.plandor@vsb.cz, lenka.landryova@vsb.cz

***Abstract*—This paper describes a system from software application research, development, implementation and testing. The final software tool is a web-based application used as a finance module by SMEs within the FutureSME portal, which supports small and medium companies for better competition on the market. The finance module offers three bankruptcy and financial standing models for evaluation of their financial health. The results are crucial for their ability to get a loan from banks. The self-assessment way of getting results does not require any advisor or bank representative. The described tool was used in practice by partnering companies.**

Keywords–finance module; financial health; bankrupcy model; financial standing model; SME.

I. INTRODUCTION

Nowadays, every SME (Small and Medium Enterprise) is in a situation when a loan is needed for future development, expansion or research. It is a long process to get the desired amount of money in a company's bank account. A company needs to hire a financial consultant or contact a bank directly. They have to fill out several forms. They have to expose their financial health to many people, which could be sometimes insecure, as when the company is in red numbers and crucial information leaked out to the public could cause a real problem. Why not use tools that are utilized by banks and use them by a company itself? We would like to research all available tools for stating a company's health and focus on creating a tool that would be accessible via the FutureSME portal and every SME will be able to get its financial health status by filling all requested data in a form and the system will process and generate results. Input data should be in the form of a profit & loss report and/or balance sheet. We would like to apply more than one financial model. The overall result will be presenting a well arranged table with simply expressed results colored or highlighted according to the current company's health status.

II. SMES ANALYSIS AND REQUIREMENTS

At the first phase of the FutureSME project, a long analysis was carried out by all FutureSME R&D partners [2, 5]. The academic research focused on emerging technologies and ICT (Information and Communication Technologies) standards and methods suitable to be used for SMEs [2, 5, 17], the industrial partners of the project were surveyed during seminars and panels organized by the project Consortium. The results, also published in several case studies [7], helped us to focus on financial models [1, 14, 16] and to choose those that are frequently used by banks and other financial institutions. Also this helped to determine a customer's solvency [8, 15], then to select the most suitable form and content to deliver our implementation in order to have it accepted by SMEs in their daily practice, and finally, to create an application using tools for web publishing.

III. FINANCIAL MODELS

All development and programming has a very powerful theoretical background in the financial area. We had been searching for usable models related to a company's profitability and bankruptcy, which would correspond to SMEs specifications. Based on the requirements collected during the analytical phase of the project [2, 5], such as keeping the entry data simple, using the same company's data available for shareholders and/or annual reports, being easy to operate by not technically skilled SME staff and similar areas, we selected three models – two models for profitability – Kralicek's quick test [14] and the DuPont analysis [10] and one bankruptcy model – Altman's Z-Score model [1]. Our research confirmed that Kralicek's quick test is robust method utilizing whole potential of balance sheet and profit & loss report. DuPont analysis was chosen as it is a great tool to show the typical employee, who has little or no financial and accounting background, on how their work and efforts impacts the financial results of their company. Altman's Z-Score model is a common simple and accurate calculation used by investors and plays a relatively easy addition to an investment checklist.

There are many evaluation tools using these models Kralicek's Online Quick Test [13], DuPont Financial Analysis - Easy Calculator [4] and Altman Z-Score Spreadsheet [11], but they are separated standalone utilities requesting almost same input data. We need just one comprehensive application on the portal sharing common input data for all analyses. User interface must be created for various types of SMEs, so the final application is generic and each company works with paragraphs and fields that are relevant for them.

A. *Kralicek's quick test*

This test was created by Professor Kralicek in 1991. This test uses an annual report (profit & loss report and balance sheet) and calculates ratio indicators and each of them gets a grade from various perspectives – capital strength, indebtedness, profitability and financial position; see Table 1. This test has four partial results and one overall result [14].

Equity Ratio = Total Owner's Equity / Total Assets

Debt Settlement Period from Cash Flow = (Liabilities - Cash) / Cash Flow)

Cash Flow = P/L Acc. Period + Assets Depreciation + Reserves and Deferred Income

Operating Cash Flow / Sales = Operating Cash Flow / Net Sales (Revenue) - This ratio, which is expressed as a percentage, compares a company's operating cash flow to its net sales or revenues, which gives investors an idea of the company's ability to turn sales into cash.

ROA (Return on Assets) = Net Income / Total Assets - An indicator of how profitable a company is relative to its total assets. ROA gives an idea as to how efficient management is at using its assets to generate earnings. Calculated by dividing a company's annual earnings by its total assets, ROA is displayed as a percentage. Sometimes this is referred to as "return on investment" [9].
If we get a grade 1 or 2, our company is profitable [14]; otherwise a company is threatened by bankruptcy.

B. *DuPont Analysis*

DuPont analysis (also known as the DuPont identity, DuPont equation, DuPont model or the DuPont method) is an expression which breaks ROE (Return On Equity) into three parts. The name comes from the DuPont Corporation that started using this formula in the 1920s [10].

ROE = (Profit margin) * (Asset turnover) * (Equity multiplier)
= (Net Profit/Sales) * (Sales/Assets) * (Assets/Equity) = (Net Profit/Equity)
The DuPont identity breaks down Return on Equity (that is the returns that investors receive from the firm) into three distinct elements. This analysis enables the analyst to understand the source of superior (or inferior) return by comparison with companies in similar industries (or between industries). The DuPont identity, however, is less useful for some industries, such as investment banking, that do not use certain concepts or for which the concepts are less meaningful. Variations may be used in certain industries, as long as they also respect the underlying structure of the DuPont identity. The DuPont analysis relies upon the accounting identity, that is, a statement (formula) that is by definition true.

TABLE 1 – KRALICEK'S QUICK TEST GRADING

	Grading Scale				
Indicators	1 excellent	2 very well	3 well	4 poor	5 dangerous
Equity / Total Assets	> 30 %	> 20 %	> 10 %	> 0 %	negative
Debt Settlement Period from Cash Flow	< 3 years	< 5 years	< 12 years	< 30 years	> 30 years
Financial Stability	arithmetic mean of total assets and Debt Settlement Period from Cash Flow				
Operating Cash Flow / Sales	> 10 %	> 8 %	> 5 %	> 0 %	negative
ROA	> 15 %	> 12 %	> 8 %	> 0 %	negative
Profit Situation	arithmetic mean of Operating Cash Flow and ROA				
Total Grading	arithmetic mean of all four indicators				

C. *Altmans Z-Score model*

The Z-Score formula for predicting bankruptcy was published in 1968 by Edward I. Altman, who was at the time an Assistant Professor of Finance at New York University. The formula may be used to predict the probability that a firm will go into bankruptcy within two years. Z-scores are used to predict corporate defaults and an easy-to-calculate control measure for the financial distress status of companies in academic studies. The Z-Score uses multiple corporate income and balance sheet values to measure the financial health of a company. The Z-Score is a linear combination of four or five common business ratios, weighted by coefficients. The coefficients were estimated by identifying a set of firms which had declared bankruptcy and then collecting a matched sample of firms which had survived, matching them by industry and approximate size (assets) [1].

Z-Score definitions:
X1 = Working Capital / Total Assets
X2 = Retained Earnings / Total Assets
X3 = Earnings before Interest and Taxes / Total Assets
X4 = Market Value of Equity / Total Liabilities
X5 = Sales/ Total Assets

Z-Score bankruptcy model:
$Z = 1.2X1 + 1.4X2 + 3.3X3 + 0.6X4 + .999X5$

Zones of Discrimination:
$Z > 2.99$ - Safe Zone
$1.81 < Z < 2.99$ - Grey Zone
$Z < 1.81$ - Bankruptcy Zone

IV. SOFTWARE IMPLEMENTATION

Our final software tool is called “Finance Module” and is implemented into the FutureSME portal; see Fig. 1.

futureSME helps grow and sustain your business performance. Develop the capability to adapt to changing business conditions and make the most of opportunities in the marketplace.

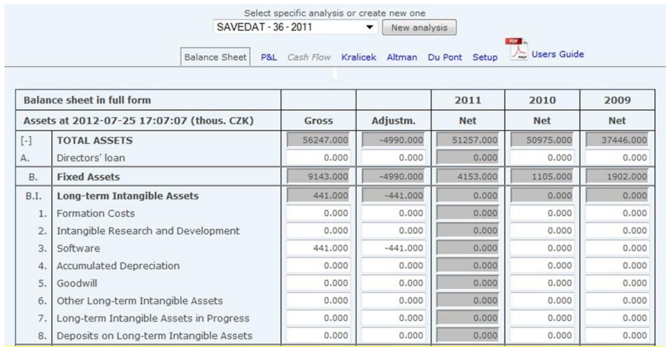
Figure 1. FutureSME portal

This portal, as the deliverable of the FutureSME project, supports SMEs for their better competitiveness on the market and, once they log in, offers them free tools to be used for improvement of their business and managing the company transformation dependent on the changing business environment they must face.

Our Finance Module offers them to get their financial grades and check their ability to get a loan from a bank without any further consultancy help, just by filling out their Profit & Loss report and Balance Sheet data; see Fig. 2.

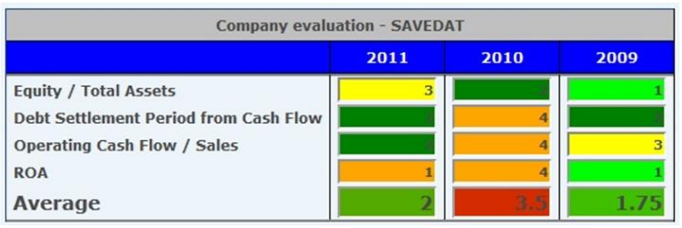
All entered data are processed on the server and then the final reports with results from all three financial models applied using common source data are produced. Just a moment after all data are entered results are generated and presented in the form of simple tables; see Fig. 3, 4 and 5. The final software application is in a form of a webpage. It is programmed in PHP and uses java script for client-side scripting and the Microsoft SQL server for data storing. All input data are related to a company and an analysis. A company is defined by its name, description and type (privately or publicly held). Once a company is stored the user is allowed to create a new analysis. Every analysis is related to a business year. Then, data from the profit & loss and balance sheet are required. There is a cash flow form as well but its data are not mandatory for final results generation.

The balance sheet always requests data from the analysis year and two previous years. It is divided into four areas – the director's loan, fixed assets, current assets and accruals



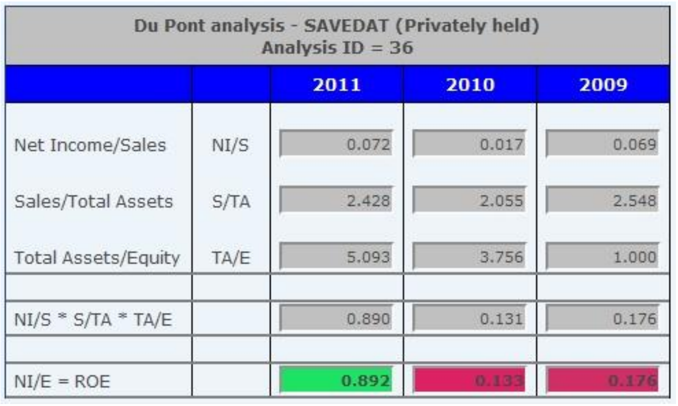
Balance sheet in full form				2011	2010	2009
Assets at 2012-07-25 17:07:07 (thous. CZK)		Gross	Adjustm.	Net	Net	Net
[-]	**TOTAL ASSETS**	56247.000	-4990.000	51257.000	50975.000	37446.000
A.	Directors' loan	0.000	0.000	0.000	0.000	0.000
B.	**Fixed Assets**	9143.000	-4990.000	4153.000	1105.000	1902.000
B.I.	**Long-term Intangible Assets**	441.000	-441.000	0.000	0.000	0.000
1.	Formation Costs	0.000	0.000	0.000	0.000	0.000
2.	Intangible Research and Development	0.000	0.000	0.000	0.000	0.000
3.	Software	441.000	-441.000	0.000	0.000	0.000
4.	Accumulated Depreciation	0.000	0.000	0.000	0.000	0.000
5.	Goodwill	0.000	0.000	0.000	0.000	0.000
6.	Other Long-term Intangible Assets	0.000	0.000	0.000	0.000	0.000
7.	Long-term Intangible Assets in Progress	0.000	0.000	0.000	0.000	0.000
8.	Deposits on Long-term Intangible Assets	0.000	0.000	0.000	0.000	0.000

Figure 2. Finance module interface with balance sheet tab



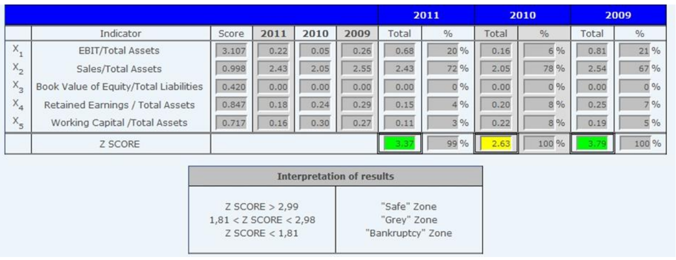
Company evaluation - SAVEDAT	2011	2010	2009
Equity / Total Assets	3	[illegible]	1
Debt Settlement Period from Cash Flow	[illegible]	4	[illegible]
Operating Cash Flow / Sales	[illegible]	4	3
ROA	1	4	1
Average	2	3.5	1.75

Figure 3. Kralicek's model results



Du Pont analysis - SAVEDAT (Privately held) Analysis ID = 36		2011	2010	2009
Net Income/Sales	NI/S	0.072	0.017	0.069
Sales/Total Assets	S/TA	2.428	2.055	2.548
Total Assets/Equity	TA/E	5.093	3.756	1.000
NI/S * S/TA * TA/E		0.890	0.131	0.176
NI/E = ROE		0.892	0.133	0.176

Figure 4. DuPont Analysis



						2011		2010		2009	
	Indicator	Score	2011	2010	2009	Total	%	Total	%	Total	%
X_1	EBIT/Total Assets	3.107	0.22	0.05	0.26	0.68	20 %	0.16	6 %	0.81	21 %
X_2	Sales/Total Assets	0.998	2.43	2.05	2.55	2.43	72 %	2.05	78 %	2.54	67 %
X_3	Book Value of Equity/Total Liabilities	0.420	0.00	0.00	0.00	0.00	0 %	0.00	0 %	0.00	0 %
X_4	Retained Earnings / Total Assets	0.847	0.18	0.24	0.29	0.15	4 %	0.20	8 %	0.25	7 %
X_5	Working Capital /Total Assets	0.717	0.16	0.30	0.27	0.11	3 %	0.22	8 %	0.19	5 %
	Z SCORE					3.37	99 %	2.63	100 %	3.79	100 %

Interpretation of results	
Z SCORE > 2,99	"Safe" Zone
1,81 < Z SCORE < 2,98	"Grey" Zone
Z SCORE < 1,81	"Bankruptcy" Zone

Figure 5. Altman Z-Score test

and deferred expenditure. The areas have their own areas and subareas. The user is supposed to fill out either subareas or total numbers. While data are being updated, the results are changing in real time, and there is no need of starting a process. The user only needs to choose a particular model and to click that tab to get the results.

Model results are presented by a table with highlighted numbers according to the particular status. Mostly green, amber and red colors are used. The data used in this paper are provided by one of our partnering companies for testing. The company's name is not published.

V. CONCLUSION

The goal of this research has been reached. A new Finance Module was developed and implemented into the FutureSME portal. The module was tested by partnering companies and external SMEs as well. The final version of the application was accommodated according to their comments. Companies reviewed this tool as very useful and helpful when a loan is needed or simply when they want to determine their company's health or development from previous years during current situation into the future. The Finance Module is offered to all registered users of the FutureSME portal and its use is free of charge. A user manual is included and is accessible via the program menu.

VI. ACKNOWLEDGMENTS

The authors would like to acknowledge the FutureSME project team who contributed greatly to the data collection and analysis of requirement specifications for this work, as well as the European Commission for funding and supporting CP-IP 214657-2 FutureSME, (Future Industrial Model for SMEs), EU project of the 7FP in the NMP area.

VII. REFERENCES

[1] E. I. Altman, Predicting Financial Distress of Companies, pp. 15–22. Available from URL <http://pages.stern.nyu.edu/~ealtman/Zscores.pdf> [retrieved: August, 2012].

[2] M. Assarlind and I. Gremyr, Quality Management in Small and Medium Sized Enterprises, Irish Academy of Management 12th Annual Conference, Conference Proceedings, Galway, 2009, pp. 135-138.

[3] R. A. Brealey and S. C. Myers. 2000. Principles of Corporate Finance. Irwin: McGraw-Hill, 10th edition, 2010, ISBN-13: 978-0077356385.

[4] K. Bernhardt, Dupont Financial Analysis - Easy Calculator, Available from URL <http://cdp.wisc.edu/wk1/DuPont%20EasyCalc.xls> [retrieved: August, 2012].

[5] U. Bititci and A. Ates, The appropriateness of current intervention policy patterns and delivery mechanisms to address the manufacturing SME needs in Europe. In Configuring manufacturing value chains - Responding to an uncertain world - 14th Cambridge Symposium on International Manufacturing. 11 pp. Contribution. University of Strathclyde. Available from URL <www.ifm.eng.cam.ac.uk/cim/symposium2009/.../20_umit_bititci.pdf> [retrieved: August, 2012].

[6] E. F. Brigham and M. C. Ehrhardt, Financial Management: Theory and Practice, USA: Thomson South-Western, 2007, ISBN-13: 978-1439078099.

[7] FutureSME Consortium, Case Studies, Available from URL <http://www.futuresme.eu/case-studies/futuresme> [retrieved: August, 2012].

[8] A. A. Groppelli and E. Nikbakht (2000). Finance, 4th ed. Barron's Educational Series, Inc. pp. 444–445. ISBN 0-7641-1275-9.

[9] Investopedia, Return of Assets, Available from URL <http://www.investopedia.com/terms/r/returnonassets.asp#axzz21YKTWO00> [retrieved: August, 2012].

[10] InvestorWords.com, Du Pont Analysis Definition, Available from URL <http://www.investorwords.com/6496/Du_Pont_Analysis.html>, [retrieved: August, 2012].

[11] J. Jun, Free Altman Z - score Spreadsheet, Available from URL <http://www.oldschoolvalue.com/blog/investment-tools/free-altman-score-spreadsheet/> [retrieved: August, 2012].

[12] E. Kislingerová and J. Hnilica, Finanční analýza krok za krokem. 1. vyd., Praha: C. H. Beck, 2005, 137 s. ISBN 80-7179 -321-3.

[13] P. Kralicek, Online Quicktest, Available from URL <http://www.kralicek.at/index.php?gr=-30> [retrieved: August, 2012].

[14] P. Kralicek, Základy finančního hospodaření: Bilance. Účet zisků a ztrát. Cashflow. Finanční plánování. Systémy včasného varování, Prague: Linde, 1993, ISBN 80-85647-11-7.

[15] B. E. Needles and M. Powers, Financial Accounting, 2007, Boston: Houghton Mifflin Company, ISBN-13: 978-0547193281.

[16] The Manage Mentor, Finance – DuPont Analysis, Available from URL <http://www.themanagementor.com/EnlightenmentorAreas/finance/CFA/DUPontAnalysis.html> [retrieved: August, 2012].

[17] M. Valas, O. Winkler, P. Osadník, and L. Landryová, Graphic Data Display from Manufacturing on Web Pages. In *Transactions of the VŠB- Technical University of Ostrava, Mechanical Series*, No. 2/2009, volume LV, article No. 1705, VŠB-TU Ostrava 2009, pp. 149-154, ISBN 978-80-248-2144-3. ISSN 1210-0471 (Print). ISSN 1804-0993 (Online). ISSN-L 1210-0471.

Automated Test Code Generation Based on Formalized Natural Language Business Rules

Christian Bacherler, Ben Moszkowski
Software Technology Research Lab
DeMontfort University
Leicester, UK
christian.bacherler@email.dmu.ac.uk, benm@dmu.ac.uk

Christian Facchi, Andreas Huebner
Institute of Applied Research
Ingolstadt University of Applied Sciences
Ingolstadt, Germany
{christian.facchi|andreas.huebner}@haw-ingolstadt.de

***Abstract*—The paper addresses two fundamental problems in requirements engineering. First, the conflict between understandability for non-programmers and a semantically well-founded representation of business rules. Second, the verification of productive code against business rules in requirements documents. As a solution, a language to specify business rules that are close to natural language and at the same time formal enough to be processed by computers is introduced. For more domain specific expressiveness, the language framework permits customizing basic language statements, so called atomic formulas. Each atomic formula has a precise semantics by means of predicate and Interval Temporal Logic. The customization feature is demonstrated by an example from the logistics domain. Behavioral business rule statements are specified for this domain and automatically translated to an executable representation of Interval Temporal Logic. Subsequently, the example is utilized to illustrate the verification of requirements by automated test generation based on our formalized natural language business rules. Thus, our framework contributes to an integrated software development process by providing the mechanisms for a human and machine readable specification of business rules and for a direct reuse of such formalized business rules for test-cases.**

***Keywords*-Requirements engineering; business rules; natural language; testing; logic.**

I. Introduction

In software development, different stakeholders with different knowledge and intention cooperate, typically domain experts and developers. Requirements engineers are acting as negotiators between these two worlds and prepare requirements specifications in a way that can be understood by both sides. Nonetheless, unstructured natural language in requirements documents does not ensure identical interpretations by different readers, which has always been a fundamental problem in software engineering [1]. Moreover, machine-readability of a requirements document can be a big asset but requires a formal syntax that is not provided by unstructured natural language [2].

By the introduction of *AtomsPro Rule Integration Language* (APRIL) [3], we propose a means to develop a formalized version of business rules specifications by precise semantics that support human- as well as machine-readability. The APRIL statements representing business rules are easy to design and can be customized by the construction of tailored statements, a feature, which we introduce via a novel combination of pattern building mechanisms. In this paper, we show how to extend APRIL's expressiveness using atomic formulas that constitute the link between statements that are like natural language and formal frameworks.

Formal specifications enhance the established software development process (V-Model). As a general advantage, such specifications allow consistency checking of business rules (e.g., reveal conflicts or proof properties). The aspect we want to focus on in this work is based on the fact that in the established software development process, code and corresponding tests are developed based on the natural language specification. In order to reduce complexity of the development process, we support automated creation of tests based on formal APRIL statements representing business rules. With our method, human understandable formal specifications can be used to directly generate formal logical conditions and behavior specifications for testing. This approach shifts the creation of the test code from the developer to the requirements engineer, which helps to improve test-driven development projects [4] [5].

The paper is structured as follows: The next Section II will give an impression of the context and the facets of the work presented. Section III presents the framework for our language to describe business rules close to natural language. After laying down the fundamentals, we demonstrate in Sec. IV the transformation of example statements in our language into computer processable test code. After the discussion of related work (Section V), a conclusion will be drawn and future work will be presented (Section VI)

II. Overview

The APRIL framework can be embedded into standard software development processes. As an example, the seamless integration into the V-Model is shown in Figure 1. Aspects that will be detailed in this paper are highlighted in dark grey.

Our Framework aims at supporting the generation of computer executable test code from formal specifications that are close to natural language and thus enable the verification of the productive code against the original user specification. In Section III, a detailed explanation of the substantial concepts of the APRIL language is given, exemplifying the formalization of business rules as APRIL statements in Section III-A.

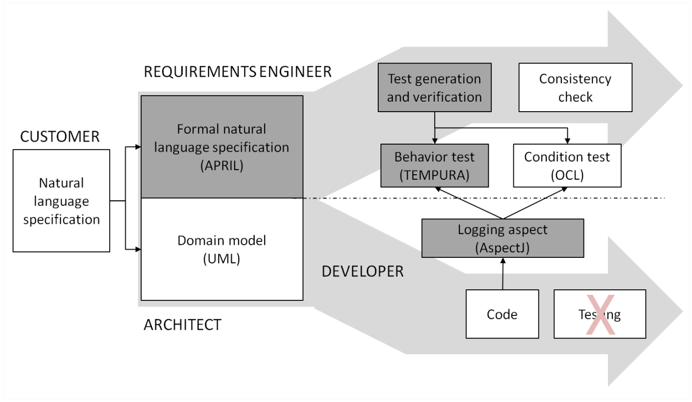


Fig. 1. Overview of the software development process using APRIL.

The treatment of complex real-world business rules using mixfix notation and decomposition into reusable sub-statements (APRIL-Definitions) is presented in Section III-B. Section III-C deals with support for customizing parts of the language using so-called atomic formulas. These are verbalized versions of operations on sets, predicate-logic formulas and special common constraints. Atomic formulas provide a precise semantics for APRIL Definitions.

Tests based on APRIL statements can be generated to check conditions using invariants, pre- and post-conditions in the Object Constraint Language (OCL) [6] notation. Checking process behavior is done by the use of a subset of Interval Temporal Logic (ITL) called Tempura. The rationale for applying our testing-framework is laid down in Section IV-A. Section IV-B presents the testing-framework by example, taking into account the significant concepts for defining a custom atomic formula for modeling a simple example-process and the relation to the semantic frameworks presented in Section III-D. This section will also include a presentation of the automated test generation for behavior testing using Tempura. Due to space limitations, the detailed presentation of generating OCL-statements is omitted. Some translation examples are shown alongside the introduction of the APRIL language.

After the discussion of related work (Section V), a conclusion will be drawn and future work will be sketched (Section VI).

III. The APRIL framework - Specifying business rules in formal natural language

Business rules are restrictions of certain object constellations and behaviors based on domain models [2]. Typically in software development, requirements engineers produce business rules in natural language and hand them to developers along with the respective domain-models to enable the development of a software-system compliant to these input artifacts. Mostly, those natural language business rules are informal and suffer from ambiguity and imprecision. APRIL supports the specification of business rules that are formal enough to be processed by computers, but still close enough to natural language to ensure readability and comprehensibility for humans.

A. Business Rules in APRIL

In general, the different types of business rules in the industrial practice are: Integrity Rules, Derivation Rules and Rules to describe behavior [7]. Despite the fact that there are fundamental intentional differences, these rule types have one aspect in common: The projection of the semantics of parts of the real world into formal representations by means of logic. In APRIL we use UML-class models [8] to formally represent business domain models. The reason is that the UML-class model is widely used for representing conceptual schemas and is easily understood by people. APRIL requires UML-class models as the domain of discourse to specify business rules as constraints, which are of the following types: **invariant, pre-, post-condition and behavioral rules**. Invariants describe allowed system states that must not be violated during any point in time. This is unlike the pre- and post-conditions, which have a restricted scope right before and after a transition. The fourth rule type describes behavior explicitly. Behavioral rules can describe operations lasting over multiple state transitions [2], which is not possible with a single pair of pre- and post-condition.

In Figure 2, a simple domain model of a car-rental system, with the basic concepts Car, Rental and Customer, is shown as UML-class model. As an example of APRIL usage on the class-model, the corresponding statement for the invariant underageCustomers can be seen in Listing I.

1	***Invariant*** *underageCustomers* ***concerns*** *Rental:*
2	*All underage customers who rent a Porsche must pay*
3	*plus 150 percent.*

Listing I
Top-level rule, composed of several APRIL Definitions.

The header (line 1) of a rule contains its name *(underageCustomers)* and the token after the keyword **concerns**, which represents the context set (represented by the class name *Rental*) of the business rule to which the formula after the colon applies. With respect to UML-models, the context in invariant rules is represented by a class name and by a qualified method name in the case of pre- and post-conditions respectively. The rule body (lines 2-3) contains the actual business rule. In order to use a natural language sentence in the needed formal way, a couple of definitions have to be installed, which are explained in Section III-B continuing this example.

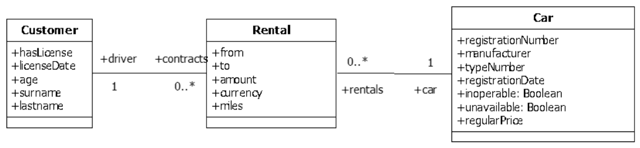


Fig. 2. UML-model of the car rental example.

Moreover, a detailed specification of APRIL including default logic- and set- operators, is given in [3].

B. APRIL-Definitions

APRIL Definitions are special mix-fix operators, which allow the intuitive construction of patterns that decompose large business rules into smaller, comprehensible and reusable sub-statements. Mix-fix is a particularly useful technique to form natural language statements [9]. Mix-fix operators allow to compose an operator's constants and placeholders in arbitrary order. The design of the APRIL-Definition's headers is based on sequences of static name parts and placeholders. Both static name parts and placeholders can be arbitrarily composed to express a business statement reflected as a natural language sentence pattern. This makes them particularly easy to construct for humans [1].

Despite the convenience that mix-fix operators provide to humans, it is quite challenging to implement the parser logic [10], especially for nested definition calls. The problem is that the parser has to recognize a *definition call* embedded inside an ID-token sequence in what is in the grammar specification another *definition call* (see highlighted EBNF-grammar rules in Listing II). As a consequence, a context free grammar provides only insufficient means to specify sub ID-token streams with a different semantics to their embedding ID-token streams. To overcome this, we use the ANTLR v3 [11] parser-/compiler-generator framework. The framework allows to specify semantic annotations [12], which are actually user defined code snippets (e.g., in Java) that get inserted into the proper positions of the grammar to guide parser decisions based on the semantics of tokens. Consider Listing II, where the Boolean return-values of the semantic annotations indicated by α_0 and α_1 influence the generated parsers resolution algorithm. The semantic annotations indicated by the symbols α_n represent java code that gets integrated into the parser. The implemented logic performs the link between syntax and semantics. E.g., when a token with the value *Rental* gets recognized, the semantic annotation allows to conclude on further decision steps for the parser. Or also trigger some type-checking mechanism. However, for parsing mix-fix operators, we limit the nesting depth to three, which was shown to be sufficient in our preliminary case study.

```
definition::= 'Definition' nameSignature 'yielding'
                typeDef 'is defined as' ruleBody '.'
nameSignature::= (ID | parameterDef)+
parameterDef::= '(' name=ID 'as' type=ID ')';
typeDef::= ID | ID '(' typeDef ')';
ruleBody::= statement+ ;
statement::= ... | referenceOrDefinitionCall | ...;
referenceOrDefinitionCall::= {α0}modelReference
                             |{α1} definitionCall | ...;
definitionCall::= ID (ID | referenceOrDefinitionCall)* ;
```

Listing II
GRAMMAR SNIPPET FOR APRIL DEFINITIONS

Given the car rental example from Section III-A, the APRIL-Definitions (D.1)-(D.3) decompose the business rule statement from Listing I into reusable and easy to define sub-statements with a signature in mix-fix notation.

(D.1) ***Definition*** *All (customers* ***as*** *Collection(Customer)) must pay plus (ratio* ***as*** *Number) per cent* ***yielding*** *Boolean* ***is defined as every*** *customer* ***satisfies that*** *contracts.amount = contracts.car.regularPrise * (1 + ratio).*

(D.2) ***Definition*** *underage customers who rent (type* ***as*** *Integer)* ***yielding*** *Collection(Customer)* ***is defined as each*** *customer* ***in all instances of*** *Customer* ***where*** *customer.age < 21* ***and*** *customer.contracts.car.typeNumber=type.*

(D.3) ***Definition*** *a Porsche* ***yielding*** *Integer* ***is defined as*** *911.*

In (D.1), the exact offset ratio is mapped to a set of customers. On the other hand, (D.2) is a set-comprehension on the set of all customers defining, what an underage customer is that rents a certain car type. Furthermore, (D.3) maps an identifier-constant (911) of type integer to a name representing the intended car type.

In order to provide a precise semantics to the Definitions, APRIL **atomic formulas** are used. They are verbalized versions of operations on sets, predicate-logic formulas and special common constraints sketched by Halpin [9]. For example, the *every-satisfies-that*-statement of Definition (D.1) is an atomic formula in APRIL that constitutes a universal quantification that is by default incorporated into the language. Some more operators are described in [3]. Default atomic formulas are for maintaining sufficient expressive power and straight-forward translation into executable representations. Therefore, APRIL uses OCL as target language for translating invariants and pre-and post conditions. Behavioral rules are translated into Tempura, which is briefly explained later.

In order to extend APRIL's expressiveness over general purpose operators provided by OCL, we allow the customization of atomic formulas that can be tailored to a certain domain. Moreover, this approach delegates the design of the atomic formulas as natural language statements to the human user, who is still the best choice for this creative task.

C. Extending APRIL with Custom Atomic Formulas

Like Definitions, customizable atomic formulas are defined using textual business patterns *(bp)*. Here, a requirements engineer can, e.g., reuse his already existing, informal textual business patterns [1], which, unlike the more abstract Definitions, express a very basic business rule- or business process pattern that regulates the business concepts and facts under consideration. For example if a requirements engineer wants to verbalize business process statements which specify that in a warehouse all elements in a goods-stock move to a dedicated truck-loading bay and have to pass a certain gate on their way, she would have to specify parts of the grammar. Therefore, a state of practice language implementation mechanism described by Parr [12] is used. First, a formal production rule of the new atomic formula must be specified. Formal production rules are parts of a context-free grammar [13] and are used to generate text recognition algorithms of a parser that processes statements of a language to generate a parse tree. Second, a parse tree rewrite rule has to be specified along with the production rule. Parse tree rewrite rules are instructions for the parser on how to construct the *abstract syntax tree* (AST) from the parse tree.

The AST is a condensed version of the parse tree that can be influenced by semantic considerations to form a concise and expressive logical representation of the parsed statements. For APRIL the AST provides the necessary flexibility to incorporate user defined language parts and also makes it particularly easy to extract the necessary parameters for the compiler. For clarification, Listing III sketches the language extension mechanisms that APRIL provides. It formalizes the example operator that reflects the scenario mentioned above. In line 1 the production rule with the name of the atomic formula moveTo is introduced. The definition of the new atomic formula's regular syntax is defined in the lines 2-7. Here, the non-terminal *referenceOrDefinitionCall* is similar to that in Listing II. It can either refer to an element of the related domain model (e.g., to class names Store, Bay, Gate) or to values in the scope of the parent rule or definition, in which the formula is used. The references to the parse tree nodes of type *referenceOrDefinitionCall* in the lines 3, 5 and 7 are stored one by one in the local variables source, target and routeNode. Line 9 concludes the specification of the grammar rule with the parse tree rewrite rule. It is delimited from the syntax rule by the "→" sign. It tells the parser to construct a tree with the MOVETO-terminal as root node having three leaves: source, target and routeNode.

The grammar rule and the parse tree rewrite rule in Listing III get injected into dedicated areas of the APRIL core grammar. Parameterization of the APRIL-compiler is straight forward, which is depicted in Figure 3. In the second pass a so called tree parser interprets the AST (of the rewrite rule MOVETO) and decides, which target language template to apply to the AST of the atomic formula. It then passes the values of the leaf-nodes (here the values of the variables $source, $target and $routeNode) to the parameters of the respective template. The instantiated template is the actual

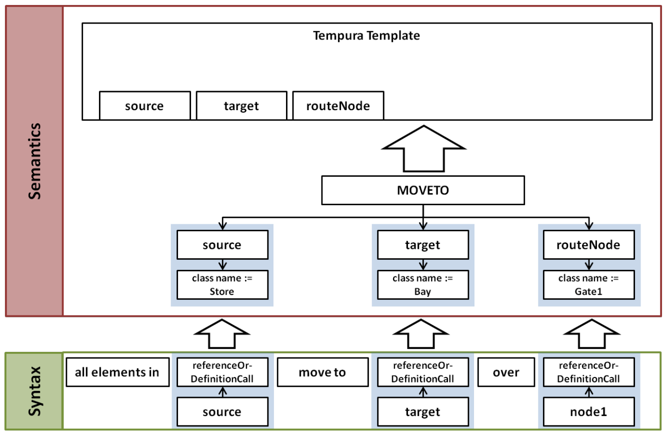


Fig. 3. Translation example of the atomic operator **moveTo**.

```
1  moveTo :
2  'all elements in'
3  source=referenceOrDefinitionCall
4  'move to'
5  target=referenceOrDefinitionCall
6  'over'
7  routeNode=referenceOrDefinitionCall
8
9  → ^(MOVETO $source $target $routeNode);
```

Listing III
GRAMMAR RULE AND PARSE TREE REWRITE RULE FOR THE OPERATOR **MOVETO** IN ANTLR 3.0.

translation of the atomic formula.

D. APRIL's Target Languages

APRIL makes use of the logical frameworks OCL and Tempura to underpin its language constituents with a well defined semantics. Both languages are briefly introduced in the subsequent sections.

1) OCL: OCL 2.3.1 is the target language for APRIL-invariants, pre- and post- conditions. For the sake of brevity, we give a rudimentary introduction to OCL because it is well known. The interested reader should consult the literature on OCL. The specification of OCL 2.3.1 can be found on [6].

OCL restricts UML-class models using predicate logic and operations on sets. Arithmetic-, Boolean- and relational operators are used in the conventional way. The well known existential and universal quantifiers allow to quantify on propositions holding on an object population derived from a class model. In order to give an idea of the OCL syntax, we provide in Listing IV a translation into OCL of the car-rental example mentioned earlier in Listing I and the definitions from (D.1)-(D.3). Here, we used OCL's decomposition mechanisms to cater to an improved readability.

```
context Rental inv underageCustomers:
Customer::
All_customers_must_pay_plus_ratio_per_cent(
  Customer::underage_customers_who_rent_type(
    Car::a_Porsche),150)

context Customer def:
All_customers_must_pay_plus_ratio_per_cent(
  customers:Collection(Customer), ratio:integer) :
    Boolean = customers→forAll(customer|
      customer.contracts.amount =
      customer.contracts.car.regularPrise *
      (1+ratio))

context Customer def:
underage_customers_who_rent_type(
  type:integer) : Collection(Customer) =
  allInstances()→select(customer: Customer |
    customer.age < 21 and customer
    contracts.car.typenumber = type)

context Car def: attr a_Prosche : integer = 911
```

Listing IV
OCL-TRANSLATION OF THE INTRODUCTORY CAR-RENTAL-EXAMPLE

2) Tempura: Tempura is an executable subset of Interval Temporal Logic (ITL) [14]. Like some other temporal logics, ITL enhances predicate calculus with with a notation of discrete time and associated operators. A key feature of ITL and Tempura is that the states of a predicate are grouped together as nonempty sequences of states called intervals σ_{plus}. They are called intervals. For example the shortest interval (of states) σ on a predicate P can be represented by $< s >$ with length σ := $|\sigma|$ = 0, which is generally the number of states in σ minus 1. The semantics of ITL keeps the interpretations of function and predicate symbols independent of intervals. Thus, well known operators like {+, -, *, and, or, not,...} are interpreted in the usual way. The characteristic operator for ITL is the operator *chop (;)*. Conventional temporal logic operators such as *next* ($\bigcirc$) and *always* ($\Box$) examine an interval's suffix subintervals whereas chop splits the interval into two parts and tests both. Furthermore, Moszkowski [14] shows how to derive operators such as always and sometimes from chop. In ITL, the formula $w := w_1; w_2$ is true if $\Im_{\langle\sigma_0..\sigma_i\rangle}$ $[\![w_1]\!]$ and $\Im_{\langle\sigma_i..\sigma_{|\sigma|}\rangle}$ $[\![w_2]\!]$ are true in the respective sub-formulas. Note that w_1 and w_2 share the same subinterval σ_i. We adopt some examples from [14], which are as follows:

σ	P	R
s	1	2
t	2	1
u	3	1

The lenght of interval σ is expressed by $|\sigma|$ and is defined as the number of the states in σ minus one. Thus, in our example, $|\sigma| = 2$.

The following formulas on the predicates P and R are true on the interval $< stu >$:

- $P = 1$. The initial value of P is 1.
- $\bigcirc(P) = 2$ and $\bigcirc(\bigcirc(P)) = 3$. The next value of P is 2 and the next next value of P is 3.
- $P = 1$ and P gets $P + 1$. The initial value of P is 1 and P gets increased by 1 in each subsequent state.
- $R = 2$ and $\bigcirc(\Box(R)) = 1$ The initial value of R is 2 and R is always 1 beginning from the next state.
- $P \leftarrow 1$; $P \leftarrow P + 1$; $P \leftarrow P + 1$. The formula $e_2 \leftarrow e_1$ is true on an interval if $\sigma_0(e_1)$ equals $\sigma_{|\sigma|}(e_2)$. Thus, $\leftarrow$ is called temporal assignment.

We adopt Tempura because it is able to model operations lasting over multiple state transitions, which would not be possible with a single pair of OCL pre- and post-conditions. Moreover, the reader will recognize similarities with the rationale of the test-definitions given in Section IV-A.

IV. GENERATING TEST CODE FROM APRIL STATEMENTS

This section clarifies the connection between APRIL and its target languages utilizing the *moveTo*-operator example introduced earlier. Section IV-A describes the basic rationale that influence the test framework presented in Section IV-B. The test framework is applied to an application, which helps to track movements of goods in a logistics centre. For testing the correct routing, we use the example operator *moveTo* described in Section III-C.

A. Testing

For generating proper test-code based on APRIL statements, the classification of different test types into black- and white-box testing has to be clarified. Our definition of the test types is as follows: Each function f_i in the set of functions F ::= $\{f_0$ to $f_n\}$ of a component under test (CUT) triggers a state transition and obeys a predefined signature. This signature requires a tuple of input values (f_{IN}) and yields a tuple of output values (f_{OUT}). A signature of a function is an interface describing a contract [15] with IN- and OUT-data, which is specified in UML-class models. We assume that a composite function g_{ik} is a conglomerate of some functions f_i to f_k, for some natural numbers $0 <= i < k <= n$. Then, any OUT-signature of a proceeding function f_j must correspond to the IN-signature of the succeeding function f_{j+1}, for some natural numbers $k < j <= i$. This convention of the inner structure can be formalized by $OUT(f_j) == IN(f_{j+1})$, which we want to abbreviate with D_j. It represents an element of a function sequence. Moreover, the following holds $IN(g_{ik}) == IN(f_i)$ and $OUT(g_{ik}) == OUT(f_k)$.

A white-box test necessitates the knowledge of the entire sequence of $D_{D_0,...,D_n}$ as the internal structure of g (g_{ik}), which is normally the case as the user knows the source code. If $D(g)$ is unknown, tests are limited to reason on the data given by $IN(g)$ and $OUT(g)$, they are called black-box tests. In APRIL black-box tests are issued to the invariants, pre- and post-conditions.

For the specification of behavioral models, we extend our recent definition of white-box tests beyond reasoning on D. We use Interval Temporal Logic (ITL) [14] for modeling behavior in white-box-tests. Therefore, we introduce behavioral constraints in APRIL, which we regard as orthogonal to the invariants as well as pre- and post-conditions. Assume D represents a state σ_1 that maps a set of values to their corresponding variables at one certain point in time. Then let σ be an ordered set of states σ_0 to σ_n, each of which describes a different D at different subsequent, discrete points in time. In our understanding, the knowledge of σ is sufficient for applying white-box-tests, which we want to utilize in our framework.

B. Test Framework and Case Study

In this section, we build a representative example around the behavioral all-elements-move-to-operator introduced in Section III-C. The Definitions of the previous section are used in our test framework, which deals with logistic processes to handle the material flow in a warehouse. It consists of a simple 3-tier architecture with RFID-readers and light sensors at the field-level and an ERP-system at the top level. Between these two levels, we use an RFID-middleware -Rifidi [16]- for information exchange and filtering.

The connection between a specification in Tempura and a function in the productive code is the test data. Therefore, the user has to provide initial test data $IN(f_0)$, constituting an important part of a test-case. The productive code affects the data $OUT(f_i)$ in the memory for each invocation of f_i, which marks a new interval at the same time. Thus, each time a function under test f_i gets invoked a snapshot of the input data (f_{IN}) prior to the invocation and output data (f_{OUT}) when f_i is left gets generated. The test data for the Tempura-statements is provided by recorded history-data that is stored in a properly formatted log-file containing a condensed version of the data-snapshots. The retrieval of the test data from the running system is achieved via AspectJ [17]. Therefore, AspectJ point-cut statements are generated based on the reference-nodes (see Listing III) to class-attributes found in the AST of an APRIL statement. The use of AspectJ permits us to leave the original code of the productive system untouched.

The use case for the earlier mentioned example with the behavioral operator *moveTo* formalized in Listing III is as follows: Imagine a warehouse that has a high-bay storage and a loading bay for lorries. Both, storage and lorry-bay are connected with a conveyor belt. Each of the three components is equipped with one RFID-reader that can detect tagged-goods in its near field to allow tracking whether the correct thing takes the right path in the right direction. For a customer order, all goods in store contained in the order must go from the store to the lorry-bay via the conveyor belt. For simplicity we assume that each good will be detected by exactly one of the three RFID-readers at a time. This simplification is an abstraction of the real world, which does not influence considerations regarding the presented methodology.

The described scenario can be reflected by a log file as depicted in Table I, if the actual memories of the readers holding the IDs of the tags can be accessed in the productive application via the following reference-IDs: STORE for the RFID-reader observing the near-field of the storage, GATE1 for the conveyor and BAY for the lorry-bay. The data in the log file is formatted as array with the symbolic name OUTPUT.

	σ_I	STORE	GATE1	BAY
OUTPUT	I=1	”a”,”b”		
	I=2	”b”	”a”	
	I=3	”b”		”a”
	I=4		”b”	”a”
	I=5			”a”,”b”

TABLE I
REPRESENTATION OF LOG-FILE RECORDED FOR EXAMPLE-OPERATOR

With regard to the model, the Tempura statements in Listing V hold. They are actually an instantiation of a template that is used by the APRIL-compiler for translating the move-to-operator if used in an APRIL statement like in Listing VI. The formatting of the statements is according to String-Template described by Parr [18] and contains generic parts that get filled according to the parameters of the operator in Listing VI.

```
define store_moves_to_Bay_over_Gate1 () = {
  len(|OUTPUT|-1) and
  I = 0 and
  I gets I+1 and
  moveAtoB(OUTPUT[I][Store], OUTPUT[I][Gate1]) and
  moveAtoB(OUTPUT[I][Gate1], OUTPUT[I][Bay]) and
  OUTPUT[|OUTPUT|-1] [Bay] ← OUTPUT[0] [Store]
}.

define moveAtoB (A,B) = {
  if (|A| > 0) then {
    first(A) gets last(B) and skip
  }
}.
```

Listing V
TEMPLATE FOR THE ALL-ELEMENTS-MOVE-TO OPERATOR.

```
all elements in Store move to Bay over Gate1.
```

Listing VI
USAGE OF THE ALL-ELEMENTS-MOVE-TO OPERATOR.

V. RELATED WORK

SBVR-Structured-English (SE) and similarly RuleSpeak [19] are so-called controlled languages to express business rules in a restricted version of natural language. Both are based on SBVR, which defines semantic parts, e.g., terms and facts to determine business concepts and their relations. The syntactic representation of these parts is achieved by text formatting and coloring, which could be used to aid parsing SE-statements. From our viewpoint, mixing technical information with the textual representation is problematic because formalized and natural language semantics have to be maintained in one and the same statement. However, natural language does not utilize text formatting information for transporting semantics.

Nevertheless, SE is used for model representation, which Kleiner et al. [20] utilize as a starting point for translating schema descriptions (in SE) into UML-class models, which is helpful for software development. Unfortunately, they leave the treatment of business rules for further work. Regarding the customizability aspect of business statements, the approach of Sosunovas et al. [21] presents another way, utilizing regular patterns. They pursue a three-step approach to constructing business rule templates that are first defined on an abstract level and then tailored to fit a specific domain with every further refinement step. Therewith, they provide precise meta-model-based semantics to the template elements but -as they admit- not to the business rule resulting from using the template.

Another interesting approach in generating tests from requirements specifications is introduced by Nebut et al. [22]. They utilize UML use-case models combined with contracts represented by pre- and post- conditions to specify sequences of state transitions. Based on these contracts, they simulate the modeled behavior by intentionally ”instantiating” the use case model. This approach could be a worthy extension to ours, which uses historical data that could also be generated by simulation. Moreover, Nebut et al. show how to generate test-cases from sequence diagrams and test objectives, that cater to a defined test coverage.

VI. CONCLUSION AND FUTURE WORK

With APRIL we want to provide a customizable and semantically well-founded notation that is close to natural language and suitable for humans as well as for computers. A core design principle of APRIL is the ability to define abstract mix-fix operators that are particularly useful to define natural language expressions as reusable patterns. We consider this pattern building technique as sufficiently intuitive even for untrained persons. The semantic underpinning of the mix-fix operators is achieved by customizable atomic formulas. The syntax of atomic formulas can be tailor-made for any domain. This is exemplified by a new atomic formula taken from the logistics domain to model behavior. We extend APRIL's grammar and present a mapping to the interpretation function based on Interval Temporal Logic. With the use of the new atomic formula and the transformation into the instantiated Tempura statement, executable test code is generated. This way our framework contributes to an integrated software development process by providing unambiguous and understandable business rules that can be reused for automatically generating tests.

From the current viewpoint, some issues are still open. Further evaluation is needed to determine wether the specification of the grammar rules and their corresponding rewrite rules are suitable to a typical requirements engineer. Also, the use of OCL and especially Tempura, for creating the templates requires a considerable amount of skills. Moreover, using APRIL requires a basic understanding of logic and set-theory. It has to be discovered if the aforementioned challenges are manageable by the typical requirements engineer. Hence, future work will target on refining the presented approach with a focus on methodologies to improve APRIL's usability.

ACKNOWLEDGEMENTS

The authors are greatful for many hours of inspiring discussion and feedback received from Hans-Michael Windisch.

REFERENCES

[1] C. Rupp, *Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis*, 5th ed. München and Wien: Hanser, 2009.

[2] A. van Lamsweerde, *Requirements engineering: from system goals to UML models to software specifications*. Chichester: Wiley, 2009.

[3] C. Bacherler, C. Facchi, and H.-M. Windisch. (2010) Enhancing Domain Modeling with Easy to Understand Business Rules. HAW-Ingolstadt. [retrieved: 09,2012]. [Online]. Available: http://www.haw-ingolstadt.de/fileadmin/daten/allgemein/dokumente/Working_Paper/ABWP_19.pdf

[4] K. Beck, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[5] P. Liggesmeyer, *Software-Qualität*. Spektrum, Akad. Verl, 2002.

[6] Object Management Group. (2010) OCL Specification: version 2.3.1. [retrieved: 09,2012]. [Online]. Available: http://www.omg.org/spec/OCL/2.3.1/PDF/

[7] J. Cabot, R. Pau, and R. Raventós, "From UML/OCL to SBVR specifications: A challenging transformation," *Information Systems*, vol. 35, no. 4, pp. 417–440, 2010.

[8] Object Management Group. (2010) UML Specification: version 2.2. [retrieved: 09,2012]. [Online]. Available: www.omg.com/uml

[9] T. A. Halpin, "Verbalizing Business Rules: Part 14," *Business Rules Journal*, vol. 7, no. 4, 2006.

[10] N. Danielsson and U. Norell, "Parsing mixfix operators," *Proceedings of the 20th International Symposium on the Implementation and Application of Functional Languages (IFL 2008)*, 2009.

[11] T. Parr. (2012) ANTLR v3. [retrieved: 09,2012]. [Online]. Available: http://www.antlr.org/

[12] ——, *The Definitive ANTLR Reference*. Pragmatic Bookshelf, 2007.

[13] A. Aho, M. Lam, R. Sethi, and J. Ullman, *Compilers: principles, techniques, and tools*. Pearson/Addison Wesley, 2007.

[14] B. Moszkowski, *Executing Temporal Logic Programs*. Cambridge, 1986.

[15] B. Meyer, "Applying Design by Contract," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[16] Rifidi Community. (2012) Rifidi-Platform. [retrieved: 09,2012]. [Online]. Available: http://www.transcends.co/community

[17] Eclipse Open Plattform Community. (2012) AspectJ: Version 1.7.0. [retrieved: 09,2012]. [Online]. Available: http://www.eclipse.org/aspectj/

[18] T. Parr. (2012) String Template: Version 4.0. [retrieved: 09,2012]. [Online]. Available: http://www.stringtemplate.org/

[19] Object Management Group. (2008) SBVR Specification: version 1.0. [retrieved: 09,2012]. [Online]. Available: http://www.omg.org/spec/SBVR/1.0/

[20] M. Kleiner, P. Albert, and J. Bézivin, "Parsing SBVR-Based Controlled Languages," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, A. Schürr and B. Selic, Eds. Springer Berlin / Heidelberg, 2009, vol. 5795, pp. 122–136.

[21] S. Sosunovas and O. Vasilecas, "Precise notation for business rules templates," *Databases and Information Systems, 2006 7th International Baltic Conference on*, pp. 55–60, 2006.

[22] C. Nebut, F. Fleurey, Y. Le Traon, and J. Jézéquel, "Automatic test generation: A use case driven approach," *Software Engineering, IEEE Transactions on*, vol. 32, no. 3, pp. 140–155, 2006.

Using SSUCD to Develop Consistent Use Case Models: An Industrial Case Study

Mohamed El-Attar
Information and Computer Science Department
King Fahd University of Petroleum and Minerals
P.O. 5066, Al Dhahran 31261, Kingdom of Saudi Arabia
melattar@kfupm.edu.sa

***Abstract*- In software development projects that utilize a use case-driven development methodology, it is crucial to develop high quality use case models to ensure the development of a quality end product. There are many quality attributes for use case models. One of these qualities is consistency. A structure named SSUCD (Simple Structured Use Case Descriptions) was developed to guide use case authors while authoring their use cases. SSUCD was developed in previous work to specifically tackle the issue of consistency in use case models. In particular, SSUCD ensures structural consistency in use case models. Thus far, SSUCD has been validated using exemplars. While exemplars provide beneficial preliminary validation, a more thorough validation process is required to ensure the industrial applicability of SSUCD. To this end this paper presents an industrial case study that was used to validate SSUCD. The result of the case study shows that SSUCD can be effectively used to develop consistent use case models of industrial strength.**

Keywords – Use Cases; SSUCD; Model Consistency.

I. INTRODUCTION

Use case modeling [7, 13] is a very popular technique used to elicit and model functional requirements in object-oriented software development projects. In a use case driven development methodology, the use case model is used to drive the development of other UML (Unified Modeling Language) [13] design artifacts at the design phase. This process is vulnerable to human injected defects since naturally there is a gap between the analysis and design phases. Consequently, this will cause system architects to create designs that provide different functionality from that was required (i.e., developing the 'wrong' system), leading to costly reworks and schedule overruns, in addition to the intangible cost of unsatisfied customers. It is, therefore essential to develop high quality use case models in order to ensure the development of end systems that delivers the required functionalities; while allowing them to be understandable by all stakeholders, including "non-technical" stakeholders.

The literature has identified a number of use case models quality attributes that can be categorized into five main categories: consistency, correctness, completeness, analytical and understandability [11]. The harmful consequences of lacking in any of these quality attributes have been documented in the literature. Many research works have been devoted towards improving these quality attributes or at least targeting a subset of these quality attributes. Consistency in particular is a highly sought after quality attribute [1-6, 10-12]. The current state of practice to develop use case models depends on the modeler's discipline to create consistent use case models. Such discipline seldom in exists in practice. In previous work, a structure named SSUCD [11] was developed to specifically target the issue of inconsistencies in use case models. In particular, SSUCD can be used to ensure structural consistency in use case models. SSUCD does not directly improve other quality attributes. Therefore, it is recommended that SSUCD be used in addition to other researched techniques to improve the overall quality of use case models.

The remainder of this paper is organized as follows: Section 2 provides a brief background and discusses related works. Section 3 presents the SSUCD structure. In Section 4, the MAPSTEDI case study is presented. Finally, Section 5 concludes and provides suggestions for future work.

II. BACKGROUND AND RELATED WORK

A use case model consists of a use case diagram, a set of use case descriptions and a glossary. The glossary is an artifact that is shared by all artifacts developed in a project to document relative terminology in a consistent manner. The use case diagram serves as a visual summary of the functional requirements of the underlying system. The functional requirements are textually detailed in use case descriptions.

In a use case model, inconsistency can occur between the use case descriptions, the various diagrams (if more than one was used), and most commonly inconsistency may occur between the use case diagrams and the corresponding set of use case descriptions. The cost of inconsistencies depends on the form it exists in.

The literature has repeatedly warned against inconsistencies in use case models. A taxonomy of use case modeling defects and their harmful consequences were presented in Anda et al. [3]. The taxonomy states that inconsistencies in a use case model have a detrimental effect on every aspect of the development process and in turn severely hampering the overall quality of the end product. In Lilly [10], a number of inconsistency defects were outlined. For example, an inconsistent system boundary has been found to cause ambiguity with respect to the functionality that needs to be developed. Development teams may suffer from costly redundant and unnecessary development leading to schedule overruns. Conversely, development teams may miss some of the required functionality. Inconsistencies in use case models has also been found to be symptomatic of an ambiguous domain model and a use case model that might be handling concepts that are not defined or understood properly [5]. Inconsistencies may also be a result of missing or vague information [5]. Ambler [2] warns that a high level of inconsistencies in use case models may render it useless as it becomes too outdated.

Naturally many research works have been devoted towards improving consistency in use case models. For example, Armour and Miller [6] and Kulak and Guiney [8] have highly recommended various mechanisms of reviewing use case models as means to ensure their quality by assuring that they possess a great deal of consistency. An automated approach was proposed by McCoy [12]. McCoy [12] presents a tool that provides a template for use case authors to write their use cases. The template aids in ensuring consistency during the data entry process. Butler et al. [4] introduced the concept of refactoring to the use case modeling domain. A number of use case refactorings improve consistency.

III. THE SSUCD STRUCTURE

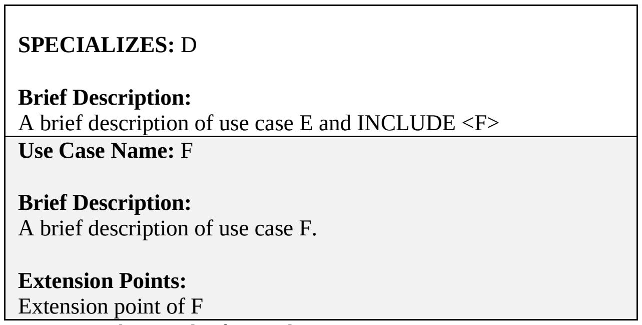
The structure SSUCD was devised to specifically tackle the issue of inconsistencies. SSUCD employs a template of commonly used fields in popular use case description templates such as those presented by Cockburn [1]. Use cases described using the SSUCD structure contains four main sections, these are: (a) Use Case Name, (b) Associated Actors, (c) Description, (d) Extension Points and Extended Use Cases. With the exception of the "Description" section, these sections utilize a handful of keywords to embed the required structure. All keywords are written in uppercase for readability purposes. The "Description" section on the other hand is populated using natural language to allow for maximum flexibility and expressiveness by use case authors. Other sections can be added to cater to specific needs; the additional sections must be contained as subsections of the "Description" section.

The design of the SSUCD structure accounted for readability. This is achieved by using a limited set of English keywords that are inserted within various sections of the templates. All keywords pertain to the use case modeling domain and thus greatly reducing the required learning curve. A brief description of each keyword is shown in Table 1. Figures 1 and 2 illustrates the concepts explained above and demonstrates the visually the mapping of the keywords in Table 1 using a mock example.

Table 1 A summary of the SMCD structure constructs

Section	Keyword	Diagram Representation
Use Case Name	ABSTRACT	*Abstract* use cases are depicted in italic font in the diagrams.
	SPECIALIZES	A generalization relationship link is depicted in the diagram.
	IMPLEMENTS	A generalization relationship link is depicted in the diagram. This is due to the fact that the generalization and implementation relationships are depicted using the same notation.
	The name of the use case	A use case with the given name is displayed in the diagram.
Description	INCLUDE	Results in the creation of an *include* relationship directed towards the use case stated in the INCLUDE statement.
Extended Use Cases	Base Use Case	An *extend* relationship link is created and directed towards the stated base use case.
	Extension Point	Optional to the user. Results in the augmentation of the targeted extension point name on the *extend* relationship link.
	IF	Optional to the user. The condition is displayed on the *extend* relationship link in square brackets.
Extension Points	The names of public extension points	Each extension point stated is depicted within the oval of the given use case in the diagram.

Mock Example Textual Descriptions
Actor Name: A **Brief Description:** A brief description of actor A
Actor Name: B **SPECIALIZES:** A **Brief Description:** A brief description of actor B
Use Case Name: C **ABSTRACT** **Brief Description:** A brief description of use case C
Use Case Name: D **Brief Description:** A brief description of use case D **Extended Use Cases:** **Base UC Name:** F **AT**: extension point of F **IF**: is true
Use Case Name: E **IMPLEMENTS:** C



SPECIALIZES: D

Brief Description:
A brief description of use case E and INCLUDE <F>

Use Case Name: F

Brief Description:
A brief description of use case F.

Extension Points:
Extension point of F

Figure 1. Mock Example of Textual Descriptions

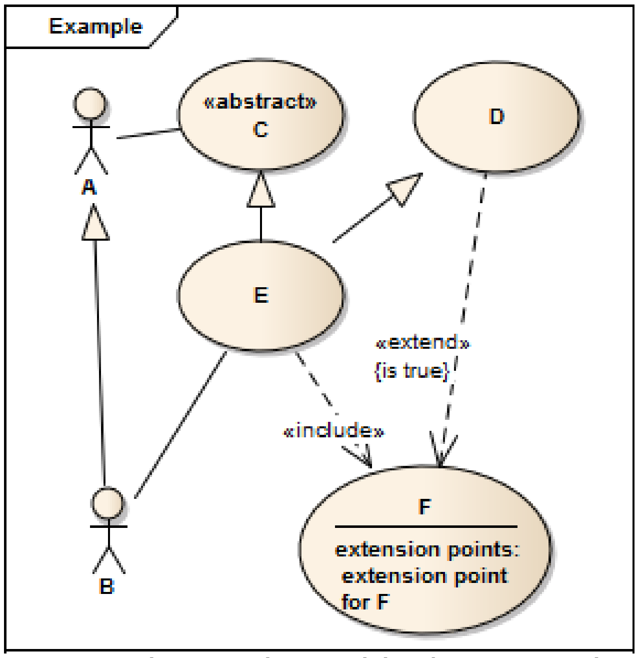


Figure 2. Example use case diagram including the entire notational set supported by the SSUCD structure

The SSUCD structure is supplemented with the REUCD (Reverse Engineering of Use Case Descriptions) process. There are two perform key functionalities that are performed by REUCD [11]: (a) REUCD constructs a use case diagram that accurately represents the textual descriptions of use cases and actors, (b) REUCD can generate skeletons of use case descriptions. Once these descriptions are completed, REUCD can once again be used to generate a use case diagrams that accurately represents the textual descriptions.

A. Consistency and Mapping Rules Between Use Case Descriptions and Diagrams

In this section, we will introduce the REUCD (Reverse Engineering of Use Case Diagrams) process, which is used to systematically map SSUCD's structural constructs to diagrammatic notations that form use case diagrams. This systematic process is automated using the tool SAREUCD (see Section 5), which will ensure the consistency and speed of the process.

The process of generating use case diagrams from use case descriptions and vice versa is analogous to generating complete and accurate UML class diagrams from code and generating code structures from UML class diagrams. The reason UML class diagrams cannot be used to generate complete programs is because they act as a visual summary of a program's static structure. UML class diagrams are at a higher level of abstraction compared to code. On the other hand, a complete program will contain more than enough details required to generate complete and accurate UML class diagrams.

Use case descriptions (analogous to code) contain far more details than use case diagrams (analogous to class diagrams). Use case diagrams are at a higher level of abstraction than the descriptions. Therefore, given a set of use case descriptions, a complete and accurate use case diagram can be systematically produced. However, if modelers choose to create use case diagrams manually first, which is often the case; a 'skeleton' of the use case descriptions can be systematically produced. Detailed descriptions of the use case are later added manually by analysts to 'flesh out' the generated 'skeletons'. After the use case descriptions are complete, an updated version of the use case diagram can be systematically generated. Users of SSUCD and REUCD will not be burdened with performing these transformations since they will be carried out by a tool.

B. The REUCD Process

When given a set of SSUCD use case description, the REUCD process is applied by iteratively parsing through the text of the descriptions. Each iteration has several purposes and these are described below:

Iteration 1: Identify actors and create XML components to represent these actors to be displayed by a UML modeling tool.

Iteration 2: Identify use cases and create XML components to represent these use cases to be displayed by a UML modeling tool.

Iteration 3: Identify relationships between actors and use cases and create to corresponding XML components. This step will require cross-referencing with XML components previously created in the previous two iterations.

When given a use case diagram, the REUCD process is applied on the XML file the represents the given use case diagram. The process is applied by iteratively parsing through the text of the XML file. Each iteration has a purpose as defined below:

Iteration 1: Identify actors and create a text area for each actor with its name and the appropriate fields.

Iteration 2: Identify use cases and create a text area for each use case with its name and the appropriate fields.

Iteration 3: Identify the relationships between actors and use cases and amend the corresponding text area to reflect these relationships.

Finally, the text areas are combined into one file.

IV. THE MAPSTEDI SYSTEM CASE STUDY

In this section, we present an industrial case study where SSUCD was applied successfully. This case study is concerned with the MAPSTEDI (Mountains and Plains Spatio-Temporal Database Informatics) use case model [9]. The MAPSTEDI system was built for research purposes by geocoders to help them analyze biodiversity data in the northern plains as well as the southern and central Rocky Mountains both spatially and temporally. It was developed by the Denver Botanic Gardens (DBG), Denver Museum of Nature and Science (DMNS) and University of Colorado Museum (UCM). The project's aim is to merge their separate collections into one distributed biodiversity database to include over 285,000 biological specimens.

The use case model of the MAPSTEDI system originally five use case models representing five subsystems. The use case diagrams of three subsystems were later merged as a result of a refactoring process. A brief description of each subsystem is provided below:

- **Database Queries:** The purpose of this subsystem is to perform queries on local and distributed databases for collections data. There are two distributed databases.
- **Database Integrator:** The purpose of this subsystem is to handle how the collections data from separate databases are integrated after being updated.
- **Database Edits:** The purpose of this subsystem handles the operational mechanisms for editing and updating the databases. The databases are updated whenever a geocoder edits the collections data.
- **Administrative Process:** The purpose of this subsystem outlines the administrative functionalities and responsibilities. This subsystem backups and restores collections data and application code. Moreover, the subsystem is used to install any new updates.
- **Database Access:** The purpose of this subsystem handles access control of the database; who may access the database and how. Public users have access to search and download collections data and visualize biodiversity analysis. However, only researchers have access to sensitive data.

The "Database Access" and "Administrative Process" subsystems each had a separate use case diagram. Meanwhile, the "Database Edits", "Database Queries" and "Database Integrator" subsystems are represented by a single merged use case diagram.

The purpose of this case study is to validate the SSUCD structure and the REUCD process. In this case study, the use case and actors descriptions were developed using the SSUCD structure. The textual descriptions were then used as input by the REUCD process to produce the corresponding use case diagrams. The successful application of this case study is if use case diagrams generated by the REUCD process were structurally similar. Figures 1, 3 and 5 below contain the textual descriptions of the use cases and actors in each use case diagram. The use case diagrams generated by REUCD based on the descriptions in Figure 1, 3 and 5, are shown in Figure 2, 4, and 6, respectively.

Database Access
Actor Name: User **Brief Description:** <A brief description about the User actor>
Actor Name: Public User **Specializes:** User **Brief Description:** <A brief description about the Public User actor>
Actor Name: Research User **Specializes:** User **Brief Description:** <A brief description about the Research User actor>
Use Case Name: Download Collections Data **Associated Actors:** User **Basic Flow:** … INCLUDE <Search Collections Data>
Use Case Name: Search Collections Data **Associated Actors:** User **Basic Flow:** …this use case allows the user to search collections data…
Use Case Name: Visualize Biodiversity Analysis **Associated Actors:** User **Basic Flow:** …this use case allows the user to visualize biodiversity analysis…
Use Case Name: Access Sensitive Data **Associated Actors:** Research User **Basic Flow:** …this use case allows the research user to access sensitive data…

Figure 3. The descriptions of the use cases and actors of the "Database Access" subsystem

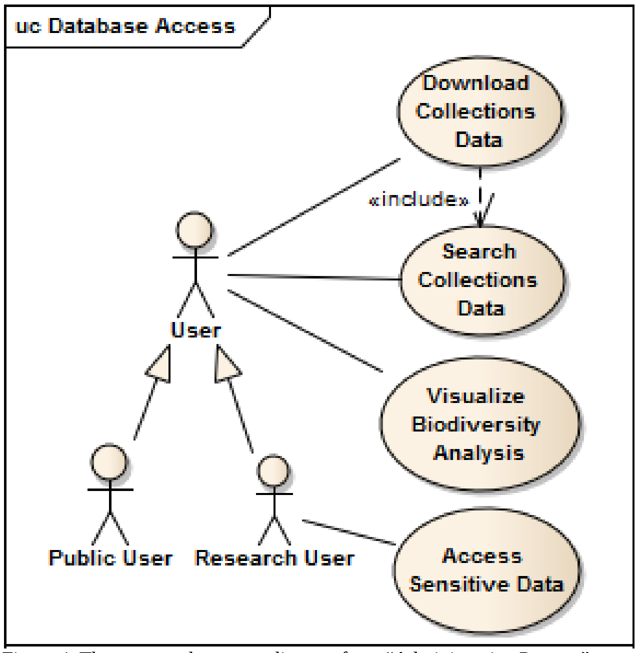


Figure 4. The generated use case diagram from "Administrative Process"

Administrative Process

Actor Name:
Administrator

Brief Description:
<A brief description about the Administrator actor>

Actor Name:
Database Administrator

Specializes:
Administrator

Brief Description:
<A brief description about the Database Administrator actor>

Actor Name:
ArcIMS Administrator

Specializes:
Administrator

Brief Description:
<A brief description about the ArcIMS Administrator actor>

Use Case Name:
Backup Process

Associated Actors:
Administrator

Basic Flow:
…this use case name allows the administrator to perform process backup…

Use Case Name:
Restore Process

Associated Actors:
Administrator

Basic Flow:
…this use case name allows the administrator to perform process restoration…

Use Case Name:
Install Software Updates

Associated Actors:
Administrator

Basic Flow:
…this use case name allows the administrator to install software updates…

Figure 5. The descriptions of the use cases and actors of the "Administrative Process" subsystem

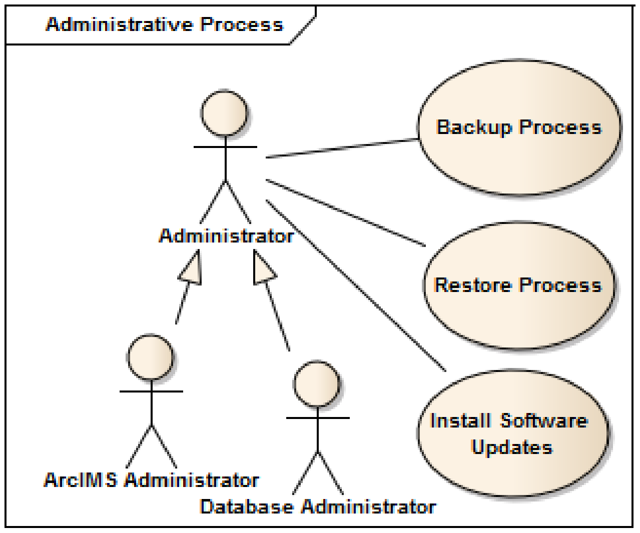


Figure 6. The generated use case diagram based on reverse engineering the textual descriptions of use cases and actors in the "Administrative Process" subsystem.

Merged Subsystems

Actor Name:
Geocoder

Brief Description:
<A brief description about the Geocoder actor>

Actor Name:
Database Integrator

Brief Description:
<A brief description about the Database Integrator actor>

Use Case Name:
Geocode Specimen

Associated Actors:
Geocoder

Basic Flow:
…this use case name allows the administrator to geocode a specimen and it INCLUDE <Update Collections Data>…

Use Case Name:
Update Collections Data

Associated Actors:
Database Integrator

Basic Flow:
…this use case name allows the administrator to update collections data…

Extended Use Cases:
Base Use Case Name: Query Remote Database

Use Case Name:
Query Remote Database

Specializes:
Query Database

Basic Flow:
…this use case name allows the administrator to query remote database…

Use Case Name:
Query DMNS Database

Specializes:
Query Remote Database

Basic Flow:
…this use case name allows the administrator to query DMNS database…

Use Case Name:
Query DIGIR Database

Specializes:
Query Remote Database

Basic Flow:
…this use case name allows the administrator to query DIGIR database…

Use Case Name:
Query Database

Basic Flow:
…this use case name allows the administrator to query database…

Use Case Name:
Query Local Database

Specializes:
Query Database

Basic Flow:
…this use case name allows the administrator to query local database…

Use Case Name:
Integrate Query Results

Associated Actors:
Database Integrator

Basic Flow:
…this use case name allows the administrator to integrate query results and it INCLUDE <Query Remote Database> and INCLUDE <Query Local Database>…

Figure 7. The descriptions of the use cases and actors of the merged subsystems

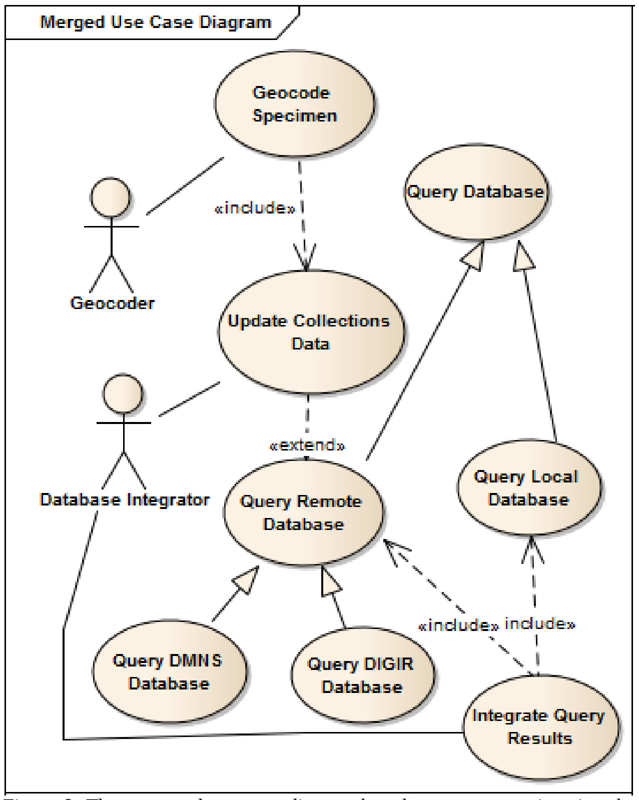


Figure 8. The generated use case diagram based on reverse engineering the textual descriptions of use cases and actors in the merged subsystems.

A. *Verifying the Correctness of the Generated Use Case Diagrams*

The correctness of the generated use case diagrams was verified through two distinct means. The first approach involved the use of the *UseCaseDiff* tool [14] to check for differences between the generated use case diagrams and the original use case diagrams. UseCaseDiff is an open source use case diagram differencing tool that was developed as part of previous work [14]. Both sets of use case diagrams were provided as input into the *UseCaseDiff* tool. The tool generated a report showing no structural differences.

The second approach used to verify the correctness of the generated use case descriptions was via manual inspection. The two sets of diagrams were juxtaposed manually by three independent researchers. The reviewers did not find any structural differences between the two sets of diagrams.

V. Conclusion and Future Work

In this paper, we report on the successful use of SSUCD to develop a structurally consistent industrial use case model that represents the functionality of the five subsystems comprising the MAPSTEDI system. The case study has shown that SSUCD can be utilized by industry practitioners to develop consistent use case models and to help them detect structural inconsistencies in existing models.

Future work can be directed towards developing an approach to transform use cases written using SSUCD into other types of models, such as UML Activity and Sequence Diagrams.

Acknowledgements

The author would like to acknowledge the support provided by the Deanship of Scientific Research (DSR) at King Fahd University of Petroleum & Minerals (KFUPM) for funding this work.

References

[1] A. Cockburn, Writing Effective Use Cases. Addison-Wesley, 2000.

[2] S. Ambler, Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. Wiley, 2002.

[3] B. Anda, D. Sjøberg, and M. Jørgensen, “Quality and Understandability in Use Case Models,” 15th European Conference Object-Oriented Programming (ECOOP), edited by J. Lindskov Knudsen. Springer-Verlag, Budapest, Hungary, pp. 402-428, 2001.

[4] G. Butler and L. Xu, “Cascaded refactoring for framework evolution,” Proceedings of 2001 Symposium on Software Reusability, ACM Press, pp. 51-57, 2001.

[5] P. Chandrasekaran, “How Use Case Modeling Policies Have Affected The Success of Various Projects (or How to Improve Use Case Modeling),” Addendum To The 1997 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 6-9, 1997.

[6] F. Armour and G. Miller, Advanced Use Case Modeling. Addison-Wesley, 2000.

[7] I. Jacobson, M. Ericsson, and A. Jacobson, The Object Advantage. ACM Press, 1995.

[8] D. Kulak and E. Guiney, Use Cases: Requirements in Context. Addison-Wesley, 2000.

[9] M. El-Attar, Analysis of the MAPSTEDI system Use Case Model. Available online: http://www.steam.ualberta.ca/main/research_areas/MAPSTEDI%20Analysis.htm. [retrieved: October 2012].

[10] S. Lilly, “Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases,” Proceedings of TOOLS USA '99, IEEE Computer Society,- pp. 174-183, 1999.

[11] M. El-Attar and J. Miller, “Producing Robust Use Case Diagrams via Reverse Engineering of Use Case Descriptions,” Journal of Software and Systems Modeling, vol. 7, no. 1, pp. 67-83, 2008.

[12] J. McCoy, “Requirements Use Case Tool (RUT),” Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 104-105, 2003.

[13] OMG 2003, “UML Superstructure Specification”, Object Management Group, http://www.omg.org/docs/ptc/03-08-02.pdf, 2003. [retrieved: October 2012].

[14] M. El-Attar, “UseCaseDiff: An Algorithm for Differencing Use Case Models,” 9th International Conference on Software Engineering Research, Management and Applications, pp. 148-152, 2011.

A Systematic Mapping Study on Domain-Specific Languages

Leandro Marques do Nascimento[1,2], Daniel Leite Viana[1], Paulo A. M. Silveira Neto[1,3],
Dhiego A. O. Martins[1], Vinicius Cardoso Garcia[1], Silvio R. L. Meira[1]
[1] Informatics Center, Federal University of Pernambuco (UFPE)
Recife, Brazil
[2] Department of Informatics, Federal Rural University of Pernambuco (UFRPE)
Recife, Brazil
[3] Department of Informatics, Federal Rural University of Pernambuco (UFRPE)
Serra Talhada, Brazil
{lmn2, dlv2, pamsn, daom, vcg, srlm}@cin.ufpe.br

***Abstract*—Domain-Specific Languages (DSLs) offer substantial gains in expressiveness and ease of use compared with general purpose languages. This way, DSLs have gained significant attention in industry and academy, as can be seen by the increased number of related publications in key conferences and journals. This paper aims to provide a broad view of the DSL research field by performing a Systematic Mapping Study. Adopting a detailed search strategy, 4450 studies were initially identified, and, after filtering, 1440 primary studies were selected and categorized using a particular classification scheme. So, this work presents the most popular application domains where DSLs have been applied, identifies different tools for handling DSLs, including language workbenches, and enumerates several techniques, methods and/or processes for dealing with DSLs.**

Keywords: Domain-specific languages; systematic mapping study; programming languages; mini languages; little languages.

I. INTRODUCTION

Software systems are built upon computer languages or better called programming languages. A programming language is a notation for expressing computations (algorithms) in both machine and human readable form. Appropriate programming languages and tools may drastically reduce the cost of building new applications as well as maintaining existing ones [1]. For humans, it would be easier to write computer programs if a natural language could be used, such as English or Portuguese, for instance. However, computer languages must follow a rigid predefined structure, with a specific grammar and syntax, and to learn this structure is not so easy for many people, taking significant time for someone to be "fluent" in that kind of language.

In the context of programming languages, a Domain-Specific Language (DSL) is a language that provides constructs and notations tailored toward a particular application domain [2]. Usually, DSLs are small, more declarative than imperative, and more attractive than General-Purpose Languages (GPL) for their particular application domain due to easier program understanding, reduced semantic distance between the problem and the program, and enhanced productivity. Some well-known examples of DSLs are BNF (syntax definition), HTML (hypertext markup), SQL (database queries), and VHDL (hardware design).

DSLs trade generality for expressiveness in a limited domain, and this can bring several benefits to software engineering. However, these benefits do not come for free. The cost of DSL design, development and maintenance has to be taken into account. Without appropriate methodologies and/or tools these costs can be higher than savings. Although DSLs have been developed from the beginning of computer science (an early example is APT, a DSL for numerical control of machine tools developed back in the 1950s at MIT [3]), many unanswered questions remain regarding when and how to develop a DSL.

Therefore, this paper presents a systematic mapping study in order to better understand the DSL research field, through synthesizing evidence to suggest important implications for practice, as well as identifying research trends, open issues, and areas for improvement. A Mapping Study (henceforth abbreviated to 'MS') [4] is an evidence-based approach, applied in order to provide an overview of a research area, and to identify the quantity and type of research and results available within it. Hence, the goal of this investigation is to identify, evaluate, and synthesize state-of-the-art domain-specific programming practices in gathering evidence of what has been achieved so far in this discipline. We are also interested in cataloging which are the domains that have taken advantage of using DSLs. This way, researchers and/or practitioners may know which DSLs have been applied to a particular domain and then reuse or adapt it for any other specific needs. This systematic mapping process was conducted from November, 2011 to April, 2012.

The remainder of this paper is organized as follows: Section 2 presents the related work. In Section 3, the research methodology used in this paper is described including the research questions, the search strategy and the classification scheme. Section 4 reports the main findings. In Section 5, the threats to validity are shown, and at last, Section 6 draws some conclusions and provides recommendations for further research on this topic.

II. RELATED WORK

The literature on DSLs provides a large number of studies, regarding both general and specific issues, as will be discussed later in this paper. However, a general search for (*"mapping study"* OR *"systematic literature review"*) AND *"domain-specific languages"* in well-known search engines have shown that no publication have tried to address the issues of this research field using specifically the MS approach. Actually, many papers presented the state-of-the-art in this field using other approaches than a MS and they are next described as related work.

One of the first published papers to coin the concept of a DSL is from 1965 [5]. It presents a family of unimplemented computing languages that is intended to span differences of a given application area by a unified framework.

In the 1980s, Bentley [6] tried to summarize the concept of the so called *Little Languages*. The paper describes examples of small languages that could be developed with the technology available back there, e.g. COBOL and FORTRAN.

In a paper from 2000, Deursen et al. [7] list a selection of 75 key publications in the area. It discusses terminology, risks and benefits, examples of domain-specific languages, design methodologies, and implementation techniques.

In a more recent work from 2005, Mernik et al. [2] try to answer the question "*When and How to Develop Domain-Specific Languages?*". The paper brings a list of DSLs developed until then for different domains. The work identified five DSL development phases: decision, analysis, design, implementation, and deployment, and then relates the listed DSLs with their development phases. At last, the work enumerates domain analysis tools and language development systems, giving a full view of the open issues in the area.

One of the most recent related work that could be identified is [8], from 2011. It compares four different approaches for DSL implementation: ANTLR, Ruby, Stratego and Converge. From their comparative study, it was observed that each approach has its merits and demerits and there is no single approach that would apply to all scenarios. The work does not mention directly the use of language workbenches.

Indeed, we believe our study states current and relevant information on research topics that can complement others previously published. By current, we mean that, as the number of studies published has increased rapidly, as shown in Figure 2, it justifies the need of more up to date empirical research in this area to contribute to the community investigations. Moreover, applying a MS approach to map out the research area of DSL gives us a full overview of what is being done and what is lacking attention from academia/ industry, as well as allow future extensions and replications.

III. RESEARCH METHODOLOGY

The experimental software engineering community is working towards the definition of a standard processes for conducting literature reviews. There are mainly two different approaches to be cited: Systematic Literature Reviews (SR) and Systematic Mapping Studies (MS) [9]. While a SR is a mean of identifying, evaluating, interpreting and comparing all available research relevant to a particular question [9], a MS intends to "map out" the research undertaken rather than to answer detailed research questions [4]. A MS comprises the analysis of primary studies that investigate aspects related to predefined research questions, aiming at integrating and synthesizing evidence to support or refute particular research hypotheses.

In this study, we merged ideas from Petersen et al. [4] with some good practices defined in the guidelines proposed by Kitchenham and Charters [9], such as the protocol definition. Therefore, we could apply a process for a mapping study, including best practices for conducting systematic reviews, making the best use of both techniques.

A MS is basically performed in three phases. All phases are detailed in following sections: *1)* Definition of the protocol, which comprises the research questions and the search strategy. This phase is commonly used in systematic reviews. *2)* Conducting the study with screening of relevant papers. During this phase, a classification scheme is used. *3)* Keywording relevant topics, data extraction and systematic mapping.

A. *Research Questions*

This mapping study intends to identify relevant publications about Domain-Specific Languages, understanding how they can be created and which ones have been created so far. In addition, this study tries to enumerate the domains in which DSLs have been applied, which knowledge is necessary from the domain experts to start using the language, and so forth which are the open issues of the whole research field.

In summary the main research question of this study is: ***In which manner are Domain Specific Languages (DSLs) being created, used and maintained?***

B. *Research Sub-questions*

Moreover, in order to make the mapping study main objective more clear and repeatable, some research sub-questions are defined, as following:

Q1. *Which techniques, methods and/or processes are used while working with DSLs, i.e. creation, application, evolution and extension of DSLs?*

Q2. *Which DSLs have been created and are available for use or are described in some type of publication?*

Q3. *In which domains are these DSLs being used?*

Q4. *Which tools are used for the development and usage of DSLs and how such tools support those activities?*

C. *Search Strategy, Data Sources and Studies Selection*

According to our research questions and in order to increase the coverage of our search, we decided to use the following search string, which brings only general terms grouped by an **OR** clause:

"*domain-specific language*" **OR** "*domain-specific modeling language*" **OR** "*generative programming*"

Therefore, instead of restricting our search items with other keywords, we understand that any work that mentions one of the three items listed is going to be returned by the search engine anyway. Although the number of manuscripts returned could increase considerably, few or even no relevant studies would be left over. Indeed, experts in the

DSL research field may say there are other related terms, such as "*little/small language*" or "*Architecture Description Language*" (ADL). Despite of including those terms in our automatic search, we decided to look for those terms in the manual search and snow-balling process (which follows up the reference list of each selected manuscript), since papers that have those terms and do not have the term "*domain-specific language*" are quite rare and can be easily found during a fine-grained and non-automatic search process.

We ended up adding "*domain-specific modeling language*" and "*generative programming*" because we noticed that these terms are extremely related to the research field just by checking at the most relevant papers according to the search engines relevance ordering.

The study was conducted using automatic and manual search. We did not establish any inferior year-limit. For automatic search, six search engines and digital databases of scientific sources were used: *ACM Digital Library*, *IEEEXplore*, *SpringerLink*, *Science Direct*, *Scopus* and *Engineering Village* (also known as *El Compendex*). Besides, the manual search includes the most important international, peer-reviewed journals published by Elsevier, IEEE, ACM and Springer, and 26 different conferences.

After performing the automatic and manual search, a total of 4450 papers were identified, 93 of them from manual search. During manual search, a snow-balling process was done. Next, the studies were submitted to the inclusion and exclusion criteria, as we detail in the following section.

The studies selection involved a screening process composed of three filters, in order to select the most suitable results, since the likelihood of retrieving not adequate studies might be high. Figure 1 details each filter.

Regarding the **inclusion criteria**, the studies were submitted to the following conditions:

- Books, papers, technical reports and 'grey literature' regarding Domain Specific Languages, Domain Specific Modeling Languages and/or Generative Programming. No date filtering was applied.
- While verifying if a given article may be included in our study, we can check if it is possible to answer 'yes' for at least one of the following questions:
  - Is it a DSL or DSML?
  - Is it a technique, method or process for handling DSLs/DSMLs?
  - Is it a tool (language workbench) for handling DSLs/DSMLs?
  - Is it any type of philosophical paper that discusses concepts of DSLs, DSMLs and/or any related generative programming technique?

Considering the **exclusion criteria**, the studies were submitted to the following conditions:

- Articles not written in English.
- Literature that was only available in the form of abstracts or Powerpoint presentations. Posters, short papers (less than 2 pages) and invited conference talks with no relevant results can be excluded.
- Articles in press, journals and conferences editorials/reviews can also be excluded.
- Duplicated and/or incomplete studies.

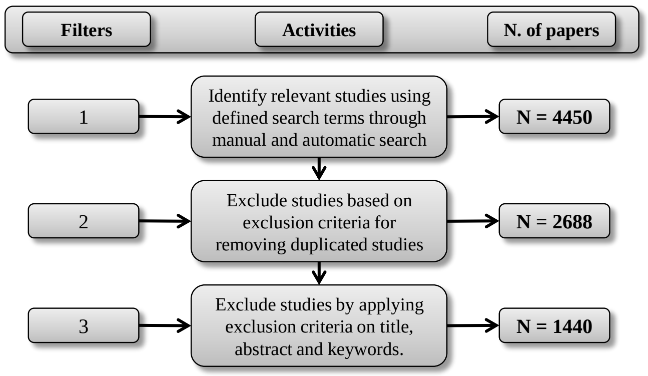

Filters
Activities
N. of papers
1
Identify relevant studies using defined search terms through manual and automatic search
N = 4450
2
Exclude studies based on exclusion criteria for removing duplicated studies
N = 2688
3
Exclude studies by applying exclusion criteria on title, abstract and keywords.
N = 1440

Figure 1. Stages of the selection process and the corresponding number of papers.

After performing the selection process, some results can be seen in Figure 2 which shows the distribution of the primary studies, considering the publication year. The Figure 2 clearly gives us the impression that many correlated areas in software engineering and computer science in general are taking more and more advantage of DSLs in practice, as we can check by looking at the growth curve.

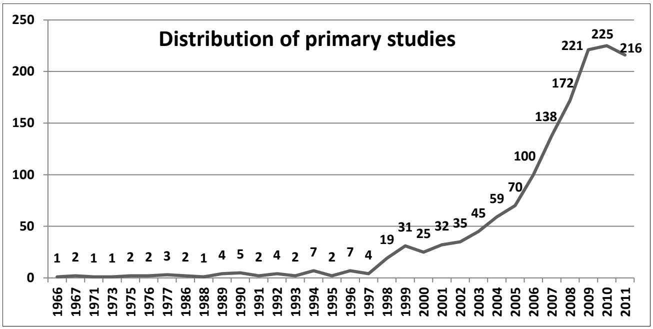


Figure 2. Distribution of studies by their publication years after 3rd filter.

We were able to identify the most common locals of publication. Conferences such ICSE (*International Conference on Software Engineering*), OOPSLA (*Object-Oriented Programming, Systems, Languages & Applications Conference*) and GPCE (*Generative Programming and Component Engineering Conference*) had the highest number of studies published. Similarly, the most popular journals were ACM SIGPLAN Notices, IEEE Software and ENTCS (Electronic Notes in Theoretical Computer). We catalogued manuscripts from 548 different sources (418 conferences and 130 journals).

D. *Classifying Selected Studies*

Our classification scheme assembled three facets. Facet one lists the classes of research based on [4]: *Validation Research, Evaluation Research, Solution Proposal, Philosophical Papers, Opinion Papers, Experience Papers*. Details of each class of research can be found on [4]. The two others are directly related to our research questions.

Facet 2 – DSL Research Type – considered in our study is directly related to the research sub-questions ***Q1***, ***Q2*** and ***Q4***. We tried to identify studies that report specifically the usage of a given DSL to solve a problem and also studies that report any kind of technique, method or process to handle DSLs, i.e., create, evolve, integrate, debug. What is more, we tried to enumerate what tools have been used to apply those techniques, methods and/or processes. TABLE I presents the details of Facet 2. Some concepts of this facet are based on [10].

TABLE I. FACET 2 – DSL RESEARCH TYPE.

1. ADL	Architecture Description Languages (ADLs) are aimed at the specification of high level system architectures, described in terms of components and connectors.
2. DSAL	A Domain-Specific Aspect Language combines benefits from DSLs and Aspect-Oriented Programming (AOP). It is a aspect language tailored to a specific domain.
3. DSML	A domain-specific modeling language is a special type of DSL that can be used for modeling domain-specific systems. The concept of a DSML comes originally from the adaptation of UML to specific domains.
4. External DSL	A completely separate language, for which you write a full parser, usually using a parser generator.
5. Internal DSL	An internal (or embedded) DSL is an idiomatic way of using a general-purpose language.
6. Method or Process	Any type of generic solution for a class of problems which usually involves technical and non-technical aspects. A method/process involves a set of steps to be performed in order to make it repeatable for anyone to try using it. A method/process may use a group of techniques which combined represent a generic solution for a class of problems.
7. Technique	Any type of solution for a specific problem. For example: a technique to generate Java code based on C# input; a technique for teaching how to create parsers, a technique to analyze model coupling.
8. Tools	Any type of software engineering tool used for handling DSLs.

Facet 3 addresses the domains in which DSL techniques are somehow applied and is directly related to ***Q3***. Inspired by previous publications that tried to do the same [2], [7], we identified many different domains ranging from bioinformatics to robotics and control systems, for example. We were able to enumerate 30 different domains. Among them, we selected the top 15 most referenced domains to be used as facet in this study. Since the final number of papers included in our study was quite large (1440), some domains were mentioned few times (1 or 2), then those ones are not considered to our classification. TABLE II displays Facet 3.

It is important to notice that none of the three facets are exclusive, it means, a paper may be classified in two or more categories of any of the three facets. For example, a paper may be categorized as a Solution Proposal and Validation Research, as a DSML and a Tool and also with the domains of Web and Control Systems.

TABLE II. FACET 3 – DOMAINS

1. Web	Every study that uses any type of web technology
2. Embedded Systems	Hardware and software co-design
3. Low-level Software	Low-level programming, for instance, operating systems, device drivers, etc.
4. Control Systems	Any type of control systems, for example: flight control, automation systems, etc.
5. Parallel Computing	High-performance computing, multithreaded programming
6. Simulation	Any type of simulation software
7. Data Intensive Apps	Studies that present ways of handling databases using DSL techniques
8. Real-time Systems	Systems where the time is a crucial variable
9. Security	Studies that handle security issues such as intrusion detection, access control, etc.
10. Dynamic Systems	A type of software system that can adapt to the context it is immersed
11. Visual Language	Apart from textual languages, this type of study describes a DSL with visual appealing
12. Testing	DSLs applied to the software engineering discipline of testing
13. Education	Any type of publication that mentions education as the primary goal, e.g. as in [11]
14. Network	DSLs for manipulating computer networks and/or distributed systems issues
15. Others	In this category, we gathered domains with at most 5 publications, covering several divergent topics, such as Chemistry, Geometry and Engineering, among others

In addition, it is important to highlight that TABLE II is missing some important domains due the total amount of manuscripts included in this study. Hereby, we cite one sample publication of these domains that were left over: healthcare [12], pervasive computing [13], graphics [14], cloud/grid computing [15], robotics [16], ontology [17], games [18], multi-agent systems [19], requirements engineering [20], bioinformatics [21], mobile apps [22], multimedia [23], user interface [24], hardware description [25], automation [26].

IV. MAIN FINDINGS

In this section, each topic presents the findings of a research sub-question, highlighting evidences gathered from the data extraction process. These results populate the classification scheme, which evolves while doing the data extraction. It is important to mention that this study is not going to enumerate all the references we found, as it makes no sense at all to list 1440 references. Instead, we are going to choose sample references to demonstrate our results.

Our first results are shown in Figure 3, which presents the distribution of papers according to Facet 1 – Classes of research. As can be seen, there is a majority number of Solution Proposals, which indicates that there are many proposals yet to be validated. The number of Validation and Evaluation Research together represents about one third of those proposals, which means that a representative number of proposals are somehow tested in industry and/or academy.

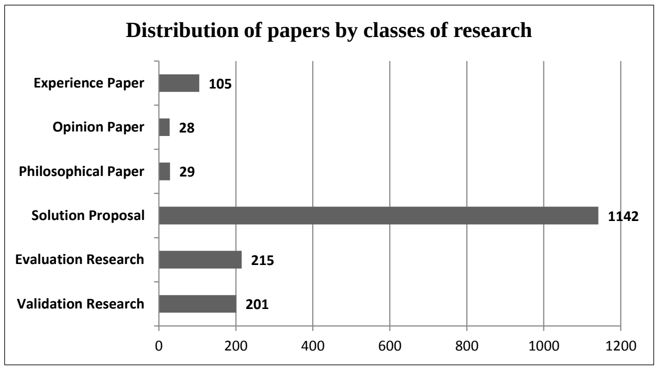


Figure 3. Distribution of papers by classes of research.

A. *Techniques, Methods and/or Processes for Handling DSLs*

Several techniques, methods and/or processes could be found during the execution of this mapping study. Methods for software construction using generative techniques are not new as we can see in [27], although they did not directly mention the construction of DSLs.

At an abstract level, a language is a means of communication; in the case of computing that communication is generally between a human and a machine. In order to be usable, a language needs to have a way that participants can share communications (syntax) and an agreed shared meaning (semantics). Languages may form parts of larger languages (e.g. the sub-part of English used only in computing could be detached and reattached to the main language); they may be parameterisable (e.g. American and British English can be seen as variations on the single, abstract, language English); they may have variable syntaxes (e.g. Serbian is written in both the Cyrillic and Latin alphabets); and so on [28].

One of the processes for handling DSL catalogued by this mapping study is called *Language Factories* [28]. Language Factories break languages down into components, including the following parts:

- **Abstract syntax**: The single definition of its Abstract Syntax Tree (AST).
- **Concrete syntax(es) and syntactic mapping**: A definition of its concrete syntax(es) specified as e.g. a context free grammar, and a mapping from that concrete syntax to the abstract syntax.
- **Semantic aspect(s)**: Each semantic aspect defines (a possibly incomplete part of the) semantics. Semantic aspects may overlap with each other (e.g. an operational and denotational semantics) or describe completely different elements of the semantics (e.g. semantics of language types and semantics for text editors supporting tool-tips).
- **Constraints**: Describes constraints on how the language can be composed with others (both in terms of what the component provides, and what it requires of other components).

These parts of language development could help us in citing the findings of this study. The development of formal DSLs contains concepts of metamodels or grammars (syntax) [29], [30], context conditions (static analysis and quality assurance) as well as possibilities to define the semantics of a language [31]. Many references highlight techniques directly related to compiler construction [11], [32], [33]. Along with the concept of DSL, we catalogued some publications describing DSMLs and its peculiarities [34]. Over the last few decades, DSLs have proven efficient for mastering the complexities of software development projects. The natural adaptation of DSLs to the model-driven technologies has in turn established domain-specific modeling languages (DSMLs) as vital tools for enhancing design productivity.

A widespread approach to the design of a DSML is to make use of the so-called profile mechanisms and to reuse the UML metamodel as the base language. By extending UML elements with stereotypes and their attributes, it is possible to define new concepts to better represent elements of a domain. Despite the ever increasing number of profiles defined and successfully applied in many applications.

The technique of UML profile is mentioned in 21 publications of our catalogue, as for example, [34–36]. We noticed that many of those techniques are well supported by tools, as we exemplify in the corresponding section.

We found quite a large number of techniques, methods and/or processes as can be seen in Figure 4. These are some examples of techniques for creating new DSLs: [37–39]. A total number of 160 publications mention some topic related to DSL creation, 69 other publications mention DSML creation and 53 mention embedded DSL creation.

Among different methods/processes for creating [40], implementing [41] and evolving [42], [43] a DSL, one of the methods that caught attention was the one that mentions directly the concept of *Language-Oriented Programming* [44] or even DSL oriented software engineering. The authors’ fundamental principle is promoting the use of the right domain specific tool for each problem, instead of some universal tool coupled with a way of working that tries to wrap it so that it becomes usable in various contexts. The primary meta-tool promoted in [44] is usage of high level, strictly domain specific languages, based on formal concepts used and widely understood by domain experts who may have limited or no software engineering knowledge. This concept of language-oriented programming is fully aligned with other similar concept called *Language Factories* [28], already mentioned.

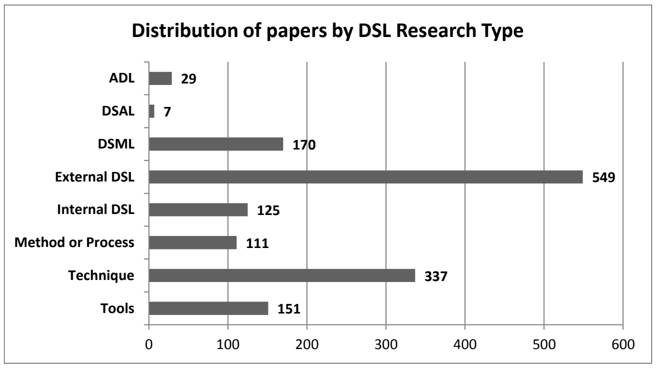


Figure 4. Distribution of papers by DSL research type.

Moreover, other relevant aspect identified in this study involves DSL integration/composition, as shown in [45–47]. Development of and tooling for a single DSL is well-studied, but surprisingly little is known about the interplay of different DSLs in a single system. Multiple DSLs are required when moving from toy examples to real enterprise applications. Methods and tool support are needed if multiple DSL development is to succeed. One of these methods is described in [48]. The method specifically tackles the problem of overlapping concerns between different DSLs. It has three steps: 1) *Identification*, 2) *Specification*, and 3) *Application*. The purpose of the *Identification* step is to uncover the overlaps between different languages and identify connections among them. The *Specification* step encodes these connections in a way that will make them amenable to various analyses. The last step of the method is *Application* where the encoded connections from the previous two steps are used. The authors also provide tools and case studies for using their method.

B. *Domain-Specific Languages and their Respective Domains*

As can be seen in Figure 5, several DSLs were catalogued according to their domain. We separated the studies that simply report the usage of a DSL in two categories: external DSL and internal (embedded DSL). For each embedded DSL, we also identified in which technology it was implemented. The most common technology in which DSLs are embedded is Haskell with 46 concurrencies, as for example in [49]. However many other host languages are used, such as Java, C/C++, Ruby, Scala, SmallTalk, Python, Prolog, XML and even some unpopular languages like Clean, Galois, Dylan and Curry.

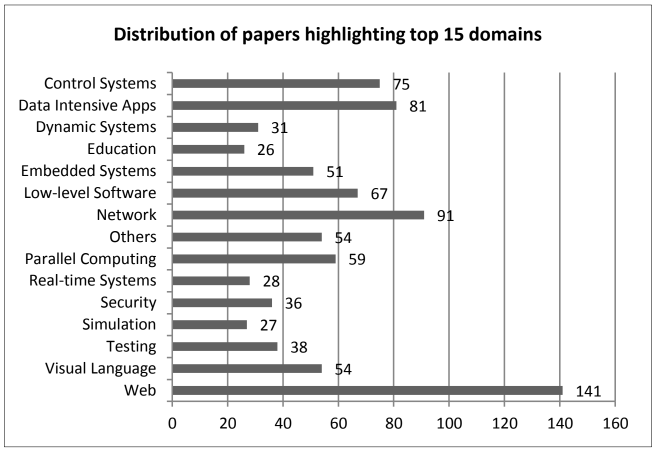


Figure 5. Distribution of papers highlighting top 15 domains.

Different types of DSLs have been identified other than the ones we previously knew. We identified FSML, ADL and DSAL.

A Framework-Specific Modeling Language (FSML) [50] is a kind of Domain-Specific Modeling Language that is used for modeling framework-based software. FSMLs enable automated round-trip engineering over non-trivial model-to-code mappings and thereby simplify the task of creating and evolving framework-based applications.

An Architecture Description Language (ADL or ADSL) [51] is a language that directly expresses a system's architecture. In this sentence, "*directly*" means that the language's abstract syntax contains constructs for all the ingredients of the conceptual architecture. Developers can thus use the language to describe a system on the architectural level.

A Domain-Specific Aspect Language (DSAL) [52] is a custom language that allows special forms of crosscutting concerns to be decomposed into modularized constructs. Examples of domain-specific aspect languages include languages for dealing with coordination concerns, object marshaling concerns, and class graph traversal concerns.

Many different domains that make use of DSL could be identified in our study. The most popular domain was the horizontal domain of web applications, in which several publications states the use of web services, and terms like *services composition*, *services orchestration* and *services mash up* are common. Figure 6 shows a full cross reference view of the DSL research type and their respective domains.

In this context, web services composition refers to the creation of new (web) services by combining functionalities provided by existing ones. A number of domain-specific languages for service composition have been proposed, with consensus being formed around a process-oriented language known as WS-BPEL (or BPEL). The kernel of BPEL consists of simple communication primitives that may be combined using control-flow constructs expressing sequence, branching, parallelism, synchronization, etc. Some examples of BPEL identified in this study: [53–55].

C. *Tools*

Tools play an essential role in software engineering and it is not different when we are talking doing language engineering. Our study identified 151 manuscripts that are related to DSL tools.

Some studies do not actually describe a new tool, but discuss about other tools as in [56] or just make use of a set of tools and report the experience as in [57]. Although, there are few studies comparing DSL tools, we were able to identify two of them as can be seen in [8], [58].

Observing the available publications, we could identify 3 subcategories of tools:

- **Tools for using DSLs**: this type of tool is actually the more comfortable for the user once he/she is supposed to be familiarized with the domain being manipulated. No knowledge about language engineering or domain engineering is necessary for using this type of tool, as well as it is projected be used by domain experts. A good example listed in our study is the tool Scratch [59], appropriated for introductory programming courses.
- **Tools for DSL creation (specification)**: these are a more intuitive way of creating compilers. At this level, the tool is nothing more than a compiler of compilers and, in the end of the process of DSL creation, there will be no integration with other software engineering tools (IDEs), pretty printer,

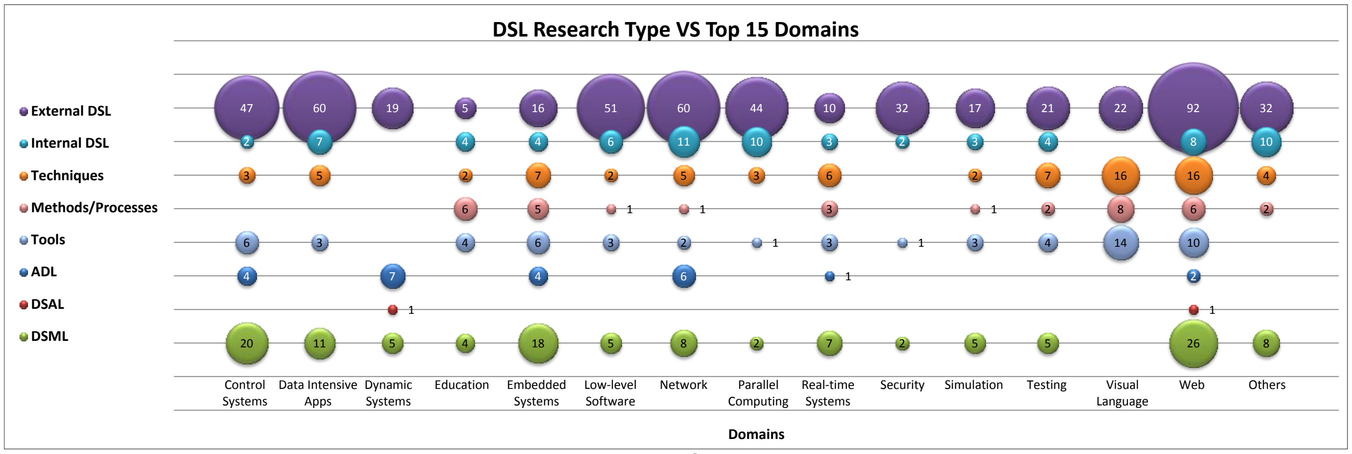


Figure 6. DSL Research Type VS Top 15 Domains.

code assistant and so on. An example of this type of tool in our study is JTS (Jakarta Tool Suite) [60].

- **Language workbench**: these tools support DSL creation not just in terms of parsing and code generation but also in providing a better editing experience for DSL users. In particular, language workbenches let a DSL author create custom DSL editors of similar power to modern IDEs. Language workbenches are still in their early days, but if their potential is realized, they could change the face of programming [10]. Our study identified some examples of language workbenches: XText [61], MetaEdit+ [62], Spoofax [63], and MPS JetBrains [64].

Another way of classifying tools is considering textual and visual languages as described in [65]. Instead, we decided to try other classification to highlight the real power of language workbenches. Unfortunately, the number of publications regarding language workbenches is still low. However, some relevant studies have been published such as [64–66].

V. Threats to Validity

There are some threats to the validity of our study. First one is regarding our set of research questions. The set we defined might not have covered the whole DSL research field, mainly because language implementation in general overlaps other several research fields, for example, model-driven approaches. As we considered this as a feasible threat, we had several discussion meetings and decided to use questions as broad as possible. This way, we knew that the number of primary studies would be bigger but there would be a smaller chance of leaving any important study out of this MS.

In addition, it is possible that we have not chosen the most appropriate keywords. In general, several research fields that use computer science as a mean to solve problems also use DSLs to provide practical solutions where the domain experts can be more effectively involved. However, these types of research and their associated publications may not directly mention DSL keywords. To mitigate this threat we added the terms "generative programming" and "domain-specific modeling language", although we noticed that rarely the term DSL is left off completely.

Other two possible threats to the validity of our study are: Search engines providing incoherent information in BibTeX and, to mitigate this threat, we developed a tool to extract BibTeX information which considers the peculiarities of each search engine, reducing the number of possible mistakes; and we may have not selected the most representative studies but, to mitigate this threat, we revised the paper selection spreadsheet in pairs until we reached a common sense.

VI. Concluding Remarks

The main motivation for this work was to investigate the state-of-the-art in engineering DSLs, through systematically mapping the literature in order to determine what issues have been studied, as well as by what means, and provide a guide to aid researchers in planning future research.

After performing this mapping study, we catalogued 1440 relevant studies from an initial set of 4450, which helped us to investigate several approaches regarding different aspects of DSL engineering. Our findings could show which are the domains where DSLs are most suitable. For instance, four domains of applications draw our attention, as following (with the respective number of publications): *Web* (141), *Network* (91), *Data Intensive Apps* (81), and *Control Systems* (85). In addition, we were able to catalogue which types of DSL are being created (*internal/external DSL*, *DSML*, *ADL*, *DSAL*), we listed several techniques, methods/processes to handle DSL, and we identified different tools to create and maintain DSLs, including language workbenches.

Moreover, in Figure 6, this study presents a bubble chart with a full cross reference view of DSL research types and their respective domains. This way, it is easy to identify which areas in this research field have been deeply explored and which are lacking attention from academy/industry with only a few publications listed.

In our future agenda, we will investigate more deeply the area of language workbenches through a SR, gathering even more evidence of the area. Moreover, we intend submit an extended version of this study to a journal because the page limit here is restraining us to present more details.

ACKNOWLEDGMENTS

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES [67]), funded by CNPq and FACEPE, grants 573964/2008-4, APQ-1037-1.03/08 and APQ-1044-1.03/10 and Brazilian Agency (CNPq processes number 475743/2007-5 and 140060/2008-1).

REFERENCES

[1] M. Fowler, *Domain-Specific Languages*, 1st ed. Addison-Wesley Professional, 2010, p. 640.

[2] M. Mernik, J. Heering, and A. Sloane, "When and how to develop domain-specific languages," *ACM Computing Surveys (CSUR)*, vol. 37, no. 4, pp. 316–344, 2005.

[3] D. Ross, "Origins of the APT language for automatically programmed tools," *ACM SIGPLAN Notices*, vol. 13, no. 8, pp. 61–99, 1978.

[4] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering*, 2008, pp. 71–80.

[5] P. J. Landin, "The Next 700 Programming Languages," *Communications of the ACM*, vol. 9, no. 3, pp. 157–166, 1965.

[6] J. Bentley, "Programming pearls: little languages," *Communications of the ACM*, vol. 29, no. 8, pp. 711–721, 1986.

[7] A. van Deursen, P. Klint, and J. Visser, "Domain-specific languages: An Annotated Bibliography," *ACM SIGPLAN Notices*, vol. 35, no. 6, pp. 26–36, Jun. 2000.

[8] N. Vasudevan and L. Tratt, "Comparative Study of DSL Tools," *Electronic Notes in Theoretical Computer Science*, vol. 264, no. 5, pp. 103–121, Jul. 2011.

[9] B. Kitchenham, "Guidelines for performing systematic literature reviews in software engineering, version 2.3," Keele University, EBSE Technical Report. EBSE-2007-01, 2007.

[10] M. Fowler, "A pedagogical framework for domain-specific languages," *Software, IEEE*, vol. 26, no. 4, pp. 13–14, 2009.

[11] T. R. Henry, "Teaching compiler construction using a domain specific language," *ACM SIGCSE Bulletin*, vol. 37, no. 1, p. 7, Feb. 2005.

[12] J. Munnelly and S. Clarke, "ALPH: a domain-specific language for crosscutting pervasive healthcare concerns," in *Proceedings of the 2nd workshop on Domain specific aspect languages*, 2007, p. 4–es.

[13] P. Barron and V. Cahill, "YABS: a domain-specific language for pervasive computing based on stigmergy," in *Proceedings of the 5th international conference on Generative programming and component engineering - GPCE '06*, 2006, pp. 285–294.

[14] F. Jacob, "CUDACL+: a framework for GPU programs," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, 2011, pp. 55–58.

[15] A. Manjunatha, A. Ranabahu, A. Sheth, and K. Thirunarayan, "Power of Clouds in Your Pocket: An Efficient Approach for Cloud Mobile Hybrid Application Development," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, 2010, pp. 496–503.

[16] M. Bordignon, U. P. Schultz, and K. Stoy, "Model-based kinematics generation for modular mechatronic toolkits," in *Proceedings of the ninth international conference on Generative programming and component engineering*, 2010, pp. 157–166.

[17] I. Ceh, M. Crepinsek, T. Kosar, and M. Mernik, "Ontology driven development of domain-specific languages," *Computer Science and Information Systems*, vol. 8, no. 2, pp. 317–342, 2011.

[18] P. Moreno-Ger, R. Fuentes-Fernández, J.-L. Sierra-Rodríguez, and B. Fernández-Manjón, "Model-checking for adventure videogames," *Information and Software Technology*, vol. 51, no. 3, pp. 564–580, 2009.

[19] M. Amor, A. Garcia, and L. Fuentes, "Agol: An aspect-oriented domain-specific language for mas," in *Proceedings of the Early Aspects at ICSE Workshops in AspectOriented Requirements Engineering and Architecture Design*, 2007, pp. 4–11.

[20] P. Sawyer, N. Bencomo, D. Hughes, P. Grace, H. J. Goldsby, and B. H. C. Cheng, "Visualizing the Analysis of Dynamically Adaptive Systems Using i* and DSLs," in *Second International Workshop on Requirements Engineering Visualization REV*, 2007, pp. 1–10.

[21] T. Antao, I. Hastings, and P. McBurney, "Ronald: A Domain-Specific Language to study the interactions between malaria infections and drug treatments," in *International Conference on Bioinformatics Computational Biology*, 2008, pp. 747–752.

[22] H. Behrens, "MDSD for the iPhone Developing a Domain-Specific Language and IDE Tooling to produce Real World Applications for Mobile Devices," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, 2010, pp. 123–128.

[23] X. Amatriain and P. Arumi, "Frameworks Generate Domain-Specific Languages: A Case Study in the Multimedia Domain," *IEEE Transactions on Software Engineering*, vol. 37, no. 4, pp. 544–558, 2011.

[24] S. Michels and R. Plasmeijer, "iTask as a new paradigm for building GUI applications," in *Proceedings of the 22nd international conference on Implementation and application of functional languages (IFL'10)*, 2010, pp. 153–168.

[25] C. Kulkarni, G. Brebner, and G. Schelle, "Mapping a domain specific language to a platform FPGA," in *Proceedings of the 41st annual conference on Design Automation DAC 04*, 2004, pp. 924–927.

[26] M. Jiménez, F. Rosique, P. Sánchez, B. Álvarez, and A. Iborra, "Habitation: A Domain-Specific language for home automation," *Software, IEEE*, vol. 26, no. 4, pp. 30–38, 2009.

[27] J. M. Neighbors, "The Draco approach to Constructing Software from Reusable Components," *IEEE Transactions on Software Engineering*, vol. 10, no. 5, pp. 567–574, 1984.

[28] T. Clark and L. Tratt, "Language factories," in *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications - OOPSLA '09*, 2009, pp. 949–955.

[29] H. Meng, "Semiautomatic acquisition of semantic structures for understanding domain-specific natural language queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 1, pp. 172–181, 2002.

[30] C. Brabrand and M. I. Schwartzbach, "The metafront system: Safe and extensible parsing and transformation," *Science of Computer Programming*, vol. 68, no. 1, pp. 2–20, Aug. 2007.

[31] J. Evermann and Y. Wand, "Toward formalizing domain modeling semantics in language syntax," *IEEE Transactions on Software Engineering*, vol. 31, no. 1, pp. 21–37, Jan. 2005.

[32] K. Kennedy et al., "Telescoping Languages: A System for Automatic Generation of Domain Languages," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 387–408, Feb. 2005.

[33] C. Consel and F. Latry, "A generative programming approach to developing DSL compilers," in *Fourth International Conference on Generative Programming and Component Engineering (GPCE)*, 2005, pp. 29–46.

[34] F. Lagarde, H. Espinoza, F. Terrier, C. André, and S. Gérard, "Leveraging Patterns on Domain Models to Improve UML Profile Definition," in *FASE'08/ETAPS'08 Proceedings of the Theory and*

practice of software, 11th international conference on Fundamental approaches to software engineering, 2008, vol. 4961, pp. 116–130.

[35] T. Ritala and S. Kuikka, "UML Automation Profile: Enhancing the Efficiency of Software Development in the Automation Industry," in *5th IEEE International Conference on Industrial Informatics*, 2007, pp. 885–890.

[36] I. Weisemöller and A. Schürr, "A Comparison of Standard Compliant Ways to Define Domain Specific Languages," in *ACM/IEEE 10th International Conference On Model Driven Engineering Languages And Systems (MoDELS 2007)*, 2008, pp. 47–58.

[37] F. Javed, M. Mernik, and A. Sprague, "Incrementally inferring context-free grammars for domain-specific languages," in *Proceedings of the Eighteenth International Conference on Software Engineering and Knowledge Engineering - SEKE'06*, 2006, pp. 363–368.

[38] H. Krahn, B. Rumpe, and S. Völkel, "Integrated definition of abstract and concrete syntax for textual languages," in *10th International Conference Model Driven Engineering Languages and Systems (MoDELS 2007)*, 2007, pp. 286–300.

[39] R. T. Lindeman, L. C. L. Kats, and E. Visser, "Declaratively defining domain-specific language debuggers," in *Proceedings of the 10th ACM international conference on Generative programming and component engineering - GPCE '11*, 2011, pp. 127–136.

[40] R. Martinho, J. Varajão, and D. Domingos, "Using the semantic web to define a language for modelling controlled flexibility in software processes," *IET Software*, vol. 4, no. 6, p. 396, 2010.

[41] B. Selic, "A systematic approach to domain-specific language design using UML," in *Proc. 10th IEEE Int'l Symp. Object and Component-Oriented Real-Time Distributed Computing*, 2007, pp. 2–9.

[42] L. Tratt, "Evolving a DSL implementation," in *ICSE '08 Proceedings of the 30th international conference on Software engineering*, 2008, pp. 425–441.

[43] S. Wenzel and U. Kelter, "Analyzing model evolution," in *Proceedings of the 13th international conference on Software engineering - ICSE '08*, 2008, pp. 831–834.

[44] A. Vajda and J. Eker, "Return to the language forrest," in *Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER '10*, 2010, pp. 389–392.

[45] H. Krahn, B. Rumpe, and S. Völkel, "MontiCore: a framework for compositional development of domain specific languages," *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 5, pp. 353–372, 2010.

[46] M. Brambilla, P. Fraternali, and M. Tisi, "A Transformation Framework to Bridge Domain Specific Languages to MDA," in *ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2008)*, 2009, pp. 167–180.

[47] E. Wyk and E. Johnson, "Composable Language Extensions for Computational Geometry: A Case Study," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, 2007, pp. 258–267.

[48] H. Lochmann and A. Hessellund, "An integrated view on modeling with multiple domain-specific languages," in *Proceedings of the IASTED International Conference Software Engineering SE 2009*, 2009, pp. 1–10.

[49] P. Thiemann, "An embedded domain-specific language for type-safe server-side web scripting," *ACM Transactions on Internet Technology*, vol. 5, no. 1, pp. 1–46, Feb. 2005.

[50] M. Antkiewicz, K. Czarnecki, and M. Stephan, "Engineering of Framework-Specific Modeling Languages," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 795–824, Nov. 2009.

[51] M. Voelter, "Architecture as Language," *IEEE Software*, vol. 27, no. 2, pp. 56 – 64, 2010.

[52] M. Shonle, K. Lieberherr, and A. Shah, "XAspects: an extensible system for domain-specific aspect languages," in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications - OOPSLA'03*, 2003, pp. 28–37.

[53] S. Brahe and B. Bordbar, "A pattern-based approach to business process modeling and implementation in web services," in *ICSOC'06 Proceedings of the 4th international conference on Service-oriented computing*, 2007, pp. 166–177.

[54] W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar, "An integrated approach for identity and access management in a SOA context," in *Proceedings of the 16th ACM symposium on Access control models and technologies - SACMAT'11*, 2011, pp. 21–30.

[55] J. Boubeta-Puig, I. Medina-Bulo, and A. García-Domínguez, "Analogies and Differences between Mutation Operators for WS-BPEL 2.0 and Other Languages," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, 2011, pp. 398–407.

[56] D. Spinellis, "The Tools We Use," *IEEE Software*, vol. 24, no. 4, pp. 20–21, Jul. 2007.

[57] T. Kosar, M. Mernik, and P. E. M. Lopez, "Experiences on DSL Tools for Visual Studio," in *2007 29th International Conference on Information Technology Interfaces*, 2007, pp. 753–758.

[58] M. Freudenthal, "Using DSLs for developing enterprise systems," in *Proceedings of the Tenth Workshop on Language Descriptions Tools and Applications*, 2010, pp. 11:1–11:7.

[59] M. Resnick et al., "Scratch: Programming for All," *Communications of the ACM*, vol. 52, no. 11, p. 60, Nov. 2009.

[60] D. Batory, B. Lofaso, and Y. Smaragdakis, "JTS: tools for implementing domain-specific languages," in *Proceedings. Fifth International Conference on Software Reuse*, 1998, pp. 143–153.

[61] M. Eysholdt and H. Behrens, "Xtext - Implement your Language Faster than the Quick and Dirty way," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion - SPLASH'10*, 2010, pp. 307–309.

[62] R. Pohjonen, "Metamodeling made easy–metaedit+ (tool demonstration)," in *Generative Programming and Component Engineering (GPCE 2005)*, 2005, pp. 442–446.

[63] L. C. L. Kats and E. Visser, "The spoofax language workbench," *ACM SIGPLAN Notices*, vol. 45, no. 10, p. 444, Oct. 2010.

[64] M. Voelter, "Embedded software development with projectional language workbenches," in *Proceedings of the 13th international conference on Model driven engineering languages and systems Part II (MoDELS 2010)*, 2010, pp. 32–46.

[65] B. Merkle, "Textual modeling tools: overview and comparison of language workbenches," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion - SPLASH '10*, 2010, pp. 139–148.

[66] M. Völter and E. Visser, "Language extension and composition with language workbenches," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion - SPLASH '10*, 2010, pp. 301–304.

[67] *National Institute of Science and Technology for Software Engineering (INES)*. Available in: www.ines.org.br. Last accessed in September, 2012.

Specifying and Designing Exception Handling with FMEA

Tsuneo Nakanishi, Kenji Hisazumi and Akira Fukuda
Faculty of Information Science and Electrical Engineering
Kyushu University
744 Motooka, Nishi, Fukuoka 819-0395, Japan
Email: {tun, nel, fukuda}@f.ait.kyushu-u.ac.jp

***Abstract*—This paper proposes a methodology to specify and design exception classes and exception handling codes used in the `try-catch-finally` exception handling control structure, which is available in C++, Java and similar programming languages. Poorly described specifications of exceptional operations cause ad-hoc, individual dependent use of the `try-catch-finally` exception control structures and fail in poorly designed exception classes and duplicated codes in the exception handling codes. Therefore, the methodology employs HAZOP (hazard and operability analysis) and FMEA (failure modes and effects analysis) to specify the exceptional operations in a consistent manner. HAZOP is used to find failure modes of the specified normal operations and then FMEA is applied to the failure modes to specify their countermeasures (namely, exception handling). Commonality and variability analysis of the specified countermeasures is performed. The result of this analysis is used to design exception classes and exception handling codes, which leads disciplined use of the exception handling control structure and elimination of duplicated codes in exception handling.**

***Keywords*-*FMEA; HAZOP; exception handling; commonality and variability analysis*

I. INTRODUCTION

The system required higher safety and reliability must provide valid behaviors, even if it cannot provide the specified normal services due to failures. The system should avoid occurrence of anticipated failures as much as possible. Furthermore, if a failure occurs unfortunately, the system should detect its occurrence and localize, compensate, or mitigate negative effects brought by the failure to minimize the damage.

Such exceptional services are realized in a valid and correct manner through a sound development process, especially in large and complicated systems. Exceptional services must be analyzed, specified, designed, implemented and then tested, as normal services are realized so. Apart from distinguishing normal and exceptional services explicitly in development, system behaviors on failures are usually specified to a greater or lesser extent during requirements and specifications phase. However, it is impossible to identify failures sufficiently that will occur at the component level, since the system is not decomposed at all at this phase. As the system is refined and decomposed in later phases, more design decisions are made and more failure modes become visible. Countermeasures to the failure modes identified in later phases must be studied and specified. Their specifications must be integrated in the system specification as exceptional services. We should keep it in mind that two thirds of system failures are due to design faults hidden in exception handling that occupies over two thirds of the system [1].

Exceptional services tend to be specified insufficiently in immature development sites. They are often decided individually by designers or programmers. That brings duplicated and/or irregular design of exceptional operations as well as a considerable amount of development rework. Furthermore, that will cause large scale modification in exceptional operations in case we enhance the existing system with additional functions.

The similar problem occurs also in implementation phase. The `try-catch-finally` statement is an exception handling control structure available in C++, Java and other similar programming languages. This exception handling control structure contributes to increase readability and reusability of exception handling codes as long as its usage is well disciplined. However, if exceptional operations are not specified and designed in a systematic manner, the exception handling control structure tends to be used in an ad-hoc, individually dependent manner. Sometimes, exceptions are ignored without taking responsible actions, although they should be processed or transferred to the caller. Absence of comprehensive view on exception handling fails in distribution of code clones doing the almost same but a little bit different things in the exception handling control structure. Moreover, poorly designed exception classes make exception handling chaotic.

This paper presents a methodology to specify exceptional operations at the class member function level and defines exception classes for the `try-catch-finally` exception handling control structure. The proposed methodology employs HAZOP (Hazard and Operability Analysis) [2] and FMEA (Failure Modes and Effects Analysis) [3]. HAZOP is a risk analysis method to identify risks to the system and its stakeholders brought by the system under consideration when a concerned property deviates from its intended extent. FMEA is a failure analysis method to study countermeasures to failures of the system under consideration. The proposed method identifies failures of the normal operation with HA-

ZOP and then, studies countermeasures to the failures with FMEA. FMEA establishes comprehensive view on exception handling and helps consistent and disciplined design of exception handling. Moreover, the proposed methodology performs commonality and variability analysis of the countermeasures, which the authors believe the novel idea in the field of exception handling. Commonality and variability analysis is a technique that has been commonly performed in software product line engineering [4] to separate common and variable structures and behaviors among products. Introduction of this idea into exception handling contributes to eliminate duplicated codes in exception handling codes for different exceptions and design well-structured exception classes.

The paper is organized as follows: Section 2 describes HAZOP and FMEA and its application to software. Section 3 gives a brief description on the `try-catch-finally` exception handling control structure of C++, Java and similar programming languages. Section 4 presents the proposed methodology with an example. Section 5 describes related works. Section 6 concludes the paper.

II. HAZOP AND FMEA

HAZOP [2] is a risk analysis method to identify possible risks to the system and its stakeholders, which was originally used in chemical process engineering. In HAZOP, risks are identified by drawing up hazardous scenarios caused when a concerned property of the system such as temperature, pressure, velocity *etc.* deviates from its intended extent. Guide words are applied to the properties to facilitate imagination of hazardous scenarios; *more*, *less*, *none*, *reverse*, and *other than* are examples of the guide words. Countermeasures to the hazardous scenarios are studied. The result of HAZOP is summarized in the tabular format.

FMEA [3] is a failure analysis method used to assess and improve reliability and safety of the system. FMEA is so generic that it has been used (maybe, more than HAZOP) in various industries such as aviation, space, automotive, nuclear *etc.* to develop and operate safety critical systems for several decades.

In FMEA, various stakeholders of the system come together; identify failure modes for each component of the system; analyze what negative effects will be brought to the component, the subsystem, and/or the system for each failure mode; and study countermeasures to compensate or mitigate the effects. Moreover, for each failure mode of the component, the stakeholders evaluate criticality of the negative effects and, based on the evaluation, prioritize the countermeasures to be realized. The criticality is basically evaluated in terms of probability of failure mode occurrence and severity of the negative effects. FMEA with this probability and severity evaluation is sometimes referred to as FMECA (Failure Modes, Effects, and Criticality Analysis). (See [5], [6], for some methods of criticality evaluation in FMECA.) The result of FMEA is also summarized in the tabular format.

Since FMEA is inherently a bottom-up analysis method starting from failure modes of the component, it may seem impossible to apply FMEA to the system extent until the system is completely decomposed into the components. However, it is absolutely unreasonable for large and complicated systems to perform FMEA and modify the system to increase reliability and safety after the system is completely decomposed. That will force us to abandon a large part of detailed design artifacts, or require a huge amount of development rework to satisfy non-functional requirements such as performance or for other reasons. Therefore, FMEA has evolved from a simple, component oriented method toward a process oriented method that performs analysis at various granularity of system decomposition along with stepwise refinement of the system. That is, FMEA is applied to each subsystem after decomposition of the system first; design of the system is improved at the subsystem level based on its result; the subsystems are decomposed into components; FMEA is applied to each component of the subsystem to improve the design of the subsystem in the similar manner.

FMEA is sometimes time-consuming. However, negative effects to the system brought by failure modes are not fully diverse; rather, we can observe a considerable amount of duplication. It is possible to deal with multiple failure modes having the identical negative effect as a group. This group is referred to as *Fault Equivalent Class* [7]. This concept contributes to reduce duplicated works in analysis and the scale of FMEA. It is reported that 12,401 failure modes are shrinked into 1,759 failure equivalent classes in the case study of the cabin management system of Boeing 777. The failure modes obtained in FMEAs of different abstraction levels are also grouped in the same failure equivalent class if their negative effects to the system are identical. That enables partial reuse of FMEA results performed in earlier phases in later phases.

HAZOP and FMEA are used for similar objectives and their results are summarized in similar tabular formats. The proposed method uses the “deviation” concept and the idea of guide words of HAZOP to identify failures of the normal operation. Countermeasures to the failures are studied with FMEA, not with HAZOP.

III. EXCEPTION HANDLING CONTROL STRUCTURE

The proposed methodology assumes use of the `try-catch-finally` exception handling control structure used in C++, Java and other similar programming languages. This section is dedicated to remind the readers of exception handling control structure.

These programming languages have the exception handling control structure of the form shown below:

```
try {
  ...
}
catch (ExClass formalExInstance) {
  ...
}
finally {
  ...
}
```

Normal operations are implemented in the `try` block, exceptional operations are implemented in the `catch` block, and clean-up operations are implemented in the `finally` block. Each `try` block can follow one or more `catch` blocks with different exception classes. The `finally` block is optional. `try`, `catch` and `finally` blocks can include another `try-catch-finally` block.

If it is impossible to continue execution of the `try` block or its callee functions, `throw` statement shown below should be executed:

```
throw actualExInstance;
```

Execution of the `throw` statement terminates execution of the `try` block. A `catch` block with the formal instance (`formalExInstance`) of an exception class (`ExClass`), which is compatible to the exception class of the thrown actual instance (`actualExInstance`), is responsible for the exception issued by the `throw` statement. The processor tries to find such a `catch` block out of the `catch` blocks of the current `try` block. If it is not found, the processor tries to find a `catch` block to be executed out of the `catch` blocks of another `try` block containing the current `try` block directly. The processor continues this backward traversal of nested `try` blocks recursively until it finds the `catch` block to be executed. Furthermore, if the `catch` block to be executed is not found in the function, the processor try to find it out of the `catch` blocks of the `try` block containing the current invocation point in the caller. The processor performs this backward traversal of function calls recursively, whenever there is no more `try` block to be checked in the function. After the proper `catch` block is found and executed, the processor transfers its execution to the point immediately after the exception handling structure having the executed `catch` block and the stack frames for the terminated `try` blocks and functions are released. The `finally` block is executed whenever control leaves the exception handling structure to which it belongs, regardless of whether a `catch` block is executed or not.

The actual instance of the exception class specified in the `throw` statement (`actualExInstance`) can be referenced in the corresponding `catch` block as its formal instance (`formalExInstance`). Therefore, the exception class is used not only to distinguish exceptions but also to pass data and control information required for exception handling by embedding them as its attributes.

IV. Specifying and Designing Exception Handling with FMEA

In this section, we propose the methodology to specify and design exception handling with FMEA. The methodology assumes that the system has already been decomposed into classes and the classes have already been designed for normal operations.

The methodology is described below in a stepwise manner with an example of the login form of the graphical user terminal. A user inputs his/her name and password in the text fields and then presses the login button on the form shown in Figure 1 to use the terminal. A member function `Login()` of class `LoginForm` is called on the login button press. The member function inquires of the user DB if the input name and password are correct by calling a member function `Auth(username, passwd)` of class `UserDB`. If they are correct, the member function closes the form and notifies its caller that the login is accepted. Otherwise, the member function waits user's further input of his/her name and password without closing the form or notifies its caller that the login is failed with closing the form.

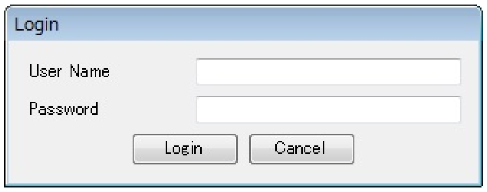


Figure 1. Login Form

A. Describing Normal Specifications

First, we describe the normal specification of each member function which are identified in class design. The normal specification of a member function is a sequential description of its internal operations from invocation to termination where the function successfully provides its service specified in the class specification. The steps must be in even granularity. The step should be in active verbal form, namely "do ...", and should not include multiple operations. The step can include conditional or iterative operations. A pseudo-code describing only normal operations of the member function can be dealt with a normal specification.

Normal specifications of `UserDB.Auth(username, passwd)` and `LoginForm.Login()` are shown in the *Normal Operation* column of Tables I and II, respectively.

B. Deriving Failure Modes

Second, we apply the methodology deriving failure modes, which was proposed by the authors [8], to normal specification steps in active verbal form "do ..." and look for failure modes of each step by imaging cases where the step

Table I
FMEA RESULTS ON `UserDB.Auth(username, passwd)`

Normal Op.	Failure Modes/Causes	Detection	Effects	Countermeasures	Not. to Callers
1) Checking if the user DB is connectable.	The user DB cannot be accessed.	The handle for user DB access is NULL.	I: User authentication is impossible.	**Run-Time:** Throw an exception.	The user DB is not accessible.
2) Reading the password of the user specified by the user name from the user DB.	Reading of the user DB is failed. / The user DB is locked.	The return value and the error code from the DB library	I	**Run-Time:** Throw an exception.	The user DB is locked.
	Reading of the user DB is failed. / Other reasons	The return value and the error code from the DB library	I	**Run-Time:** Throw an exception.	The user DB is unreadable for a certain reason.
	The specified user is not found.	The number of the matched records is zero.	I	**Run-Time:** Throw an exception.	The specified user is not found. + The specified user name
	Two or more specified user is found.	The number of the matched records is equal to or greater than two.	II: The user DB is logically corrupted. Further accesses may destroy the user DB completely and prohibit even partial recovery.	**Run-Time:** Throw an exception.	The specified user is found too much. + The specified user name
3) Checking if the password is correct.	The password is not correct.	The password is not equal to one in the user DB.	I	**Run-Time:** Terminate the function without authenticating the user.	Return value: false
4) End user authentication.			(The user is authenticated successfully.)	**Run-Time:** Terminate the function with authenticating the user.	Return value: true

cannot be accomplished successfully. The methodology assumes abnormal deviation of properties of objects, relationships among objects and behaviors as failures. Four HAZOP guide words *no*, *less*, *more* and *other than* are applied to the properties to derive failure modes by their deviation. To derive failure modes on behaviors, the methodology examines properties common to all the behaviors and specific to each behavior. The common properties are beginning time, ending time, duration, frequency, rate, interval, order *etc.*

The second normal operation step of `UserDB.Auth(username, passwd)`, "Reading the password of the user specified by the user name from the user DB." has a behavior *read* and objects *password*, *user*, *user name* and *user DB*. The HAZOP guide words are applied to their properties. The properties of the behavior *read* are the common ones mentioned above as well as *success or fail* and *the number of the records* which are specific to *read*. The properties of the objects *user* is *the number of the users* and the properties of the object *password* are *the number of the passwords* and *length*. Table III shows candidates of failure modes found by HAZOP application to these properties.

They are all candidates of failure modes. The candidates which are actually hazardous are adopted as the failure modes and listed in the *Failure Modes/Causes* column of the FMEA table after adjustment of their terms.

Countermeasures can be different depending on the cause of the failure. Therefore, if the abstraction level of a derived failure mode is too high to find a single countermeasure for the failure mode, we apply FTA [9] to the failure mode to identify its causes and study countermeasures depending on the causes. For example, in Table I, causes of the failure mode "Reading of the user DB is failed." are analyzed. A couple of causes, "The user DB is locked." and "Other reasons", are found and the countermeasures for them are studied separately. (The countermeasures for the both causes are same, namely, "Throw an exception." However, note that exception objects representing different meanings are thrown.)

If the function calls other functions, the contents of the *Notification to Callers* column must be duplicated in the *Failure Modes/Causes* column. For example, the failure

Table II
FMEA RESULTS ON `LoginForm.Login()`

Normal Op.	Failure Modes/Causes	Detection	Effects	Countermeasures	Not. to Callers
1) Getting the user name from the *User Name* field.	The user name is not specified.		I	**Run-Time:** i) Displaying a message telling the user name is not specified; ii) Clearing the *User Name* and *Password* fields; iii) Focusing the *User Name* field; iv) Continuing the login form.	Nothing.
	Invalid characters are used in the user name.		III: The user DB will reject login at Step 3.	(Unnecessary)	Nothing.
	The user name is too long.		III	**Dev. Time:** Set the maximum length of the user name field to the maximum length of the user name.	Nothing.
2) Getting the password from the *Password* field.	The password is not given.		IV: No problem because some users may not set their passwords.	(Unnecessary)	Nothing.
	Invalid characters are used in the password.		III	(Unnecessary)	Nothing.
	The password is too long.		III	**Dev. Time:** Set the maximum length of the password field to the maximum length of the password.	Nothing.
3) Asking the user DB if login is possible.	The user DB is not connected.	Exception from UserDB.Auth()	I	**Run-Time:** i) Displaying a message that the user DB is unavailable; ii) Terminating the login form.	Nothing.
	The user DB is locked.	Exception from UserDB.Auth()	I	**Run-Time:** i) Displaying a message telling that the user DB is locked; ii) Focusing the login button; iii) Continuing the login form. (Because the user DB may be unlocked later.)	Nothing.
	The user DB is not available for other reasons.	Exception from UserDB.Auth()	I	**Run-Time:** i) Displaying a message telling that the user DB is unavailable; ii) Terminating the login form.	Return value: false
	No user is found.	Exception from UserDB.Auth()	I	**Run-Time:** i) Displaying a message telling that the user is unknown; ii) Clearing the *User Name* and *Password* fields; iii) Focusing the *User Name* field; iv) Continuing the login form.	Nothing.
	Two or more users are found.	Exception from UserDB.Auth()	II	**Run-Time:** i) Displaying a message telling that the user DB is logically corrupted; ii) Terminating the login form.	Return value: false
	Password is not correct.	Return value from UserDB.Auth()	I	**Run-Time:** i) Displaying a message telling that password is incorrect; ii) Clearing the *Password* field; iii) Focusing the *Password* field; iv) Continuing the login form.	Nothing
4) Terminating the form.			(The user accomplishes login successfully.)		Return value: true

Table III
HAZOP GUIDE WORDS APPLICATION TO "READ" (UPPER), "USER" (MID) AND "PASSWORD" (LOWER)

Properties	Type	No Guide Word	Applied Guide Words			
		ϕ	*no*	*less*	*more*	*other than*
beginning time	time instant	Start reading at right timing	Do not start reading	Start reading too early	Start reading too late	/
ending time	time instant	End reading at right timing	Do not end reading	End reading too early	End reading too late	/
...	...	...	...	...	...	...
success/fail	boolean	Read successfully	Fail in reading	×	×	×
# of the records	number	Read a right number of the records	Read no record	Read too few records	Read too many records	/
# of the users	number	A right number of the users	No user	Too few users	Too many users	/
# of the passwords	number	A right number of the passwords	No password	Too few passwords	Too many passwords	/
The length of the password	number	A right length of the password	Null password	Too short password	Too long password	/
...	...	...	...	...	...	...

modes at the third normal operation step of the operation `LoginForm.Login()` in Table II are from the *Notification to Callers* column of the `UserDB.Auth(username, passwd)` in Table I.

C. Assessing Each Failure Mode

Third, we assess each failure mode; study how to detect the failure mode, how the failure mode brings negative effects, how to devise countermeasures against the failure mode, and what kind of information should be given to the caller; and complete the FMEA table. The FMEA table for the proposed methodology has, in addition to *Normal Operation* and *Failure Modes/Causes* columns, *Detection*, *Effects*, *Countermeasures* and *Notification to Callers* columns described below.

Detection: If the failure mode can be detected in the member function, we describe how to do it briefly.

If the failure mode can be detected in the callee of the member function, we describe how the callee notifies the member function that the failure mode is detected. See the *Notification to Callers* column described below for the details.

Effects: We describe how the failure mode brings negative effects with grouping them into fault equivalent classes described in Section 2 and assigning a sequential number referred to as *Fault Identifier Number* (*FIN*). Negative effects which are previously described in earlier and current phases are referenced by FINs, instead of describing the same thing twice or more. In the *Effects* column of Tables I and II, negative effects appeared first time are described with new FINs in the Roman numeral, however, ones appeared previously are just referenced by their FINs.

Countermeasures: We describe countermeasures to the failure mode, namely, how to avoid the failure mode and/or how to localize, compensate or mitigate the effect brought by the failure mode.

Various types of countermeasures are possible. Development time countermeasures are ones taken before release of the product, while run-time countermeasures are ones taken after release of the product. Some of run-time countermeasures are executed as exception handling.

Note that the countermeasures must be subject to the specification on the exceptional behaviors specified in earlier phases if they are available.

Notification to Callers: We describe manners and contents of notification to the caller when this member function encounters the failure mode. This notification is not mandatory and should not be performed in vain to keep information hiding and loose coupling. The notification is needed in the following cases:

- This member function cannot manage the failure mode within its specified responsibility.
- The caller of this member function can provide worthful actions to localize, compensate, or mitigate the effect of the failure mode.
- Expected reactions to the failure mode can be different depending on the caller of this member function.

There are some manners to notify the caller of failure mode occurrence such as the return value or reference parameters of the function, exceptions thrown by the function, global variables, invocation of prescribed or preregistered functions *etc.* The contents of the notification are data used in exception handling and/or control information which changes behaviors of exception handling in the caller. Note that description in this column will appear in the *Detection* column of the FMEA tables for the callers of this function.

Others: We can describe criticality, namely probability of failure mode occurrence and severity of the negative effects, for each failure mode to prioritize countermeasures to be taken.

D. Performing Commonality and Variability Analysis of Failure Mode Countermeasures

Fourth, we perform commonality and variability analysis of run-time failure mode countermeasures and construct a variability model of them. As the variability model, the proposed methodology uses an extension of the feature model [10], which is often employed in product line engineering [4] to describe commonality and variability among products in terms of their features.

The original feature model is a static model to represent variability among products in the product line (or product line variability), that is, the feature model represents how each product can select the features to be equipped. The variability model used in the proposed methodology represents not only product line variability but also variability in dynamic behaviors of the system (or run-time variability), that is, the variability model additionally represents how the system changes its behavior in terms of the features.

The variability model represents product line variability in an almost same manner as the original feature model: mandatory features are represented by nodes without any decoration, optional features are represented by ones with decoration of the white circle, and the nodes corresponding to a set of alternative features are grouped by the solid arc across the edges from the nodes to their parent node. It is also same as the original feature model that the solid thin edge represents the feature of the parent node partially consists of the feature of the child node and the solid thick edge represents the feature of the parent node is implemented by the feature of the child node. However, generalization/specialization relationship between the features is represented by the white arrow from the child node corresponding to the specialized feature to the parent node corresponding to the specialized feature, although the dotted thin edge is used in the original feature model.

Furthermore, the variability model represents run-time variability with symbolic extension to the original feature model. The solid edge means that the feature of the child node is always executed if the feature of the parent is executed. The dotted edge means that the feature of the child node may or may not be executed if the feature of the parent is executed. The dotted arc across the edges means that the features of the children of the bundled edges are alternatively executed if the feature of the parent node of the bundled edges is executed.

Figure 2 shows a variability model for the countermeasures described in Table II. The model does not include any variability on software construction, since the example system is not in product line development, but variability on software execution. For example, the variability model shows that the error message is always displayed but the message to be displayed is alternatively selected out of “Unknown user name”, “User DB is locked”, *etc.* The variability model tells us that the input fields may not be cleared, but if cleared, the user name field is always cleared and the password field may or may not be cleared.

There are some reasons why the authors use this variability model. While the class of the class diagram can represent only structural aspect of exception handling, the feature of the variability model can represent relating structural and behavioral aspect of exception handling in a consolidated manner. Moreover, variability modeling is a powerful technique to facilitate separation of concerns. For example, in Figure 2, features “Clearing fields” and “Focusing UI Object” are modeled separately. Since this separation enables independent handling of these behaviors, we can avoid scattering of similar or duplicated codes on these behaviors in the exception handling. The variability model can be used also in product line development, not only in single product development, since it inherits the properties of the original feature model.

E. Describing Exceptional Specifications

Fifth, we describe the exceptional specification of the member function. At the same time, we add and/or modify the normal specification of the member function if necessary.

We classify features in the variability model into *normal features* executed as normal operations in the `try` block and *exceptional features* executed as exceptional operations in the `catch` block. The features which terminate a series of normal operation steps for its execution should be exceptional features. The other features can be executed as either. After the classification, we describe the exceptional specification relating to the exceptional features in the same format as the normal specification. Moreover, we add and/or modify the existing normal specification to add the normal specification relating to the normal features identified in run-time failure mode countermeasures.

For the example of Figure 2, we can regard all the features in the variability model as exceptional features, since all the behaviors relating to the features require termination of the normal operation.

F. Designing Exception Classes

Finally, we design exception classes with referencing variability models. The variability model, which has already been constructed for each member function of the class, contains features that represent run-time behaviors as countermeasures for failure modes, namely exceptional operations, and data used in the exceptional operations. Moreover, note that the variability model represents control information specifying which feature should be executed conditionally in exceptional operations.

Before designing exception classes, to reduce the number of exception classes, variability models of member functions

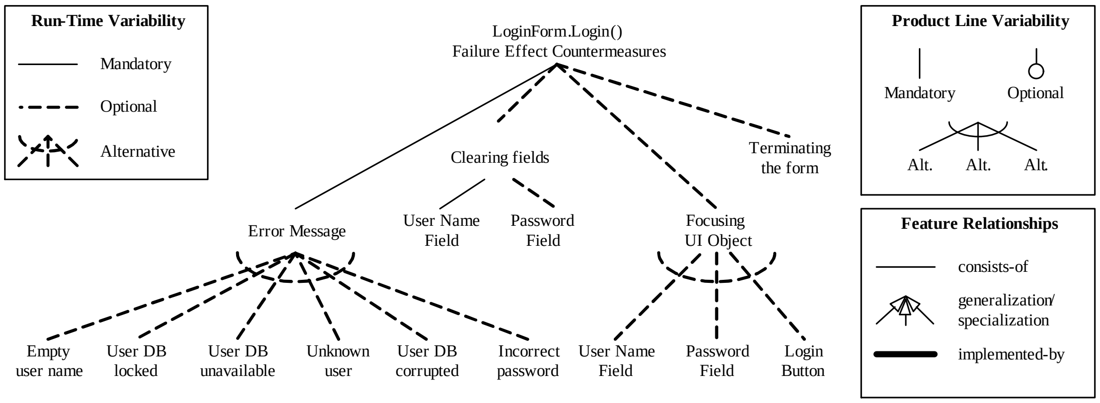


Figure 2. A Variability Model for Run-Time Failure Mode Countermeasures

should be merged and integrated if they are closely related and share a lot of features. The merging and integration are performed based on the names of the features. The variability model may be required to be refactored in this activity. For example, we have to rename the features, if they have the same name although they have different meanings. Moreover, we have to restructure the variability model for integration, if the same features from different variability models are organized in different tree structures.

We define an exception class for each integrated variability model. The features which represent control information changing run-time behaviors of the exceptional operations are reduced into attributes for flagging. The features which represent data used in the exceptional operations are reduced into attributes. Reference and conversion of these attributes are defined as member functions of the exception class.

The exception class reduced from the variability model shown in Figure 2 is as follows:

```
class ExLoginForm : public Exception
{
public:
  enum LoginFormError {
    ERR_EMPTY_USER_NAME,
    ERR_USERDB_LOCKED,
    ...
  } login_form_error;
  enum ClearingFields {
    UserNameField = 0x01,
    PasswordField = 0x02,
  } clearing_fields;
  enum FocusingUIObject {
    UserNameField,
    PasswordField,
    LoginButton
  } focusing_UI_object;
  bool terminating;
  string getErrorMessage();
  ...
};
```

In this reduction, a feature representing data or behaviors taken alternatively such as "Error Message" and "Focusing UI Object" is reduced to the enumeration data member. A feature representing data or behaviors taken multiply such as "Clearing fields" is reduced to the bit field data member. A feature representing data or behaviors taken optionally such as "Terminating the form" is reduced to the Boolean data member.

Considering overheads, use of `try-catch-finally` exception handling control structure should be carefully limited. Countermeasures can be implemented by general control structures and variables instead of exception handling control structures. If we can write the same thing without losing readability and producing duplicated codes in exception handling, use of general control structures is preferable. The proposed methodology will be helpful also in such a case to realize exception handling codes in a consistent manner and without including duplication.

V. Related Work

This work is refined from authors' previous work [11] presented as a non-peer-reviewed, ongoing paper in Japanese.

Although FMEA was originally applied to mechanical and hardware systems, efforts applying FMEA to software has been continued so far [8], [12], [13], [14], [15], [16], [17].

HAZOP is an effective methodology to find possible failure modes in a comprehensive manner. The failure mode derivation methodology used in this work applies HAZOP guide words to properties of behaviors and their targetting objects [8]. The methodology is an extension of Kouno's work [18]. HAZOP is applied to software for UML in Hansen's work [19] and for state transition diagram in Kim's work [20], for example. Both works are for descriptions in higher abstraction level than this work.

In this work, FTA is also used to seek for the causes of the failure and study countermeasures depending on the causes, in case the abstraction level of the failure mode is higher. The approach looking for the causes from the failure mode is taken by Lutz *et al.* [12] and Goddard [13], for example.

On the other hand, Maxion *et al.* presents an approach based on the viewpoints given by fish-bone diagram that classifies exceptional conditions into computation, hardware, input and output, library, data, return value, external environment, and null pointer and memory problems, which can be abbreviated as "CHILDREN". [21]

VI. Conclusion and Future Work

In this paper, the authors proposed a methodology to specify and design exception handling for `try-catch-finally` exception handling control structure of C++, Java and other similar programming languages.

The proposed methodology applies a HAZOP based failure mode derivation method proposed by the authors to each normal operation step of the member function. FMEA is performed to study countermeasures to the identified failure modes. The table produced in FMEA facilitates consistent and disciplined design of exception handling. Commonality and variability analysis is applied to the countermeasures studied in FMEA. Commonality and variability analysis contributes to eliminate duplicated codes in exception handling codes. A variability model constructed by the analysis is used to design exception classes.

The proposed methodology will be used not only for newly development but also for refactoring of exception handling codes in the existing system. The future work includes large scale application of the methodology to real applications and empirical validation of the methodology.

References

[1] Flaviu Cristian, "Exception Handling and Tolerance of Software Faults," *Software Fault TOlerance*, M. R. Lyu, ed., Chapter 4, John Wiley & Sons, 1995.

[2] IEC Standard, *Hazard and Operability Studies (HAZOP Studies): Application Guide*, IEC 61882 ed1.0, 2001.

[3] IEC Standard, *Analysis Techniques for System Reliability: Procedure for Failure Mode and Effects Analysis (FMEA)*, IEC 60812 ed2.0, 2006.

[4] Paul Clements and Linda Northrop, *Software Product Lines: Practice and Patterns*, Addison-Wesley, 2001.

[5] John B. Bowles, "The New SAE FMECA Standard," *Proc. Annual Reliability and Maintainability Symp. (RAMS) 1998*, pp. 48–53, Jan. 1998.

[6] John B. Bowles, "Fundamentals of Failure Modes and Effects Analysis," Tutorial Notes, *Annual Reliability and Maintainability Symp. (RAMS) 2003*, Jan. 2003.

[7] C. Steven Spangler, "Equivalence Relations within the Failure Mode and Effect Analysis," *Proc. Annual Reliability and Maintainability Symp. (RAMS) 1999*, pp. 352–357, Jan. 1999.

[8] Tsuneo Nakanishi, Kenji Hisazumi, and Akira Fukuda, "A Software FMEA Method and Its Use in Software Product Line," *IEICE Technical Report*, Vol. 111, No. 481, pp. 19–24, Mar. 2012. (in Japanese)

[9] IEC Standard, *Fault Tree Analysis (FTA)*, IEC 61025 ed2.0, 2006.

[10] Kyo-Chul Kang, Jaejoon Lee, and Patrick Donohoe, "Feature-Oriented Product Line Engineering," *IEEE Software*, Vol. 9, No. 4, pp. 58–65, July/August 2002.

[11] Tsuneo Nakanishi, Kenji Hisazumi, and Akira Fukuda, "Specifying and Designing Exception Handling with using FMEA," *IPSJ SIG Technical Report*, Vol. 2012-SE-175, No. 14, pp. 1–8, Mar. 2012. (in Japanese)

[12] Robyn R. Lutz and Robert M. Woodhouse, "Experience Report: Contributions of SFMEA to Requirements Analysis," *Proc. 2nd Int. Conf. on Requirements Engineering (ICRE '96)*, pp. 44–51, Apr. 1996.

[13] Peter L. Goddard, "Software FMEA Techniques," *Proc. Annual Reliability and Maintainability Symp. (RAMS) 2000*, pp. 118–123, Jan. 2000.

[14] John B. Bowles and Chi Wan, "Software Failure Modes and Effects Analysis for a Small Embedded Control System," *Proc. Annual Reliability and Maintainability Symp. (RAMS) 2001*, pp. 1–6, Jan. 2001.

[15] Dong Nguyen, "Failure Modes and Effects Analysis for Software Reliability," *Proc. Annual Reliability and Maintainability Symp. (RAMS) 2001*, pp. 219–222, Jan. 2001.

[16] Nathaniel Ozarin, "Failure Modes and Effects Analysis during Design of Computer Software," *Proc. Annual Reliability and Maintainability Symp. (RAMS) 2004*, pp. 201–206, Jan. 2004.

[17] Ajit Ashok Shenvi, "Software FMEA: A Learning Experience," *Proc. India Software Engineering Conf. (ISEC) 2011*, pp. 111–114, Feb. 2011.

[18] Tetsuya Kouno, "An Application of HAZOP to Risk Analysis of Software Requirement Specification," *Proc. Japan Symp. on Software Testing 2012*, pp. 37–42, Jan. 2012. (in Japanese)

[19] Klaus M. Hansen, Lisa Wells, and Thomas Maier, "HAZOP Analysis of UML-Based Software Architecture Descriptions of Safety-Critical Systems," *Proc. 2nd Nordic Workshop on the Unified Modeling Language (NWUML) 2004*, pp. 59-78, Aug. 2004.

[20] Zoohaye Kim, Yutaka Matsubara, and Hiroaki Takada, "A Safety Analysis Method Based on State Transition Diagram," IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, Vol. J95-A, No. 2, pp. 198-209, Feb. 2012. (in Japanese)

[21] Roy A. Maxion and Robert T. Olszewski, "Eliminating Exception Handling Errors with Dependability Cases: A Comparative, Empirical Study," *IEEE Trans. on Software Engineering*, Vol. 26, No. 9, Sep. 2000.

Orchestration Definition from Business Specification

Charif Mahmoudi
Logics, Algorithm, Complexity Laboratory
LACL, Paris 12 University
Creteil, France
charif.mahmoudi@lacl.fr

Fabrice Mourlin
Logics, Algorithm, Complexity Laboratory
LACL, Paris 12 University
Creteil, France
fabrice.mourlin@wanadoo.fr

Abstract—The implementation of service orchestration is often seen as a convoluted process for business analysts, lead developers and architects. In this document, we propose a new approach based on a continuous process starting from the analysis phase to the architecture phase as an attempt to standardize the implementation of service orchestration. Our ultimate goal is to have a BPEL (Business Process Execution Language) script which will be interpreted by an engine residing inside a middleware generating a composition of elements where each element can be considered as an independent component equipped with a Web service. Orchestration definition contains several facets such as logical, pragmatic and architectural aspects; each of them is complementary and the interaction between them usually raises conflicts. In our approach, these issues are addressed and solved by adaptation rules and the problem of adapting the software architecture onto a physical architecture is solved by the pragmatism method.

Keywords-SOA; Architecture; Web service orchestration; business process design and specification

I. INTRODUCTION

The component-oriented approach has emerged and has become widespread in the industry to meet the scalability of information systems [1]. It reduces software costs and allows rapid adaptation to changing business and technological developments. It also enables software components to create highly modular and integrated. The development of certain parts of the information system may be too independent.

This component-oriented approach allows governing the evolution of technical and functional information system based on standard software. In addition, it covers all aspects of development and life cycle of the software. With the emergence of the component-based modeling paradigm, the OMG (Object Management Group) did not remain inactive and has proposed a new architecture based on MDA rules [2]. The development has facilitated UML modeling components. In 2001, the OMG defined the MDA approach with the aim to facilitate the integration of applications and make the specification of independent application development technologies. It also sets rules for mapping the standard specifications of different technologies [3].

UML Modeling tools generate the code source structure of applications. There is a transformation of a logic model to design model to the platform on the basis of design pattern templates and code [4].

The SOA is not far away removed from the component-oriented approach; see Peter Herzum [5]. He is one of the first authors having clearly defined the concept of components and component architecture. From his point of view, there are three types of components: Components "business process", components "business entity" that implement a core business concept, and finally, the component "business tool" used in various system components. He proposed to build a system specification based on four models. A business process model is used to identify components known as "business process" that manage one or more use cases. A model of "business entities" supports one or more business processes. A model is created to define business events. Another model is created for the definition of business rules.

The component-oriented approach has been developed within companies, but the purpose of sharing common components is often wishful thinking. Projects are organized into business lines. The application needs vary greatly over time. The services are requested too often, and the code of common components is duplicated and modified directly in new applications as alternatives. Reuse requires the establishment of specific resources such as the development of cataloging tools, dissemination of information about the components, creating a team to administer the transverse components. It also requires the definition of a target upstream of urbanization of the information system. We present in this contribution our approach to defining orchestration from business specification, and to mix it with other reused components. In the next section, we explain our design process for SOA architecture. Then, we give details about the semantic model of our approach and pragmatic model also. The following section is about logical model and how we declare it. The last part is about architecture and implementation of orchestration. Finally, we provide a case study of our approach.

II. DESIGN PROCESS FOR SOA ARCHITECTURE

The concept of enterprise architecture management was gradually adopted in enterprises to address the problems of organization and urban information system. Different methodological approaches have been developed or framework to build and maintain this architecture such as Zachman [6], [7] and TOGAF [8], [9] or EUP (Enterprise Unified Process) [10], [11]. Our approach provides a method for developing SOA and managing the complexity of the enterprise by integrating the evolution of new technologies. It is based on modeling different aspects of the system and a process of construction and derivation of the models. Basically, there are two views. An external view for describing the company level: business data manipulation, organization and business processes. An internal view can gradually develop the system: logic model and technique to build the software deployed on the hardware. The physical model describes the implementation and deployment. It extends to the logical architecture with the definition of service components and the technical architecture design. Our approach is providing a comprehensive urbanization of information systems by using a semantic model and a pragmatic model. We add to this a logical architecture with a design of the technical architecture.

A. *Several models for a given project*

We start our approach by a first pre-model, which is used to define a common vision to the various players in the enterprise. The description languages used are UML [12] and Business Process Modeling Notation (BPMN) [13]. The pre-model provides a dictionary of terms in the application field, an analysis of objectives and business needs. We identify high-level processes and key use cases and the fundamental business rules.

Our semantic model is intended to describe the basic business concepts of the company. This model can be established at two levels: The overall level contains the definition phase of urbanization. The local business domain contains the definition of business service. The purpose of the construction of these models is to achieve stability of business concepts. We build mostly by diagrams such as UML class diagram semantics, OCL constraints. For instance, Figure 1 provides a semantic model about bank operations:

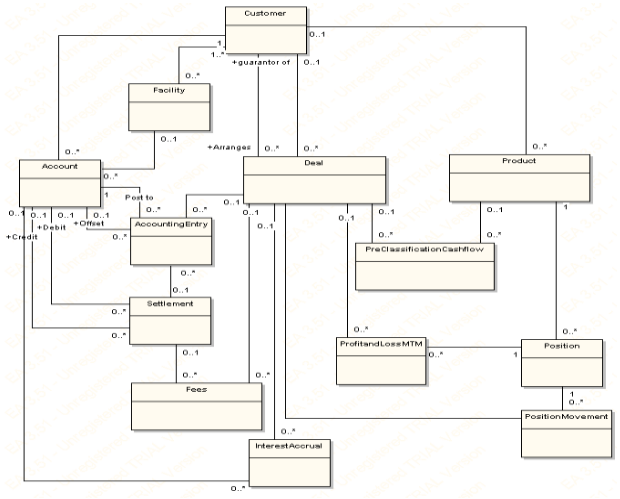


Figure 1. External view - business class diagram as semantics model.

Behind this kind of diagram, vocabulary is bound and a first set of constraints is taken into account. This business model is the knowledge of the company. This business model acts as a common language between all company projects. It can be built by successive iterations. In that diagram, Semantic classes and main attributes are defined. The life cycle of the business classes is also described. Relations between business classes are also provided clearly. Relations could be evolved with precise detailed information. For instance association between Deal class and PreClassificationCashflow class can change into a composition under business conditions.

The different projects feed into the common semantic repository. The difficulty of this analysis involves the construction of an observation with no prerequisites. The aim is to describe the business concept and not handled the technical which has been used in existing projects. The management of business objects needs a workflow description. Also, we have added such diagram attached to the fundamental business class. As an example, we give a description of the Order Fulfillment Process of a bank product (OFP) (Figure 2). The diagram shows that secondary business classes can be added for describing the process. It also focuses on the responsibilities of each step of the workflow.

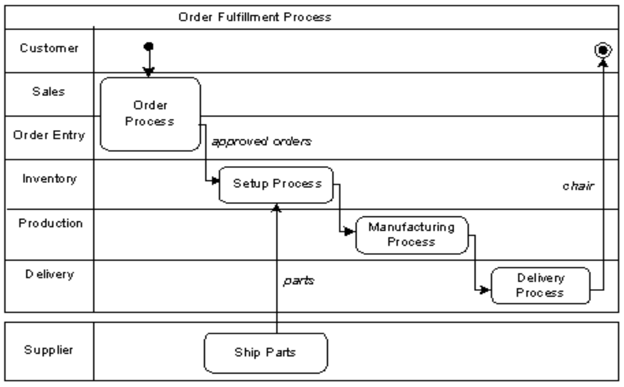


Figure 2. Workflow with partitions showing who does what work.

This global diagram provides main collaborations into business process and precisely causality between main events in the process. It also highlights synchronization between processes. This automaton is generally deterministic. Controls can be done with other workflows and conflicts are then detected which improve our models.

Business modeling is to improve the abstract concepts. The diagram should be simple and generic. It can anticipate the consideration of future developments.

The repository contains semantic early different business areas and key objects. It is enriched in with new projects. A review of models is to perform when they are stable.

B. Architecture and semantics

Then, the specification of services is driven by the business analysis. It involves business managers and technical managers. It requires defining relationships between the UML diagrams that were constructed. Then, all descriptions can be observed as a multi layers diagram as follows (Figure 3).

A business layer is based on the functional layer, this functional layer depend on the application layer. The implementation of this application layer is described by the technical layer.

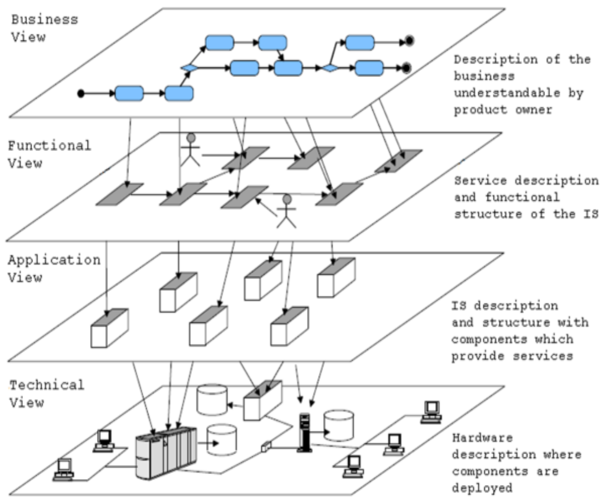

Business View
Description of the business understandable by product owner
Functional View
Service description and functional structure of the IS
Application View
IS description and structure with components which provide services
Technical View
Hardware description where components are deployed

Figure 3. Methodology approach.

The use of four views allows traceability of the business to the computer. It provides the logical sequence between the business views, functional and application.

This top-down functional approach may have a downside: how to structure the architecture of services into a stable structure? And witch levels of abstraction in the information system will we consider. Our answer is a dual approach. First, learn the basic services starting from the semantic model (Figure 4, functional view) and also complement the services based on the principles of the organization (Figure 4, application view): use case of the information system and business process details.

The modeling principles presented apply with a global reach and local levels. Of course, there are analysis and design and the need to reorganize and streamline processes. Use case diagrams of the information system are the business functional requirements that must leave the system in a consistent state. It ensures the unity of actor, unity of place, time unit.

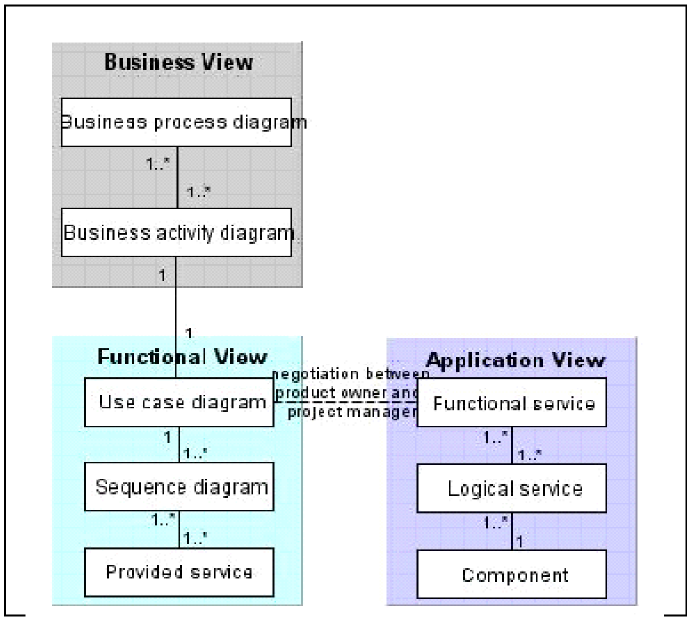


Figure 4. UML models used in our approach.

Our method for identifying use cases is classic. It identifies the actors, list the types of events. Finally, we deduce the interactions with the system. Then we structure all the use cases to a hierarchy or order planning for a future project.

Of course, in some projects, it is necessary to take into account organizational or operational constraints. In this case we show a view of the organization. It describes the process in relation to the business organization.

Other constraints must also be considered as the geographical dispersion of actors and business processes. These features may require significant optimizations. For example, the choice of appropriate new technologies can help streamline and simplify processes. For example, the nomadic operation belongs to that kind of constraints.

The underlying logical model is intended to specify and organize the services of our SOA. This is accomplished by the use of semantic diagrams (such as Figures 1 and 2). This defines the logical architecture that will be derived within the meaning of the Model Driven Architecture (MDA) approach [14] in software components. The word “logical” denotes the sense that the logic model should remain free of any technical choice. It contains a Platform-Independent Model (PIM) [15]. They can be derived to different technical platforms: J2EE, .NET, ESB, Web services, etc. Our logical architecture defines the components and services based on semantic aspects, pragmatic aspects and geographic aspects.

Our logic model is not deprived of any technical concern. It must produce a coherent model; this model must be implementable effectively while respecting technical choices.

C. Multi layer architecture

We structure our layered architecture. We distinguish a logical level, related to semantic classes, an organization level, linked to the workflow or orchestration and finally a technical level related to the problems transverse. Our service concept is seen as elementary grain architecture. Services are provided at the logical level. They correspond to elementary operations directly related to the state machine of the main class of business concept. These operations are the transitions of this automaton. They normally have been specified in the semantic model.

The types of data exchanged between services must be precisely specified. They correspond to the design pattern DTO (Data Transfer Object) J2EE applications. It is the pivot language used in the process orchestrations. They reduce the number of parameters of services.

This data are formalized as classes belonging to a mapping utility. They are grouped in a factory to make them accessible to all services. These classes allow access to properties using getter and setter. All instances of those classes can be exposed through the use of context.

All components must provide at least one interface, which is to say all the public services exposed to the outside. The interfaces are not accessible. A component can provide different interfaces for different tasks of the components. A component also includes one or more data structures combining the exchange of trade data structures internal components. After deployment time these components will be exposed via a component server.

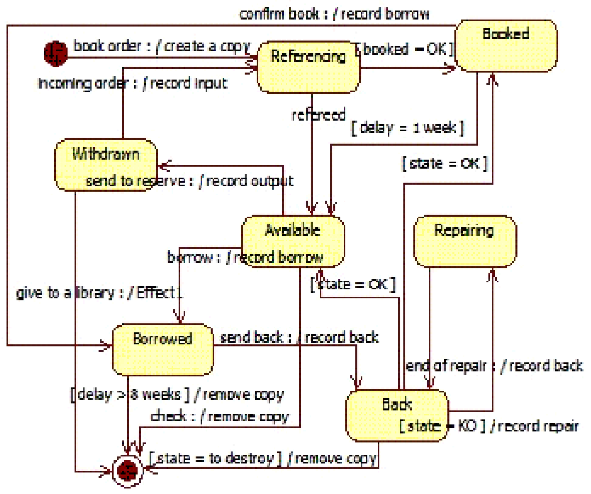

confirm book : / record borrow
book order : / create a copy
Referencing
[booked = OK]
Booked
incoming order : / record input
refereed
[delay = 1 week]
Withdrawn
send to reserve : / record output
[state = OK]
Available
Repairing
borrow : / record borrow
[state = OK]
give to a library : / Effect1
Borrowed
send back : / record back
end of repair : / record back
[delay > 8 weeks] / remove copy
check : / remove copy
Back
[state = KO] / record repair
[state = to destroy] / remove copy

Figure 5. State chart of a copy

Use cases should have been structured to eliminate any redundancy. Each use case is then derived in an elementary workflow where each activity is a candidate to become distributed logical service. Basic activities correspond to simple exchanges with users. The others are often called business services.

III. SEMANTIC MODEL OF OUR APPROACH

Our semantic model is intended to gradually create a stable business model describing the general business concept business fundamentals. It corresponds to the concepts and business objects of the project field. It is described with UML notation: class diagrams with semantic attributes, relations between the business concepts, business rules that constrain them, the life cycle of the business classes.

A. *Constraints on semantics model*

The requests on the semantic model are: tractability upstream. It is useful to be able to justify the model in relation to its inputs (functional requirements, legislation, regulations, etc.). Other requirements relate to restitution: the diagrams must be interpreted in natural language. We must keep the synonyms of the terms in a thesaurus. Finally, the model should express the semantics of the domain while excluding of any other aspect.

Gradually, it evolves the model is documented in the project including different aspects such as the number of handled instances, their persistence...

The quality of such a model is assessed with reference to classical properties. The non-coupling expresses that each capture a single semantic entity. The homogeneity requires that we do not aggregate various aspects of business semantics. Sufficiency occurs when classes are all the information. Completeness is achieved when all the relevant features are included in the model.

This model is important because it is a communication medium between the project partners. Moreover, it must be easily usable by all members intervened in the project, whether internal or not.

Constraints and business rules are encapsulated in classes. The constraints are described as the sources of method or attribute.

The life cycle of business objects is described using finite state automata. Its purpose is to identify all the events changing the state of business objects and operations of a semantic nature. A second goal is to identify all the disturbances affecting the cycle of the object: the trigger events, operations performed during the transition.

B. *Example of semantics model*

We have studied workflow of copies of books which are managed into a library. The library has several sites into a town and books can be transferred from one site to another if there are not borrowed more than eight weeks. Other business rules are defined by business expert. This kind of diagram (Figure 5) is an ideal support for expressing rules and constraints because all existing cases are taken into account.

When constraints change, such diagram catches all new constraints, even if they involve the refactoring of the whole diagram. Because this diagram is linked to a business class, new methods enrich its behavior. Of course, such a description is rich in information and can update the business class diagram. These events are in addition to methods for the question of the life cycle of objects. Then we created a package diagram by business domain. This constitutes around major business classes such as Library, User, Copy, etc. The semantic model is valid by reviewing models. It is also useful in cases of unit test on the model. This means checking state machines and the events received and forwarded. The model is simple and it is likely to be stable.

IV. PRAGMATIC MODEL AND PHYSICAL DISTRIBUTION

A. *Process Analysis in UML*

UML provides good modeling tools [16]. Use cases provide an overview of the system: reactions reports by the needs of actors. They can be used at two levels. Informally, they are used to identify the processes and use cases in a phase of reflection. Finally, more formally, they are useful for describing use cases of information system.

Activity diagrams provide a graphical representation to represent the largely consensual process. Sequence diagrams are used to describe the use case scenarios. But they are limited in their use to cases nominal and do not necessarily provide added value. The state diagram can also model the status of an activity of a business use case. They clear the events that affect its performance. Class diagrams are used to rank the players.

The difficulty of the analysis process is the level of granularity of modeling. The distinction is between processes and sub processes, tasks and activities. We often encounter a meta-level model of the company to agree on the level of detail of this modeling. The issue is the granularity of the service and its reusability often introduces unnecessary and lengthy discussions.

In this type of study, there are little methodological approaches completely formalized and really consensual. The existing methodological approaches are usually the owners of these processes. We preconize to use an open source approach to be able to use instrument allows fully shared processes by the tools.

B. *The organizational view*

1) *Process definition*

It contains descriptions of the process. This description comprises both system responses to external events and also the information flows (flows, object flows, internal events), it also includes coordination between the activities of different actors.

The view also contains organizational decisions and their explanations. This includes configuring services, distribution of responsibilities, the profile of players and possibly other constraints such regulations. This view is also about the definition of classes related to problems of organization and management. It also contains classes related to business events that are in the semantic model.

We consider a process as an ordered set of activities to produce a result: the production or transformation of an object. Our pragmatic model describes the process the processing steps acting on objects in the semantic model are already described as state machines associated with these objects.

Activity corresponds to an action or a set of actions. The mastery of activities requires organization and rules that are not present in the semantic model.

The process space is hierarchical, it is important to begin the descriptions of the most important processes. It is unnecessary to describe the process with a full level of detail rendering them incomprehensible except to experts. We establish a general map of the process. We determine the key process, that is to say those criticisms vs. strategic objectives. It is important to analyze the risks.

Our approach to analyze the process remains a classic. First, we outline the beginning and end of the process, and then describe the goal. This means knowing the customer's expectations. In addition, we describe the interface with other processes. We detail the resources used: objects manipulated. We add the traceability rules. Finally, we define the associated skills: entities contributing to the process. We list the actions that can be activated with the possible exception thrown.

We use activity diagrams for our performances. They contain the events sent or received, the conditions of the transitions, the parallel workflows and exchanged business objects whose states are monitored.

When existing processes are described, it is possible to reconfigure to improve efficiency, improve flexibility. Further improvements are possible to provide a better level control and smoother operation and even reduce the execution time of processing. As an example of improvement, there is research into the causes of waiting by grouping tasks within a single activity or eliminating seizures or occasions of data.

2) *Modelling approach*

Our approach aims to extract the organizational aspects. It is important to analyze the process by respecting their borders. For this, we focus on objects involved in the process. Processes frequently collaborate within the same activities and it is important to specify the transactional aspects. This is specified as a string of treatments which obeys the rule of all or nothing. The scope should be as far as possible, as small as possible.

Of course, there arises the problem of transaction management long term which could several hours or several days. There is no question of pausing transactional locks on objects handled; the rollback is managed by a compensation mechanism.

3) *Use view*

An actor performs a series of transactions during his dialogue with the system. But restrictions apply: a scenario is not interruptible in business perspective view. In addition, a use case is single-player. Use cases describe the purpose of use. These are functional requirements that must leave the business information system in a consistent state.

Our method for identifying use cases is very simple. We first identify the actors and list the events and infer interaction decomposition systems. This view must be comprehensive to describe all interactions between the actors and the system. Each use case typically handles one main purpose. It is important to ensure that all use cases described covers many transitions in the state machines of the major classes of the semantic model.

C. Physical distribution

To complete the functional requirements, it is important to locate users and geographic information systems. We add the technical requirements for equipment: technical guidance on network configuration requirements for the workstations. Other sensitive issues include competition within use cases and the constraints of scalability.

Nonfunctional constraints are taken into account as the degree of availability desired. Requirements related to quality of service, sometimes, bring the definition of categories of use cases. And a class level of service quality is associated with accessibility criteria.

Constraints related to the geographic dispersion are difficult because they require difficult technical choices. A site is described by its location, its capacity. He mentioned the players it hosts and the type of activities taking place there. Communication between sites is via the media. They should list them: communication network, transport, etc.

We use deployment diagrams, collaboration diagrams eventually. The collaboration diagram is often used early in the project to be within the information system and show the flow of information. Deployment diagrams are used in hand continuously from one project to another.

V. Logical Architecture model

A. Approach

The logic model is used to specify and organize the service of SOA, based on semantic and pragmatic views. It defines the logical architecture of SOA will be derived in software components. Phase logical architecture of the system is similar to all phases of project management methodology. One of the rules is to minimize dependencies. Finally it is important to consolidate services related to business classes or use cases.

We chose to group services by field of business objects. We added a set of factories. These factories are the first level of urbanization. They correspond to the main structure of the information system. A factory has no interface but represents a logical division of classes. It involves important properties of the SOA. This multilevel structure organizes the data flow between the parties of the information system. Encapsulation manages the relationship between different components of the architecture: The level of services in terms of mask data.

Our methodological approach can be summarized. First, there is the structuring of the logic model in key areas. Then, there is the recovery of the semantic model and the derivation of detailed models, definition of complex data types. They are used to exchange information between services. They are grouped with the utilities. Then, we treat the analysis of use cases to discover the additional services. This involves grouping into packages. Finally, there is a detailed description of services (business and technical).

It is necessary to designate the services exposed to the outside world. For efficiency reasons, it is crucial to choose the type of invocation (web service, SOAP, XML, RMI, etc.). Transaction management also has an impact, especially for the resilience (the backup orchestration of context)

B. Structuring of the semantic model

Each main class of the semantic model is the heart of a logic component. The division into logical component follows the same structure as the division into business components. Dependence after a combination of the semantic model can be managed in several ways. First, it can be passed as a parameter of the service to remove a strong dependency. Secondly, it is possible to have data at the service orchestration. Do keep the dependencies in the types of data exchanged.

It is important to distinguish two types of services at the level of elementary components. On the one hand the services those run on a single component instance. These are the services associated with managing the life cycle. On the other hand the service for handling collections and navigation. These are services that work on sets to calculate a subset of data: search, sort, query, verification of existence.

The data exchanged are specified in detail because they are the pivot language that is used in the process orchestration. They reduce the number of service settings. The data related to a secondary class masked by a main class are managed by incorporation of a subtype in the main type. The data related to a semantic class of another component are managed by reference. They are retrieved by accessing a directory.

A logical component is described by an interface. All utilities are exposed to the outside. It also includes one or more internal data structure. Of course this interface is not accessible directly but through an access server. Optionally, a component can provide multiple interfaces for different missions.

C. Structuring of the pragmatic model

Each use case results in a Transactional service to validate a customer dialog. The transactional service logic starts a transaction that contains technical information transfer. This means that inspections are carried out with the use of a rules engine. If the checks are correct, the service validates the transaction and returns the information resulting from the transaction. In the pragmatic model, processes are described in terms of activity diagram. This diagram is attached either to the functional field or it’s a package associated with crosscutting activities. The activity diagram is shown in the logic model and built to represent the functional area. It shows the services that keep coming into the process. Human interventions are indicated by actions that refer to use cases.

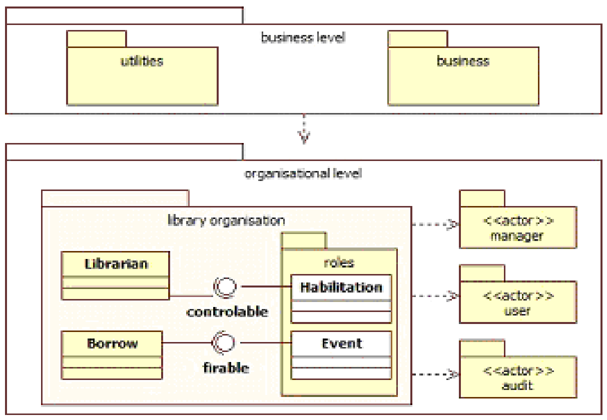


Figure 6. Structural diagram of the organization

In our previous example, organizations are run by factories logic. Organizations share the same core but can describe their operations as separate. In Figure 6, the decomposition of the system comprises both the context of use (organization, actor, ID), the informal context (set of information related to the trade).

VI. ARCHITECTURE AND ORCHESTRATION

A. *Service specification*

In service specification step, we may choose the signature and the names that will be present in the WSDL.. The parameters in reading, writing are prohibited. In an SOA, the transition is performed by parameter value to allow only based implementation web service. Additional parameters are the context, they provide information about the user context when the invocation. This information may influence the functionalbehavior of the service. They help minimize the number of services.

It is important to provide an exhaustive list of error codes. Non-blocking errors do not throw program level exception because these kinds of exceptions are catched and managed in framework level. We can provide services in signing a complex type that allows logging errors non-blocking.

Preconditions and post conditions can guarantee the conditions for running a service. They are based on the parameters passed as input. They are typically implemented by a direct appeal of the checking methods. This is a prerequisite for triggering of the main algorithm. We chose to externalize their code of the service implementation not the use of AOP (aspect oriented programming) [17].

Service quality is also specified. It is a guarantee of performance, availability and security. It relates to the average response time, the number of calls per second web service, the number of sessions, etc. We specify the end user monitoring: indicators and measures used. The documentation part of the services is important that a supplement should not be overlooked because life depends on it.

B. *Technical aspects*

Technical aspects that we addressed in our case study are the persistence, security, object-relational mapping, archiving issues, and the implementation of business rules and data architecture. Management communication was done by the web service call usually asynchronous.

We have implemented the business classes by POJO (Plain Old Java Object). We create factory for façade to delegate calls to the implementation classes. We distinguish the services handling a single component instance handling collection of services. These services must appeal directly to a data service that handles requests multiple instances of the database.
Queries performing joins on several business classes should be modified to remove dependencies between elementary concepts.

The process service calls keep coming. It is necessary to implement a facade since the methods have directly initiated the process. If several processes contain an identical set of activities, these can be managed using a process as implemented by a class in each package of integrated process that contains it. Implementation of this principle respects the principle of SOA but uses conventional technologies.

Processes are the orchestrators of calls to operations of business services. In terms of architecture, the process includes a presentation layer. A process is conventionally implemented as a component state full or using an orchestrator. In the first case, we must manage the execution context and make the system fault tolerant. We constructed an intermediate layer adaptation allows both to transform data and orchestrate existing transactions. The problem with JavaEE arises with the use of external transaction via tools such as SAP via JCA connector. We use a SOAP wrapper to trigger the transaction from operations.

We used the framework WISIF (Web Service Invocation Framework) from Apache to call the JCA connector. For security aspects, we wanted to make confidentiality and identification of access rights. The use of SSL is possible to exchange point to point but quickly becomes difficult with the spread and use of web services. The security management which is integrated directly within the SOAP messages [18]. The standard WS-Security OASIS framework provides a stabilized security manager. It enables strong authentication based on Kerberos ticket and is based on a W3C specification.

For transaction management, three aspects are taken into account with different frameworks. WS-Coordination provides a protocol to coordinate the actions of a distributed application (creation and propagation of context between the services). WS-Atomic Transaction defines transactions with a simple method of two-phase commit. Finally, WS-BusinessActivity can coordinate distributed activities with long transactions.

We used a middleware for the exchange of asynchronous messages. It has several properties: the ordonnancement messages, persistent messages in the event of service interruption, the integration of new components. All of our orchestration is made with the BPEL language. As defined above WSDL, it can describe a collection of Web services. Activities can be combined with additional elements to form structured activities. For asynchronous calls, callbacks are defined by use of framework WS-Addressing.

CONCLUSION AND FUTURE WORK

We have presented our approach of orchestration definition. We have structured the design time in a sequence of model definition: semantics, logic, pragmatic. We have shown that relationships exist between them. These allow us to check and update our design. Then we have explained that BPEL scripts are derived and deployed into the business part of our multi-layer application.

Our future work will focus on extending our approach to other orchestration languages like CAMEL DSL [19]. Our goal is to enrich Java DSL's routing for managing dynamic mobile Participant that implements WS-Coordination. The participants are defined in functional layer; our approach offers a solution to adapt a functional definition to a generated Participant implementation to the constraints of the technical layer.

REFERENCES

[1] C. Szyperski, “Component Software: Beyond Object-Oriented Programming”, 2nd Edition. Addison Wesley. 2002, pp. 139–150. ISBN: 978-0201745726

[2] J. Rumbaugh, I. Jacobson and G. Booch, “Unified Modeling Language Reference Manual,” 2nd Edition, Pearson Higher Education, 2004, pp. 139-150, ISBN:0321245628

[3] Object Management Group. OMG Unified Modeling Language Specification, Version 1.4, 2001. http://www.omg.org/technology/documents/formal/uml.htm, retrieved: October, 2012.

[4] R. S. Pressman. “Software Engineering, A practitioner’s approach”. 7th edition edition, Mc Graw-Hill Education, 2000, pp 603-630. ISBN: 978-0071267823.

[5] P. Herzum, “Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise”, Kindle Edition, January 2000, pp. 477-527.

[6] W. H. Inmon, J. A. Zachman, and J. G. Geiger, “Data Stores, Data Warehousing, and the Zachman Framework: Managing Enterprise Knowledge.” McGraw-Hill, 1997, pp. 105-140, ISBN 0070314292.

[7] J. Zachman. The zachman framework for enterprise architecture. http: //www.zifa.com/, 1997, retrieved: October, 2012.

[8] TOGAF Version 9. The Open Group, 2009. http://www.togaf.info/togaf9/index.html, retrieved: October, 2012

[9] T. Erl, SOA Principles of Service Design, 1 edition, Prentice Hall, July 18, 2007, pp. 211-252, ISBN: 978-0132344821

[10] S. Hussain, B. Ahmad, S. Ahmad, and S. M. Saqib, “Mapping of SOA and RUP: DOA as Case Study,” Journal of Computing, January 2010, pp. 2-4.

[11] S. W. Ambler, J. Nalbone, and M. Vizdos, “Enterprise Unified Process: Extending the Rational Unified Process”, Prentice Hall. 2002, www.enterpriseunifiedprocess.com.

[12] C. Hofmeister, R. L. Nord, and D. Soni, Describing software architecture with UML. In Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1), San Antonio, TX, February 1999. , pp. 7-12.

[13] S. White, Using BPMN to model a BPEL process, BPTrends 3 (3) (2005) 1–18.

[14] A. Kleppe, J. Warmer, and W. Bast, “MDA Explained, The Model-Driven Architecture” Practice and Promise. Addison Wesley, 2003

[15] M. Elammari and Z. Issa, “Using Model Driven Architecture to Develop Multi-Agent Systems” the International Arab Journal of Information Technology (IAJIT), Volume 10, No. 4, July 2013, pp. 19-24.

[16] M. Peltier, J. Bézivin, and G. Guillaume, “A general framework based on XSLT for model transformations,” In WTUML’01, Proceedings of the Workshop on Transformations in UML, Genova, Italy, April 2001, pp. 5-7.

[17] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J-M. Loingtier, and J. Irwin, "Aspect-oriented programming," In Proceedings of the 11th European Conference on Object-Oriented Programming, June 1997, pp. 3-5.

[18] S. Santesson, R. Housley, and T. Polk, "Internet X.509 Public Key Infrastructure Qualified Certificates Profile," http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-I pp. x-y, retrieved: October, 2012.

[19] C. Ibsen and J. Anstey. “Camel in Action”. Manning, 2010, pp. 113–122.

Value-Based Technical Debt Model and Its Application

Marek G. Stochel, Mariusz R. Wawrowski, Magdalena Rabiej
Motorola Solutions
Kraków, Poland
{marek.stochel, mariusz.wawrowski, magdalena.rabiej}@motorolasolutions.com

***Abstract*—The majority of software development today is being conducted in a value-neutral setting, where each functionality once being locked down as a part of software release is treated as equally important. This limited visibility of the real value perceived by customer inside software engineering organizational departments has significant consequences in the way the technical quality of the product is being evaluated and maintained. The relentless pursuit of efficiency in the software engineering domain requires a broader view of long-term economical consequences of any product-related decision. Technical debt typically is an internalized (engineering-based) assessment. We propose to expand the understanding and visibility of the technical debt by introducing a model driven approach to provide the means to assess the technical debt impact on perceived product quality parameters, such as codebase/design and architecture, engineering productivity, and finally the company's business return on the engineering investment. Furthermore, the case studies presented in this paper are focused on the application of the technical debt concept—how it could be identified, measured and what are the consequences of not managing it. The key principles of this concept were proved to be valid while evaluating the development of a major software system release. Finally, the need for balanced view for the technical debt management strategy is discussed, to ensure pay-off benefits are aligned with time-to-market expectations.**

Keywords—Technical Debt; Software Life Cycle; Software Economics; Software Development and Maintenance; Wisdom of Crowds; Value Based Software Engineering.

I. Introduction

The technical debt metaphor refers to the product's deficiencies, caused by shortcuts or incomplete engineering knowledge, which may speed up software development and delivery, but inevitably have their drawbacks and incur additional, delayed costs [7][8]. Unfortunately, due to its mostly internalized nature (engineering-based), technical debt is sometimes hardly recognized even by the very business unit where it was created and which is responsible for business return on the engineering investment. We claim that focus should be not only on the product/solution deficiencies, but also on how the value perceived by customers and profitability of a business unit are directly or indirectly impacted. The significant problem with managing technical debt and establishing pay-off strategy lies in the definition of the value to the customer/product/business return. Therefore, we believe that binding technical debt to the real value for the customer and hence the products return on investment (ROI) for the business (profitability) provides a better understanding of the potential consequences of not managing it. Additionally, awareness of the current and predicted product/solution condition is raised as a leading indicator.

The realization of value during software development was also the rationale for Boehm's Value Based Software Engineering concept [1][2], for which we propose a slight adjustment (marked by the dashed line) of the value realization feedback process to explicitly measure and manage technical debt (Fig. 1).

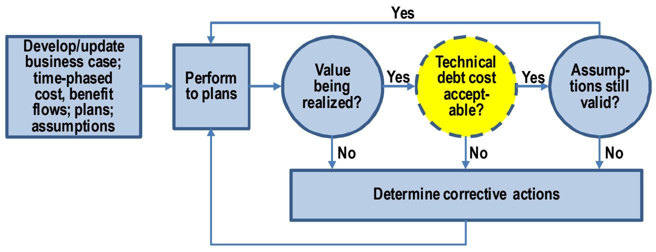

Develop/update business case; time-phased cost, benefit flows; plans; assumptions
Perform to plans
Value being realized?
Yes
Technical debt cost accept-able?
Yes
Assump-tions still valid?
Yes
No
No
No
Determine corrective actions

Figure 1. Value realization feedback process enhanced with technical debt cost assessment

As a consequence, technical debt is analyzed and discussed in this paper as a three-layered model, consisting of codebase/design debt, architecture debt, and portfolio debt. From this perspective long-term consequences for engineering and business are visible and understandable, leading to an established communication strategy across these functional areas. This aligns the effort spent in the oftentimes numerous departments of an organization during the planning and execution phases. Missing any of these critical layers of value-added granularity will provide a considerably less robust prioritized definition, and as a result, a distorted view of potential consequences of the technical debt the organization has already experienced and accumulated and may well continue to do so.

While introducing the model-based approach and the discussing underlying rationale, this paper provides also specific examples of how to calculate the technical debt value for subsequent layers, and how the total technical debt value may be coherently assessed. This discussion is supported by the results of two experiments. The first one was conducted to compare the subjective and objective ways of assessing the technical debt, the second—to understand the consequences of the missing knowledge about technical debt and related management strategy. Finally, we discuss the consequences the proposed model incurs and suggest further areas of the research and investigation.

II. Review of Literature

Although Cunningham refers to technical debt as a result of gaining knowledge during development [7][8], rather than deliberate choice, we should not neglect the fact that sometimes the latter is justifiable, as suggested by Brazier [3]. Some may suggest that technical debt should be tightly coupled only with the coding aspects that were caused deliberately, as it was referred by Fowler [12]. Furthermore he states that the question should be whether the technical debt metaphor is useful to discuss the design problems. This was supported by the technical debt quadrant concept referring to whether the debt is prudent or reckless, and running into it was deliberate or inadvertent [12]. Other authors expand the technical debt metaphor and propose the Modern Portfolio Theory [15] to calculate technical debt and the resulting associated risk. The risk based approach for software architectural decisions was also discussed by Fairbanks [10]. Following that path (usefulness of the metaphor), we propose to integrate these aspects tying technical debt to real business value in order to enhance communication and visibility.

III. The Model

Understanding technical debt and how it changes over time should be performed consistently across the organization for both the product and the portfolio. In order to achieve this goal we propose a three-layered model aligned with a typical software product development lifecycle (Fig. 2). Note that codebase/design debt is aggregated by architectural debt, and this in turn is aggregated by portfolio debt.

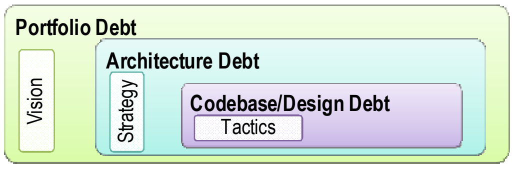


Figure 2. The three-layered model of technical debt

The rationale for this approach stems from a software development process analysis. The P-Diagram approach [17], usually employed for process risk assessment and impact analysis, was used for this purpose (Fig. 3). From this perspective technical debt can be thought of as an unintended byproduct of the software product development process as well as a noise factor of current development practices. Thus, we define technical debt as:

- Caused by the very nature of the software product development practices,
- Creating negative feedback loops, which impact all the intended outputs: product quality, engineering productivity, and organizational profitability,
- Being impacted by control factors, including—but not limited to—technical debt management strategy.

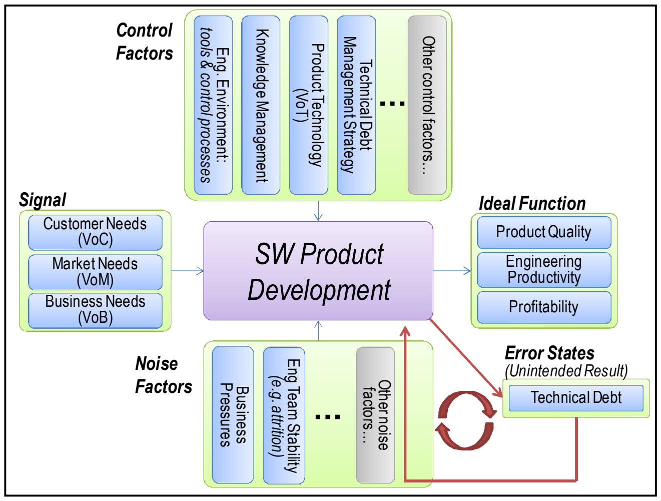


Figure 3. Technical debt in the context of software product development process

A. *Technical debt—"the process view"*

The software product development process constitutes the framework for technical debt assessment. Any omission of these aspects inevitably leads to an incomplete view of the technical aspects concerned and may result in a technical debt management strategy counterproductive from the organizational perspective.

The P-Diagram components (Fig. 3) are defined as follows:

Inputs (Signal): requests for a change in the product, driven by the customer (Voice of Customer—VoC), the market (VoM), and/or the business (VoB).

Outputs—Intended results (Ideal Function)

- Product Quality: product characteristics, as perceived by both customers and producers,
- Engineering productivity (Voice of Process, organizational capability): functionality delivered considering effort spent, may be presented as the engineering organization throughput (optimized cost, quality, schedule, scope according to baseline capability index),
- Profitability (marketing/sales): the business return on the engineering investment between and within projects. This can be measured as the percentage gross margin.

Outputs—Unintended results (Error States): the technical debt itself, which may be referred to as product and process characteristics, internally perceived by the organization (business unit).

Control factors: They comprise all factors, which influence and control the software development process, decreasing its variability thus improving predictability. For example:

- **Engineering development environment**: Tools, quality control processes, continuous software integration

- **Knowledge management/sharing**: Activities pertaining to information interchange, supporting seamless communication, and ensuring optimal work assignments
- **Technology/product management**: Activities controlling *technical inflation* [11] with focus on technology alternatives, ensuring robust testability
- **Technical debt management strategy**

Noise Factors: They are out of the direct control of the engineering organization, for example: business pressures, stability of engineering organization (attrition), and the technical debt itself.

B. *Technical debt trend—assessment and prediction*

Despite the technical debt value being calculated, it will naturally evolve. The question how to address these changes should be reflected in the technical debt management strategy. As technical debt from an unintended byproduct becomes the noise factor later (Fig. 3), the trend impacting predictability of intended results needs to be constantly monitored.

Understanding the measured impact and process capability we are able to predict whether a technical debt level is acceptable. The metric here should provide information how the product's current owner cares about maintenance and quality of the product, and how the value of technical debt changes over time according to the information provided by the portfolio and architecture teams. We also need to understand how value is being realized, and this information may be provided by a Value Based Software Engineering approach (presented earlier in Fig. 1). As a result, the current value of the technical debt may be determined in the broader context of the three-layered technical debt model. Additionally we are able to establish a threshold of how high the tech debt value might be when immediate action or a change in strategy is required and whether a technical debt pay-off is justifiable from the cost perspective.

IV. CODEBASE/DESIGN DEBT

The codebase/design debt is the most objective aspect of technical debt. It is a concept claiming that various source code quality indicators may be combined into one meaningful value easier to manage. This metric may be expressed as the effort required to change the existing codebase into an easy to maintain, well structured and testable code.

There are various aspects of both static and dynamic code characteristics that can be taken into consideration to create one meaningful technical debt metric. One can use code complexity measures provided by standard tools, such as Klocwork [16] or FXCop [13], to ensure that good coding practices are followed. Other tools may provide information that robust unit test coverage was assured. Enforcing well defined coding standards leads to a lesser number of defects introduced during integration to the version control engine, thus reducing the debt. Delaying integration of local software changes leads to a debt increase: the greater the delay, the less probability of an effortless integration. In principle, the set of measures and tools should explicitly reflect the software product's true characteristics in its intended environment.

A. *Measurement Approaches*

One approach, proposed for Java-based projects, is the Sonar approach [14]. Sonar is an Open Source Software quality management tool, which leverages the existing ecosystem of quality open source tools (for example: Checkstyle [4], PMD [20], Maven [18], and Cobertura [6]), to offer a fully integrated solution for development environments and continuous integration tools. Being accompanied by technical debt plug-in, Sonar is able to monitor static and dynamic metrics on the project and enforce coding best-practice rules, supporting defect prevention effort.

Another way to assess technical debt in codebase/design may be Wisdom of Crowds technique, which is based on the approach proposed by Surowiecki [22]. This technique has already proved successful in the prediction of defect distribution among system areas [23] and, although not mentioned explicitly, it was a major component of a proposed test case prioritization approach [24]. The software development team, if mature, can readily assess the code quality with precision and predictable results. However, in order to properly assess technical debt using the Wisdom of Crowds method, the following conditions must be met:

- Diversity of opinion—each person should have an opportunity to voice private information (even if it is his/her view of known facts),
- Independence—we have to assure that people can voice their concerns/opinions, and not repeat those of more senior, influential ones,
- Decentralization—we have to ensure the opportunity to present different perspectives, as people are able to specialize and provide conclusions based on local knowledge,
- Aggregation—the mechanism to turn private judgments into a collective opinion.

A more detailed overview of importance of these factors and rationale behind them in the context of experimental setup may be found in [23].

B. *Context of the three-layered model*

Technical debt introduced in the codebase/design phase is tightly coupled with the code being implemented. It has direct impact on the codebase cohesion, coupling, process flow, etc. Additionally, in the Agile approach, design phase is reduced to minimum. So the overall code and design quality is the responsibility of a software development team and can be assessed together using the codebase debt in the broader context of the three-layered model. Importantly, the technical debt ratio in a product is more of a metric how the historical decisions were made, so it should not be used for comparison of the organizations' maturity. Technical debt trend shows the efforts of the current software development team.

Monitoring technical debt trend in the product can give a software development team an early problem indicator, before these software problems are actually released (where technical debt ratio has significantly grown during the project). Another way to use technical debt ratio is to compare particular areas of the same code, to identify where investment is required before any new feature is implemented. In order to establish consistent technical debt management strategy the team may define a baseline ratio for codebase/design debt that reflects the acceptable debt level for the product and optimize the particular code areas to limit this debt ratio below the desired threshold. However, some questions have to be answered: *When to stop optimizing? Should the code be optimized until it is "clean" and 100% testable?* Software development team may not possess this knowledge. It is the business/architecture team that knows what the software product's roadmap and strategy is: whether it is a onetime development with no maintenance planned or mission critical software that will be maintained for years. A decision on acceptable level of codebase/design technical debt should be taken by the software development team, on the basis of a feedback loop from business/architecture to avoid sub-optimization.

C. *Experiment 1: Sonar vs. Wisdom of Crowds*

As it was mentioned earlier in this section, some tools exist, which provide the technical debt calculation, for example Sonar for Java-based products. However, more complex products or development environments may pose a serious challenge to find a consistent approach to evaluate the status of the product. This was the rationale behind comparing objective assessment provided by Sonar to the subjective measurement strategy based on the Wisdom of Crowds approach.

This experiment was conducted in one of the main components of a major system release. The developers prioritized modules according to the following criteria:

1. Where introducing changes is the most difficult
2. Which are poorly written (refactoring required)
3. Where there is high amount of latent defects

Survey results were compared to technical debt ratio calculated by Sonar. The results were analyzed for any correlation between objective measurements and subjective assessment of the technical debt value. The Pearson Correlation coefficient was calculated between each of the criteria and values provided by Sonar. In each case, correlation for the objectively measured technical debt (by Sonar) and the engineering assessment was significant. The first criterion is the closest to how technical debt is usually perceived—the consequences of earlier short-cuts of incomplete knowledge for current work (state of the product being maintained). Not surprisingly, the highest correlation with objective Sonar assessment was observed—the correlation coefficient reached 0.84 (Fig. 4). For criteria 2 and 3, the correlation was also significant, reaching 0.75 and 0.67, respectively.

There is one more aspect, which was considered in this assessment. An additional question was asked: Do you think it is necessary to pay-off technical debt in an area before any changes are introduced? The correlation with technical debt value reported by Sonar is significant (0.61); however we should also focus on the specific measurement points, where although debt is low, engineers insist on paying off the debt. This provides further insight into architectural or portfolio debt concerns.

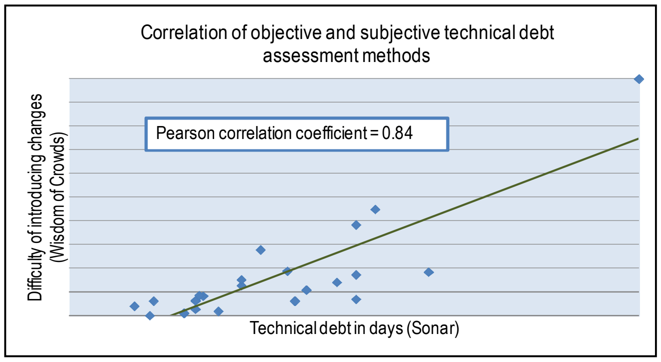


Figure 4. Correlation of objective and subjective assessment methods (difficulty of introducing changes in software modules).

These three criteria mentioned earlier by no means exhaust technical debt concept, but may provide a representative guideline for the assessment. Modules, which accumulated higher technical debt, are perceived as:

- Most difficult to have changes introduced,
- Badly written (require refactoring),
- Highly riddled by defects.

Statistically significant correlation was observed. This assessment provides us with a good overview of the quality perceived by engineering (e.g. maintainability) as well as overall quality visible to the customer (e.g. error proneness of certain system areas).

If we use all the answers as a basis for aggregation, instead of aggregating them question by question, the Pearson correlation coefficient still remains high (0.77). Therefore, we claim that having common understanding of technical debt and its consequences, the subjective measurement based on the Wisdom of Crowds method can provide reliable and consistent results in comparison with objectively defined measurements (in this case Sonar output). This fact opens a possibility of measuring the technical debt in more complex software products, even of a heterogeneous nature.

D. *Experiment 2: Naturalistic observation—technical debt management*

The following study was conducted to answer the question whether it is worthwhile to consider technical debt—not only as quality indicator for already existing code, but perhaps as a quality gate for software development activities. Unfortunately, delayed payment for what cannot be easily measured, or not knowing the potential consequences and value (real costs) usually causes technical debt to be neglected and accumulate over time, until it is very difficult and costly to address. Without a value associated to this, it is usually omitted during task prioritization. In this experiment, we wanted to understand how two similar modules behave if one is optimized against technical debt concerns and the second is not.

Typically, preparation of an experimental set-up focuses on establishing a controllable environment. Unfortunately the very activity of preparing for the experiment may impact its results. This is manifested in the social proof phenomenon described by Cialdini [5]; in our case, engineers may try to figure out what is the expected "good" result and therefore inadvertently optimize their work against it. Providing some framework before the experiment, even without any explanation about the quality of the value to be compared against, may also cause additional obstacles—such as cognitive bias, known as anchoring [25], impacting the results. In this particular case, we were in a very good position as the study was actually a *naturalistic observation* [21], i.e., no direct researcher influence was disturbing the engineering activities. The retrospective analysis of the results was performed.

We were assessing two similar database replication handler modules, which were developed for two similar database engines. The first one was taking into consideration technical debt and adhering to the rules defined by Sonar plug-in (module_1), and the second one was developed without the knowledge and proper application of that concept (module_2). Both modules were handling data synchronization and monitoring mechanisms, and the same development approach was used: pair programming during similar timeframe. Looking into technical debt as reported by Sonar, the value for module_1 was 2.5 times smaller than for module_2.

The chart presented below shows the defects found and their distribution among testing phases (Fig. 5). What can be observed is that number of defects found in module_1 was 2.6 times smaller than number of defects found in module_2. Moreover for module_1 test screening effectiveness was much better at earlier phases (80%), as only 20% escaped this activity and was found during system tests. No defects were found post release. For the second module, similar box test effort was capable of finding 8% of defects, 69% were found by system tests, and finally 23% were found post release.

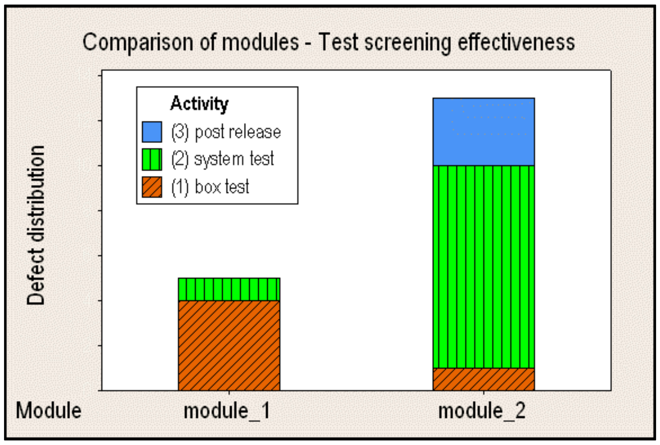


Figure 5. Comparison of defect arrival distribution between modules.

The development cost of the module written neglecting technical debt concerns was 10% smaller. But, as a result, the cost of fixing the problems in this module was almost two times higher than the initial creation effort. The results confirmed that neglecting technical debt incurs not only problems with maintenance later, but also causes higher amount of defects found both during development process and post release ones. We claim that technical debt value is a reliable prediction of future software product quality.

V. ARCHITECTURE DEBT

Numerous publications emphasize the fact that architecture is not only the macroscopic recognition of design, but constitutes the backbone of the system, orthogonal to its functionality, for example [10]. More importantly, the selection of architecture is a choice between multiple ways of implementing the system that accomplish an established set of functionalities. The choice of technology solutions—a decision made in the early phases of the project—implies a more or less explicit choice of the relevant architecture and may prevent the efficient refactoring later (inherent activity in projects using Agile approach). Therefore, reconstruction of the architecture, especially based on a specific technology, becomes much more costly. In this respect, the architecture related decisions should be the result of continuous, long-term analysis of customer needs, leading to an optimal solution selection. Moreover, changing the architecture or technology of software development needs to be a result of a complex analysis of the business process, and qualitative assessment against technical debt. So, the question arises: what metrics should be used to evaluate the architecture? One approach may be the Architecture Tradeoff Analysis Method (ATAM), which offers "utility tree" analysis [19]. Architecture technical debt assessment can be done using quality factors like: modifiability, scalability, and latency. For example, modifiability directly affects the characteristics of the cost of change, which can be treated as expression of technical debt at an architecture level. It is essential that the assessment of architecture should be done in reference to the Voice of Business. Following the conceptual model of "Portfolio Management" [15], it is necessary to assess technical debt for each artifact created in the software production process. Nevertheless, information on technical debt from the architecture level is critical because it accumulates technical feedback from the engineering teams (involved in software development and test activities), and directly takes into consideration information from the business (portfolio). Fig. 6 shows a model of decision making at the architecture level, which provides a roadmap for software product, technology, and development process based on all factors mentioned earlier. These roadmaps define engineering strategy, which is driven by value and considers organizational capability.

Technical debt in this model is presented as a trend in the cost of software changes, and its calculation should take into account the estimated cost of changes expected by the customers in time. This prediction should also include the estimated cost of change as if it were to be done in alternative architecture or technology. As a result, the architecture technical debt can be expressed as a set of characteristics representing the cost of changes for the current architectural solution and alternative ones. This assessment should constitute a feedback to the process of business analysis on portfolio level, where profitability trend of the investment should be determined.

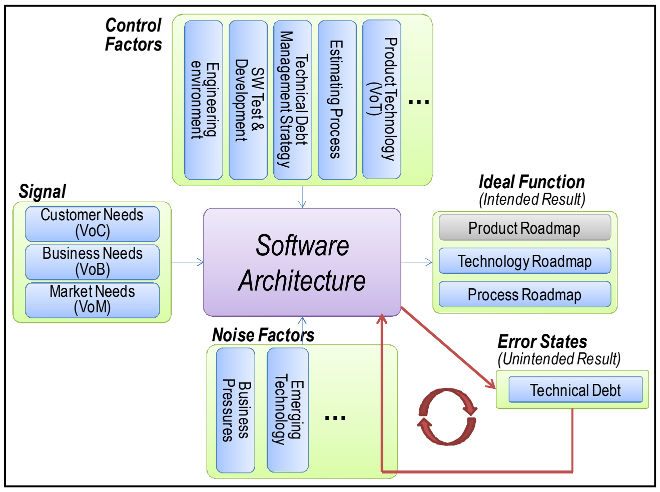


Figure 6. Technical debt limited to the context of architecture deliverables

VI. Portfolio Debt

The rationale behind portfolio debt layer (highly dependent on external factors) lies in a strategic alignment of effort spent by different organizational departments and common understanding of the goal to be pursued. Referring to the technical debt model shown in Fig. 2, a portfolio level is accompanied by *Vision* for a given product in terms of sales, market needs, technology, etc. Vision guides the strategy definition (architecture layer) and helps to align the software development effort (codebase/design layer). As a result, the profitability aspect needs to be taken into account, what can be expressed as:

$$Max\big(ROI(t)\big) \sim \frac{Sales(t) - Cost(t)}{Cost(t)} \qquad (1)$$

where *Cost(t)* is defined as:

$$Cost(t) = C_{tools,IDE,COTS} + C_{migration} + C_{SW\ changes} + C_{other} \qquad (2)$$

The selected technology should enable the organization to make an optimal decision, which aims at the highest ROI considering present and future customer needs and sufficient time to market. The result of optimizing the objective function (1) is a certain amount of observed technical debt. For example, lack of investment in new technology and refactoring at some point may just not yield the targeted ROI based upon VoB. Continued analysis of this trend can lead to a decision change, however still vulnerable to the risk of established customer goals volatility, emerging alternative technologies, or significant changes in market conditions.

For example, technical debt may show the estimate variation of the cost of change in time for the three possible technologies. Customers waiting for new and more complex features push the existing solution to the technological frontier. This prevents the effective implementation of new products or improvement of the existing ones.

Let us assume that a new technology emerges, which is very promising and the software team might be prone to immediate migration. However, this incurs a risk that expensive migration might not be justifiable from ROI perspective, as the benefits will not be realized in the expected timeframe. In such a case, a better strategy might be an evolutionary migration, where the current software is gradually modified (code refactoring) to compensate technical debt in relation to the new technology. The decision about final migration can be made when technical debt is reduced accordingly (Fig. 7).

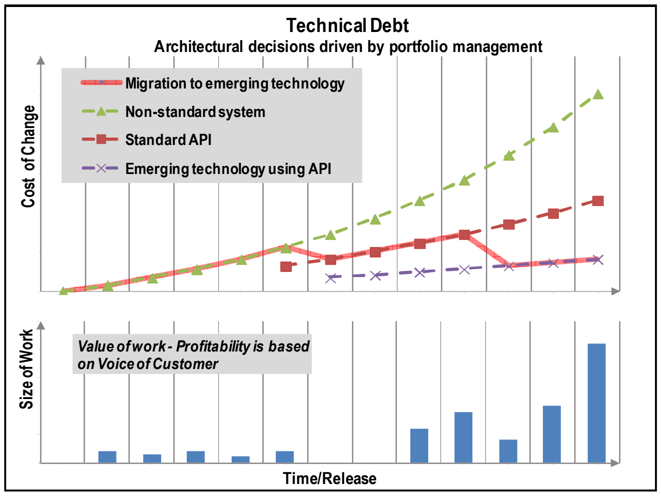
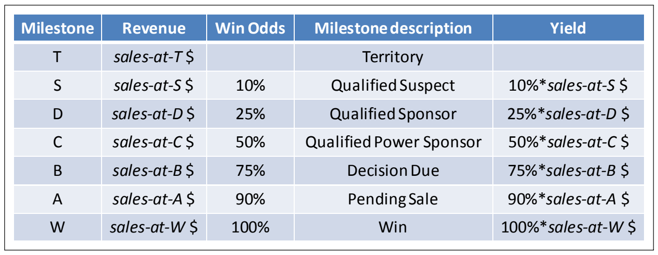


Figure 7. Technical debt driven by Voice of Customer optimized to control the refactoring of architecture

The decision about migration to a new technology is dictated by potential profitability of the estimated cost of change reflected by the Voice of Customer (VoC). The new technology may require additional investment but it will be balanced by potential financial benefits that can be calculated on the basis of VoC and VoB information. *Modern Portfolio Management* offers a statistical method to calculate this risk, making decisions about technical debt reduction economically justifiable [15].

The need for accurate assessments and quantification of future customer/market needs and its associated ROI is evident. The approach of sales prediction (proposed by Eades [9]) may be used as a reference for measuring impact of technical debt on the value perceived by customers and company profitability. Having the sales pipeline properly filled in, the potential value (yield) for a particular functionality can be estimated at any point of time (Fig. 8), as well as technical debt value associated with it. However, when assessing market opportunity, not only value (interpreted as sales prediction) should be taken into account but also how it is aligned with product long-term portfolio planning and potential market needs. In summary, the nature of the portfolio debt trend may not be linear, as the debt reveals itself as a result of certain external factors. Therefore codebase/design and architecture layers have to take some degree of assumptions and risks (even if unconsciously). This further emphasizes the importance of communication links presented in the next section.



Milestone	Revenue	Win Odds	Milestone description	Yield
T	*sales-at-T* $		Territory	
S	*sales-at-S* $	10%	Qualified Suspect	10%**sales-at-S* $
D	*sales-at-D* $	25%	Qualified Sponsor	25%**sales-at-D* $
C	*sales-at-C* $	50%	Qualified Power Sponsor	50%**sales-at-C* $
B	*sales-at-B* $	75%	Decision Due	75%**sales-at-B* $
A	*sales-at-A* $	90%	Pending Sale	90%**sales-at-A* $
W	*sales-at-W* $	100%	Win	100%**sales-at-W* $

Figure 8. Solution Selling Pipeline Milestone Chart

VII. Communication Needs

Assessing technical debt as an inherent aspect of the software development process reveals the critical need for seamless communication and organizational alignment to understand and manage all three defined layers of the technical debt model. To remove obstacles with information interchange the communication links as presented in Fig. 9 should be established.

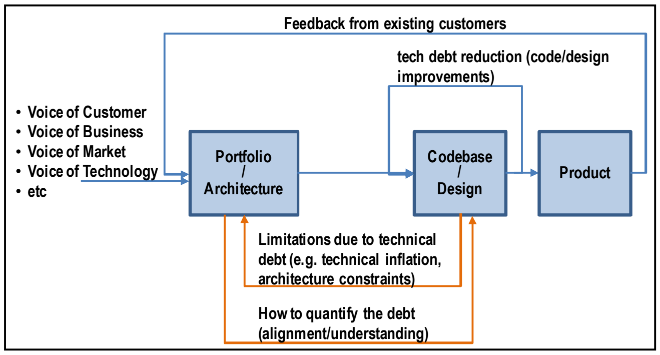
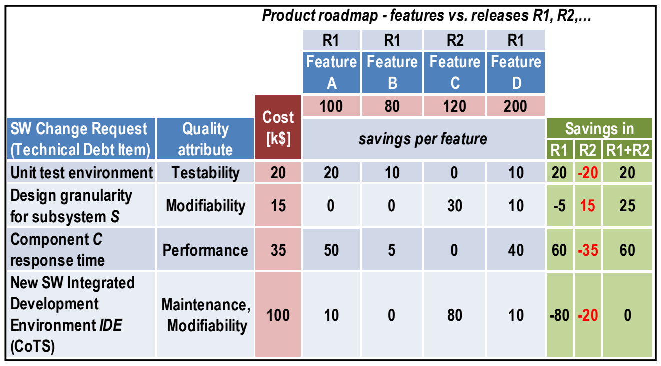

Feedback from existing customers
tech debt reduction (code/design improvements)
• Voice of Customer
• Voice of Business
• Voice of Market
• Voice of Technology
• etc
Portfolio / Architecture
Codebase / Design
Product
Limitations due to technical debt (e.g. technical inflation, architecture constraints)
How to quantify the debt (alignment/understanding)

Figure 9. Addressing communication needs related to technical debt

The discussed communication needs may be addressed using a matrix matching feature development schedule with potential benefits to be gained from technical debt pay-off. Such a matrix (table) complements the practical use of the three-layered model discussed in this paper. The sample matrix implementation is presented in Fig. 10. *SW Change Request* rows represent **Technical Debt Items (TDI)** assessed against quality attributes. For each TDI pay-off cost (column *Cost*) and savings per each roadmap feature (columns *Feature A, B, etc.*) are estimated. Each feature is assigned to a planned software release (top row: releases *R1, R2, etc.*) and has a total development cost associated with (row *Cost*). As a result different scenarios may be evaluated, for example: what is the benefit of having a particular TDI paid off in releases R1 or R2, how does it change when scope of releases R1 and R2 will be released together? Such a matrix supports also discussions on the budgeting scenarios. Let us assume that a fixed budget (450k$) was planned for a given project. The total cost of the functionality planned for releases R1 and R2 reaches 500k$.

An investment in paying off technical debt items (170k$) provides us with the savings of 275k$ only when features A, B, C, D are implemented together. Summarizing, having all technical debt items paid off, the gained benefits enable the company to develop all features within the fixed budget (3):

$$450\text{k\$} > 500\text{k\$} + 170\text{k\$} - 275\text{k\$} \tag{3}$$



			Product roadmap - features vs. releases R1, R2,...						
			R1	R1	R2	R1			
			Feature A	Feature B	Feature C	Feature D			
		Cost [k$]	100	80	120	200	Savings in		
SW Change Request (Technical Debt Item)	Quality attribute		*savings per feature*				R1	R2	R1+R2
Unit test environment	Testability	20	20	10	0	10	20	-20	20
Design granularity for subsystem *S*	Modifiability	15	0	0	30	10	-5	15	25
Component *C* response time	Performance	35	50	5	0	40	60	-35	60
New SW Integrated Development Environment *IDE* (CoTS)	Maintenance, Modifiability	100	10	0	80	10	-80	-20	0

Figure 10. Technical debt vs. portfolio assessment matrix

However, the time to market for individual features may be different, what may impact selling value. Such an assessment may lead to the decision of making separate releases, acknowledging the technical debt presence and maximizing the profits. As a result the portfolio analysis makes the roadmap change frequently, reflecting changing market conditions and embracing engineering feedback. Any external change is being reflected in ongoing synchronization of the roadmap, thus optimizing profits and ROI stemming from an engineering investment. Additionally having technical debt properly assessed and being paid off according to the long term needs, engineering organization achieves easier maintainability of the owned code base over a longer period of time.

VIII. Conclusions

The model proposed in this paper exemplifies the need for a framework, in which the technical debt has to be assessed and reveals clear rationale behind it. Moreover, various approaches to evaluation and quantification of technical debt were presented.

To answer the question *what if the cost prohibits using advanced analytical approaches, or there are no tools to support such an analysis (e.g. heterogenic solutions, auto-generated code)?*, we claim that a properly conducted subjective assessment based on the Wisdom of Crowds is capable of providing sufficient and reliable information to help in understanding the technical debt in such environments and may also support prioritization of refactoring tasks.

Technical debt prioritization may be considered in a three-layered perspective. The consequences of this model show how the defined layers (codebase/design, architecture and portfolio) depend upon and are related to each other. Neglecting any one of these dependencies may result in sub-optimization. Moreover, from a business perspective such an effort may be treated as a waste (overproduction), which may impede further improvement activities initiated by engineering.

Technical debt management strategy should be the key concern, not relentless pay-off. The understanding and prioritization of the debt may be done on a value-basis, providing a bridge between the business and engineering, and a common strategy for the technical debt management. Furthermore, experiments indirectly stressed the importance of portfolio analysis, to confirm for which software components refactoring effort may be prioritized.

Even a high technical debt value may be discarded if a particular product is close to its retirement. A different approach may be taken when the product is planned to be expanded, to constitute the baseline for other products (according to predicted market needs). Another concern is maintenance time—for long-term projects, where maintenance is scheduled for years, the debt will have a different value comparing to the solutions for products, which have a significantly shorter life. Lastly, the criticality of issues which may occur is also a very important aspect, as mission critical communications should be treated differently than cell phone games.

IX. Future Directions

Current work is focused on the validation of the three-layered technical debt model and calibration of measurement approaches among several organizational departments. Additionally there are further concerns which may be addressed by the future research. Several of them are mentioned below:

Technical debt communication:

- How to address the communication needs avoiding known "traps" in the organizations' psychological and sociological composition (as team cooperation and software development are social activities).
- How much proprietary information is to be shared with broader audience? How to ensure it is properly handled?

ROI assessment:

- What factors should be taken into account assessing technical debt impact on engineering productivity? What is their relative impact and importance?
- How to assess and address the nonlinear traits of value-based technical debt trend?

Mathematical formulae:

- How the interdependence among technical debt layers can be approximated by mathematical formula—expanding the model proposed by Guo and Seaman [15]? How it may be deployed to measure technical debt in the company with rich software legacy, with no technical debt evidence consistently tracked?

X. Acknowledgments

We want to express our gratitude to James Joseph Waskiel and Elżbieta Stochel who supported the work, relentlessly editing and reviewing the paper.

References

[1] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, Eds. Value-Based Software Engineering. Springer, 2005.

[2] B. Boehm and L.G. Huang, "Value-Based Software Engineering: A Case Study," IEEE Computer, March 2003, pp. 33-41.

[3] T. Brazier, "Managing Technical Debt," Overload Journal, Vol. 77, Feb. 2007, retrieved on Feb. 13, 2012, from: http://accu.org/index.php/journals/1301

[4] Checkstyle, http://checkstyle.sourceforge.net

[5] R. Cialdini, Influence: Science and Practice, 4th ed. Pearson Education, 2000.

[6] Cobertura, http://cobertura.sourceforge.net

[7] W. Cunningham, "Ward Explains Debt Metaphor," retrieved on Feb. 11, 2012, from: http://c2.com/cgi/wiki?WardExplainsDebtMetaphor

[8] W. Cunningham, "The WyCash Portfolio Management System," OOPSLA'92 Experience Report, Mar. 26, 1992, retrieved on Feb. 11, 2012, from: http://c2.com/doc/oopsla92.html

[9] K. Eades, The New Solution Selling: The Revolutionary Sales Process That is Changing the Way People Sell, 2nd ed. McGraw-Hill, 2003.

[10] G. Fairbanks, Just Enough Software Architecture. A Risk-Driven Approach. Marshall & Brainer, 2010.

[11] M. Fowler, "Technical Debt", Feb. 26, 2009, retrieved on Feb. 13, 2012 from: http://www.martinfowler.com/bliki/TechnicalDebt.html

[12] M. Fowler, "Technical Debt Quadrant", Oct. 14, 2009, retrieved on Feb. 10, 2012 from: http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html

[13] FXCop, http://msdn.microsoft.com/en-us/library/bb429476.aspx

[14] O. Gaudin, "Evaluate your technical debt with Sonar," Jun. 11, 2009, retrieved on Feb. 5, 2012 from: http://www.sonarsource.org/evaluate-your-technical-debt-with-sonar/

[15] Y. Guo, C. Seaman, "Portfolio Approach to Technical Debt Management," Proceeding of the 2nd working on Managing technical debt (MTD '11), May 2011, pp. 31-34

[16] Klocwork, http://www.klocwork.com

[17] E. Maas and P.D. McNair, Applying Design for Six Sigma to Software and Hardware Systems, Prentice Hall, 2009.

[18] Maven, http://maven.apache.org

[19] R. L. Nord, M. R. Barbacci, P. Clements, R. Kazman, M. Klein, L. O'Brien, and J. E. Tomayko, Integrating the Architecture Tradeoff Analysis Method (ATAM) with the Cost Benefit Analysis Method (CBAM). Carnegie Mellon University, 2004.

[20] PMD, http://pmd.sourceforge.net

[21] J.J. Shaughnessy, E.B. Zechmeister, and J.S. Zechmeister, Research Methods in Psychology, 5th ed. McGraw-Hill, 2000.

[22] J. Surowiecki, Wisdom of Crowds. Why the Many are Smarter Than the Few. Abacus, 2005.

[23] M. G. Stochel, "Reliability and accuracy of the estimation process. Wideband Delphi vs. Wisdom of Crowds," Proceedings of 35th Annual IEEE International Computer Software and Applications Conference, Jul. 18-21, 2011, Munich, Germany, pp. 350-359.

[24] M. G. Stochel and R. Sztando, "Testing optimization for mission-critical, complex, distributed systems," Proceedings of 32nd Annual IEEE International Computer Software and Applications Conference, Jul. 28-Aug. 1, 2008, Turku, Finland, pp. 847-852.

[25] A. Tversky and D. Kahneman, "Judgement under Uncertainity: Heuristics and Biases," Science, New Series, Vol. 185, No. 4157, Sep. 27, 1974, pp. 1124-1131.

An Advanced Interactive Visualization Approach for Component-Based Software: A User Study

Jaroslav Šnajberk, Lukas Holy, Kamil Jezek and Premek Brada
Department of Computer Science and Engineering, Faculty of Applied Sciences,
University of West Bohemia, Pilsen, Czech Republic
{snajberk, lholy, kjezek, brada}@kiv.zcu.cz

***Abstract*—We present a user study that compares user performance during architectural analysis using two different approaches to visualizing component-based applications structure: AIVA (Advanced Interactive Visualization Approach) and UML (Unified Modeling Language). AIVA is a research proof of concept focused on extensive use of interactivity in visualization of structure. UML is an industrial standard in the field of software visual modeling. Participants of this user study tested how fast they could perform six basic tasks which were selected so as to gain understanding of component dependencies in a medium-sized OSGi application. The results show that AIVA helps to find answers on average three times faster than UML. The study and its results provide a quantitative support for our hypothesis that the structure of component-based applications should be visualized interactively using dedicated notation rather than in static UML diagrams to improve understanding of the whole application architecture.**

***Keywords*-software visualization; component; UML; user study; interactivity.**

I. Introduction

Component-based software engineering is a modern software discipline that encapsulates objects into black box structures called components to maximize re-usability and to improve logical composition of the application. Although the concept of a component is not new, approaches able to fully visualize the structure of these applications are lacking. However, such visualization is important for engineers to thoroughly understand the applications.

AIVA (Advanced Interactive Visualization Approach) [1] is our research visualization approach that is built on the idea that interactivity is beneficial for the study of structure and that different interactive techniques should be adopted to maximize the impact of interactivity. This idea is in contrast with the commonly used static approach of standardized UML [2] and its component diagrams. Such an interactive approach should be able to describe any component in at least the same level of detail as UML can, without introducing more occlusion in the diagrams. Interactivity should lead to a simplification of structure visual representation, especially when combined with techniques that are able to provide all details to the user just when he needs them.

To validate the approach and assess its practical implementation, we have performed a controlled user study focused on performance (time required to provide a correct answer) during architectural analysis tasks. This paper describes the whole study in terms of its design and results, together with necessary background information on the AIVA approach and further discussion of the results.

A. Current State of the Art

The Unified Modeling Language (UML) provides three groups of diagrams to model both static and dynamic features of software [2], including the component diagram. The notation used in this diagram, however, captures components only on a general level, while their details differ greatly between component models, both commercial and research ones – OSGi [3], EJB [4], SOFA 2 [5] and others.

A component model can further introduce its own unique features like behavior protocols [6] or hierarchical decomposition. To this end, UML supports user-defined profiles that are able to model most – but not all – of the features satisfactorily; although, their visual representation tends to be difficult to read. An opposite approach is sometimes used, namely to augment a given component model with its own visual notation as, for example, in the case of SaveCCM [7]. A brief study of the currently used approaches and tools is provided by Holy [8].

Even when a UML profile is complemented with a tool providing good visualization of the profile, an essential problem still remains: that the structure of component applications tends to be more complex than most class structures. Contracts between components often involve several features like event queues, provided interfaces, imported packages, etc., leading to many diagram lines per component pair. The resulting structure is therefore more complex and harder to read.

B. Related Work

Research efforts related to our visualization of software architectures fall in two broad categories: displaying the structure and dealing with interactivity. The efforts to display these structures are most commonly oriented on extensions of the UML itself, without taking interactivity under consideration; while Dumoulin [9] introduces layers to support multiple views in one diagram, Byelas [10] suggested the use of colored areas of interest to improve orientation in classical UML component diagrams. Other works are even less relevant to the work presented in this paper.

Telea's [11] work on (interactive) visualization of component-based software is generic and mostly similar to the work presented in this paper, but it does not provide many details about components themselves and can hardly be compared with UML. Wettel and Lanza [12] visualize software as cities – they define three dimensions: two are used as the base of a building and the third is used as the height of a building. This approach could be easily used on component software, but again it does not provide details needed to get full comprehension of the structure.

Interactivity should help primarily with the creation of a mental model, so that one will be able to reason about the architecture and make decisions. It is important to lighten the cognitive load, namely hide unnecessary details, as Ric Holt highlighted with several examples in [13]. The importance of interactivity for the ability to make decisions about a mental model is mentioned in several studies, one of which is Meyer et al. [14]. He goes even further and defines a new science of visually enabled reasoning, implying that interactivity is its key enabler.

Finally, works concerned with the evaluation of new information visualization approaches are also related, as we studied them prior to designing our own user study. Therefore, the work of Camilla Forsell [15] should be highlighted, as it provides a clear guide for similar studies. Laidlaw [16] uses a similar comparative study of performance on 2D vector field visualization methods. Evaluation of software visualization was also described by Sensalire et. al. in [17], cooperating with Telea, mentioned earlier.

C. Structure of the Text

This paper first describes AIVA and UML in Section II in order to provide sufficient understanding of them. Section III describes the design of the presented user study in detail. Technical information related to the hardware and specific software technologies used are provided in Section IV.

Section V presents the results of the performed user study, whose conclusions are then discussed in Section VI. Finally this paper is concluded by Section VII with a summary of findings.

II. Overview of Compared Approaches

This section discusses both AIVA and UML, where the former is described in greater detail, as it is an experimental approach which is not commonly known. UML is touched only briefly, being a common standard. Special care is given to the techniques and features that were used by participants in the study. Both approaches were already compared in a case study [18] which focused solely on the readability of the diagrams in these two approaches – compared to their performance testing, which is the subject of this paper.

A. AIVA

The Advanced Interactive Visualization Approach (Project page: `http://www.assembla.com/spaces/comav`) uses an oriented graph to visualize components and their relations. Its visual notation is unified for all component models (including the hierarchical ones) and is partially similar to that of a UML class diagram. However, it differs in the representation of the list of elements inside the component "box" – AIVA prefers hierarchical ordering based on the characteristics of elements. The resulting groups of elements stand each for itself to provide better orientation. A thorough description of AIVA was published in [1], so below we discuss mainly its interactive features that can help increase user performance, in line with the goal of this paper.

The navigation and explore features that are most frequently used when studying the diagram accommodate these interactive techniques: scrolling, zooming, panning, outline view and quick search (move the view on the diagram to show the component selected in the project overview). AIVA improves the zooming technique: when zoomed out, standard zooming simply changes the scale of the standard output, which makes the details of a component unreadable and thus useless. AIVA in this case hides the details and enlarges only the information that is always important – the name of the component. This change should improve orientation in the diagram.

Highlighting helps to further improve orientation in the diagram. Almost any interaction will highlight the subject of interaction with bright, easily distinguishable color in both the diagram and the overview. One click on a component highlights the component; one click on a connection line highlights the line together with both components connected by the line and the elements involved in this relation. Double click on an element highlights all connection lines related to this element as well as all connected components.

Decreased complexity of the diagram is yet another way to improve diagram readability; AIVA uses information-hiding techniques that help to achieve this goal, together with details on demand (discussed next). A very effective complexity reduction method is to collapse the connection lines, since the reduction of the number of lines in the diagram makes it less complex and easier to read; in AIVA, therefore, there is only one connection line between each pair of connected components. In addition, the information labels identifying the connection type and connected elements are by default hidden. Finally, interfaces and other component interface features are not diagram nodes and the structure is therefore less complex.

Details on demand are used to access the hidden information. When one clicks on the connection line, an information box appears showing a detailed list of all connected elements; therefore, no information value is lost. Additional details are also provided when the user clicks on the component, revealing information about, e.g., its version, license, symbolic name, etc. Similar information about every component element is accessible by hovering over its name.

Other features that are supported by AIVA are used mostly to offer a different view on the same thing. However, these features are not in the scope of this paper, so we mention them only to present AIVA completely:

1) Grouping and filtering, based on the characteristics of the elements. Support for user defined sets of groups.
2) Conditional formatting able to work with component-

model-specific information and various secondary data that are normally hidden.
3) Reconfiguration features, namely change of representation of components or diagram layout.

B. UML

Unified Modeling Language on its own is a static approach in which the diagrams look the same on a computer screen and on paper. For full component modeling support, user profiles are necessary, which considerably shortens the list of available tools. The standard interactive techniques used across all these UML tools are navigation ones – scrolling, panning, zooming, and overview. The overall usability of UML component diagrams is related more to the specific features of concrete tools than to the UML notation itself. Therefore, tools with some "added value" should be selected to objectively investigate UML usability.

IBM Rational Software Architect (RSA) can be considered as such an advanced UML tool. Besides all standard navigation features, it supports some advanced ones that allow users to manipulate the diagram, like changing the layout of nodes, changing the line routing and modifying the look of components and interfaces. Added value is in its "*properties view*", displayed at the bottom of the screen. This view shows all the details about components and relations and, most importantly, it can be used to navigate to related components. For example, the "*Relationships*" tab shows a list of all elements that use or are used by the component. This list clearly specifies which kind of relation is used and which component is related. The name of the related component has the form of a link, so the user can easily find more information about it.

III. Design of the Study

This section provides the details about the goal and mechanics of the performed user study.

A. Goal of the Study

The main goal of this study is to evaluate the performance of users during architecture analysis in two different approaches – UML and AIVA. The hypothesis tested was: "It is faster for engineers to study the structure of component-based applications interactively rather than using static diagrams." The null hypothesis was that studying component-based applications' structure is comparably fast when using interactive and static visualization methods.

The results of this user study can therefore help in finding out to what degree interactivity is useful. These questions are important because the level of interactivity used in AIVA is high and could negatively affect the user's performance while he collects some more detailed information, specifically because a lot of this information is hidden and revealing it requires user interaction.

The set of tasks used in the study simulates the activities performed during one step of architecture analysis. These tasks are focused on collecting knowledge about one component – its features, dependencies and overall context consisting of related components. When analyzing the whole architecture, one needs to repeat this step for most of its components. The concrete set of tasks is discussed thoroughly further below.

B. Profile of Participants

The structure of component-based applications is studied by software engineers who work on these applications. They have a deep knowledge of components and UML to be able to understand the diagram presented. Such people are hard to get to participate in a user study that takes at least one hour; thus we decided to ask our colleagues to participate. Use of academics and Ph.D. candidates was encouraged by Sensalire et. al. in [17], based on their lessons learned: "They are willing to take part in studies for the sake of gaining knowledge and may require less or no additional motivation", while still clearly being professionals in the field of software engineering.

The participants were young software engineers selected from different groups at our department. They were confident in most UML diagrams; their knowledge of component diagrams was tested specifically before participation. Most of the participants use components on a daily basis and the rest were briefly trained before the study. All of the participants were confident in the required basics of component-based development before undertaking the questions.

All participants were also trained in both tools that were used to test the two approaches: UML/RSA and the AIVA research prototype. First, the tool was presented to them. We shared our working experience on how to get various types of information effectively. Then every participant had unlimited time to test all types of tasks that he would encounter. Participants were handled individually and guidance was given when asked. The training ended when the participant felt confident and able to perform all types of tasks used in this user study.

C. How the Study Was Performed

The process repeated with every participant was as follows:

1) Verification of knowledge
2) Training in Tool 1
3) Performing all tasks in Tool 1
4) Training in Tool 2
5) Performing all tasks in Tool 2

Verification of UML and component knowledge took about 20 minutes to ensure the participant's expertise. Training in both tools took about 40 minutes, until the participant felt confident. All tasks were performed in under 10 minutes, because the tasks were quite short.

All participants were observed for the whole time of the study and there were no interruptions nor any advice while they searched for the answers to a given task. The time was measured from the moment the question was read and understanding was confirmed by the participant, to the point when a correct and full answer was given. We required users to visually verify the information as part of the answer, i.e. to pinpoint the found information in the diagram.

All participants were divided into two groups of the same size. Group #1 started with AIVA and group #2 with UML. The set of tasks was identical for both approaches; however, the tested approaches provide such different diagrams that the participants could not gain an advantage by performing the same tasks again with the other tool. Moreover, training in the second tool was performed after testing first tool to distract participants from the tasks. During the training phase, similar tasks were practiced by the participants on different components in a completely different application to ensure that they could not gain any knowledge related to the test tasks.

The study was intentionally designed this way to prevent bias of participants which could erode the validity of results.

IV. Technical Background

This section provides technical details required to recreate the user study described. An overview of the OSGi component model and CoCoME application used is provided, after which the set of tasks (bound to the application) is described in its final form. Finally, details about hardware are presented to provide the whole picture.

A. OSGi Component Model

OSGi [3] is a multi-platform Java component solution focused on dynamic deployment and assembly of components. OSGi components are black-boxes; their nature and features are described in a manifest file. The communication between components is realized by *services* which are implementations of interfaces, thus keeping their black-box nature.

Apart from services, both provided and required, these components can depend on *Imported packages* and other components – *Required bundles*. Thus there are three completely different types of relations in the OSGi environment. Complex analysis of OSGi and its key features is described in [19], where the author suggests an OSGi profile for the ENT meta-model which is used by AIVA.

After a thorough study of this profile, we developed a similar one for UML, so it can model the same information as the ENT meta-model. We already described this profile in our previous study [18], which was more concerned with the question of how this information is visualized.

B. CoCoME

CoCoME stands for Common Component Modeling Example [20]. It models an information system for supermarket chains and is used for the purpose of comparing different approaches to component-based software development. It has been officially implemented in 13 component models; we created an OSGi implementation (`http://www.assembla.com/spaces/comav/documents/tag/CoCoME`).

The CoCoME application consists of three main parts. First is a Cashdesk part, which contains the cashdesk, including barcode scanners, credit card readers, etc. The second part is a store backoffice server and a store client. Finally, the chain part consists of an enterprise server and client applications. CoCoME is assembled from 37 components using 12 interfaces, thus representing a medium-size application. The complete diagram of the whole application is accessible in both AIVA and UML forms on our project page (`http://www.assembla.com/spaces/comav/wiki/Comparison_of_AIVA`).

C. Tasks

The tasks described below were tested on the CoCoME application. Because of this, they are formulated directly for its components; however, they can be easily generalized. The tasks are basic and contribute to answering one complex question: how is a particular component (*cocome-osgiDS-store.impl*) integrated in the CoCoME application. One has to find out what this component offers and requires and uncover its ties to other components, simulating the activities performed during one step of the architecture analysis. The most complex component of the application was chosen for these tasks.

The tasks were identified based on our experience with the structure of component-based applications and hints obtained during interviews with several software engineers from local software companies. The exact wording was then designed to cover all aspects of one concrete component.

Q1. Which packages are imported by component *cocome-osgiDS-store.impl*?
Q2. Which elements of component *cocome-osgiDS-store.impl* are unused (i.e. have no relationship)?
Q3. Which components use the service *StoreIf* provided by *cocome-osgiDS-store.impl*?
Q4. Which components depend on *cocome-osgiDS-store.impl*?
Q5. Which components are required by *cocome-osgiDS-store.impl*?
Q6. Which elements does *cocome-osgiDS-store.impl* need from *cocome-osgiDS-data*?

D. Hardware

Computer hardware did not influence the results of the study since the bottleneck for performance was the user's ability to interact and read the information from the diagram. However, to provide complete technical background, here are the specifications of the testing computer: Intel Core i5 3Ghz CPU, 4GB DDR3 1066Mhz RAM, 7200RPM HDD and, most importantly, 24" LCD with 1920x1080 resolution. This computer proved to be fast enough to ensure a comfortable working experience and the screen resolution was sufficient for visualization purposes.

V. Results

This section provides detailed results of this study for each approach and their comparison. As the reader may note, the results differ greatly depending on the participant. This was caused by individual perception, orientation abilities and how quickly they were able to click the mouse. (A lot of attention was paid to preparing all of them thoroughly, see Section III.)

Twelve users participated in the study, identified as A-L in the tables below. The last two participants (K and L) are co-authors of this paper and mentored the rest of the participants. Our performance is listed in the results to show the peak performance of the tasks, as we knew exactly what we were looking for and how to retrieve this information. We followed the same rules as any other participants and accepted the answer only after visual confirmation. Our results are not used in later statistics. The Q1-Q6 identifiers in the result tables refer to the tasks from Section IV-C.

Statistical tables present important statistical values calculated per question. The "Total" column provides the sum of values per row, calculated from results data with millisecond precision (thus, the simple sum of provided values may differ).

A. Performance in AIVA

Results of all participants are presented in Table I, while statistical values are in Table II. The biggest strength of AIVA was the search for unused elements (Q2), as it provides the answer immediately and most participants were able to read it right away; however, a few of them did not at first understand this information. The biggest weakness was finding the dependent components (Q4), because most of the participants forgot to read the type of arrow indicating the type of the connection and got a little confused – searched for the answer elsewhere before they found it. As a result, the time needed for a correct answer was longer, as we waited for them to solve the problem themselves.

TABLE I
RESULTS OF USERS IN AIVA [MIN:SEC].

ID	Q1	Q2	Q3	Q4	Q5	Q6	SUM
A	0:46	0:11	0:20	0:32	0:33	0:28	2:50
B	0:16	0:09	0:42	0:25	0:35	0:25	2:32
C	0:22	0:08	0:18	1:02	0:23	0:27	2:40
D	0:51	0:33	0:20	0:41	0:44	0:50	3:59
E	0:12	0:10	0:11	1:20	0:22	0:10	2:25
F	0:25	0:09	0:23	0:27	0:31	0:38	2:33
G	0:23	0:22	0:19	0:40	0:23	0:29	2:36
H	0:29	0:06	0:16	0:37	0:29	0:16	2:13
I	0:07	0:04	0:08	0:24	0:16	0:24	1:23
J	0:15	0:24	0:17	0:31	0:25	0:17	2:09
K	0:08	0:03	0:08	0:13	0:19	0:10	1:01
L	0:12	0:04	0:08	0:15	0:17	0:11	1:07

TABLE II
STATISTICS OF USERS IN AIVA.

Measure	Q1	Q2	Q3	Q4	Q5	Q6	Total
Avg	0:24	0:13	0:19	0:39	0:28	0:26	2:32
Median	0:22	0:09	0:18	0:34	0:27	0:26	2:18
Min	0:07	0:04	0:08	0:24	0:16	0:10	1:09
Max	0:51	0:33	0:42	1:20	0:44	0:50	5:00
Std dev.	0:13	0:08	0:08	0:17	0:07	0:10	N/A

B. Performance in RSA

Results of all participants are presented in Table III, while statistical values are in Table IV. The biggest strength of RSA was looking up service clients (Q3), as it provided the answer almost immediately, while the biggest weakness was finding the dependent components (Q4) due to worse RSA support in connecting components through ports. Participants gave a stable performance as they are familiar with UML notation. The graphical user interface of RSA is more user friendly, which also helped users in orientation. Often, Participants were delayed by accidental clicking on the connection line – RSA had centered the screen on it and they lost the context of the studied component.

TABLE III
RESULTS OF USERS IN RSA [MIN:SEC].

ID	Q1	Q2	Q3	Q4	Q5	Q6	SUM
A	1:04	2:40	0:12	2:56	2:43	2:10	11:45
B	1:22	2:06	0:11	1:40	2:39	2:08	10:06
C	1:13	2:30	0:25	1:58	2:49	2:14	11:09
D	1:19	1:30	0:27	1:23	2:25	2:10	9:14
E	0:36	0:59	0:17	0:43	1:41	1:00	5:16
F	1:24	1:05	0:21	1:01	1:40	2:49	6:12
G	0:43	0:30	0:07	0:39	1:46	1:20	5:05
H	0:46	1:14	0:09	0:54	2:17	0:52	6:12
I	0:52	0:34	0:08	0:28	1:00	0:36	3:38
J	0:59	1:06	0:21	0:40	1:48	1:28	6:22
K	0:32	0:42	0:10	0:34	1:07	0:46	3:51
L	0:39	0:52	0:11	0:25	0:54	0:39	3:40

TABLE IV
STATISTICS OF USERS IN RSA.

Measure	Q1	Q2	Q3	Q4	Q5	Q6	Total
Avg	1:01	1:25	0:15	1:14	2:04	1:40	7:42
Median	1:01	1:10	0:14	0:57	2:02	1:48	7:14
Min	0:36	0:30	0:07	0:28	1:00	0:36	3:17
Max	1:24	2:40	0:27	2:56	2:49	2:49	13:05
Std dev.	0:16	0:43	0:06	0:43	0:33	0:41	N/A

C. Comparing the Results

Apart from the measured values, a useful piece of information resulting from this study is the performance ratio of AIVA to RSA. Comparing this ratio for every participant can bring more insight than comparing the global numbers. The highest ratio was for participant A, who was 4.15 times faster in AIVA than in RSA. The lowest ratio was for participant G, only 1.96 times faster in AIVA than in RSA. The rest of the participants were within these extremes; however, they were on average 3 times faster in AIVA – the average test time in RSA was 462 seconds, compared to 152 seconds in AIVA.

The average results are compared with the standard deviation in Figure 1. Normal distribution says that 70% of users would fall within these limits. This figure clearly shows that AIVA was faster in tasks Q1, Q2, Q4, Q5 and Q6, that is in 83% of cases, while it was slower in task Q3, which was the strongest task in RSA.

Figure 2 comprehensibly presents minimum, maximum and median values in a comparable way, so that these values can be conveniently studied in one place.

Lastly, Figure 3 contrasts the longest times measured in AIVA with the shortest times measured in RSA. This figure

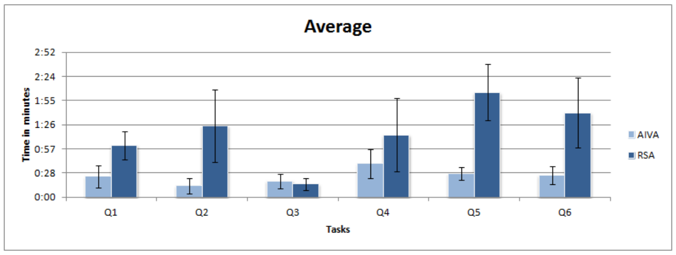


Fig. 1. Comparison of average results.

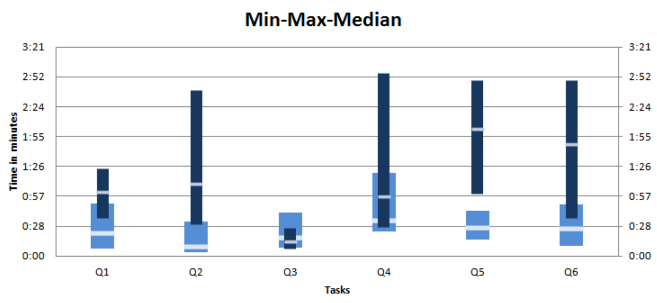


Fig. 2. Minimum and maximum values with median marked.

presents a different look at these extreme values, showing how a poor use of AIVA compares with best-performing RSA users. The numbers show that even in this case, AIVA is comparable in two thirds of tasks; RSA has its best results significantly faster in tasks related to service dependencies.

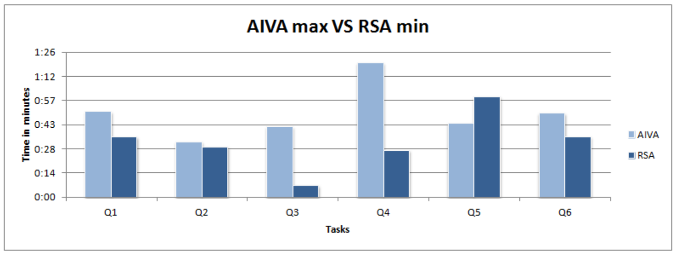


Fig. 3. Comparison of maximal AIVA results with minimal RSA results.

Two scenarios were tested to compare results in more depth. The first scenario tested if users who perform best in UML are also very fast in AIVA. All participants were ordered from fastest to slowest in both AIVA and UML and their order was compared. Four participants from the UML top 5 were also in the AIVA top 5. The notes on the remaining participant who did worse in AIVA showed that he was overconfident because of his expertise in UML. He started the test in AIVA and had to think longer about how to finish the given tasks. As a result of this scenario, it is possible to conclude that good analysts will benefit from using AIVA.

The second scenario tested where in the distribution are users who were really slow in UML. All participants were ordered from fastest to slowest in UML and, also, their performance ratio between AIVA and UML was ordered from highest to lowest. Four participants from the bottom 5 in UML were among the top 5 users overall in the performance ratio. (The one who was not in the top 5 also had significantly worse results in AIVA – he also felt confident in AIVA although it turned out he should have kept training for some more time.) The five slowest participants in UML were on average 3.5 times faster in AIVA, while the top 5 UML participants were on average only 2.5 times faster. It is possible to conclude that casual users of UML will benefit the most from using AIVA.

VI. Discussion

The previous section provided results of this user study and compared them. By studying these results it is possible to conclude several things on which this section comments. It is important to realize that AIVA did trade-off some characteristic features of UML to get these better results. The most important trade-offs are that AIVA can not be used on paper; it is usable only for component-based applications and, moreover, only for the structure of these applications.

A. Measures of AIVA Performance

First of all, the time required to finish a task in AIVA is more consistent – the standard deviation in RSA (48 seconds) is almost four times higher that that of AIVA (14 seconds). The reason is that UML itself and thus also RSA has different levels of recognition and therefore handling for different elements – work is really fast with some (tasks that depend mostly on interfaces) but slow with others (tasks that depend mostly on ports). On the other hand, AIVA provides the same level of support for all types of elements on both visual and interaction levels.

The previous conclusion leads to a more important one – the choice of tasks is not so important for AIVA as it is for UML. In other words, AIVA should be able to provide stable user performance for any task set, in any component model. In contrast, a user's performance in UML depends on the selected tasks, the selected UML tool and the component model.

Task Q4 (searching for clients of the studied component) was the slowest one in both approaches. The reason is that the component was widely used by many other components in the CoCoME application. Therefore, it took time to find them all.

One should also look at the fastest task for UML – Q3, which worked with dependencies of interfaces in a tool which is able to list all these dependencies at once. This can be recognized as a best case UML scenario, with the fastest time of 7 seconds, while AIVA required 8 seconds. Median values are 14 seconds for UML and 18 seconds for AIVA – that is, AIVA is 28% slower. The worst case scenario for UML would be Q5, which worked a lot with dependencies of packages (ports). The fastest user finished this task in 16 seconds in AIVA but took 60 seconds in UML. Median values are 27 seconds in AIVA and 122 seconds in UML – UML is 450% slower. These numbers again indicate that AIVA would be faster in any mixed task set.

From the results provided, it is thus possible to conclude that the level of interactivity used in AIVA is useful and that the null hypothesis is not valid. Interactivity helped AIVA to provide a simpler diagram, so users could orientate themselves easier, and the overall user performance was better even when the interaction was required to gather the necessary information.

B. Participant Opinions

This user study confirmed that AIVA is faster than UML, but we also asked the participants a few subjective questions after they finished:

1) Do you consider the AIVA or UML diagram clearer? Why?
2) Which was more comfortable to work in, AIVA or RSA? Why?
3) Do you have other suggestions?

All participants answered that AIVA provides a clearer diagram that is more readable and understandable. They mentioned these reasons: fewer lines, hidden details on zoom, all information in one place and very readable structure of elements.

One participant felt more comfortable in RSA because labels were always visible and a click on lines centered the screen. The rest of the participants felt more comfortable in AIVA, giving these reasons: clearer GUI, packages shown inside components, much faster operation, information easier to reach, better interactive overview. These participants also did not like the RSA feature that centered the screen after they clicked on a line because it happened often by accident and they lost thecontext.

VII. Conclusion and Future Work

This paper described a complete user study that compared performance in component architecture analysis tasks using two different component visualization approaches – AIVA and UML. AIVA is implemented as a research proof of concept, while UML is supported by a lot of commercial tools. Rational Software Architect was chosen to represent these tools because of its ability to easily study relations, which was most needed in this experiment.

The data obtained show that users working interactively (i.e. in AIVA) are approximately three times faster than those using UML. In only one of six tasks was UML faster, while AIVA performed better in the remaining 5/6 of tasks. The discussion section above provides insight into the reasons and on how different tasks could affect the overall performance.

Results of this user study therefore confirm that advanced visualization of component-based application architecture using a high level of interactivity is beneficial for users. Even the increased interaction required to uncover hidden information does not introduce significant problems.

Our future work will evaluate if AIVA can be used by software engineers in real-life scenarios. In particular, we want to test if it helps users to understand the application structure starting from the beginning of the learning process to the point where they have sufficient insight to make decisions and answer complex questions.

Acknowledgment

The work was supported by the University of West Bohemia grant SGS-2010-028 Advanced Computer and Information Systems.

References

[1] J. Snajberk and P. Brada, "Interactive Component Visualization," in *Proceedings of International Conference on Evaluation of Novel Approaches to Software Engineering*. SciTePress, 2011, pp. 218–225.

[2] Object Management Group, "UML Superstructure Specification," Object Management Group, OMG Specification formal/2009-02-02, 2009.

[3] OSGi Alliance, "OSGi Servise Platform Core Specification," OSGi Alliance, OSGi Specification, 2009.

[4] Sun Microsystems, Inc., "Enterprise JavaBeans(TM) Specification," Sun Microsystems, Inc., SUN Specification, 2001.

[5] T. Bures, P. Hnetynka, and F. Plasil, "SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model," in *SERA*. IEEE Computer Society, 2006, pp. 40–48.

[6] F. Plasil and S. Visnovsky, "Behavior Protocols for Software Components," *IEEE Trans. Software Eng*, vol. 28, no. 11, pp. 1056–1076, 2002.

[7] H. Hansson, M. Akerholm, I. Crnkovic, and M. Tarngren, "SaveCCM - A Component Model for Safety-Critical Real-Time Systems," in *EUROMICRO*. IEEE Computer Society, 2004, pp. 627–635.

[8] L. Holy, J. Snajberk, and P. Brada, "Evaluation Component Architecture Visualization Tools," in *Proceedings of International Conference on Information Visualization Theory and Applications*. SciTePress, 2012.

[9] C. Dumoulin and S. Gerard, "Have Multiple Views with one Single Diagram! A Layer Based Approach of UML Diagrams," Institut National de Recherche en Informatique et en Automatique, Universite des Sciences et Technologies de Lille, Research report INRIA-00527850, October 2010.

[10] H. Byelas, E. Bondarev, and A. Telea, "Visualization of areas of interest in component-based system architectures," in *Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 160–169.

[11] A. Telea and L. Voinea, "A Framework for Interactive Visualization of Component-Based Software," in *Proceedings of the 30th EUROMICRO Conference*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 567–574.

[12] R. Wettel and M. Lanza, "Visualizing software systems as cities," in *In Proc. of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. Society Press, 2007, pp. 92–99.

[13] R. Holt, "Software Architecture as a Shared Mental Model," in *Proceedings of International Workshop on Program Comprehension*, 2002.

[14] J. Meyer, J. Thomas, S. Diehl, B. Fisher, and D. A. Keim, "From Visualization to Visually Enabled Reasoning," in *Scientific Visualization: Advanced Concepts*, ser. Dagstuhl Follow-Ups, H. Hagen, Ed. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010, vol. 1, pp. 227–245.

[15] C. Forsell, "A guide to scientific evaluation in information visualization," in *Information Visualisation (IV), 2010 14th International Conference*, july 2010, pp. 162 –169.

[16] D. H. Laidlaw, J. S. Davidson, T. S. Miller, M. da Silva, R. M. Kirby, W. H. Warren, and M. Tarr, "Quantitative comparative evaluation of 2d vector field visualization methods," in *Proceedings of the conference on Visualization '01*, ser. VIS '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 143–150.

[17] M. Sensalire, P. Ogao, and A. Telea, "Evaluation of software visualization tools: Lessons learned," in *Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009. 5th IEEE International Workshop on*, 2009, pp. 19 –26.

[18] J. Snajberk, L. Holy, and P. Brada, "AIVA vs UML: Comparison of Component Application Visualizations in a Case-Study," in *Proceedings of 16th International Conference on Information Visualization*, 2012.

[19] L. Valenta and P. Brada, "OSGi Component Substitutability Using Enhanced ENT Metamodel Implementation," Department of Computer Science and Engineering, University of West Bohemia, Tech. Rep. DCSE/TR-2006-05, 2006.

[20] A. Rausch, R. Reussner, R. Mirandola, and F. Plasil, *The Common Component Modeling Example: Comparing Software Component Models*, 1st ed. Springer Publishing Company, Incorporated, 2008.

Representing Topic Event-Based Systems using Pluggable Units

Fernando J. Barros
Departamento de Engenharia Informática
Universidade de Coimbra, Portugal
Email: barros@dei.uc.pt

***Abstract*—In this paper, we present Pluggable Software Units (PUs), a formalism aimed to represent independent and hierarchical software units. PUs extend the request/reply paradigm by introducing full support to anonymous invocation. PUs is a reflective approach supporting the definition of dynamic software topologies. We show that these features enable the representation of modular topic event-based systems using gate-to-gate (g2g) communication. PUs provide a unification of request/reply and event-based paradigms. Our results contradict current research that suggests event-based and request/reply approaches to be independent and intrinsically incompatible. Application examples are described in JUSE, a Java/Groovy implementation of PUs.**

Keywords-topic event-based programming; pluggable software units; dynamic software topologies.

I. INTRODUCTION

Hierarchical and modular design has its origins in the field of General Systems Theory [1] and it has been later adapted by the area of software engineering [2], [3]. Although the advantages of independent software are evident, the definition of reusable software has been elusive for many decades. Early formal frameworks for defining independent software lack the compliance with object-oriented design, becoming virtually useless for software engineering projects. The area of software architecture has produced specifications that have little support from programming languages [4]. Earlier executable specifications supporting the independence between software components have been introduced in the area of event-based programming [5]. Although the event-based paradigm has many features enabling reuse, it is not compatible with the request/reply principles of object-oriented programming, imposing an exclusive choice between programming paradigms [6]. Given the known advantages of both event and request/reply programming it would be desirable to develop a unifying paradigm exhibiting the best of their features.

We have developed PUs [7], a modular and hierarchical software specification framework, based on the General Systems Theory [8]. The PU approach is fully compatible with the request/reply paradigm introducing the complete independence between software units [9]. This approach the anonymous request/reply paradigm, as defined in [6].

In this paper we unify topic event-based and anonymous request/reply programming. In particular, we express events using PUs *gate-to-gate* (g2g) primitives to achieve a framework supporting both styles of programming. This unification allows software models to combine the best features of both paradigms, giving the choice to the modeler to represent parts of the model using the multicast feature of event programming, simultaneously with gate-to-gate communication provided by request/reply.

To obtain the unification of both paradigms we map event publish/subscribe operators into g2g links supported by PUs. Since publish/subscribe operators can be made during application runtime, the key to the unification is given by the ability to support dynamic software topologies that adapts links to these operators.

We shown that design patterns based on implicit invocation, like the Observer pattern [10], can also be represented in PUs, showing the generality of this approach. Application examples are provided in JUSE, a Java/Groovy implementation of PUs.

The paper is organized as follows. Section II provides a formal definition of basic and network pluggable software units (PUs). Section III introduces a representation of topic event-based programming using PUs with a dynamic topology. Related work is described in Section IV. Conclusion and future work are presented in Section V.

II. PLUGGABLE SOFTWARE UNITS

PUs comprises two types of software units: basic and network. Basic PUs provide the actual method invocation, whereas networks are a composition of PUs and provide message passing. In PU composition, both basic and network PUs can be used indistinctly. PUs supports a hierarchical and modular type of software construction. Network definition is dynamic, permitting the specification of adaptive software topologies.

A. Basic PU

Basic PUs define a set of input and output gates. Input gates correspond to object methods, whereas output gates represent an abstract access to external PUs. Output gates are an amendment to the object-oriented protocol and they remove the need for PUs to refer to others explicitly. This construct supports effectively the anonymous request/reply programming. Since PU communication is made exclusively through gates they are completely independent and can be arbitrarily composed. Each basic PU has its own description, referred to as the *PU model*. Let $\widehat{B}$ be the set of names of basic PUS. The PU model associated with $\chi \in \widehat{B}$ is given by:

$$M_\chi = \Big(inGates, \{inSign_g\}, S, s_0, \{a_g\}, outGates,$$
$$\{outSign_k\}, \{outDSign_k\}, \{outFunction_k\}\Big)_\chi$$

where

$inGates$ is the set of PU input gates
$inSign_g$ is the input-to-output signature of every gate g in $inGates$
S is the set of PU states
s_0 is the PU initial state
a_g is an action for every gate g belonging to the set $inGates$

$outGates$ is the set of PU output gates
$outSign_k$ is the output-to-input signature of every gate k in $outGates$
$outDSign_k$ is the intermediate signature of every output gate $k \in outGates$
$outFunction_k$ is the output function of every gate k in $outGates$

An input-to-output signature is a 2-tuple containing the range set of the incoming parameters and the range set of outgoing parameters. For example, if input gate g receives real values **R**, and responds by sending integer values **I**, then its input signature is given by $inSign_g = (\mathbf{R}, \mathbf{I})$.

The function a_g on input gate g of signature (I_g, O_g) is expressed by

$$a_g : S \times I_g \rightsquigarrow S \times O_g$$

An *action* corresponds to a method in the object paradigm. Action a_g receives input values from $(S \times I_g)$, produces a change in the PU state, and returns a value from O_g. As a side effect, an action on a PU can trigger other actions on the PUs linked to it. The action can also request values from the network where the PU is inserted. We do not formalize here these *side effects* of action behavior.

An output-to-input signature is a 2-tuple (O_k, I_k) containing the range set of the outgoing (direct) parameters O_k and the range set of incoming (return) parameters I_k.

Output functions convert the set of values received by an output gate. These functions are useful when several links are connected to an output gate, and in general, to convert values without creating special PUs.

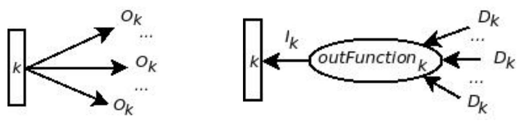
Intermediate signatures define the values, D_k, that can be received by an output gate k. These value are then converted by the output function to the set I_k.

The output function $outFunction_k$ on output gate k of intermediate signature D_k and output signature (O_k, I_k) is expressed by

$$outFunction_k : {D_k}^* \longrightarrow I_k$$

where ${D_k}^*$ is a list of values from set D_k.

Semantics of the output function $outFunction_k$ associated with gate k is graphically sketched in Figure 1, where request and reply semantics are represented in Figure 1(a) and Figure 1(b), respectively.



(a) Request. (b) Reply.

Fig. 1. Semantics of the $outFunction_k$.

In the request phase (Figure 1(a)), values from set O_k are sent to all neighbors of gate k. In the reply phase (Figure 1(b)), values from set D_k are collected and transformed by $outFunction_k$ into a value of the set I_k.

We note that input gates of basic PUs do not define intermediate signatures since these units do not have internal connections, and thus their input gates are terminal.

Given an output gate k with output signature (O_k, I_k), we assume the *head* function when the output function $outFunction_k$ is omitted and the intermediate and input signatures match $(D_k = I_k)$. This function returns the first value from a list and it is defined by:

$$head(< arg_0, \dots >) = arg_0$$

Example: `Position` *PU:* To illustrate an example of a basic PU we employ the `Position` PU represented in Figure 2. This PU has input gates: `ax` and `x`, corresponding to actions it can provide. `Position` has also the output gate `x` that sends the current position to the outside. `Position` receives piecewise constant acceleration values and computes the current position `x` by double integrating the input signal. For simplicity we describe here one-dimension positions. 2D coordinates are used in the next sections.

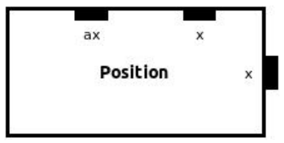


Fig. 2. `Position` PU.

`Position` state keeps the time of the last update ($time$), position (x), velocity (v_x) and acceleration (a_x) values. The PU is described by:

$$\begin{aligned}M_{\texttt{Position}} = (&\{\texttt{ax}, \texttt{x}\},\\ &\{(\mathbf{R}^2, \emptyset), (\mathbf{R}, \mathbf{R})\},\\ &\mathbf{R}^3, (time = 0, x = 0, a_x = 0),\\ &\{action_{\texttt{ax}}, action_{\texttt{x}}\},\\ &\{\texttt{x}\}, \{(\mathbf{R}, \emptyset)\}, \{\emptyset\},\\ &\{outFunction_{\texttt{x}}(\varnothing, ...) = \varnothing\})\end{aligned}$$

where $\emptyset$ represents the empty set and $\varnothing$ represents the null/absence of value.

The `ax` action sets the acceleration and is defined by:

```
$action_{\mathtt{ax}}(t, a)$
    $\delta \leftarrow t - time$
    $x \leftarrow x + v_x\delta + \frac{a_x}{2}\delta^2$
    $v_x \leftarrow v_x + a_x\delta$
    $a_x \leftarrow a$
    $time \leftarrow t$
    $out.\mathtt{x}(x)$
    $\uparrow \varnothing$
```

This action also sends the current position to the outside through gate x using the command out.x(x). The current position at time *t* is computed by:

```
$action_{\mathtt{x}}(t)$
    $\delta \leftarrow t - time$
    $\uparrow x + v_x\delta + \frac{a_x}{2}\delta^2$
```

B. PU Network

Hierarchical composition of systems has been used as a powerful construct to manage complex systems. We consider that PUs can be hierarchically composed, being the resultant PU indistinguishable from the basic PU of the last section. This ability permits to handle in a homogeneous form both basic and aggregated components. A PU network is a complex PU built by the composition of other PUs. Let $\widehat{E}$ be the set of names corresponding to PU networks, constrained to $\widehat{E} \cap \widehat{B} = \emptyset$. The model of the network PU $\chi \in \widehat{E}$ is defined by:

$$M_\chi = \Big(inGates, \{inSign_g\}, \{inDSign_g\}, \{inFunction_g\}, \varepsilon, M_\varepsilon, outGates, \{outSign_k\}, \{outDSign_k\}, \{outFunction_k\}\Big)_\chi$$

where

$inGates$ is the set of the network input gates
$inSign_g$ is the input-to-output signature of every gate $g \in inGates$
$inDSign_g$ is the intermediate signature of every input gate $g \in inGates$
$inFunction_g$ is the input function of every gate $g \in inGates$

$\varepsilon \in \widehat{\varepsilon}$ is the network executive
M_ε is the model of the network executive
$outGates$ is the set of the network output gates
$outSign_k$ is the output-to-input signature of every gate $k \in outGates$
$outDSign_k$ is the intermediate signature of every output gate $k \in outGates$
$outFunction_k$ is the output function of every gate $k \in outGates$

with $\widehat{\varepsilon}$ representing the set of all names associated with network executives, constrained to $\widehat{\varepsilon} \cap \widehat{B} = \widehat{\varepsilon} \cap \widehat{E} = \emptyset$.

The PU network has the same type of interface of a basic PU making it possible to use networks as components of other networks, enabling the hierarchical composition of PUs. The network topology is managed by a special PU termed here by *network executive* ϵ. The executive keeps a list of the PUs that compose the network. It also keeps the set of the links existing among PUs. This information is not static, and can be changed by executive actions [11]. The model of the network executive is an augmented PU model defined by:

$$M_{\varepsilon_\chi} = \Big(inGates, \{inSign_g\}, S, s_0, \{a_g\}, \sigma, \widehat{\Sigma}, outGates, \{outSign_k\}, \{outDSign_k\}, \{outFunction_k\}\Big)_{\varepsilon_\chi}$$

Function σ maps the executive state into an network topology. The *topology function* σ is expressed by:

$$\sigma : S \rightarrow \widehat{\Sigma}$$

Each topology $\Sigma \in \widehat{\Sigma}$ is given by

$$\Sigma = \Big(C, \{M_c\}, L, \Xi\Big)$$

where

C is the set of PUs
M_c is the model of each PU $c \in C$
L is the set of links
Ξ is the order function

Given that the current network topology is a function of the executive state, any change in this state can cause a topological change in the network. A link in L is a 3-tuple defined by:

$$\Big((i, g_i), (j, g_j), (dC, rC)\Big)$$

where

i is the name of the source PU
g_i is a gate of the i PU
j is the receiver PU
g_j is a gate of j
dC is the link direct converter
rC is the link reverse converter

Converters transform both the values sent and received by a PU. For example, if a PU works with values in m·s^{-1} and needs to communicate with another PU operating in km·h^{-1}, then adapting capabilities provide a solution to make this conversion without the creation of additional PUs. In this case, the direct converter is given by $dC(x) = 3.6\ x$, and the reverse converter is given by $rC(x) = \frac{x}{3.6}$ to make the conversions m·s^{-1} $\leftrightarrow$ km·h^{-1}. If omitted, converters are considered to be the identity function. We note that reverse converters are a consequence of the request/reply paradigm that imposes the compatibility of the returned values.

$\Xi : L^+ \rightarrow L^+$ is the *order function*, where L^+ is the set of all sets of links (excluding the empty set).

The order function establishes the order of the outside calls when several links are connected at the same output gate. For simplicity, when omitted, a non-deterministic order is assumed. In the JUSE implementation the order function is established implicitly by link declaration order.

The initial network topologyΣ_0 is given by $\Sigma_0 = \sigma(s_{0,\epsilon})$.

Example: Mobile Entity: To illustrate the definition of a PU network, we build a PU to represent an autonomous mobile based on the `Position` PU of last Section. The network PU is depicted in Figure 3 and it is defined by:

$$\begin{aligned} M_{\texttt{Mobile}} = &(\{\texttt{x}\}, \{(\emptyset, \mathbf{R})\}, \varepsilon_{\texttt{Mobile}},\\ &M_{\varepsilon_{\texttt{Mobile}}}, \{\texttt{x}\}, \{(\emptyset, \mathbf{R})\}, \{\emptyset\},\\ &\{outFunction_{\texttt{x}}(\varnothing, ...) = \varnothing\}) \end{aligned}$$

where

$$\begin{aligned} M_{\varepsilon_{\texttt{Mobile}}} = &(\{\}, \{\}, \{s_0\}, s_0, \{\}, \sigma, \{\Sigma_0\},\\ &\{\}, \{\}, \{\}, \{\}) \end{aligned}$$

Network single topology given by:

$$\Sigma_0 = \sigma(s_0) = (C, \{M_c\}, L)$$

where

$C = \{\texttt{Position}\}$
$\{M_c\} = \{M_{\texttt{Position}}\}$
$L = \{$
$((\texttt{Mobile}, \texttt{x}), (\varepsilon_{\texttt{Mobile}}, \texttt{x}), dC(t) = t, rC(x) = x)$
$((\varepsilon_{\texttt{Mobile}}, \texttt{ax}), (\texttt{Position}, \texttt{ax}), dC(t, a) = (t, a), rC(\varnothing) = \varnothing),$
$((\varepsilon_{\texttt{Mobile}}, \texttt{x}), (\texttt{Position}, \texttt{x}), dC(t) = t, rC(x) = x),$
$((\texttt{Position}, \texttt{x}), (\texttt{Mobile}, \texttt{x}), dC(x) = x, rC(\varnothing) = \varnothing)$
$\}$

The PU network, represented in Figure 3, is composed of one `Position` PU, linked to the executive $\varepsilon_{\texttt{Mobile}}$. The network has the input gate `x` to access the value of the current position. The executive requests the position through the call `x(time)`, where `time` represents the current time.

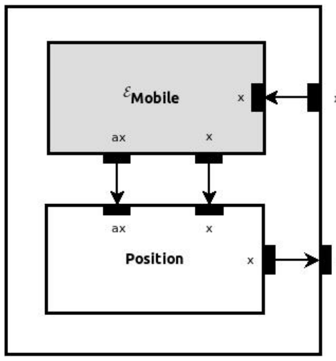


Fig. 3. Block diagram of the `Mobile` network PU.

At a random intervals, the executive updates the current value of acceleration through gate `ax`. This value is integrated by PU `Position` that computes current position and velocity as described in the last section.

C. JUSE

JUSE is a Java/Groovy implementation of PUs and it provides an executable version of software units. JUSE supports the following calls to create software units and to establish links between PUs:

`void addS(Class aClass, String aName)`, creates a PU named `aName` of class `aClass`;
`void linkS(String aName, String aGate, String bName, String bGate, Closure dConverter, Closure rConverter)`, links a PU named `aName` gate `aGate` to gate `bGate` of PU `bName`, establishing direct converter `dConverter` and reverse converter `rConverter`.

Input/output functions are associated with input/output gates, respectively. In JUSE these functions are specified when gates are added to PUs by the call:

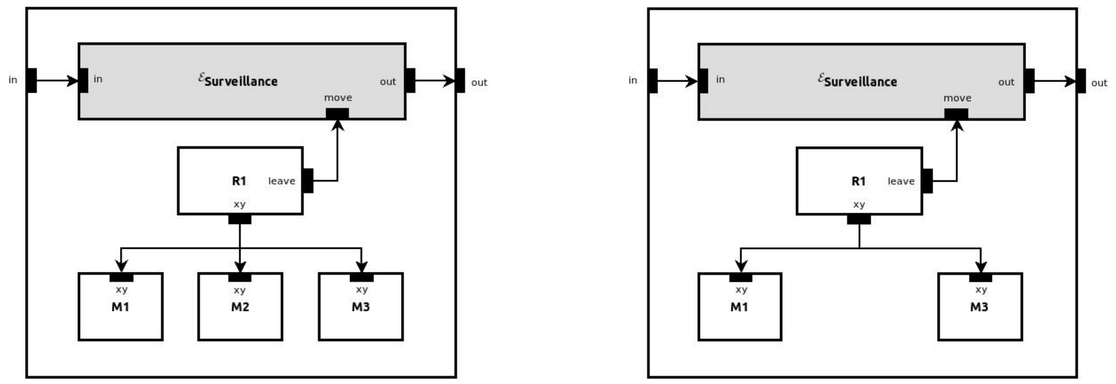
`GateCollection add(Class rSgnt, Class iSgnt, String aGate, ArrayList<Class> dSgnt, Closure aClosure)`, that adds `aGate` with return signature `rSgnt`, intermediate signature `iSgnt` and direct signature `dSgnt`, and associates it with the function `aClosure`.

These primitives are used in the next sections for describing several examples using JUSE. Anonymous Publish/Subscribe JUSE provides support for anonymous publish/subscribe programming enabling both static and dynamic topologies. We consider the surveillance system depicted in Figure 4, with one radar and a variable number of mobiles (ships, aircrafts, ...), that enter and leave radar range. The radar samples the position of all mobiles at a regular rate and receives a list of pairs with mobile position and name. Instead of modeling mobiles as entities that publish their position, as required by push event-based programming, we have considered that mobile positions are pulled by the radar. The main advantage is that we can directly express radar sampling rate instead of handling an arbitrary pushing rate from the mobiles. This representation makes the surveillance model more efficient since it can easily accommodate several radars with different sampling rates. When the radar issues the command `xy` it receives a list of mobile positions at the current time. This example exploits the bidirectional nature of request/reply that unifies the push and pull styles of event-based programming. These two styles can be used independently in some systems like CORBA [12]. To override the default behavior of radar output gate `xy`, we use the following output function that return a list of values:

```
  add(List, XY, 'xy', [], List<XY> list->
list)
```

Surveillance initial topology is given by Listing 1 where the name of mobiles is introduced by reverse converters using the method `setSource` in lines 9-11.

```
void structure() {                                                   1
  super.structure();                                                 2
  addS('R1', Radar);                                                 3
  addS('M1', Mobile);                                                4
  addS('M2', Mobile);                                                5
  addS('M3', Mobile);                                                6
  linkS('Network', 'in', 'Executive', 'in', {List<IOutput> m->       7
       [m]}, {Void x-> x});
  linkS('Executive', 'leave', 'Network', 'move', {List<String>s ->   8
       [s]}, {Void x-> x});
  linkS('R1', 'xy', 'M1', 'xy', {->}, {XY p-> p.setSource('M1')});   9
  linkS('R1', 'xy', 'M1', 'xy', {->}, {XY p-> p.setSource('M2')});   10
```



(a) Initial topology.   (b) Topology after the removal of mobile `M2`.

Fig. 4. Block diagram of the surveillance network.

```
  linkS('R1', 'xy', 'M1', 'xy', {->}, {XY p-> p.setSource('M3')});
}
```

Listing 1. JUSE definition of the surveillance initial topology

One of the radar tasks is to keep track of the mobiles within its range. Mobiles out of range are removed and sent to another surveillance system through output gate `out`. These operations are described in Listing 2 where executive action `move` receives a list of mobiles to be removed from the current surveillance system.

```
Void move(List<String>mobiles) {
  List<IOutput> leaving = new List<IOutput>();
  mobiles.each{String m-> leaving.add(remove(m)};
  out.out(leaving);
  return null;
}
```

Listing 2. Removing mobiles

III. Topic Event-Based Programming

We consider event-based programming as a special case of anonymous publish/subscribe systems with a dynamic topology. This interpretation implies that the same type of invocation is used, the difference being the way dynamic topologies are specified. The event invocation is less expressive than anonymous publish/subscribe due to unidirectional information flow. The publish/subscribe mechanism characteristic of event programming is kept since it enables simpler specifications.

A. Basic Operators

A key feature introduced in publish/subscribe systems, and not supported by many modular systems, is the ability to define changes in topology. In fact, publish/subscribe operators can be regarded as implicitly defining dynamic topologies that link publishers and subscribers. However, in a reflective framework like PUs, that provides full support for dynamic topologies, the ability to change links between components can be easily supported. While PUs topology is kept and controlled by the executive, this does not mean that changes in topology must be decided in the executive. In fact, decisions can be made anywhere in the system, but they are only effective when they are enforced by the executive. To provide similar operators as defined by event-based systems we consider the executive to act as the implicit hidden middleware supporting events. The result is an explicit construct where publish/subscribe messages can be handled. For this purpose, we provide each PU with the output gate `command`. The executive is provided with the input gate `command`, that receives a command and the origin of the command. To support hierarchical event-based systems, we extend PU network with the output gate `command`, so publish/subscribe commands can be sent to upper levels of the hierarchy.

JUSE supports the following executive commands to represent publish/subscribe systems:

`Void publish(String aName, String aGate, Closure dConverter, Closure rConverter);` where PU named `aName` publishes output gate `aGate`, with direct converter `dConverter` and reverse converter `rConverter`;

`Void subscribe(String aName, String aGate);` where PU named `aName` subscribes gate `aGate` and will receive notifications at gate `aGate`.

These operators establish g2g channels whenever there is an intersection between the interest of publishers and subscribers. The definition of hierarchical event-based systems is enabled by the possibility to send/receive messages to/from the network PU.

JUSE supports the following executive commands to destroy links in publish/subscribe systems:

`Void unpublish(String aName, String aGate);`

`Void unsubscribe(String aName, String aGate).`

Although this can be seen as an exercise of expressing one paradigm into another, this mapping combines advantages of both techniques. The unifying approach allows multicast and g2g topologies to co-exist. If we take the example of the last section, while the links between the radar and the mobiles can be easily expressed using publish/subscribe operators, a pursuer launched to cancel a specific mobile will be better specified using g2g operators. In every case the specification ends up becoming g2g, but when using publish/subscribe operators links are defined implicitly, providing a short notation that in many cases becomes more convenient. The addition of a new radar, for example, becomes quite simple, since the radar needs only to publish its output gate `xy`.

The new representation of the Mobile PU presented in Section II-B is given in Figure 5. Default gate `command` and g2g links are provided to each PU so commands can be transmitted. While the executive has the input gate `command` linked to the output gate `command` of all other PUs, executive output gate `command` is linked to network output gate `cmd` so commands can be sent hierarchically to the upper level, enabling dynamic scoping [6].

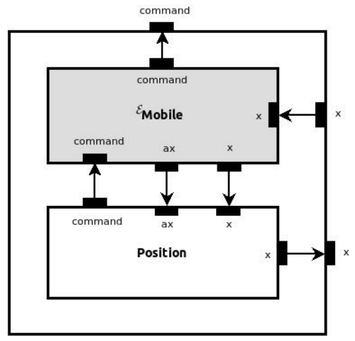


Fig. 5. Mobile PU supporting publish/subscribe operators.

A major contribution of the publish/subscribe paradigm is a set of compact operators to express structural changes. Gate-to-gate links require source and destination information in order to establish a new channel. In some situations this information is cumbersome to gather and publish/subscribe operators can represent models in a simpler manner. However, since the resulting topologies can still be expressed by g2g networks, no increasing expressiveness is actually obtained. We next provide an example that demonstrates the advantages of the publish/subscribe operators.

B. Event-Based Surveillance System

We consider the event-based representation of the surveillance system described in Section 3 and depicted in Figure 6. After the commands of Listing 3, the topology is changed from the initial structure given by Figure 6(a) to the topology of Figure 6(b).

```
out.command({String s, Executive e-> e.publish(s, 'xy', {->},         1
      {String source, XY p-> p.setSource(source)})}) "In radar R1"
out.command({String s, Executive e-> e.subscribe(s, 'xy')}) "In       2
      Mobile M1"
out.command({String s, Executive e-> e.subscribe(s, 'xy')}) "In       3
      Mobile M2"
```

Listing 3. JUSE publish/subscribe commands to create the topology in Figure 6(b).

The commands originated at mobiles `M1` and `M2` are issued by the respective executive and sent to the Surveillance executive through gates `command`. Line 1 publishes `R1` output gate `xy`. Line 2 subscribes to all output gates `xy` that have been publishes and tries to create g2g channels to `M1` input gate `xy`. Line 3 is similar and it applies to mobile `M2`. Mobile `M3` did not subscribe any gate and it becomes stealthy since it cannot be detected by any radar. The advantages of this approach are self-evident. The introduction of new radars and mobiles becomes very simple since the event-based executive handles publications and subscriptions, freeing the modeler from specifying g2g links. These links can be cumbersome to establish as shown in this particular system.

The support in JUSE for systems requiring a hybrid specification using events and request/reply can be exemplified by the creation of a pursuer launched to cancel a specific threat. Upon detection, the radar sends a signal to the executive to request the creation of a pursuer and to make g2g links between the pursuer and a specific mobile identified by the radar. Figure 7 depicts the surveillance system with pursuer `P1` attached to mobile `M1`. The radar and executive PUs are extended with gate `pursuer` so the radar can make requests for purser creation.

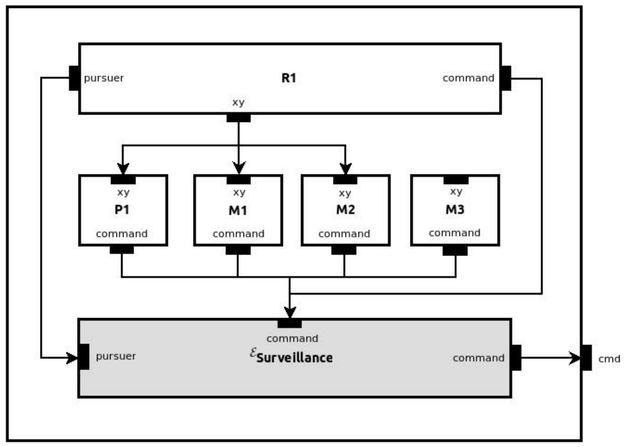


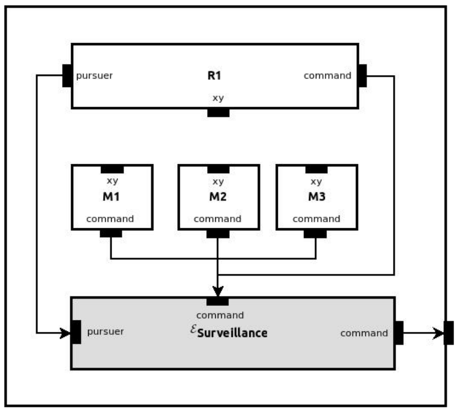
Fig. 7. Surveillance system with purser `P1` targeting mobile `M1`.

JUSE executive action `pursuer` is given in Listing 4. The executive finds the nearest purser to position aXY (Line 3), adds the purser (Line 4) and links it to the mobile aName (line 5). These links are specified using g2g operations and they would become difficult to be specified using publish/subscribe operations.

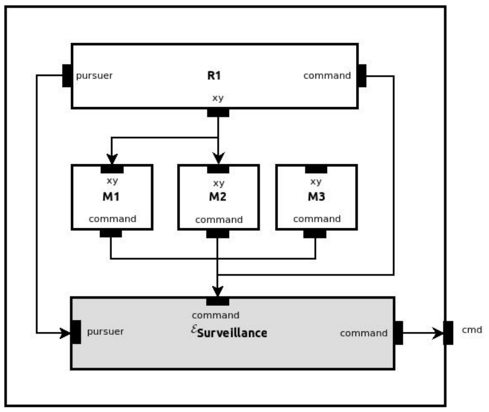
```
Void pursuer(String aName, Point aXY) {                 1
   pursuer = findPursuerAt(aXY);                        2
   add(pursuer);                                        3
   link(pursuer _name, 'xy', aName 'xy');               4
}                                                       5
```

Listing 4. Creation of pursuer `P1` using g2g specification.

This solution requires the ability to program the executive that no longer can be an implicit and hidden middleware layer and needs to be made visible and reprogramable. A solution in the event paradigm would require the use of content event-based programming and the pursuer would only receive messages from a specific mobile. This solution, however, would



(a) Initial topology.



(b) Topology after the radar `R1` has published output gate `xy` and `M1-M2` have subscribed gate `xy`.

Fig. 6. Event-based surveillance system.

not be so elegant and mostly no so efficient since it would require message conversion from a possibly large number of mobiles. This solution would also require to disclose the mobile name to the pursuer, breaking the modularity of the approach.

C. Event-Based vs. Request/Reply

The multicast nature of events poses no difficulty in being mapped into gate-to-gate communication. In the example of the previous section, when a new mobile is added to the surveillance system and subscribes gate `xy` it is immediately linked to all radars in the systems, without the need to explicitly find what are the current set of active radars, since the executive can retrieve this information and use it to make the required links. Analogously, when a new radar is created and publishes gate `xy` it starts receiving data from all mobiles, since the executive keeps tracks of all PUs having published gate `xy`. The complementary situation is treated in a similar form: when a mobile leaves the system by unsubscribing gate `xy`, all radars stop receiving mobile position. Event-based systems have been considered orthogonal to anonymous request/reply programming [6]. This seems to be the case for some request/reply systems, but it does not hold for PUs, as shown here.

Event-based programming, through publish/subscribe constructs, has introduced a set of operators that provide a compact specification for some types of changes in software topology. However, these operators can be mapped into g2g links defined in PUs. This situation is quite fortunate since it allows the expression of event-based programming with modular software units. Events can thus be integrated with anonymous request/reply instead of being an additional form of software specification.

The difficulty in integrating event-based and request/reply approaches seems to be caused by the requirements event systems impose on software topologies. As shown in the examples, anonymous invocation is just one of the key factors to integration, the other is the ability to modify software topology during runtime. Anonymous requests/reply solutions not exhibiting both features will thus provide limited support for event-based systems.

We consider that a unifying effort is currently required given the multiplicity and apparent disparate paradigms proposed for software development. This situation forces practitioners to master a large variety of paradigms, or in alternative, to map models to a known paradigm that does not yield be best representation. Since reality is, for complex systems, multi-faceted, a unifying approach, like PUs, permits to choose the best paradigm to each aspect of the system while guaranteeing the overall integration.

IV. Related Work

Hierarchical and modular principles have been used as a powerful heuristic for handling complex problems in many fields. One of the first formal descriptions of modular decomposition have been made in the area of General Systems Theory [1]. The decomposition of software into modules has later been advocated in software engineering [3]. On this latter work, however, the hierarchical decomposition of software has not been really introduced but rather hierarchy is used as synonymous of layered (software). Recently, there has been a growing interest in modular representations and large variety of formalisms have been modified/created [2], [13]–[15]. Likewise General Systems Theory representations, these formalisms are not compliant with request/reply principles, imposing awkward software specifications. Given these limitations, many formal models become virtually useless for practical use in software projects.

To overcome the limitations of formal descriptions, so called Architecture Definition Languages (ADLs) have been developed [4], [16], [17]. However, ADLs are mainly fa ades decoupling specification from implementation as pointed in [18]. ADLs description need thus to be translated into a programming language. This process is somewhat similar to the one used by the Unified Modeling Language [19] with the limitations and drawbacks of separating specification from implementation.

To bridge this gap, hierarchical and modular constructs have been introduced into existing programming languages [18], [20], [21]. However, none of theses approaches provide the general support to modular hierarchical software as provided by PUs. Limitations include the lack of converters and input/output functions. Additionally, these systems do not provide full support for topology adaptation. In particular, they lack the ability to represent hierarchical mobility, as supported by PUs [7].

The use of event-based programming has also been advocated as an alternative to anonymous request/reply [22], (the reverse we have described in this paper). However, the advantages are only apparent since signal wiring diagrams [22] need to be manually mapped into object-oriented request/reply languages becoming, in some aspects, similar to ADLs.

V. Conclusion and Future Work

PUs provide a powerful framework for developing reusable software units. This approach supports hierarchical and modular software development, permitting to handle complexity by partitioning large models into smaller and independent units. The introduction of converters and input/output functions provide a great flexibility to software interconnection. PUs also provide full support for dynamic software topologies. In particular, we have shown that the ability to add and remove software channels at runtime permits to describe the topic event-based programming style using gate-to-gate connections supported by PUs. This work has demonstrated that topic event-based programming can be regarded as a particular case of anonymous request/reply. Callbacks, used in the Observer pattern and the Composite pattern, were also shown to be particular cases of an anonymous request/reply representation supporting converters and input/output functions. As future work we intend to study the requirements for supporting content event-based programming and to introduce the required operators to represent this paradigm in PUs. The representation of event-based scoping exploiting hierarchical modeling also looks promising.

Acknowledgment

This work was supported by the Portuguese Foundation for Science and Technology under project PTDC/EIA-EIA/100-752/2008.

References

[1] A. Wymore, *A Mathematical Theory of Systems Engineering: The Elements*. Krieger, 1967.

[2] R. Allen and D. Garlan, “A formal basis for architectural connection,” *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 3, pp. 213–249, March 1997.

[3] D. Batory and S. O’Malley, “The design and implementation of hierarchical software systems with reusable components,” *ACM Transactions on Software Engineering and Methodology*, vol. 1, no. 4, pp. 355–398, 1992.

[4] D. Garlan, R. Monroe, and D. Wile, “ACME: An architecture description interchange language,” in *Conference of the Centre for Advanced Studies on Collaborative Research*, 1997.

[5] D. Luckham and J. Vera, “An event-based architecture definition language,” *IEEE Transactions on Software Engineering*, vol. 21, no. 9, pp. 70–93, 1996.

[6] G. Mühl, L. Fiege, and P. Pietzuch, *Distributed Event Based Systems*. Springer, 2006.

[7] F. Barros, “System and method for programming using independent and reusable software units,” US Patent 6851104 B1, February 2005.

[8] ——, “Modeling formalisms for dynamic structure systems,” *ACM Transactions on Modeling and Computer Simulation*, vol. 7, no. 12, pp. 505–515, 1997.

[9] ——, “Achieving reuse with pluggable software units,” in *12th International Conference on Software Reuse: Top Productivity through Software Reuse*. Lecture Notes in Computer Science, Volume 6727, 2011, pp. 183–191.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[11] F. Barros, “Representing hierarchical mobility in software architectures,” in *International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2007.

[12] OMG, *CORBA Component Model Specification*, 2006.

[13] J. Bradbury, “Organizing definitions and formalisms for dynamic software architectures,” Queens University, Canada, Tech. Rep. 2004-77, 2004.

[14] F. Oquendo, “Formally modelling software architectures with the UML 2.0 profile for π-ADL,” *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 1, pp. 1–13, 2006.

[15] F. Arbab, “Reo: A channel-based coordination model for component composition,” *Mathematical Structures in Computer Science*, vol. 14, pp. 329–366, 2004.

[16] N. Medvidovic and R. Taylor, “A classification and comparison framework for software architecture description languages,” *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp. 70–93, 2000.

[17] M. Shaw and P. Clements, “The golden age of software architectures: A comprehensive survey,” Carnegie-Mellon University, USA, Tech. Rep. CMU-ISRI-06-101, 2006.

[18] J. Aldrich, C. Chambers, and D. Notkin, “ArchJava: Connecting software architecture to implementation,” in *International Conference on Software Engineering*, 2002, pp. 187–197.

[19] J. Arlow and I. Neustadt, *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison, 2005.

[20] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J. Stefani, “The FRACTAL component model and its support in Java,” *Software Practice and Experience*, vol. 36, no. 11–12, pp. 1257–1284, 2006.

[21] V. Sreedhar, “Mixinup components,” in *International Conference on Software Engineering*, 2002, pp. 198–207.

[22] T. Faison, *Event-Based Programming: Taking Events to the Limit*. Apress, 2006.

Towards an Approach to Represent Safety Patterns

Pablo Oliveira Antonino, Thorsten Keuler
Embedded Systems Development Department
Fraunhofer IESE
Kaiserslautern, Germany
(Pablo.Antonino, Thorsten.Keuler)@iese.fraunhofer.de

Elisa Yumi Nakagawa
Department of Computer Systems
USP - University of São Paulo
São Carlos, Brazil
elisa@icmc.usp.br

***Abstract*—Safety-critical systems are complex entities, which, due to severe regulations, demand continuous development of approaches for supporting their construction. To keep safety-critical systems free of failures, it is fundamental to identify potential failure modes and their causes, and to eliminate them. One major approach to solving failure modes is the application of safety patterns at the architectural level of such systems. However, this is not trivial, since safety patterns have not been represented in a widely accepted way that would facilitate their understanding and use. In order to contribute to filling this gap, we present in this paper an approach for representing safety patterns in a way that allows them to be properly modeled and also offers means to support their application in architectural models. To this end, we propose the joint use of a UML profile and rules that are descriptive structures stating safety patterns application constraints. We have observed that our approach makes the safety patterns easy to represent and apply, thus contributing to the development of safety-critical systems.**

Keywords-Safety Pattern; UML Profile; Pattern Descriptive Rule; Architectural Model.

I. Introduction

Domains such as automotive and avionics demand high integrity levels between hardware and software to ensure proper execution of their systems, which, in turn, are constantly becoming larger and more complex [1]. A commercial airplane, for instance, contains systems that control ground proximity, navigation, and engine commands, amongst others. Almost all of these systems are safety-critical; i.e., a failure would lead to a catastrophic situation, endangering human lives and/or the environment. To minimize the probability of failure in safety-critical systems, it is necessary to integrate tactics based on well-known fault-tolerant methods to deal with failure avoidance, detection, and containment, for example, monitoring and redundancy [2]. Most of these tactics are concretized by means of specific design patterns, best known as Safety Patterns. Examples of safety patterns are Watchdog, Homogeneous Redundancy, and Sanity Check [3].

The growing popularity of model-driven approaches, such as Model Driven Architecture (MDA) and Model Transformation, has triggered the use of such tactics in the construction of safety-critical systems [1]. In this context, UML (Unified Modeling Language) [4] has been used by several MDA-based approaches for representing the required models. The reason is the diversity of elements and diagrams offered by UML, which provides means for expressing systems from diverse perspectives [5], and the existence of extension mechanisms like UML Profiles, which allows modeling particularities of domains by means of customizations of UML's syntax and semantics [6].

The UML's official specification for representing design patterns consists of using Parameterized Collaborations [7]. However, due to singularities that are inherent to safety-critical systems and safety patterns, UML and the others initiatives mentioned in the literature [8][7][9][10][11] are not appropriate for representing safety patterns in a standardized way that jointly facilitates their understanding and supports the automatized application of patterns in architectural models. Actually, the existent approaches belong to one of two extremes: (1) too complex and far from intuitive, requiring deep knowledge about formal specification techniques for representing patterns, or (2) providing only subjective information about the pattern that is useful only for reasoning on high-level concerns, which is important, but not enough to support the application of safety patterns in architectural models.

To fill this gap, we propose an approach for representing safety patterns that offers means for graphically expressing the structure and purpose of a safety pattern, and also provides information to facilitate its automatized application in architectural models by means of Model Transformations. In a nutshell, this corresponds to the joint use of UML Profile and descriptive rules stating safety pattern constraints that are worth being considered for their application in architectural models. After representing a set of safety patterns relevant in the domain of safety-critical systems described in [3] with our approach, we observed that the safety patterns became easier to represent and reuse, thus indicating our contribution to the construction of safety-critical systems.

The remainder of this paper is structured as follows: In Section II, the overall context is presented; in Section III, the related works are described; in Section IV, we present our approach; in Section V, we show the complete representation of a safety pattern using our approach; and in Section VI we conclude and present perspectives for future work.

II. Context

A. Safety-Critical Systems

Safety-critical systems are those that, in case of failure, will cause unacceptable drastic consequences to human beings and/or to the environment [12].

There are four concepts that are intrinsically related to safety-critical systems [12]: (i) *Mistake:* Cause of a fault happening during development; (ii) *Fault:* The adjudged or hypothesized cause of an error; (iii) *Error:* If the system is running, an error is an erroneous state that could lead to a failure; and (iv) Failure: An event that occurs when the system terminates its ability to provide the correct service. As perceived, a Failure is caused by an Error, which is caused by a Fault, which is a result of a Mistake. According to Avižienis et al. [12], all faults that might affect a system during its existence are part of specific Fault Classes, which, in turn, are grouped into specific Fault viewpoints. With respect to failures, Avižienis et al. [12] discuss *service failure modes*. A *service failure* happens when a service delivered by a system deviates from its correctness, and *service failure modes* are the different ways in which the deviations are perceived.

B. Safety Tactics and Safety Patterns

Safety tactics are architectural design decisions made to avoid or handle failures that safety-critical systems are subject to [2]. They are based on well-known fault-tolerant design methods, and were inspired by the notion of architectural tactics proposed by the Software Engineering Institute (SEI). Architectural tactics are "means of satisfying a quality-attribute-response measure by manipulating some aspect of a quality attribute model through architectural design decisions." [13]. Following the same principle, Wu and Kelly developed an analytic safety model focusing on the relationship between safety attributes and architectures with respect to failures. Based on this analytic safety model, they organized safety tactics into three categories: (i) tactics for failure avoidance, (ii) tactics for failure detection, and (iii) tactics for failure containment. To be compliant with the SEI's tactics approach, they proposed a hierarchical organization for such tactics, as illustrated in Figure 1.

Due to the nature of safety-critical systems, the decision about whether the tactics should be addressed at the software or at the hardware level are mainly driven by regulations, which state exactly where and how a tactic must be applied [2].

Safety tactics should be seen as the highest abstraction level of a safety pattern. A safety tactic will be addressed in a safety-critical system when any pattern (or combination of patterns) that implements the tactic is considered in the system architecture.

Safety patterns have been considered for years mainly in the Electrical Engineering field, due to the fact that safety-critical systems, in most cases, are the result of a very tight synergy between the hardware and the software embedded in electronic devices - so-called Embedded Systems [14]. However, methods originating from the Computer Science field have been widely considered in the development of such systems, mainly because the software portion of embedded systems is continuously acquiring more responsibility. In this regard, safety patterns have been considered a topic of interest for software engineers.

Some aspects of safety patterns that make them special and demand specific mechanisms for dealing with them are [3]: (i) they can be essentially formed by roles that represent software or hardware entities, or by a combination of both, demanding specific ways to show how these entities are related; (ii) a great number of roles are common to many safety patterns, differentiating basically in how they are connected and distributed along execution channels. Execution channels are pipes comprised of roles that sequentially transform input data into output data [3]; (iii) each safety pattern is meant to avoid, detect, or do the containment of a failure and faults [12]. Therefore, it is necessary means to deal with the reuse of roles while modeling safety patterns, which will ensure that the roles are properly connected and are associated with the proper execution channel. Moreover, there must exist means to indicate the fault class and service failure mode that the safety pattern is supposed to handle, as well as the safety tactic that the safety pattern implements.

With respect to the description and representation of safety patterns, we considered the general pattern description principle proposed by Alexander [15] that, when reasoned in our context, states that a pattern description also provides means with which can be observed how the resulting system architecture will look like after the application of a pattern. This generative property enforces that a pattern description should not only show the characteristics of a pattern, but coach how to apply it [16]. In this regard, we understand that a safety pattern constitutes a set of entities related to each other by specific rules, which, by definition, represent knowledge that states actions to be followed for the achievement of a purpose [17].

C. UML Profiles

UML Profiles are mechanisms for specifying rules to be used in parts of the model of a system where specific constraints are required [7]. Profiles are based on stereotypes and tags, which are the concrete entities that must be applied to UML elements such as classes, components, and connectors, with the aim of ruling parts of a system with respect to constraints of the domain or of the modeling process.

III. Related Work

In the computer science field, software pattern specification and representation have been widely discussed, mainly after the work of GoF (Gang of Four) [18]. The

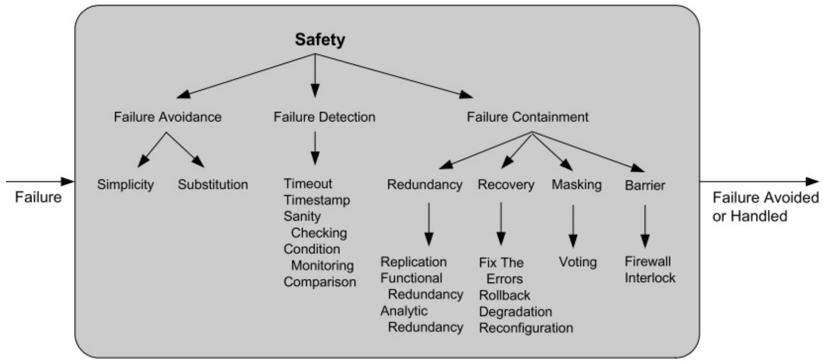


Figure 1. Hierarchical organization of safety tactics (extracted from [2]).

UML's official pattern representation approach is based on a parameterized collaborations model, rendered in a way similar to UML template classes [7]. Rosengard and Ursu [19] proposed an ontological representation for patterns. Mak et al. [20] proposed an extension to UML 1.5, using meta-modeling techniques and collaboration diagrams to specify the collaboration among the elements of the model. Guennec et al. [21] proposed the use of UML collaboration models combined with the Object Constraint Language (OCL) for representing patterns. Eden et al. [10] proposed a declarative and higher-order language, called LePUS, to represent generic solutions indicated in the patterns. Kim [9] proposed a language called Role-Based Meta modeling Language (RBML), which comprises abstract syntax, meta model level constraints, and constraint templates. Selonen et al. [11] established a language for defining profiles hierarchy, which is derived to support patterns representation using only UML Profiles.

With respect to safety patterns, Douglass [3] [22], Pullum [23], Koren and Krishna [24], and Hanmer [25] have documented a vast list of safety patterns. However, their presentation of these patterns focuses mainly on general information related to the general structure, the problem addressed, the context of use, and the consequences.

Regarding the representation of safety patterns, Armoush et al. [8] proposed an approach for representing safety patterns that consists of a traditional table template for pattern documentation, with a series of fields that address subjective information like the safety pattern's name, problems that it solves, and consequences of use. Such a canonical structural form of representing a pattern is basically useful for understanding the nature and purpose of the patterns, but does not offer any information to support the proper pattern application in architectural models.

An approach that is closer to ours is the one proposed by Tichy and Giese [26], which uses degradation rules to describe the structure and deployment restrictions of safety patterns and specify the behavior that is executed while degrading the systems functional or non-functional properties. Actually, such rules are used only to complement the structural and deployment documentation of safety patterns. On the other hand, our rules were built foreseeing automatized processes of safety patterns application in architectural models by means of model transformation mechanisms.

IV. Our Approach

Our approach for representing safety patterns aims at facilitating the modeling and application of safety patterns in the architectures of safety-critical systems. For this, it jointly uses:

1) *Graphic models of the safety pattern* that show (i) the structure of the safety pattern in terms of the roles that compose the pattern and how they are connected, modeled with elements defined in the *Safety UML Profile*; (ii) the safety tactic that the pattern is related to; (iii) the fault class, and (iv) the service failure mode for which the pattern is appropriate.
2) *Pattern Descriptive Rules*, which express detailed design constraints of the safety pattern, providing information that is useful to support the construction of statements used by mechanisms that perform automatic application of safety patterns modeled with the *Safety UML Profile*, in architectural models.

The remainder of this section covers: (i) the *Safety UML Profile*; (ii) how to model safety patterns using elements defined in the *Safety UML Profile*; and (iii) the Pattern Descriptive Rules.

A. Establishing the Safety UML Profile

We have defined an UML profile, the so-called *Safety UML Profile*, which aggregates *stereotype* elements representing the roles of each safety pattern. Actually, each role of a safety pattern becomes a stereotype of the *Safety UML Profile*. For instance, Figure 2 shows the roles of the Protected Single Channel Pattern (PSC). This pattern is composed of six roles: Input Sensor, Input Processing, Data Transformation, Data Validation, Output Processing, and Actuator. Each role corresponds to a stereotype in the *Safety UML Profile*. The role Input Processing, for example, becomes the stereotype ≪Input Processing≫.

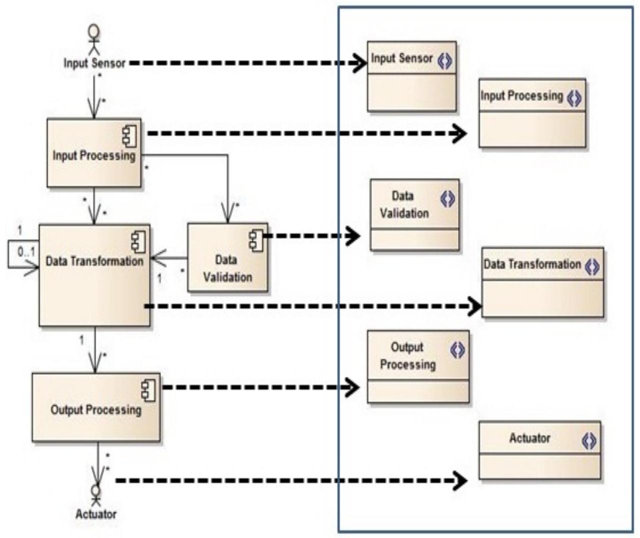


Figure 2. Roles of the Protected Single Channel pattern mapped to stereotypes of the Safety UML Profile.

We understand that a unique UML profile is enough to comprise the entities required for a concise representation of a safety pattern, due to the fact that the roles of the safety pattern are very often repeated in most patterns, differing basically in how these roles are linked, the execution channel that each role is part of, and in the quantity of elements that are present in a specific pattern. For instance, consider the PSC (shown in Figure 2) and the Triple Modular Redundancy pattern (TMR) [3] (shown in Figure 3). It can be seen that the TMR contains roles that are also present in the PSC, but that are replicated along three execution channels. TMR contains an additional role called *Voter*, while PSC has another one called *Data Validation*. To address the roles of the TMR, the Safety UML Profile shown in Figure 2 (which already contains the roles of the PSC) is modified by adding only a new stereotype that represents the role *Voter*.

Each stereotype representing a role of a safety pattern has associated with it a textual description of the role. Such information is useful for engineers to better understand the purpose of each role. There is no standardized way for the description. However, it must be detailed enough to ensure that the purpose of the role is clearly understandable.

Beyond the roles that compose the safety patterns, the *Safety UML Profile* also defines three other stereotypes: ≪Safety Tactic≫, ≪Fault Class≫, and ≪Service Failure Mode≫. These stereotypes are respectively used to indicate the safety tactic (cf. Figure 1) that the safety pattern is associated with, and the *fault classes* [12] and *service failure modes* [12] that a safety pattern solves when applied in

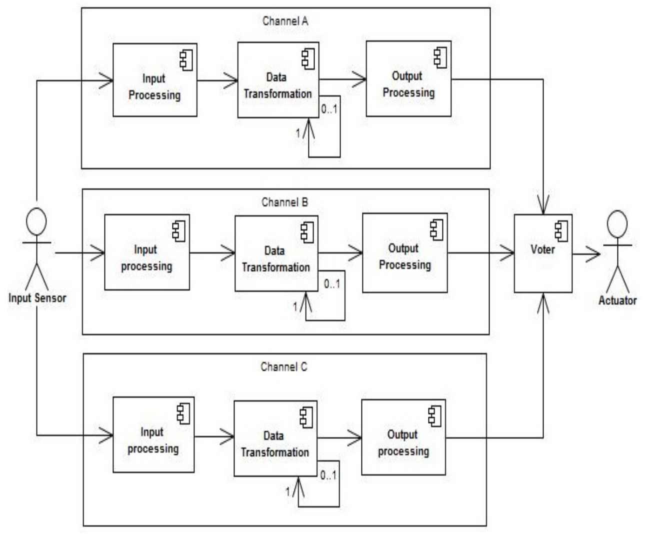


Figure 3. Triple Modular Redundancy pattern and its multiple execution channels (adapted from[3]).

the architecture of a safety-critical system. This is done by adding of three UML classes in the same diagram where the structure of a safety pattern is represented with instances of the *stereotypes* available in the Safety UML Profile. The class associated with the *fault classes* is stereotyped with the stereotype ≪Fault Class≫, and named according to the fault class that the safety pattern solves. The class associated with the *service failure modes* is stereotyped with the stereotype ≪Service Failure Mode≫, and named with the Service Failure Mode that the safety pattern solves. The class associated with the *safety tactic* is stereotyped with the stereotype ≪Safety Tactic≫, and is named according to the safety tactic that the safety pattern is associated with.

B. Representing Safety Patterns with Safety UML Profile

According to our approach, the structures of safety patterns are designed using instances of the *stereotypes* available in the Safety UML Profile. The reason for using instances is that we understand that the *stereotypes* defined in the profile are reusable elements, which are present in multiple patterns, since, as already mentioned, the same role is present in various patterns. For example, consider that the architect wants to model the PSC pattern (cf. Figure 2). In a separate diagram, he/she creates instances of the stereotypes available in the *Safety UML Profile* that comprise the PSC pattern, and connects these stereotype instances with directed links indicating the direction of the information flow, as shown in Figure 4.

Due to the fact that safety patterns can be composed of hardware, software, or both entities at the modeling level, it is important to reason on them in terms of functional entities, regardless of their roles as hardware or software. For instance, the *Data Transformation* role of the PSC pattern

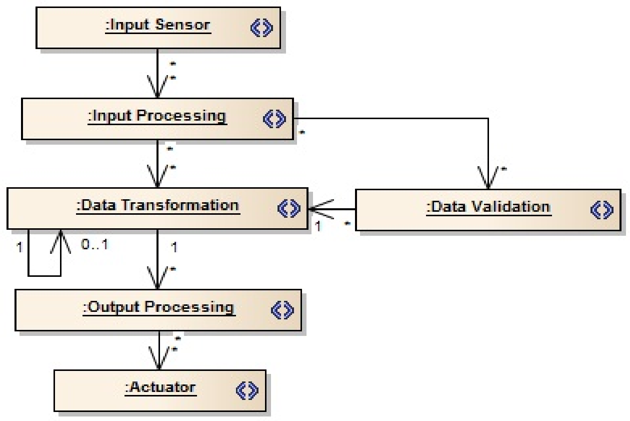


Figure 4. Protected Single Channel pattern specified with instances of stereotypes defined in the *Safety UML Profile*.

can be a hardware or a software entity. Representing the roles with stereotypes offers the flexibility of mapping the *Data Transformation* of the PSC pattern, for instance, to software components, deployment units, or any other UML element of the architectural model, once they are conceptual entities.

If we take a closer look at Figure 2, we observe that the roles of the PSC pattern, except for *Input Sensor* and *Actuator*, are surrounded by a boundary called *Channel*. A particular characteristic of safety patterns is that the roles that compose them, with the exception of Sensors and Actuators, run under execution channels [3]. Our way of dealing with this characteristic is to use Tagged Values [7]. Each stereotype instance used for designing a specific safety pattern is tagged with *Name* ≡ ***ExecutionChannel*** and *Value* ≡ ***Name of the Channel***. When the roles are under a unique execution channel, as in the PSC pattern, each stereotype instance is tagged with a tag that has *Tag Name* ≡ ***ExecutionChannel***, and *Tag Value* ≡ ***Channel A***. In other patterns, such as the TMR pattern, the roles are under multiple execution channels (cf. Figure 3). In this case, the stereotype instances of this pattern (cf. Figure 5) are tagged in accordance with the execution channel they are part of. This means that the three instances of the stereotype that represents the role *Input Processing*, for instance, are tagged differently, due to the fact that they run under different execution channels. Their tags will have *Tag Name* ≡ ***ExecutionChannel***, and *Tag Value* ≡ ***Channel A***, ***Channel B***, and ***Channel C***, respectively. On the other hand, the roles *Input Sensor*, *Voter*, and *Actuator* will have *Tag Name* ≡ ***ExecutionChannel*** and *Tag Value* ≡ ***null*** because they are not associated with any execution channel.

With respect to the Fault Class, Service Failure Mode, and Safety Tactic of the TMR pattern, consider Figure 5, which shows the TMR pattern specified with stereotypes instances. On the bottom left there are (i) a class called *Random Fault*, stereotyped with ≪Fault Class≫, (ii) a class called *Content Failure*, stereotyped with ≪Service Failure Mode≫, and (iii) a third class called *Failure Containment*, stereotyped with ≪Safety Tactic≫. This indicates that the TMR pattern is a concretization of a failure containment tactic, and is appropriate for dealing with random faults and content failures.

C. Pattern Descriptive Rules for representing Design Constraints of Safety Patterns

We understand that a safety pattern constitutes a set of entities related to each other by a specific rule. Therefore, we propose a set of rules, which we call Pattern Descriptive Rules, which were designed to support the future construction of statements used by mechanisms that perform automatic application of safety patterns in architectural models. An example of such a mechanism are Model Transformation rules, which provide indications on how the structure of a model is orchestrated in the model transformation processes, in terms of which elements should be created, updated, or deleted [27].

Our Pattern Descriptive Rules are structured to describe each specific role that participates in a safety pattern, which was previously represented with instances of stereotypes defined in the *Safety UML Profile*. As already mentioned, we argue that the participation of a role in a safety pattern composition is determined by (i) the information flow among the roles in each pattern; (ii) the quantity of instances of the same role in each safety pattern; and (iii) how the instances of the same role are distributed along execution channels. For example, in the PSC Pattern (cf. Figure 4), there is only one instance of the *Output Processing* role, which receives input from the *Data Processing* and provides output to the *Actuator*. On the other hand, in the TMR pattern (cf. Figure 5), there are three instances of the *Output Processing* role, each one in a different channel, and all of them providing output to the *Voter* role.

In this regards, our approach states that for each role that participates in a safety pattern:

1) There is one Pattern Descriptive Rule representing the participation of the role (this includes information about the execution channel where the role is located).
2) There is one Pattern Descriptive Rule describing to which other roles the output flow is directed to.

For instance, for the *Input Processing* role in the PSC pattern (cf. Figure 4), there is one Pattern Descriptive Rule related to the role itself and its execution channel, and one Pattern Descriptive Rule related to the connections between the *Input Processing* and the *Data Transformation*, and another one between the *Input Processing* and the *Data Validation*.

The Pattern Descriptive Rule that we propose for representing *a role and its execution channel* is:

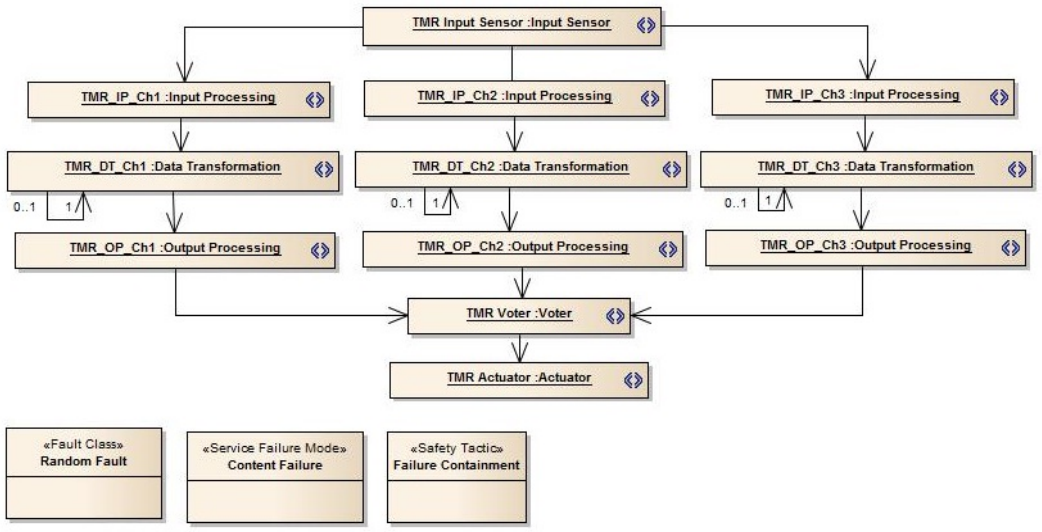

TMR Input Sensor :Input Sensor
TMR_IP_Ch1 :Input Processing
TMR_IP_Ch2 :Input Processing
TMR_IP_Ch3 :Input Processing
TMR_DT_Ch1 :Data Transformation
TMR_DT_Ch2 :Data Transformation
TMR_DT_Ch3 :Data Transformation
0..1
1
TMR_OP_Ch1 :Output Processing
TMR_OP_Ch2 :Output Processing
TMR_OP_Ch3 :Output Processing
TMR Voter :Voter
TMR Actuator :Actuator
«Fault Class» Random Fault
«Service Failure Mode» Content Failure
«Safety Tactic» Failure Containment

Figure 5. Triple Modular Redundancy pattern specified with elements of the *Safety UML Profile*.

rule *Rule Name:*
σ:UML $\cup$ (($\epsilon \in \kappa$) $\in \psi$: Safety UML Profile)
where,

$\sigma \equiv$ Architectural model, modeled according to the standard UML meta model, where the safety pattern will be applied.
$\epsilon \equiv$ UML element representing the role instance.
κ Execution channel where the role instance is located.
$\psi \equiv$ Safety pattern modeled with instance of stereotype defined in the Safety UML Profile.

This Pattern Descriptive Rule is read as: *The architectural model σ is modified by the insertion of a UML element that represents the role instance ϵ, which is tagged with information related to the execution channel κ and composes the safety pattern ψ.*

As already mentioned, if a role instance is not part of an execution channel, like the Input Sensor and Actuator in the PSC, and Input Sensor, Actuator, and Voter in TMR, the tags will have ***Tag Name*** $\equiv$ *ExecutionChannel*, and ***Tag Value*** $\equiv$ *null*.

The Pattern Descriptive Rule that we propose for representing *connections between roles in a safety pattern* is:

rule *Rule Name:*
$\tau \cup (\sum\lambda \in (\epsilon \in \kappa))$
where,

$\tau \equiv \sigma$:UML $\cup$ ($\sum\epsilon \in \psi$: Safety UML Profile), i.e., Architectural model σ containing all the roles ϵ that compose the safety pattern ψ being applied, but without presenting connections among the role instances.

$\lambda \equiv$ one connection originated in the element representing a specific role ϵ that is part of an execution channel κ.

This Pattern Descriptive Rule is read as: *The architectural model with all the roles ϵ that compose the safety pattern ψ being applied is modified by the insertion of all the connections λ originated in the element that represents a specific role ϵ, which, in turn, is part of an execution channel κ.*

For example, consider the PSC pattern (cf. Figure 4). The rule that represents the *Input Processing* role of this pattern and its execution channel is:

rule *Input Processing of the PSC pattern Rule:*
M1:UML $\cup$ ((Input Processing $\in$ Channel A) $\in$ PSC pattern: Safety UML Profile)

This rule states that a UML element, stereotyped with *Input Processing* and tagged with *Tag Name* = ***ExecutionChannel***, and ***Tag Value* = *Channel A***, must be introduced in the UML model M1 in the application of the PSC pattern. It is worth emphasizing that every role that composes the PSC pattern has a rule like this one. Consider now the existence of a model M2, containing six UML elements, one for each role that composes the PSC pattern. The Pattern Descriptive Rule that represents connections originated in the *Input Processing* role is:

rule *Connections of the Input Processing of PSC pattern:*
M2 $\cup$ (Connections Input Processing(Channel A): Data Transformation(Channel A), Data Validation(Channel A)), where:

M2 = (M1 $\cup$ ($\sum$Roles $\in$ PSC pattern: Safety UML Profile))

This Pattern Descriptive Rule states that the model M2 is modified by adding the connections between the elements representing the *Input Processing* role and the elements that receive its output: Connection 1 = Input Processing and Data Transformation; Connection 2 = Input Processing and Data Validation. In this case, all the role are under the same execution channel. However, it is important to have such indication for the case of association of roles in different execution channels.

The graphical representation of safety patterns with stereotypes instances is an appropriate front-end that allows engineers to reason on the safety pattern structure (roles and connections) in terms of abstract functional entities. Moreover, it provides means to state safety specificities (fault classes, services failure modes, and safety tactics) that are singular to each safety pattern. Our Pattern Descriptive Rules state actions that foresee the automatized application of safety patterns in architectural models, providing fundamental highlights on how the artifacts necessary to perform the complete pattern application using Model Transformation mechanisms should look like. When combining the graphical representation with the Pattern Descriptive Rules, engineers have means to represent the pattern, taking in consideration not only the pattern design, but also explicitly indicating application constraints.

V. Representing the Homogeneous Redundancy Pattern with our Approach

We have represented with our approach the safety patterns proposed by Douglass [3] and observed that they become easier to represent and reuse. Due to space constraints, however, for this section, we selected the Homogeneous Redundancy pattern to be represented with our approach. This pattern is composed of seven roles: *Input Sensor, Input Processing, Data Transformation, Data Validation, Output Processing, Actuation Validation,* and *Actuator*. Consider that initially, the *Safety UML Profile* contains only three stereotypes: ≪Fault Class≫, ≪Service Failure Mode≫, and ≪Safety Tactic≫. The first step is to create stereotypes on it that represent the roles of the Homogeneous Redundancy pattern in the *Safety UML Profile*, as shown in Figure 6. At this point, each role has an associated documentation providing general information of it, as can be seen in Table I.

Role Name	Role Description
Input Sensor	This is the source of the information used to control the actuator.
Input Processing	Acquires and performs the first processing on the data sent by the Primary Input Sensor.
Data Transformation	Performs a single transformation step on the input data.
Data Validation	Validates if the data is correct or reasonable, and also stops the processing on the current channel and begins it on the second channel when a fault is detected.
Actuation Validation	Compares the output to the commanded output, and determines when some application specific fault occurs.
Output Processing	Performs the last stage of the data transformation, and controls the Actuator.
Actuator	This is the device performing the actuation. This is the actuator used by default.

Table I
Description of roles that compose the Homogeneous Redundancy Pattern.

After having the roles available in the Safety UML Profile, the pattern structure is created in a separate diagram using instances of these stereotypes (cf. Figure 7). To clearly differentiate between the execution channels that the roles are part of, the roles in light gray are part of the *Primary Actuation Channel*, the ones in dark gray are part of the *Secondary Actuation Channel*, and the white elements are sensors and actuators that are not part of any channel. The roles that are part of the *Primary Actuation Channel* are tagged with *Tag Name* ≡ ***ExecutionChannel***, and *Tag Value* ≡ ***Primary Actuation Channel***. The roles that are part of the *Secondary Actuation Channel* are tagged with *Tag Name* ≡ ***ExecutionChannel***, and *Tag Value* ≡ ***Secondary Actuation Channel***. The sensors and actuators are tagged with *Tag Name* ≡ ***ExecutionChannel*** and *Tag Value* ≡ ***null***. The three classes on the bottom left side of Figure 7 represent the Fault Class, Service Failure Mode, and Safety tactic related to the Homogeneous Redundancy pattern, which are Random Fault, Content and Timing, and Functional Redundancy, respectively.

For each role that composes the Homogeneous Redundancy pattern, there is one rule representing the role and its execution channel, and one rule describing to which other roles its output flows are directed. For this example, consider the UML model called *SourceModel* as the original model where the Homogeneous Redundancy pattern is to be applied, and another UML model called *TargetModel* as the model that already contains elements representing every role of the Homogeneous Redundancy pattern. The rule representing the role *Input Sensor* (so called Primary Input Sensor) associated with the Primary Actuation Channel is:

rule *Primary Input Sensor of Homogeneous Redundancy:*

SourceModel:UML ∪ *((Primary Input Sensor* ∈ *null)* ∈ *Homogeneous Redundancy pattern: Safety UML Profile)*

This Pattern Descriptive Rule is read as: *The architectural UML model SourceModel is modified by the insertion of an UML element that represents the role Primary Input Sensor, which is tagged with Tag Name =* ***ExecutionChannel****, and* ***Tag Value*** *=* ***null****, and composes the safety pattern Homogeneous Redundancy.* It is worth highlighting that the

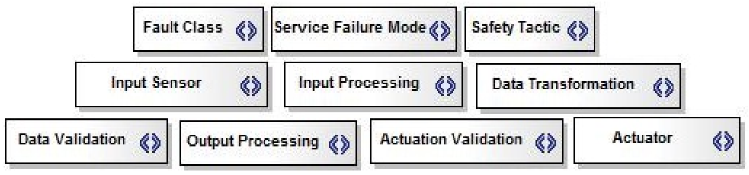


Figure 6. Safety UML Profile with the Roles of the Homogeneous Redundancy pattern.

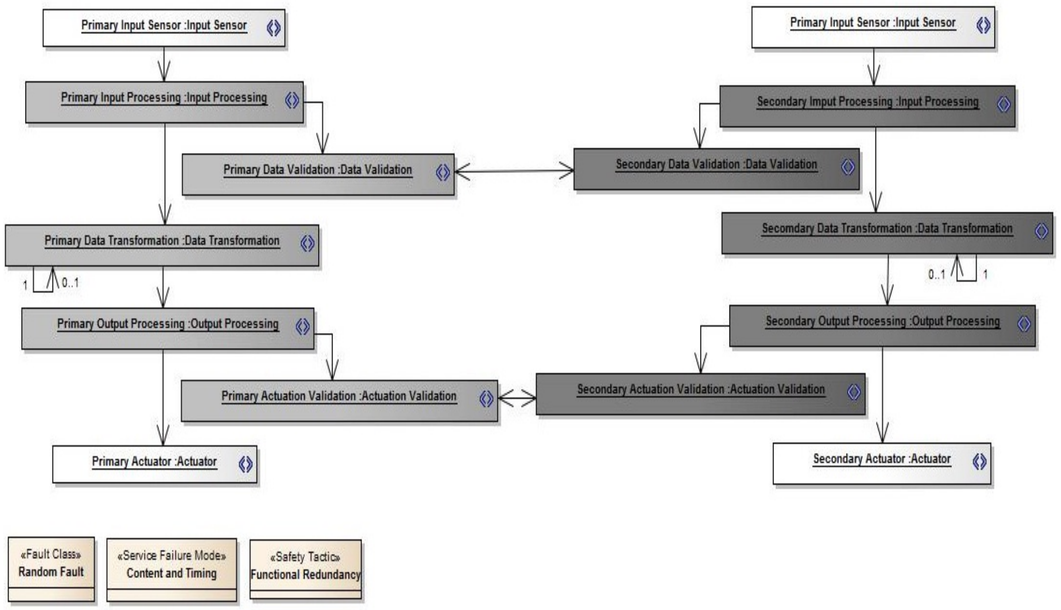

Primary Input Sensor :Input Sensor
Primary Input Processing :Input Processing
Primary Data Validation :Data Validation
Primary Data Transformation :Data Transformation
1
0..1
Primary Output Processing :Output Processing
Primary Actuation Validation :Actuation Validation
Primary Actuator :Actuator
Primary Input Sensor :Input Sensor
Secondary Input Processing :Input Processing
Secondary Data Validation :Data Validation
Secomdary Data Transformation :Data Transformation
0..1
1
Secondary Output Processing :Output Processing
Secondary Actuation Validation :Actuation Validation
Secondary Actuator :Actuator
«Fault Class»
Random Fault
«Service Failure Mode»
Content and Timing
«Safety Tactic»
Functional Redundancy

Figure 7. Homogeneous Redundancy pattern modeled with instances of stereotypes defined in the *Safety UML Profile*.

tag ***ExecutionChannel*** has a value ***null*** because it is not part of any Execution channel.

The following Pattern Descriptive Rule is related to the Primary Input Processing, and is read as: *The architectural UML model SourceModel is modified by the insertion of an UML element that represents the role Primary Input Processing, which is tagged with Tag Name =* ***ExecutionChannel****, and* ***Tag Value = Primary Actuation Channel****, and composes the safety pattern Homogeneous Redundancy.*

rule *Primary Input Processing of Homogeneous Redundancy:*

SourceModel:UML $\cup$ *((Primary Input processing* $\in$ *Primary Actuation Channel)* $\in$ *Homogeneous Redundancy pattern: Safety UML Profile)*

The rules for the others roles that are part of the Primary and Secondary Actuation channel, as well as the Secondary Input Sensor and both actuators, follow the same construction principle. For instance, the rules representing the Secondary Data Validation role and the Secondary Actuator are as follows:

rule *Secondary Data Validation of Homogeneous Redundancy:*

SourceModel:UML $\cup$ *((Secondary Data Validation* $\in$ *Secondary Actuation Channel)* $\in$ *Homogeneous Redundancy pattern: Safety UML Profile)*

This Pattern Descriptive Rule is read as: *The architectural UML model SourceModel is modified by the insertion of an UML element that represents the role Secondary Data Validation, which is tagged with Tag Name =* ***ExecutionChannel****, and* ***Tag Value = Secondary Actuation Channel****, and composes the safety pattern Homogeneous Redundancy.*

rule *Secondary Actuator of Homogeneous Redundancy:*

SourceModel:UML $\cup$ *((Secondary Actuator* $\in$ *null)* $\in$ *Homogeneous Redundancy pattern: Safety UML Profile)*

The Pattern Descriptive Rule above, which describe the Secondary Actuator of Homogeneous Redundancy pattern, is read as: *The architectural UML model SourceModel is modified by the insertion of an UML element that represents the role Primary Actuator, which is tagged with Tag Name =* ***ExecutionChannel****, and* ***Tag Value = null****, and* *composes the safety pattern Homogeneous Redundancy.* As with the Primary Input Sensor peviously mentioned, the tag ***ExecutionChannel*** has the value ***null*** because it is not part of any Execution channel.

For describing the rules that represent the connections of the roles of the Primary and Secondary Actuation Channel, as well as the Input Sensors and Actuators, consider the *SourceModelWithRoles* as the original source model *(SourceModel)* modified by the addition of representatives for every role of the Homogeneous Redundancy pattern (or, using our notation, SourceModelWithRoles = (SourceModel $\cup$ ($\sum$Roles $\in$ Homogeneous pattern: *Safety UML Profile*))).

The following Pattern Descriptive Rule represents the connections originated in the Primary Input Sensor that connects it with the Primary Input Processing, and is read as: *The architectural model containing all the roles that compose the safety pattern Homogeneous Redundancy is modified by the insertion of the connections that have origin in the element that represents the role Primary Input Sensor, and that link it with the element that represents the role Primary Input Processing, which, in turn, is part of the Primary Execution Channel.*

rule *Connections of the Primary Input Sensor of Homog. Redundancy Pattern:*
SourceModelWithRoles $\cup$ *(Connections Primary Input Sensor: Primary Input Processing (Primary Actuation Channel)*

The rule representing the connections originated in the Primary Input Processing is:

rule *Connections of the Primary Input Processing of Homog. Redundancy Pattern:*
SourceModelWithRoles $\cup$ *(Connections Primary Input Processing: Primary Data Transformation (Primary Actuation Channel); Primary Data Validation (Primary Actuation Channel))*

This rule is read as: *The architectural model containing all the roles that compose the safety pattern Homogeneous Redundancy is modified by the insertion of the connections that have their origin in the element that represents the role Primary Input Processing (part of the primary execution channel), and that link it with the elements that represent the role Primary Data Transformation and the role Primary Data Validation, which are both part of the Primary Execution Channel.*

The rule that represents the connections originated in the Primary Data Validation is:

rule *Connections of the Primary Data Validation of Homog. Redundancy Pattern:*
SourceModelWithRoles $\cup$ *(Connections Primary Data Validation: Secondary Data Validation (Secondary Actuation Channel))*

This rule should be read as: *The architectural model containing all the roles that compose the safety pattern Homogeneous Redundancy is modified by the insertion of the connections that have their origin in the element that represents the role Primary Data Validation (part of the primary execution channel), and that link it with the element that represents the role Secondary Data Validation, which, in turn, is part of the Secondary Execution Channel.*

As our approach requires two rules per role that compose a safety pattern (one rule describing the role itself and its execution channel, and another one describing the connections originated in the role), the complete representation of the Homogeneous Redundancy pattern consists of the model shown in Figure 7, and additional twenty eight Pattern Descriptive Rules. Therefore, due to space limitation it is not possible to show all the rules. However, as already mentioned, they look similar to the ones previously shown, with the appropriate changes of the connected roles instances and the actuation channel they are part of.

It is important to emphasize that the rules described in this work are not enough to ensure the application of a safety pattern in an architectural model. Actually, our Pattern Descriptive rules are to Model Transformation rules as algorithms are to Imperative languages. It means that they are language independent and, when considered together with the information available in the graphical model (for instance Figure 7), offer the basis for constructing model transformation rules in languages like TRL or ATL [27], which will ensure the automatic application of safety patterns in architectural models of safety critical systems, by means of model transformation mechanisms.

VI. Conclusion and Future Works

This work was motivated by the lack of ways to represent safety patterns in a way that engineers can reason on the safety patterns composition in terms of the functional entities that comprise them, and on the safety patterns constraints that should be considered when applying them in architectural models. To fill this gap, we have proposed a safety pattern representation approach that consists of the joint use of: (i) a graphical representation of safety pattern using elements defined in a UML profile called *Safety UML Profile*; and (ii) a set of pattern descriptive rules that describe each role participation in the safety patterns, its execution channel, and the connections of each role.

Being able to express safety patterns with elements of a UML profile and pattern descriptive rules in a unified fashion is the first step in our attempt to perform safety pattern applications in architectural models of safety-critical systems by means of Model Transformation mechanisms.

Acknowledgements: This work is supported by the Fraunhofer-Innovation Cluster Digitale Nutzfahrzeugtechnologie (Digital Engineering for Commercial Vehicles).

References

[1] P. Liggesmeyer and M. Trapp, “Trends in embedded software engineering,” *IEEE Software*, vol. 26, no. 3, pp. 19–25, May 2009.

[2] W. Wu and T. Kelly, “Safety tactics for software architecture design,” in *in Proceedings of the 28th Annual International Computer Software and Applications Conference, (Hong Kong, 2004)*. IEEE Computer Society, 2004, pp. 368–375.

[3] B. P. Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] O. M. G. Group. UML specification, version 2.0.

[5] N. Rozanski and E. Woods, *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley Professional, 2005.

[6] L. Fuentes-Fernández and A. Vallecillo-Moreno, “An introduction to uml profiles,” *Journal of UML and Model Engineering*, vol. 2, 2004.

[7] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[8] A. Armoush, “Design patterns for safety-critical embedded systems.” Ph.D. dissertation, RWTH Aachen University, 2010, http://d-nb.info/1007034963.

[9] D.-K. Kim, R. France, S. Ghosh, and E. Song, “A role-based metamodeling approach to specifying design patterns,” *Computer Software and Applications Conference, Annual International*, p. 452, 2003.

[10] A. H. Eden, Y. Hirshfeld, and A. Yehudai, “Lepus - a declarative pattern specification language,” Department of Computer Science, Tel Aviv University, Tech. Rep., 1998.

[11] P. Selonen, M. Siikarla, K. Koskimies, and T. Mikkonen, “Towards the unification of patterns and profiles in uml,” *Nordic J. of Computing*, vol. 11, no. 3, pp. 235–253, Sep. 2004.

[12] A. Avižienis, J. . Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[13] F. Bachmann, L. Bass, and M. Klein, “Deriving architectural tactics: A step toward methodical architectural design,” *Software Engineering Institute (SEI), Technical Report, No. CMU/SEI-2003-TR-004*, 2003.

[14] J. C. Knight, “Safety critical systems: challenges and directions,” in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE ’02. New York, NY, USA: ACM, 2002, pp. 547–550.

[15] C. Alexander, *The Timeless Way of Building*. Oxford University Press, 1979.

[16] B. Appleton, “Patterns and software: essential concepts and terminology,” *Object Magazine Online*, vol. 3, 1997.

[17] P. Avgeriou and U. Zdun, “Architectural patterns revisited - a pattern language,” in *Proceedings of the 10th European Conference on Pattern Languages of Programs (EuroPLoP 2005)*, Irsee, Germany, Jul. 2005.

[18] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 1994.

[19] J.-M. Rosengard and M. Ursu, “Ontological representations of software patterns,” *Proceedings of KES’04*, 2004.

[20] J. K.-H. Mak, C. S.-T. Choy, and D. P.-K. Lun, “Precise modeling of design patterns in uml.” in *ICSE’04*. Scotland: IEEE Computer Society, 2004, pp. 252–261.

[21] A. L. Guennec, G. Suny, and J.-M. Jzquel, “Precise modeling of design patterns.” in *UML*, ser. Lecture Notes in Computer Science, A. Evans, S. Kent, and B. Selic, Eds., vol. 1939. Springer, 2000, pp. 482–496.

[22] B. P. Douglass, *Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[23] L. L. Pullum, *Software fault tolerance techniques and implementation*. Norwood, MA, USA: Artech House, Inc., 2001.

[24] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[25] R. Hanmer, *Patterns for Fault Tolerant Software*. Wiley Publishing, 2007.

[26] M. Tichy and H. Giese, “Extending fault tolerance patterns by visual degradation rules,” in *Proc. of the Workshop on Visual Modeling for Software Intensive Systems (VMSIS) at the the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC’05), Dallas, Texas, USA*, 2005.

[27] K. Czarnecki and S. Helsen, “Classification of model transformation approaches,” in *OOPSLA03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

Process Modeling-based Assessment of Software Release Planning

Case Study Results and Experiences

Jos J.M. Trienekens
University of Technology Eindhoven
Eindhoven, The Netherlands
j.j.m.trienekens@tue.nl

Robbert Slooten
Philips Eindhoven
The Netherlands
slooten.robbert@gmail.com

***Abstract*- In the software industry, customer input often takes the form of improvement requests. Release planning is the process of making decisions about what new functionalities or changes will be implemented in which release of a software product. The purpose of this work in progress paper is to explore a new approach to assess and improve the release planning process in industrial software companies. The new approach consists of a combined application of software process modeling, assessment and improvement. This paper presents results and experiences from a case study in two industrial companies.**

Keywords-release planning; assessment; process modeling.

I. Introduction

An increasing part of the software produced is aimed at being offered to a general marketplace rather than to one specific customer. This type of software development is called market-driven software product development (MDSPD) [9]. In the software industry, customer input often takes the form of improvement requests. Improvement requests and other system complaints often result in an abundance of requirements [2]. Yet, often too little resources are available to implement all requirements at the same time. Proper release planning is as complex as it is important for the success of a software product [10] .

The purpose of this paper is to explore a new approach to assess and improve the release planning process in industrial software companies. This approach consists of a combined application of business process modelling, [5] , and process assessment [11]. Instead of a generic CMMI model for process assessment and improvement a more focused maturity model is used, i.e., the SPM Maturity Model. This maturity model is dedicated to software product management and addresses particular software processes such as the release planning process. In Section 2, we will first address the background of the release planning process, and the motivation for the new approach. Section 3 will present the case study and selected results and experiences of the new approach. Section 4 finalizes the paper with conclusions and further work to be done.

II. Background release planning and prioritisation

Release planning aims at selecting an optimal subset of features that satisfy as many stakeholders as possible within the budget, resource and risk constraints [10]. Many different aspects can be taken into account when prioritizing requirements. However, involving multiple aspects complicates the decision process.

Regarding the assessment and improvement of software processes, the Capability Maturity Model Integration (CMMI) is a well-known approach [7]. Applications of CMMI in Small and Medium Sized Enterprises (SMEs) have shown several problems, such as: the implementation is too complex, too time-consuming, too costly etc., see e.g., [8]. In our case study we therefore have selected a more focused assessment approach that has as scope software product management. Four main processes, are being distinguished, respectively requirements management, release planning, product planning and portfolio management. For the release planning process a number of focus areas have been defined, see Table I. Capabilities of each of the focus areas are represented by A to F in the rows. To progress through maturity levels 1 to 10, the capabilities of each focus area indicated in each respective column must be achieved. E.g., to reach maturity level 2 the capability A of Launch preparation should be achieved. Moving from left to right through the matrix matures the SPM processes. Progressing from maturity level 1 to 10, the focus areas are revisited multiple times maturing them incrementally as well. An important aspect in an assessment is the collection of information to rate the capabilities. Currently this is done by asking questions to practitioners and by studying project documents. For all the capabilities in the matrix, a company has to answer yes or no to the question ‘Is this capability implemented in your organisation?’ Advantages of using a questionnaire are that it can be distributed easily to a wide range of respondents across geographic boundaries, it is non-invasive, and it is cost and time efficient. However, there are also serious disadvantages to using questionnaires in software process assessment, respectively: questionnaires have been found to be repetitive and verbose, questions have been found to not be related to the real problem, qualitative information is needed to reflect the software process and finer granularity than yes/no questions is needed to reflect the software process [8]. Regarding our release planning assessment case study, we decided that our assessment method should not only be aimed at collecting answers to yes/no questions, but should aim at an in-depth analysis and discussion of the ‘actual’ release planning process as it is carried out in practice. Given this aim, and the disadvantages of questionnaires listed above, we applied a different assessment instrument: i.e., formal process modelling techniques. These modeling techniques have already been recognized for many years as instruments, e.g., [3][4] which both conclude that 'software process modeling facilitates human understanding and supports process improvement'. Also, research on success factors of software process improvement shows that an inhanced understanding of the process by employees, and employee involvement (because of their insight into and knowledge of the process areas), are of utmost importance for the success of a software process improvement project. To select an appropriate modelling technique we first defined the purpose of modelling. Our purpose was to describe the current processes, to determine the current process maturity, and to formulate improvement proposals. This is classified in the process modeling area as 'working towards a descriptive model for learning and process development' [1]. A number of techniques can be used for this type of modelling. In our situation, based on the characteristics from a user and a modeller perspective, formal flowcharts and data flow diagrams have been chosen. Subsequently, we selected Business Process Modelling Notation (BPMN) as our modelling language. BPMN is able to capture aspects of both flowcharts and DFDs (based on the Bunge-Wand-Weber (BWW) representation model), We used the BPMN notation, i.e., its extended modelling elements, as described in [6]. BPMN is already for quite some time recognized as the most ontological complete model [12]. Lastly, it has to be stated that our choice for BPMN was also motivated by current knowledge and expertise in our research group at TU/e,), e.g., [5]. Software process modelling facilitates human understanding and communication and supports process assessment and improvement. Business process modelling is an iterative method that provides rich information on how processes are implemented. Summarizing, our new process assessment approach consists of the application of process modelling techniques to derive information from the actual processes for the rating of the capabilities (in a SPM Maturity Matrix) of a release planning process.

III. Release Planning assessment and improvement

A. The case study content

The case study addressed in this paper was carried out at two companies (A and B) active in providing telematics solutions to the transportation industry. Companies in this transport and logistics domain are facing major challenges. Competition is large, operating costs are increasing rapidly, customers demand an ever-expanding range of services and the legislation is strict. As more information becomes available digitally, inside the truck as well as outside, data from an expanding number of sources needs to be processed and integrated. This increases the complexity of full service telematics solutions considerably.

Company B is a company that provides telematics solutions to the transportation industry. It has over 280 employees spread throughout Europe. More than 80.000 devices have been installed. It provides products to acquire mobile operations data and communicate this data between dispatch/shipment control, back-end information systems, and mobile shipping unit (e.g., a truck or trailer, etc.). Company A, a SME, has recently been taken over by company B. Release planning at company A is mainly done by a small group of high-positioned executives, and no formal process descriptions exist. At company B however, processes are more defined and certain formal protocols are in place. The different release planning processes in the distinct companies have to be aligned and should finally smoothly flow together. Preserving the best of both ways of the release planning processes of both companies will contribute to a more efficient process and a higher quality of release planning.

B. Process modelling and maturity rating

Process models of the current release planning processes at both companies have been made. In the context of this work in progress paper examples of the process models are not presented here. However, we will clarify in what way we discussed these process models and how we derived information from them to determine the capabilities (maturity) of the release planning processes. The modeling process was identical at both companies. In a first session with the persons responsible for the release planning process a list of persons of interest was put together, not limited to those who are involved directly in the process. Persons involved in requirements gathering and the product development process in general were thus included on the list. After interviews with these persons were completed, the gathered information was used to build a first version of the process models. Subsequently, the process models were used to guide a second iteration of data collection. Again, information was gathered that was now used to adjust the process models created in the first iteration. This process continued until the company supervisors agreed with the process models build in the latest iteration. Within company A, the first session with the company supervisor resulted in a list of 9 persons. Within company B, the initial list of persons of interest included 7 persons. The initial discussion sessions were all conducted using a semi-structured discussion protocol. The discussion sessions ranged in duration from 30 minutes to one hour on average. The process modelling effort at both companies shows that input from multiple stakeholders and an iterative approach is needed to get accurate process models. Using the input from multiple stakeholders singled out contradicting views on a process. In the discussion sessions we focused on a number of process model characteristics such as: the release planning focus areas recognized or implemented, importance of decisions made in process steps, type of and frequency of (formal) techniques and tools used, (internal and external) stakeholders involved, etc.

From the discussion sessions on the release planning process models of company A, we summarize two types of results. On one hand, we will present the identification of capability ratings for the distinct release planning focus areas, and on the other hand we will address interesting benefits of the usage of the process models. First, see Table II (underlined ratings), the capability ratings:

- A grouping based prioritisation method is often used, so capability B of requirements prioritisation is achieved.
- Based on formal cost-benefit criteria for prioritisation level D is also achieved. Level A and C are skipped since not all internal stakeholders provide input on the priorities and external stakeholders are not involved in the prioritisation process.
- An open-ended release planning technique is used, and no formal release definition is formed. As a consequence none of the capabilities of release definition are achieved.
- There exists hardly a release definition. This has as consequence: none of the capabilities for release definition validation have been achieved.
- No formal scope change management exists. When development on projects of features turns out to be underestimated it can be chosen to simply not release yet or exclude it from the new version.
- New products are first thoroughly tested internally and externally, thus achieving capability A and B for build validation.
- Internal stakeholders are informed about a new release by means of a formal release document. Capability A in launch preparation is thus achieved. However, the release decision is not based on formal quality rules, and therefore capability B is not achieved.

Regarding the benefits of using process models in company A we can give the following example: based on the multiple iterations of discussing process models with different stakeholders, it appeared that alternative routings were possible in the initially developed process models, e.g., request handlers often consulting sales managers for a second opinion regarding their decisions. Only because of these iterations important business process details could be identified. However, it has to be stated that in total, the release planning process at company A didn't strongly benefit from using process models. Although the various process models were slightly adjusted in subsequent iterations, this didn't lead to big changes in the maturity ratings.

Regarding the capability ratings, from the discussion sessions in company B, we give the following results; see Table III.

- Regarding requirements prioritisation, internal stakeholders are being involved and a formal prioritisation method is often used. Also a cost-revenue consideration is being made. As a consequence levels A, B and D have been achieved. Because customers are not being involved in the process, C is skipped.
- The release proposal compiled by the Product Management fits the descriptions of capability A and B of release definition.
- The release definition has to be approved by a formal Strategic Product Board thus achieving capabilities A and B of release definition validation. Capability C is not achieved because the release definition is not communicated to internal stakeholders.
- Having implemented a formal project management methodology, the company meets capabilities A and B of scope change management.
- Regarding the release preparation process a tool has been developed called the 'release clock'. The release clock defines that new features are first tested internally and consequently in a field test, thereby achieving levels A and B of build validation.
- The 'release clock' also defines a rigorous launch preparation process which includes all the capabilities of launch preparation.

Regarding the experienced benefits of the usage of process models in company B we also experienced the advantages of the multiple iterations in discussing the process models with different stakeholders. For example, in company B, it became clear, only after a couple of iterations that an automated requirements gathering system was hardly used in practice and was not favoured by most of the employees. As a consequence, we decided to exclude this system, which was formally specified in the company process standard, from the process models. However, also in company B it appeared that the usage of process models also can have disadvantages. E.g., the 'build validation and launch preparation processes' would have required very much effort to capture in process models, while the existing and available process documentation was of sufficient quality to base the maturity rating on. Summarizing, we can state that the usage of process models has clear advantages in particular process areas, but when applied 'at random' to any process it also can have serious drawbacks, e.g., too time-consuming. too cost-ineffective.

IV. Conclusions and future work

The development of software process models in the case study provided detailed and in-depth information on the capabilities of the release planning focus areas. The maturity matrices of the release planning process of both companies could be completed in a reliable and detailed way. In particular, the iterations in the discussions on the process models, with the different stakeholders, resulted in renewed insights into the current practice, e.g., with respect to existing but unused tools. Of course the detailed arguments for the capability ratings are an important add-on to the Maturity Matrices in Table II and Table III. They clarify important and detailed aspects of the release planning processes. The use of a focus area oriented SPM Maturity Model, instead of a generic software process maturity approach such as CMMI, proved to have strong advantages regarding the detailed information of strengths and weaknesses of the release planning process and its focus areas.

However, the usage of process models also showed sometimes disadvantages. In particular, we mention here the effort which is needed to develop (and maintain) the process models. From our case studies we conclude that the appropriateness of using process models depends on particular characteristics of the process and the business context, e.g., the complexity, the dynamics, and the number of stakeholders involved in a process. In further research, we will focus on these characteristics in order to become able to determine the suitability of using process models to particular types of processes and to different types of business situations. In that way, we will strive at a more efficient usage of process modeling for software process maturity rating.

REFERENCES

[1] Aguilar-Savén, R. (2004). Business process modelling: Review and framework. International Journal of Production Economics, Vol. 90, pp. 129-149.

[2] Bagnall A., Rayward-Smith V. and Whittley, I. (2001).The Next Release Problem. Information and Software Technology, Vol. 43, No. 14, pp. 883 - 890.

[3] Bollinger, T. and McGowan, C. (1991), A Critical Look at Software Capability Evaluations. IEEE Software, pp. 25-41.

[4] Curtis, B., Kellner, M. and Over, J., (1992). Process Modelling. Communications of the ACM, Vol. 55, No. 9, pp. 75-90.

[5] Dijkman R.D., M. Dumas and C. Ouyang, Semantics and analysis of business process models in BPMN, Information and Software Technology, Volume 50, Issue 12, November 2008, pp. 1281-1294.

[6] OMG (2011). Business Process Model and Notation (BPMN). Object Management Group, Inc.

[7] Paulk, M. (2002). Capability Maturity Model for Software.Encyclopedia of Software Engineering.

[8] Pino, F., García, F. and Piattini, M. (2008). Software process improvement in small and medium software enterprise: a systematic review. Software Quality Journal, Vol. 16, pp. 237 - 261.

[9] Regnell, B. and Brinkkemper, S. (2005). Market-driven requirements engineering for software products. In A. Aurum, & C. Wohlin, Engineering and Managing Software Requirements, pp. 287 - 308. Berlin Heidelberg: Springer.

[10] Saliu, O. and Ruhe, G. (2005). Supporting Software Release Planning Decisions for Evolving Systems.Proceedings of the 2005 29th Annual IEEE/NASA Software Engineering Workshop (SEW'05), pp. 14- 24. Greenbelt, MD.

[11] Van de Weerd, I., Bekkers, W. and Brinkkemper, S. (2010). Developing a Maturity Matrix for Software Product Management.ICSOB, Lectures Notes in Business Information Processing, Vol. 51, pp. 76 – 89.

[12] Weber, R. (1997). Ontological Foundations of Information Systems. Melbourne: Coopers & Lybrand.

TABLE I Example of a Release Planning Part of the Maturity Matrix for Software Product Management (Van de Weerd et al, 2010)

	0	1	2	3	4	5	6	7	8	9	10
Release planning process											
Requirements prioritisation			A		B	C	D			E	
Release definition			A	B	C				D		E
Release definition validation					A			B		C	
Scope change management				A		B		C		D	
Build validation					A			B		C	
Launch preparation		A		B		C	D		E		F

TABLE II. RELEASE PLANNING PROCESS MATURITY AT COMPANY A

	0	1	2	3	4	5	6	7	8	9	10
Release planning process											
Requirements prioritisation			A		**B**	C	**D**			E	
Release definition	▨	▨	A	B	C				D		E
Release definition validation	▨	▨	▨	▨	A			B		C	
Scope change management	▨	▨	▨	A		B		C		D	
Build validation	▨	▨	▨	▨	**A**	▨	▨	**B**	▨	C	
Launch preparation	▨	**A**	▨	B		C	D		E		F

TABLE III. RELEASE PLANNING PROCESS MATURITY AT COMPANY B

	0	1	2	3	4	5	6	7	8	9	10
Release planning process											
Requirements prioritisation	▨	▨	**A**	▨	**B**	C	**D**			E	
Release definition	▨	▨	**A**	**B**	C				D		E
Release definition validation	▨	▨	▨	▨	**A**	▨	▨	**B**		C	
Scope change management	▨	▨	▨	**A**	▨	**B**		C		D	
Build validation	▨	▨	▨	▨	**A**	▨	▨	**B**	▨	C	
Launch preparation	▨	**A**	▨	**B**	▨	**C**	**D**	▨	**E**	▨	**F**

Distributed Software Framework

For Biosphere 2 Land Evolution Observatory (LEO) Autonomic Cyber-Physical System (ACPS)

Shafiul Islam
The Department of Electrical and Computer Engineering
The University of Arizona
Tucson, U.S.A.
jacky@email.arizona.edu

***Abstract*— This paper presents the architecture, design, and implementation of a real-time Distributed Software Framework for Biosphere 2 Land Evolution Observatory Autonomic Cyber-Physical System, which uses an optimum technology mix discovered through intensive research, design, and development over a period of two years (2010- 2012) using a novel adaptable process framework named as Jacky's Universal Process. It has a Service Oriented Architecture with Publish/Subscribe interaction pattern and Object Oriented Design. It applies self-healing feature of Autonomic Computing and uses Cloud Computing and OpenSplice Data Distribution Service. The distributed system software (software + service) of this framework is a complete production quality software product that requires near zero maintenance since only sensor drivers for new sensor types need to be developed and by appropriately mixing and matching the services all required system level capabilities can be provided. This framework deployed on B2 server, is capable of handling 45x the expected load having a total of about 148,500 sensors. It is highly reliable, robust, fault tolerant, scalable (both vertically and horizontally), extensible, secured, and easy to use. It successfully resolves all technological risks, provides concept consistency, and supersedes the functional and non-functional requirements.**

Keywords-distributed software framework; data distribution service; service oriented architecture; autonomic computing, jacky's universal process.

I. INTRODUCTION

The Biosphere 2 Land Evolution Observatory (LEO) is an interdisciplinary project aimed to quantify various earth and atmospheric processes to understand the complex non-linear interaction among these processes by coupling controllable physical systems with numerical models of the interacting processes using a cyber-physical system, which is a specialized cyberinfrastructure (CI) for LEO and referred to as Autonomic Cyber-Physical System (ACPS). ACPS requires a highly reliable, robust, fault-tolerant, scalable, extensible, and easy to maintain real-time Distributed Software Framework (DSF) with a life span of about 10 years that can be deployed on any heterogeneous distributed system and resolve integration risks. The primary users of this framework are researchers and engineers interested in development of scientific domain specific applications and computational models, which in turn are meant to be used by scientists and students for research and education (e.g., CI for Atmospheric Sciences, Earth Sciences, and Engineering Research) as mentioned in NSF's CI vision for 21st century discovery [2].

In order to meet the challenging requirements and resolve technological risks, autonomic computing, cloud computing, service oriented architecture (that uses publish/subscribe interaction pattern), and object-oriented design were used. The optimum technology mix was discovered through intensive research, design, and development over a period of two years (2010-2012). In this paper, the final production quality architecture, design, and implementation of ACPS DSF for Biosphere 2 LEO are presented. A secondary outcome of this research and development effort, also introduced in this paper, is the inception of a novel adaptable process framework for software engineering of self-managing distributed systems, which is named as Jacky's Universal Process (JUP).

The rest of the paper is organized as follows: Section II provides a theoretical foundation through literature review; Section II explains the methodology used; Section IV lists the requirements, and discusses the architecture and design; Section V discusses testing, and results; Section VI discusses experimentation and results; and finally, Section VII describes conclusion and future work.

II. LITERATURE REVIEW

At present, no similar distributed software framework for cyber physical systems that use autonomic computing exist. Hence the fundamental concepts are briefly presented here to provide the theoretical foundation.

A. *Distributed Systems*

A collection of independent systems that appear as a single coherent system is called a distributed system [3], which has key goals of achieving reliability, availability, adaptability, expandability, scalability, robustness, and fault-tolerance (through redundancy) while providing distribution transparency [4]. ACPS DSF, being a distributed software framework, naturally provides the non-functional needed by Biosphere 2 LEO.

B. Autonomic Computing

The overall goal of Autonomic computing, modeled upon autonomous nervous system, is that computing systems will self-manage taking only high-level objectives from administrators (human beings) [5]. The four aspects of self-management are: 1) Self-configuration; 2) Self-optimization; 3) Self-healing; and 4) Self-protection. In ACPS DSF, self-healing aspect is implemented to provide fault tolerance for critical system-level services, and monitoring and notification for sensors of the physical system.

C. Cloud Computing

Data center hardware and software is known as a cloud [6]. Using a composability methodology, cloud computing systems can be classified into any of the five layers [7]: 1) Cloud Application Layer (SaaS); 2) Cloud Software Environment Layer (PaaS); 3) Cloud Infrastructure Layer (IaaS); 4) Software Kernel; 5) Hardware and Firmware. For example, Amazon EC2 is IaaS, Google AppEngine SaaS, and Microsoft Azure is PaaS [6]. ACPS DSF system-level services need to be run locally on Biosphere 2 servers to avoid latency issues as experienced during testing on Amazon EC2. However, ACPS DSF application-level software can easily be deployed to a cloud.

D. OMG Data Distribution Service

Object Management Group (OMG) Data Distribution Service (DDS) is an open specification for publish-subscribe (PS) data distribution systems [8] that attempts to provide formal definition for defining Quality of Service (QoS) to configure service and help connect information producers (publishers) with information consumers (subscribers). Many real-time applications, including ACPS DSF, have the need to have pure data-centric architectural pattern and take advantage of DDS. OpenSplice [9] is the most advanced, complete and widely used (commercial and open source) implementation of OMG DDS specification. This is a tried and tested commercial-of-the-shelf product and was chosen for ACPS DSF as the OMG DDS implementation of choice.

III. METHODOLOGY

This project was primarily a complex large-scale interdisciplinary engineering project with intensive technology research. The greatest risks in the project were the technical risks and the greatest challenge was to maintain conceptual integrity among interdisciplinary Biosphere 2 staff members. The Spiral model was applied when the project was completely risk driven. As critical risks were resolved by incorporating new technologies like OpenSplice DDS and as development moved from middleware / distributed system software towards distributed application software, the approach moved more towards Lean (Agile) [11] principles. All of the critical risks have been resolved by developing Proof of Concepts (PoCs) for Data Turbine, OpenSplice DDS, real-time Visualizations using Matlab compiled codes, and coming up with ways to deliver data from DDS and plots over the web using Java Server Pages (JSP). Code quality has been ensured by incorporating recommendations of S. McConnell [18] whenever applicable.

By going through the activities in the engineering notebook and through self-reflection, the hybrid (model that was being used naturally) has been extracted. John S. Miranda, a manager at Intel, proposed that any organization that tries to implement such hybrid approach should have a set of questions / criteria to decide the best mix. This research confirms that requirements stability, software generalizability, software life expectancy and dependency are some key criteria.

Although, at present, there is no quantitative data to confirm the effectiveness of such a hybrid approach (the focus of this project was in research, design, and development of ACPS DSF), but the fact that a complicated large-scale software project like ACPS DSF was very successful may set some foundation for future research. In this paper a novel adaptable process framework for Self-managing Distributed Systems that adds a third dimension to the Rational Unified Process (RUP) [1] [12] is proposed. This new process framework is named as **Jacky's Universal Process (JUP)** of Software Engineering for Self-managing Distributed Systems. As shown in Figure 1, while the pre-existing axes from RUP provide sequential increments and iterative workflows, in JUP the third dimension provides parallel augmentations. Augmentations are different from increments in that they are very loosely coupled services that can be connected or disconnected anytime as required. Each

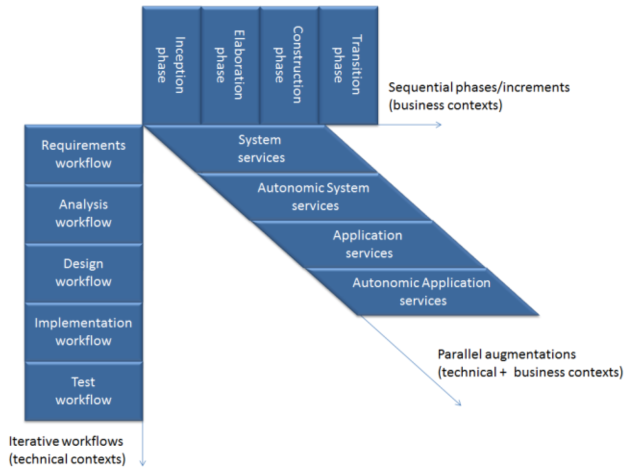

Inception phase
Elaboration phase
Construction phase
Transition phase
Sequential phases/increments (business contexts)
Requirements workflow
Analysis workflow
Design workflow
Implementation workflow
Test workflow
Iterative workflows (technical contexts)
System services
Autonomic System services
Application services
Autonomic Application services
Parallel augmentations (technical + business contexts)

Figure 1. Jacky's Universal Process (JUP)

parallel augmentation can apply Spiral, Lean (Agile), or Hybrid e.g. Spiral + Lean (Agile), or any other model as appropriate. The four universal categories of services in this third augmentation axis are:

- **System Services:** Any distributed system level services.
- **Autonomic System Services:** Any or all of Self-* features of Autonomic Computing at the system level (global).
- **Application Services:** Any distributed application level services.

- **Autonomic Application Services:** Any or all of Self-* features of Autonomic Computing at the application level (local).

The four universal categories of services can be seen in the ACPS DSF Software Architecture (Figure 3). For example, Universal Critical Services are System Services, Autonomic Managers are Autonomic System Services, Visualizations are Application Services, and Control Panel is an Autonomic Application Service. At present use of JUP in different kinds of distributed system that have some capabilities of Autonomic Computing is advocated. Further research in JUP will provide some quantitative measures of its effectiveness. For now, use ACPS DSF as the case study to learn JUP by example.

IV. Requirements, Architecture, and Design

A. *Requirements*

As any other project, the requirements of this project were very vague initially. The vision, overall requirements, and detailed requirements have been collected through interaction with Biosphere 2 Scientists and Biosphere 2 Staff members. At any point, technology risks and non-functional requirements were the key drivers for all subsequent system architecture, software architecture, software design and implementation. The overall requirement was to research, design, and develop a distributed software framework that would facilitate the establishment of LEO cyberinfrastructure by providing a standard reliable, robust, and fault-tolerant means of data acquisition, data distribution, data visualization, data assimilation, modeling, and simulation. No hard and fast metric for feature requirements were defined, but the overall requirements can be listed as follows:

- The software framework should collect data from the physical system and store it in database.
- The software framework should make data available to real-time monitoring and visualization.
- The software framework should have facility for off-line modeling and simulation.
- The software framework should use technologies best for LEO's cyberinfrastructure.
- The visualizations should be available to students/faculty members over the web.
- The software framework should be:
  - Scalable
  - Reliable
  - Robust
  - Fault tolerant

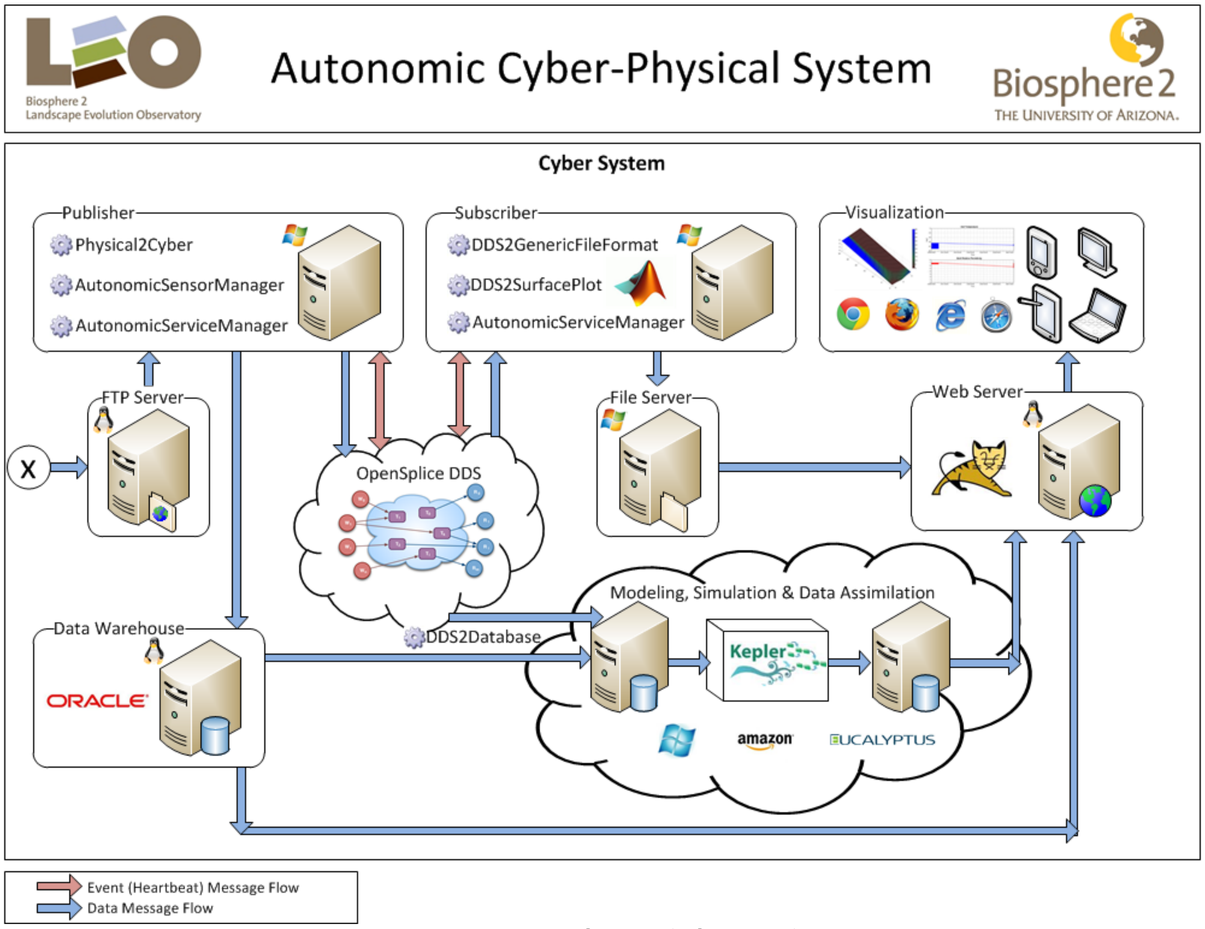


Figure 2. System Architecture (Cyber System)

- Easy to maintain (preferably no maintenance)
- Easy to extend

B. System Architecture

The system architecture, provided in Figure 2, is the final standard cyber system architecture consisting of heterogeneous systems, which can be modified in the future if desired and/or required. In the future, there will be at least one feed-back loop going from the cyber system to the physical system.

Basically, this architecture shows that the entry into the cyber system happens through the file system—a 'Source' folder where the physical system drops measurement data files. From there, Physical2Cyber uploads data to both database and DDS. From the DDS, DDS2GenericFileFormat samples the most recent sensor values in to a special generic file format for current and future web/cloud applications to read. Also, there is a DDS2SurfacePlot, a composite service, which uses CommandExecuter to generate surface plots. DDS2Database is service that can be used to sample DDS contents directly into database. AutonomicSensorManager monitors sensors and writes notifications of sensor failures according to defined policy and also stores knowledge of failures. AutonomicServiceManager monitors the heartbeats of the critical services according to the defined policy. In case of failures of any services, it takes the appropriate actions as defined in policy while storing knowledge of any failures. The web/cloud application reads data from the 'Sink' folder (in particular, from files created and updated byDDS2GenericFileFromat and notification files created by AutonomicSensorManager), and database to show text data, visuals, text and provide audio warnings when required [13].

All of the servers shown in this system architecture are important (ftp server being the most important one) and the flow of data and events through them can easily be analyzed

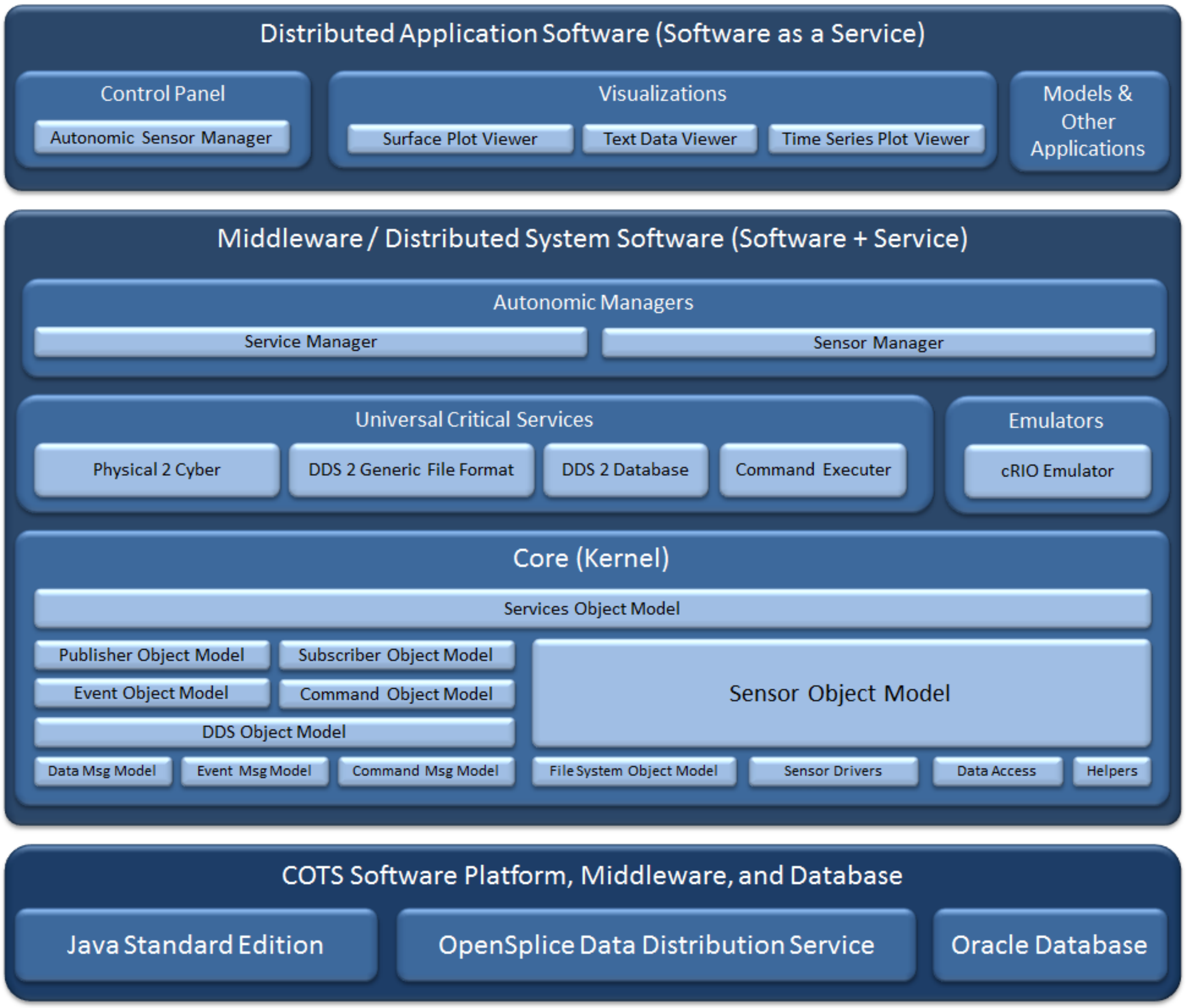


Figure 3. Software Architecture

and best understood using the system architecture, which also shows the best technology mix. The physical system is modeled as a system that generates data. As a matter of fact, details of the physical system, which was outside the scope of my responsibility, is not provided to emphasize the cyber system.

C. Software Architecture

The ACPS Distributed Software Framework (DSF) architecture is shown in Figure 3. It basically consists of three layers in an open architecture i.e. any top level layer can call any of the bottom layer(s). This architecture is a Service Oriented Architecture (SOA) that combines layered and data space architectural patterns [3]. It is SOA because it is composed of a collection of services that provide the fundamental services, which can be mixed and matched to provide the universal set of capabilities ACPS will need over its life-span. These services use Publish/Subscribe interaction pattern by applying the first open international middleware standard—OMG Data-Distribution Service for Real-Time Systems [14]. The Middleware / Distributed System Software (Software + Service) and Distributed Application Software (Software as a Service) consists of subsystems (projects) and each subsystem consists of modules (packages), which in turn contains classes/components.

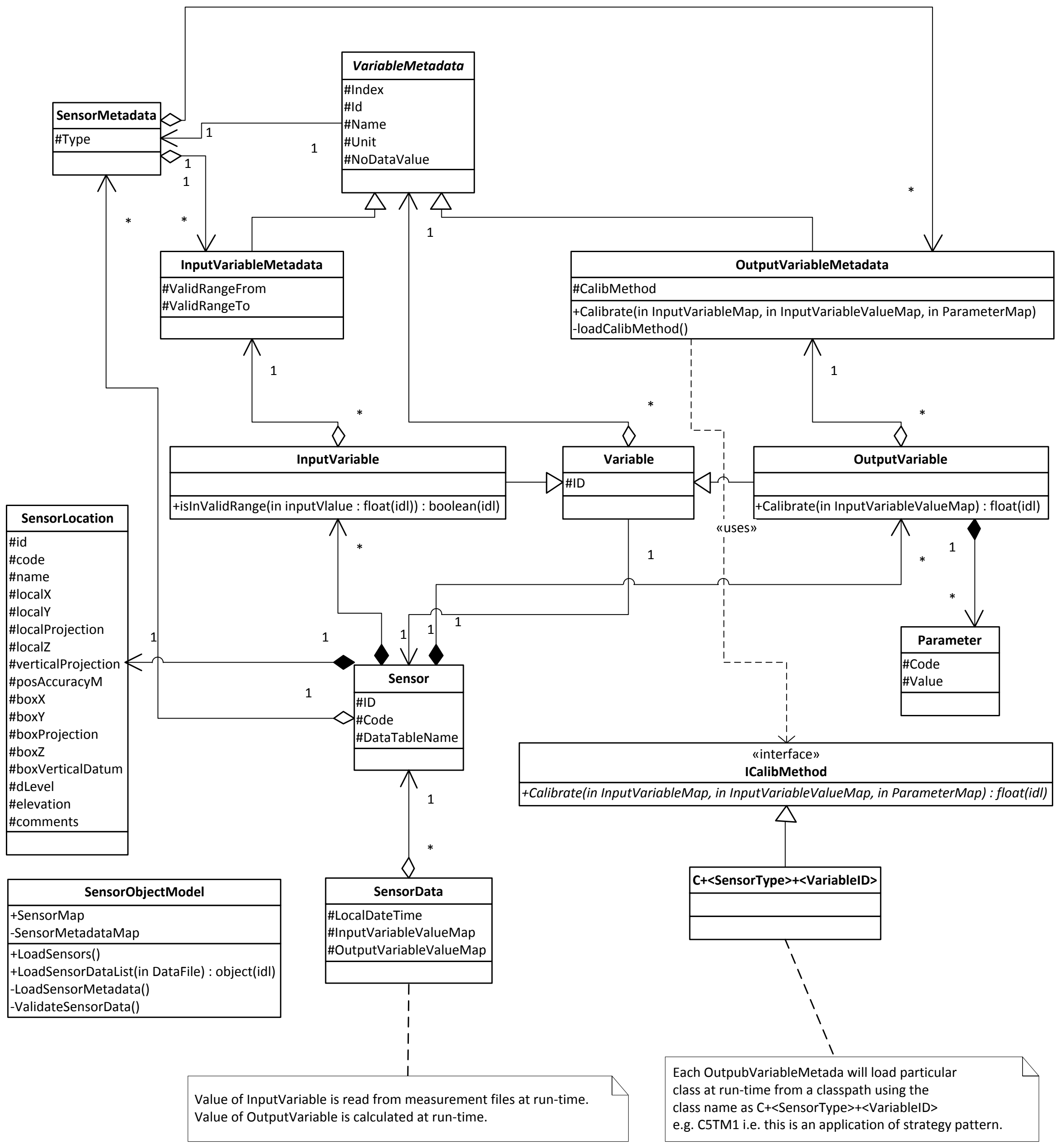
Each of the software layers are described below starting with the bottommost layer first:

Commercial of the shelf (COTS) Platform, Middleware, and Database: This layer of software represents the industry standard proven, tried and test products used as the foundation for ACPS DSF. The major technologies in this layer are:

- **Java Standard Edition:** Java was chosen as the software platform of choice for performance, versatility, portability, and security [15] that ACPS DSF requires.
- **OpenSplice Data Distribution Service (DDS):** OpenSplice DDS is the global leader in real-time data distribution middleware technology [9]. It is the strictest implementation of Object Management Group (OMG) DDS Open Standard providing high scalability, low latency, and fault-tolerance for real-time distributed systems.
- **Oracle Database:** Oracle database provides the foundation for high quality information storage and delivery [16]**.**

Middleware / Distributed System Software (Software + Service): This system layer consists of the system software and services (S+S) that should be sufficient to provide all of the system level capabilities ACPS will ever need over its entire life-span of about 10 years. This layer is complete in that by combining and configuring the universal critical services that use the core, all of the current and future system level needs can be met provided the project plan does not change radically. Other than sensor driver development, and shell script development to meet particular deployment need, no maintenance to this level is neither expected nor recommended. Overall, this layer provides the scalability, robustness, reliability, and fault-tolerance along with fundamental/core ACPS system (and significant application) logic. If ever required, the software and services may be extended without making any changes to the existing core, services, and emulator. This layer has the following subsystems:

- **Core (Kernel):** As the name implies, contains all of the core system (and some application level) logic fundamental to ACPS as a whole. In this subsystem, *Sensor Object Model* is the most important module that hosts the most important object oriented data structure / application programming interfaces (APIs) for all system level services. *Sensor Drivers* is the next most important module that hosts all of the derivers for sensors that define calibration functions. The *File System Object Model*, *Data Access*, and *Helpers* module contains relevant classes/APIs for files, database, and general helpers. The DDS Object Model, contains the APIs for talking to DDS that use data structures in *Data Msg Model*, *Event Msg Model*, and *Command Msg Model* modules. The *Publisher Object Model*, *Subscriber Object Model*, *Event Object Model*, and *Command Object Model* provide classes / wrapper APIs to DDS. The *Service Object Model*, contains the base class and exception class for any system level service.
- **Universal Critical Services:** These are the fundamental services that provide universal capabilities that ACPS will ever need at the system level. *Physical 2 Cyber*, *DDS 2 Generic File Format*, *DDS 2 Database*, and *Command Executer* classes represent the critical services.
- **Emulators:** At present only one *cRIO Emulator* is needed while B2 LEO is under construction. This emulator has the capability to read sensor definitions from database and generate data in exact established format specification between the physical and the cyber system.
- **Autonomic Managers (controllers):** The autonomic managers provide self-healing capabilities of Autonomic Computing to add fault tolerance at service level by *Autonomic Service Manager* module and monitoring/notification capabilities at sensor level by *Autonomic Sensor Manager* module.



Value of InputVariable is read from measurement files at run-time.
Value of OutputVariable is calculated at run-time.

Each OutpubVariableMetada will load particular
class at run-time from a classpath using the
class name as C+<SensorType>+<VariableID>
e.g. C5TM1 i.e. this is an application of strategy pattern.

Figure 4. Sensor object model (simplified)

Distributed Application Software (Software as a Service): This layer is meant to be extended where all future work will take place. All applications in this layer are provided as service over internet and expected to be all deployed in private and/or public clouds i.e. this is the SaaS layer. Hence, all of the control panel and visualization applications are available to any portable device in the world with an internet connection and a web browser. Essentially, all of the web/cloud applications are JavaServer Pages (JSP) that read data from appropriate sources (files and database) and present them either in text or graphics format in appropriate format. *Autonomic Sensor Manager* refreshes itself periodically and monitors contents of a notification file for any sensor failures and present such failures to the user along with audio warning. *Surface Plot Viewer* displays the appropriate surface plot image generated by *DDS 2 Surface Plots* composite service. Text Data Viewer display the appropriate data from generic file format generated and updated by *DDS 2 Generic File Format* service and *Time Series Plot Viewer* shows time series data from database.

D. *Design Overview*

The design of ACPS DSF started with the notion of loosely coupled collection of publishers and subscribers interacting through a middleware keeping in mind the need for scalability due to the massive amount of data flow that is expected. In the design, file system was used, in addition to OpenSplic DDS, as a queue, and also as a shared memory. This way, different systems are decoupled from each other through the file system. Also, during the design, parallelism was taken as a key design criterion to provide scalability. For example, the finished ACPS DSF can be used in parallel by splitting the load of one cRIO to multiple cRIOs, which in turn would mean splitting the queue, running services in parallel, and loading data to database in parallel either at the schema level or at the server level. Also, in the design, synchronization issues have been carefully assessed and file locks were always used for any writing operations. Overall, this ACPS DSF is designed to make it easier to develop ad-hoc applications, which can easily consume data either from the file system or from database. The OpenSplice DDS serve as a shared memory where most recent value of all sensors (from a hill slope) are kept up to date. Any interested application can also sample any particular number or types of sensors at the desired sampling rate.

E. Class Design

Object-oriented design has been used throughout ACPS DSF. Although most of the design evolved over time, the classes in Sensor Object Model (Figure 4) were carefully designed first, even before hitting a single key, and the design was always kept in sync with code. This object model is the most important data structure integral to ACPS DSF. In this design, it is assumed that a sensor has inputs (measurements) and outputs (calibrations) which can be a function of any number of inputs. In order to make these calibrations as general as possible, the strategy design pattern [17] was used. Thus, classes collectively referred to as sensor drivers, define calibration functions by implementing ICalibMethod interface. The next most important sets of classes (in respective packages) are those that form wrappers around OpenSplice DDS namely those in DDS Object Model, Publisher Object Model, Subscriber Object Model, Event Object Model, Command Object Model—all of these packages/modules are part of the ACPS DSF core. ACPS DSF itself is composed of a number of subsystems: Controllers, Core, Emulators, Services, and User Interface. As the name implies, Core/kernel is the most fundamental critical subsystem to the entire distributed system. Each of these subsystems is divided into packages/modules as required.

Design pattern [17] was used in three places: Strategy Pattern in implementing sensor drivers mechanism, Command Patten for commands and events mechanism, and Singleton Pattern was used to make sure that only one instance of Physical2Cyber can be executed per folder it is monitoring.

F. Message Structure Design

In this DDS-centric design, message structure is very important. In order to make sure that all of the kinds of messages that will ever flow through the middleware, are general message structures which are specialized over layers of software using command pattern. In OpenSplice DDS, these messages were defined using Interface Definition Language (IDL), and when passed through a tool (idlpp) that is part of the OpenSplice, the relevant Java classes were generated.

G. Database Design

Database is also an integral part of the system. In order to decouple the design of the database from the design of ACPS DSF, special view specifications were created that serve as interface between the full database and the view of the database in light of ACPS DSF. These views are prefixed with ACPS and all that matters to ACPS DSF is the exact number of attributes with proper data types. The query used to get these attributes may change and are not a concern for ACPS DSF as long as this interface is not broken.

V. Testing, Results, And Discussion

A. Testing Overview

The test cases designed were goal oriented as recommended by Fenton [19]. All of the test cases basically had nominal scale of measurement: Success or Failure. Since all of the test cases passed, the correctness and quality of ACPS DSF were successfully validated.

B. System Testing—Biosphere 2 LEO Server Deployment

In this deployment scenario, three physical cRIO are connected which provides the expected amount of load once the physical system is completed. In the actual physical system, each cRIO is expected to drop files in ‘Source’ folder every 10-15 mins. However, for the purpose of testing,

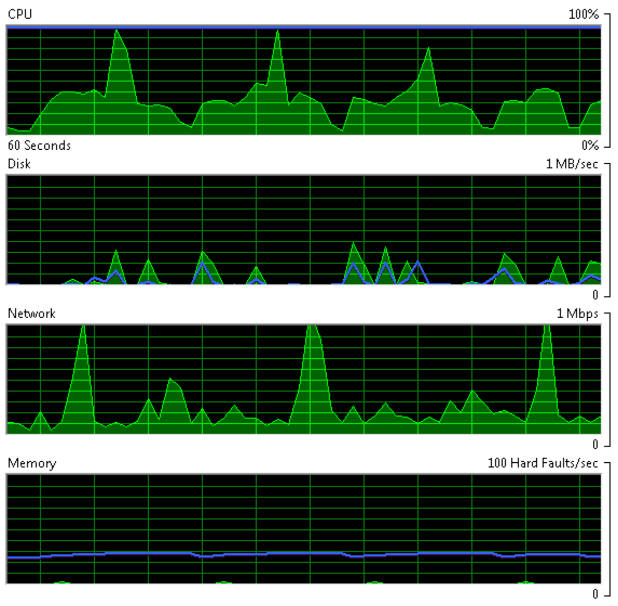


Figure 5. 'Subscriber' machine CPU, Disk, Network, and Memory usage (when DDS2SurfacePlot is running)

the connected cRIOs (x, y, and z) are configured to drop files every 2 mins. In addition to the physical cRIOs, three cRIOEmulators (x, y, and z) are also configured to drop files every 1.5 mins. In accordance with the System Architecture (Figure 2), the ‘Publisher’ machine is running Physical2Cyber, AutonomicSensorManager, and AutonomicServiceManager. Similarly, the ‘Subscriber’ machine is running DDS2GenericFileFormat, DDS2SurfacePlot, and AutonomicServiceManager.

Resource usage for 'Subscriber' machine is shown in Figure 5, and resource usage of 'Publisher' machine is shown in Figure 6. From these statistics, it is evident that resources in 'Publisher' machine can easily be more utilized by applying more load i.e., one 'Publisher' machine is capable of serving multiple hill slopes (physical systems).

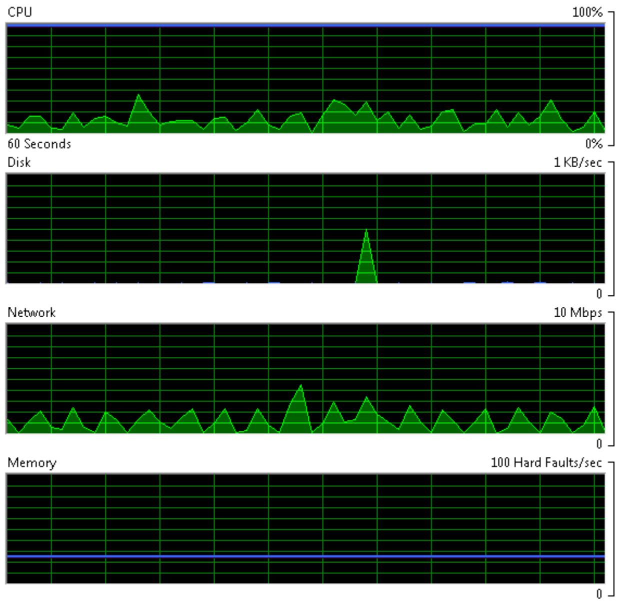


Figure 6. 'Publisher' machine CPU, Disk, Network, and Memory usage

This demonstrates that the system has the capability to provide more throughput than required. However, in 'Subscriber' machine resource utilization is more intense when plotting routines of Matlab is in use as expected. Thus it may be reasonable to have one 'Subscriber' machine per hill slope. Another option could be to have a dedicated machine to run DDS2SurfacePlot, which is indeed the most resource intensive service and in this configuration, a single 'Subscriber' machine could serve multiple hill slopes.

C. *System Testing—Amazon Web Services/Cloud Deployent*

A test bed was created on Amazon Web Services (EC2) with minimal machine (m1.small) configuration. At present, a shared folder on 'ACPSServer' to be accessed by 'ACPSPublisher' and 'ACPSSubscriber' machines could not be created. So the backup plan was to test everything on the 'ACPSServer' machine with predefined m1.small configuration. Since this server is in Amazon EC2, loading time to database was taking much longer ~ 4 minutes for cRIOEmulatorX, which has the largest number of sensors attached to it. The time to load to database (including network latency) directly determines the throughput of the system. Hence for the purpose of testing, cRIOEmulatorX, cRIOEmulatorY, and cRIOEmulatorZ were configured to generate sensor measurement data files every 5 minutes. A total of three sets of application setup were tried: 'Publisher' setup, 'Subscriber' setup, and 'Web Server' setup.

D. *Reliability Testing*

Until present time, there has never been any software downtime. The best way to understand the reliability would be to have a look at a production release of Biosphere 2 Sensor Network Data Acquisition System (Bio2SNDAS), which is the precursor to Physical2Cyber service in ACPS DSF. In the ACPS DSF, in the final release, none of the universal critical services ever failed i.e. zero crashes so far during non-stop execution.

E. *Robustness Testing*

Error could be introduced into the ACPS DSF from three primary places: sensor measurement files, configuration files, and database sensor definitions. Error checking, sensor verification and validation and lots of try-catch blocks were used. As a result, the final release of ACPS DSF never crashed due to any input error.

F. *Fault-tolerance Testing*

ACPS DSF is configured using batch files in both 'Publisher' and 'Subscriber' machines with shortcut from startup folder such that incase of any machine restart, it will start automatically. Also, self-healing capabilities provided by AutonomicServiceManager ensures that if any service dies, it is restarted according to the defined policy. As of now, none of the service ever crashed. However, for the purpose of testing, they were manually terminated, and in all of the trials, AutonomicServiceManager was always able to do its job.

G. *Scalability Testing*

ACPS DSF has been tested thoroughly, and it can be parallelized at the cRIO level, service level, database schema level, and obviously at the database server level. As a matter of fact, the current deployment of ACPS DSF far exceeds the requirements. As long as the network and the database does not become a bottleneck, ACPS DSF is highly scalable. To overcome any network latency, the database should be as geographically close as possible to the cRIOs and the 'Publisher' machine. In order to overcome any possible database insertion inefficiency, various techniques including indexing that are common in database optimization should be applied. Hence, this software product supports both scaling up and scaling out both of which are limited only by the available resources and the imagination of the administrator of the product.

H. *Maintainability Testing/Assessment*

ACPS DSF requires near zero maintenance for the middleware / distributed system software level since only new sensor drivers need to be developed for new sensor types. The universal critical services can be mixed and matched to provide all of the system level capabilities ACPS will ever need. Hence, at this level, it is a complete product and is expected to be used like a commercial of the shelf product.

I. Extensibility Testing/Assesment

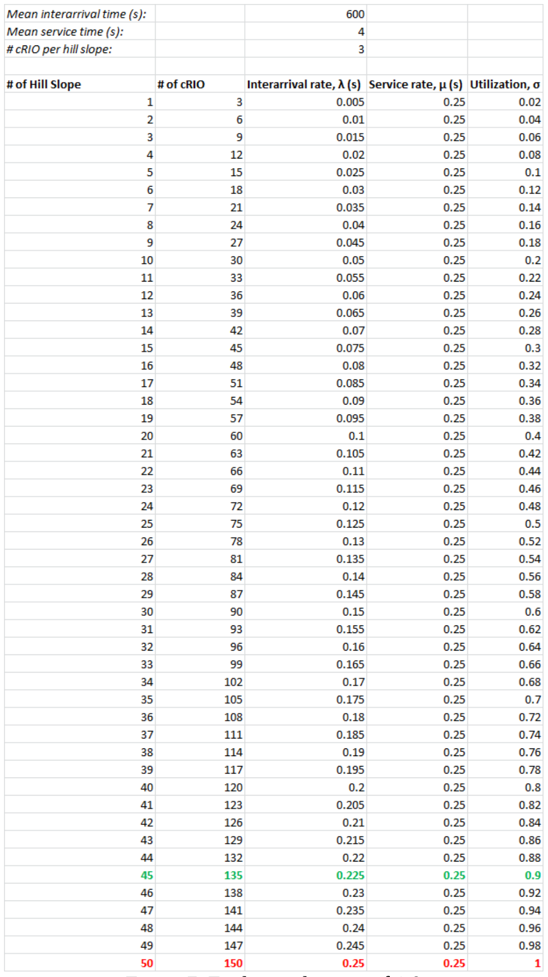
At the distributed application software level, ACPS DSF is expected to be extended throughout its entire lifespan. It has a service oriented architecture using publish/subscribe interaction pattern. Also, it has object-oriented design. Anyone can extend ACPS DSF both at the middleware / system and at the application level.

J. Usability Testing/Assessment

ACPS DSF is very easy to use once a user learns how to configure it, which is also easy. Basically, once it is configured, due to self-healing feature of autonomic computing, no human monitoring of the software is required. As a matter of fact, it can be considered as a fire and forget software system.

K. Security Testing

As of now, all user input is through button clicks. Web/Cloud application is completely decoupled from the middleware. Security was taken in to account throughout the entire design and development cycle. Also, no nice-to-have features were implemented, which often lead to security holes. Overall, ACPS DSF is highly a secured software product, however, the Web/Cloud app could have vulnerability in that user could try SQL injection from the address bar. At the same time, these critical web pages are not expected to be exposed to the general public but only restricted to the B2 employees. It would be nice to have a team of ethical hackers try to compromise it and provide further insight into security issues (if any) of the Web/Cloud. For now, no further tests were done.



Mean interarrival time (s):		600		
Mean service time (s):		4		
# cRIO per hill slope:		3		
# of Hill Slope	**# of cRIO**	**Interarrival rate, λ (s)**	**Service rate, μ (s)**	**Utilization, σ**
1	3	0.005	0.25	0.02
2	6	0.01	0.25	0.04
3	9	0.015	0.25	0.06
4	12	0.02	0.25	0.08
5	15	0.025	0.25	0.1
6	18	0.03	0.25	0.12
7	21	0.035	0.25	0.14
8	24	0.04	0.25	0.16
9	27	0.045	0.25	0.18
10	30	0.05	0.25	0.2
11	33	0.055	0.25	0.22
12	36	0.06	0.25	0.24
13	39	0.065	0.25	0.26
14	42	0.07	0.25	0.28
15	45	0.075	0.25	0.3
16	48	0.08	0.25	0.32
17	51	0.085	0.25	0.34
18	54	0.09	0.25	0.36
19	57	0.095	0.25	0.38
20	60	0.1	0.25	0.4
21	63	0.105	0.25	0.42
22	66	0.11	0.25	0.44
23	69	0.115	0.25	0.46
24	72	0.12	0.25	0.48
25	75	0.125	0.25	0.5
26	78	0.13	0.25	0.52
27	81	0.135	0.25	0.54
28	84	0.14	0.25	0.56
29	87	0.145	0.25	0.58
30	90	0.15	0.25	0.6
31	93	0.155	0.25	0.62
32	96	0.16	0.25	0.64
33	99	0.165	0.25	0.66
34	102	0.17	0.25	0.68
35	105	0.175	0.25	0.7
36	108	0.18	0.25	0.72
37	111	0.185	0.25	0.74
38	114	0.19	0.25	0.76
39	117	0.195	0.25	0.78
40	120	0.2	0.25	0.8
41	123	0.205	0.25	0.82
42	126	0.21	0.25	0.84
43	129	0.215	0.25	0.86
44	132	0.22	0.25	0.88
45	135	0.225	0.25	0.9
46	138	0.23	0.25	0.92
47	141	0.235	0.25	0.94
48	144	0.24	0.25	0.96
49	147	0.245	0.25	0.98
50	150	0.25	0.25	1

Figure 7. Finding utilization of 1.0

VI. EXPERIMENTATION, RESULTS, AND DISCUSSION

From all of the testing conducted in Section **Error! Reference source not found.**, it is evident that not only does ACPS DSF meets all of the functional and non-functional requirements of B2 LEO, but exceeds them by more than 10x. In B2 server setup, one deployed instance of ACPS DSF is already demonstrating capability to serve more than 10 hill slopes. The calculation for this load multiplier for the present experimental configuration on B2 LEO is provided in **Error! Reference source not found.**. Even with these experiments, it is evident from the resource usage profiles that the machine is not fully utilized. This is indeed an exploratory analysis. From this analysis, a theory is established that ACPS DSF have the capability by far more than 10x. To test the theory, a confirmatory analysis is conducted using Queuing Theory [20].

TABLE I. LOAD MULTIPLIER CALCULATION

	Actual system	**Experimental emulated system**	**Experimental physical system**
Sensor Measurement File interarrival time (minutes)	$t_{actual} = 10$	$t_{emulated} = 1.5$	$t_{physical} = 2$
Number of cRIO	$n_{actual} = 3$	$n_{emulated} = 3$	$n_{physical} = 3$
Load multiplier	$(t_{actual} / t_{actual}) * (n_{actual} / n_{actual}) = 1$	$(t_{actual} / t_{emulated}) * (n_{emulated} / n_{actual}) = 6.67$	$(t_{actual} / t_{physical}) * (n_{physical} / n_{actual}) = 5$
Total load multiplier (emulated + physical)	N / A	$(t_{actual} / t_{emulated}) * (n_{emulated} / n_{actual}) + (t_{actual} / t_{physical}) * (n_{physical} / n_{actual}) = 11.67 > 10$	

The Source Folder in File System can be considered as a queue where sensor measurement files arrive. The time it takes to process each measurement file, depends upon the amount of data it has, which in turn depends upon the number of sensors connected to the particular cRIO (X, Y, or Z). cRIO Y has the most number of sensors (little more than 1000). The time taken to process a measurement file from cRIO Y in the current B2 server is about 4s. For cRIO X, Z, the time taken is negligible. However, let us take the maximum service time for all of the measurement files X, Y, and Z. Hence, the interarrival rate, λ, is 3 measurement files every (10 * 60) seconds, and the service rate, μ, is 3 measurement files every 12 seconds. The utilization of server is given by equation (**Error! Reference source not found.**).

$$\rho = \lambda / \mu \quad (1)$$

By entering this information in an Excel Spreadsheet, the number of hill slopes for which utilization of the current system becomes one is found as shown in Figure 7.

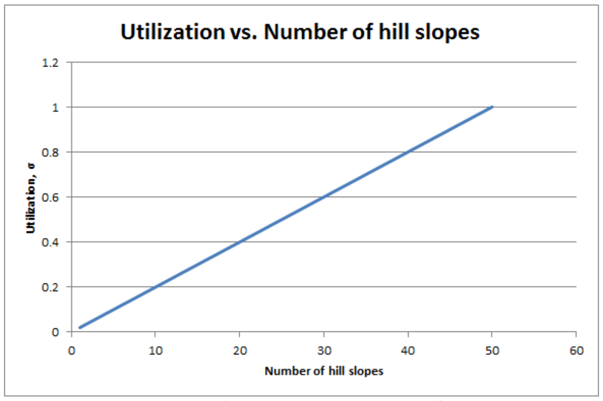


Figure 8. Graph of utilization vs. number of hill slopes

According to this calculation, the current system will attain an utilization of 1.0 when a load of 50 hill slopes i.e. 150 cRIO is provided (assuming they have the same size of sensor measurement files i.e. same number of sensors attached). A graph of utilization vs. number of hill slopes is also provided in Figure 8. It is evident that the system will have utilization of 0.9 for 45 hill slopes, even this is way more than the amount of load each cyber system is expected to have when augmented sensors are introduced in future. In order to simulate a 45x load, the cRIOEmulators (X, Y, and Z) can be configured with lower interarrival time and verify that the cyber system is still stable. Hence, an experiment is conducted with each cRIOEmulators configured to have an interarrival time of 600 / 45 = 13.33 seconds on B2 server 'Publisher' machine and are configured to use a development shared folder so that the files from physical cRIOs are not included in the experiment. The outcome of this experiment is exactly as expected. The queue (source folder) goes almost full, then Physical2Cyber completes processing them all and as soon as it finishes, more measurements file arrive. This is an excellent **utilization (0.9)** of the 'Publisher' machine and this utilization represents **45 hill slopes**. This experiment confirms the theory that ACPS DSF is not only capable of meeting the non-functional requirements but goes above and beyond—it has the ability to process the load of as many as 45 hill slopes with a server utilization of 0.9 and each hill slope having three cRIOs with each cRIO connected to about 1100 sensors (i.e. a total of about 45 * 3 * 1100 = 148,500 sensors).

It is important to note that to load measurement file from cRIO Y from Amazon cloud to LEO database server, it took about 4 minutes. Hence, both the ACPS DSF and database are highly optimized. The network latency is a major bottleneck and should be resolved by locating databases as close to the cRIO as possible to reduce latency.

VII. Conclusion and Future Work

In conclusion, this was a very exciting and challenging project. Turning various ideas into reality, shaping the ideas and finding the best architecture, design, and implementation with the perfect technology mix was a great accomplishment in an interdisciplinary team like this, where maintaining concept consistency itself was a great challenge. ACPS DSF provide the ultimate technical solution for B2 LEO so that it can now focus on integrating computational models with the confidence that the underlying middleware / distributed system software is organized physically into the standard system architecture will always provide the performance and the reliability, robustness, fault-tolerance, scalability, and extendibility it needs. This ACPS DSF provide the strategic directions for all kinds of application development—applications that all fit into the distributed software framework. No more maintenance is needed (at the distributed system software level) once all of the different sensor types are purchased, and all of the sensor drivers are developed. OpenSplice DDS and Oracle Database make real-time, near-real-time, and offline analyses possible. Moreover, this optimum technology mix found through this research project resolves all of the critical technological risks. The best architecture, design, and implementation methods are clearly demonstrated through this distributed software framework, which is expected to last for the entire life-span of B2 LEO project of about 10 years. In this research project how Matlab can be used creatively to generate plots that can be delivered over the web through simple techniques as just pooling plot files generated by Matlab from JSP pages using auto refresh mechanism is clearly demonstrated. Such simple mechanisms allow thin clients to access all of the applications as a service over web/cloud. The audio alarm in addition to text alarm are extremely helpful and provide such off-site monitoring of sensor from anywhere in the world with just a portable device with internet connection and a browser. This Service Oriented Architecture allows very loose coupling enabling each subsystem to decouple from one another. In case of maintenance only the required machines can be turned down for maintenance ACPS DSF can be configured to auto-start. The Source folder where all files are dropped from the physical system act as a queue, and the Sink folder where all data and plots are dropped serve as a shared memory. The Source folder decouples the physical system from the middleware, and the Sink folder decouples any application from the middleware. Likewise the database also decouples applications from the middleware. Hence, any physical system application and/or any application software do not have to know the details of the middleware, but need to know how to access text files, pictures, and database. The middleware in the meantime provide all of the non-functional capabilities that B2 LEO ACPS will ever need.

Overall, ACPS DSF resolves all of the technology risks by providing the middleware and by providing efficient application solutions; it provides concept consistency through the framework in which all future applications can evolve. ACPS DSF is a very successful real-time product with meets all of the functional and non-functional requirements. As a matter of fact, since one instance of ACPS DSF can handle the load of as many as 45 hill slopes having a total of 148,500 sensors with a utilization of 0.9, B2 LEO will never have to worry about any development in ACPS DSF middleware / distributed system software. Also, the hope is that JUP will add value to both industry and academia for software engineering of self-managing distributed software systems.

Future work consists of extending ACPS DSF by building new applications at the distributed application level. The immediate next set of work include creating services that read data from database and transform them into NetCDF format so that the very first model known as Kathy can be run for analysis. Along with the models simulation, new visualizations will be required. From there, it will all depend upon the priorities and progress of other new models. If there is new sensor types, drivers for them should be developed. Any change to ACPS DSF middleware / distributed system software is neither expected nor recommended. ACPS DSF is a finished product and should be used like a commercial-of-the-shelf software framework.

ACKNOWLEDGMENT

Thanks to Biosphere 2 LEO for funding this research and development.

REFERENCES

[1] P. Kruchten, The Rational Unified Process: An Introduction (2nd Edition), Addison-Wesley Professional, 2000.

[2] NSF, Cyberinfrastructure Vision for 21st Century Discovery, National Science foundation Cyberinfrastructure Council, 2007.

[3] A. S. Tanebaum and M. V. Steen, Distributeds Systems: Principles and Paradigms (2nd edition), Upper Saddle River: Pearson, Prentice Hall, 2006.

[4] P. Veríssimo and L. Rodrigues, Distributed Systems for System Architects, Springer, 2001.

[5] J. Kephart, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41-50, 2003.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, and G. Lee, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, 04 2010.

[7] M. Butrico and D. Da Silva, "Toward a Unified Ontology of Cloud Computing," in *Grid Computing Environments workshop*, Santa Barbara, 2008.

[8] G. Pardo-Castellote, "OMG Data-Distribution Service: architectural overview," in *Distributed Computing Systems Workshops*, 2003.

[9] "OpenSplice DDS Overview," PrismTech, [Online]. Available: http://www.prismtech.com/opensplice/products/opensplice-dds-overview. [retrieved: September, 2012].

[10] B. W. Bohem, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61-72, 1988.

[11] M. Poppendieck and P. Tom, Lean Software Development: An Agile Toolkit, Upper Saddle River: Addison Wesley, 2003.

[12] S. Schach, Object-Oriented Software Engineering, McGraw-Hill Science/Engineering/Math, 2007.

[13] S. Islam, A. Robertson, C. M. Kartchner, D. V. Sickinger, and J. L. Eyre, "Situational Awareness Detection and Warning for Airport Operations," The University of Arizona., Tucson, 2010.

[14] "Data Distribution Service Portal," Object Management Group, [Online]. Available: http://portals.omg.org/dds/. [retrieved: September, 2012].

[15] "Java SE Overview - at a Glance," Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/javase/overview/index.html. [retrieved: September, 2012].

[16] "Oracle Database 11g Release 2," Oracle, [Online]. Available: http://www.oracle.com/technetwork/database/enterprise-edition/overview/index.html. [retrieved: September, 2012].

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

[18] S. McConnell, Code complete: A Practical Handbook of Software Construction (2nd Edition), Microsoft Press, 2004.

[19] N. E. Fenton and S. L. Pfleege, Software Metrics: A Rigorous and Practical Approach, Revised (2nd edition), Course Technology, 1998.

[20] L. L. Peterson and B. S. Davie, Computer Networks, Elsevier Science, 2007.

Understanding the Relationships Within the Medi SPICE Framework

Derek Flood, Fergal Mc Caffery, Valentine Casey
Dundalk Institute of Technology,
Dundalk, Ireland
{derek.flood, fergal.mccaffery, val.casey} @dkit.ie

***Abstract*— Regulated domains, such as medical device software development, require organisations to have specific processes in place in order to secure regulatory approval. Software process improvement initiatives, such as Medi SPICE, help organisations to improve their process in conformance with these regulations. These initiatives, however, do not specify how an organisation implements these processes, instead detailing what the organisation must implement. This work proposes the development of a series of roadmaps that will guide an organisation through the implementation of the required processes in a regulatory compliant manner. This paper presents the first step towards achieving this aim, which involves an investigation of the dependencies between the base practices defined in Medi SPICE in order to ensure that the produced roadmaps form a complete software development process in line with regulatory requirements. The paper describes two complementary approaches, a structured representation and a graphical representation, to representing the links between practices in the Medi SPICE framework.**

Keywords-Software Process; Medical Device Regulation; Software Process Improvement Roadmaps.

I. INTRODUCTION

Advancements in technology have allowed medical practitioners to provide a greater level of care to patients by offering a wider range of treatment options. However, when technology is used, there is a risk to the patient if that device should fail. For this reason, strict regulations must be followed during the design and development of medical device software. In order for an organisation to market medical devices they must comply with the regulatory requirements of the country in which the device is to be sold [1]. For example, an organisation wishing to market new medical devices, unlike anything on the market, within the US must first submit a pre-market submission to the Food and Drug Administration (FDA) for approval prior to the distribution of the medical device. If a similar medical device is already on the market, then the medical device organisation must submit a 510k application. One exception to this is Medical Device Data Systems (MDDS), which do not require pre-approval, but must have been developed using defined development processes and have a Quality Management System (QMS) in place [2].

Increasingly, software is becoming a more important component of medical devices. This is partially due to its flexibility and its ability to enable complex changes to be made to the medical device, without the need for changes to the hardware [3] and also due to the fact that standalone software in its own right may also be considered a medical device [4]. Consequently, this increase in the proportion of software within medical devices has resulted in increased medical device software complexity [5].

In order to assist organisations improve their processes to meet regulatory compliance, Medi SPICE [6] (a medical device specific software process improvement framework) provides organisations with the goals of the required processes and a number of base practices that must be implemented in order to achieve these goals.

The Medi SPICE framework is divided into a number of processes each detailing a different aspect of the software development process. However, there are a number of dependencies between these processes making it more difficult to focus upon individual processes in isolation. This work aims to identify these dependencies through an analysis of the base practices defined within Medi SPICE both internally within individual processes and externally across different processes.

Upon obtaining a detailed understanding of these dependencies a series of roadmaps may then be developed that will guide organisations through the implementation and improvement of their medical device software development processes in an efficient manner.

In this paper, we detail the process used for the identification of these links and how the representation scheme that has been used will allow for validation upon the completed roadmaps. In addition, the paper outlines the types of relationships that were identified in the Medi SPICE framework and provides examples of each type of relationship.

The paper is structured as follows: Section II outlines the importance of medical device software. Section III introduces the Medi SPICE framework. Section IV outlines the objectives of this research. Section V describes how the relationships in Medi SPICE were modelled using both a human readable and machine readable representation. Section VI discusses how these representations will be used during the construction of a series of process roadmaps to guide organisations through the implementation of the necessary standards for developing medical device software. Section VII contains our conclusions for this research.

II. MEDICAL DEVICE SOFTWARE

Software is playing an increasingly integral part in medical devices and is now included in approximately 50% of the medical devices available for sale in the US [7]. Consequently, generic software development organisations are now becoming medical device software organisations both due to the software development opportunities within this domain and also because their software development applications may now be classified as medical devices if they meet the Medical Device Directive's (MDD's) definition of a medical device [8]. The MDD defines a medical device as

"any instrument, apparatus, appliance, material or other article, whether used alone or in combination, including software necessary for its proper application intended by the manufacturer to be used for human beings for the purpose of:

- *diagnosis, prevention, monitoring, treatment or alleviation of disease,*
- *diagnosis, monitoring, treatment, alleviation of or compensation for an injury or handicap,*
- *investigation, replacement or modification of the anatomy or of a physiological process,*
- *control of conception,*

and which does not achieve its principal intended action in or on the human body by pharmacological, immunological or metabolic means, but which may be assisted in its function by such means."

This means that software development organisations creating applications which meet this definition must now conform to the same regulatory requirements as traditional medical device manufacturers.

Therefore, organisations that are new to medical device software development must be aware of the relevant regulations that are applicable to the medical device domain within the particular region they wish to market their device [9]. Medical devices marketed in the US must comply with the FDA regulations, while devices to be marketed within the European Union (EU) must conform to the regulations set out by the European Council.

As part of these regulations [2], a QMS must be in place during the design, development, delivery, installation and servicing of medical devices. The QMS ensures that high quality processes are used through-out the entire product lifecycle and that adequate documentation is maintained for review by the appropriate authority.

To guide these organisations a number of regulations and standards have been produced by the relevant regulatory authorities. In the EU, the *ISO 13485- Medical Devices - Quality management systems – Requirements for regulatory purposes* [10], has been produced outlining the main requirements of a QMS. Similarly, the FDA has produced the FDA 21 CFR Part 820 Quality Systems Regulations (QSR) [11].

III. MEDI-SPICE

Despite the regulatory bodies outlining the necessary regulations, standards, technical reports and guidance documents for medical device software development, they do not provide specific methods for performing the required activities to achieve regulatory approval. This often leads to medical device organizations becoming compliance centric in their approach to software development. As a result, there has been very limited adoption of software process improvement within the medical device domain [12]. Previously, this was not a critical issue due to the limited proportion of software contained within medical devices, but this is no longer the case. Today, there is a particular requirement for highly effective and efficient software development processes to facilitate medical device software development [13].

Existing generic Software Process Improvement (SPI) models are available, which include the Capability Maturity Model Integration (CMMI®) [14] and ISO 15504-5:2006 [15] (SPICE), but these models were not developed to provide sufficient coverage of all the areas required to achieve medical device regulatory compliance [16]. To address the requirement for a medical device software process assessment and improvement model the Regulated Software Research Group at Dundalk Institute of Technology undertook extensive research in this area [6] [17]. This initiated the development of Medi SPICE, a medical device specific SPI framework, which is being developed in collaboration with the SPICE User Group [19].

The objective of undertaking a Medi SPICE assessment is to determine the state of a medical device organisation's software processes and practices. Medi SPICE is an integration of the regulatory requirements of the medical device industry and software engineering best practice [14]. It can also be used as part of the supplier selection process when an organisation wishes to outsource or offshore part or all of their medical device software development to a third party or remote division [16].

Medi SPICE is based upon the latest version of ISO/IEC 15504-5 (currently under ballot) and ISO/IEC 12207:2008 [17]. It is being developed in line with the requirements of ISO/IEC 15504-2:2003 [18] and contains a Process Reference Model (PRM) and Process Assessment Model (PAM). It also incorporates the requirements of the relevant medical device regulations, standards, technical reports and guidance documents.

The Medi SPICE PRM consists of 42 processes and 15 subprocesses which are fundamental to the development of regulatory compliant medical device software. Each process has a clearly defined purpose and outcomes that must be accomplished to achieve that purpose.

Medi SPICE also contains a PAM, which is based upon the PRM, which forms the basis for collecting evidence that may be used for rating the process capability. This is achieved by the provision of a two-dimensional view of process capability. In one dimension, it describes a set of process specific practices that allow the achievement of the process outcomes and purpose as defined in the PRM; this is termed the process dimension. In the second dimension, the PAM describes capabilities that relate to the process capability levels and process attributes, this is termed the capability dimension.

IV. Research Objectives

The aim of this research is to understand and identify the relationships between the base-practices defined within the Medi SPICE PAM. In order to achieve this aim, two research questions (RQs) were constructed to examine the relationships between base-practices both within individual processes and across different processes.

- RQ1: What relationships exist between base-practices in each process included within the Medi SPICE framework?
- RQ2: What relationships exist across processes of the Medi SPICE framework?

RQ1 was posed to examine each process in isolation to determine the relationships that exist between the base practices.

In contrast to RQ1, RQ2 examined the relationships between the processes by identifying base-practices that are dependent upon base practices in other processes. For example Fig. 1 shows the relationship between Eng1, which details a process for obtaining stakeholder requirements, and ENG2, which defines the system requirements analysis process. It can be seen that, before establishing the system requirements (*ENG2.BP1*), an organisation must first agree on the requirements with stakeholders (*ENG1.BP7*).

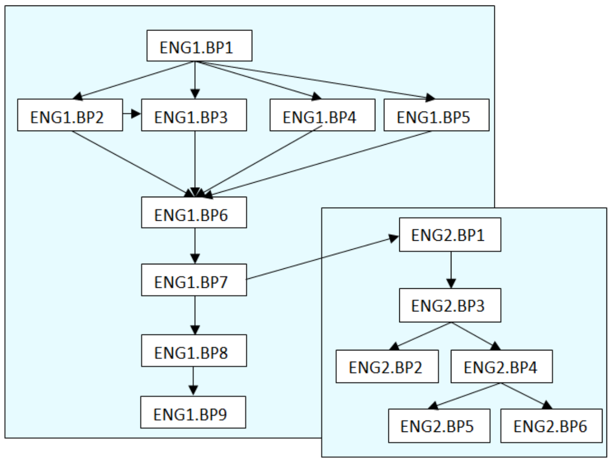


Figure 1. Across Process Relationship

In order to answer the research questions posed above, an analysis of the Medi SPICE PAM was performed. The base practices in each process were examined and the relationships between the practices were determined. The identified relationships were then independently validated by the authors of Medi SPICE. The identified relationships were represented using both a human readable (graphical) representation and a machine readable (structured) representation (XML).

V. Representation of Links

Once the links were identified between the practices, they were represented in two ways. To aid the understanding of the relationships between practices in each process, a graphical representation of each process was produced. In addition to this, a machine readable structured representation was also produced to allow for a quick identification of practice dependencies.

A. Human readable representation

As one of the aims of this work was to understand how the base practices in the Medi SPICE framework relate to one another, a human readable representation of each process was created.

In this representation, each practice is represented as a rectangle and the links between them are represented as an arrow pointing to the depending process. It can be seen in Fig. 2 that there is an arrow pointing from *AGR1B.BP1* to *AGR1A.BP2*. This means that base practice 2 *(AGR1B.BP2)* is dependent upon base practice 1 *(AGR1B.BP1)*.

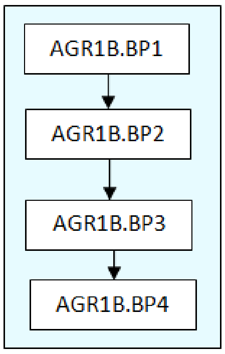


Figure 2. Human-Readable Visualisation

In this representation, it was decided to use the full process ID to help users distinguish between practices of different processes when the graph is used to represent a relationship between multiple processes.

The nature of the dependencies between the base practices usually stems from the need of information to pass from one base practice to another. For this reason, the dependency graphs designed during this work were produced to replicate the information flow between the base practices.

B. Structured representation

In addition to the visual representation, it was necessary to produce a machine readable format that could be used during the production of the roadmaps to identify all base practices necessary to meet those required by the standards.

It was decided to use a custom XML schema to represent the links as most languages provide support for reading in XML files. An example process is presented below.

```
<process title="Acquisition Preparation"
id="AGR1A">
    <basePractice
    id="AGR1A.BP1">Establish the
    need</basePractice>
    <basePractice id="AGR1A.BP2">Define
    the requirement</basePractice>
    <basePractice id="AGR1A.BP3">Review
    Requirements</basePractice>
    <basePractice id="AGR1A.BP4">Develop
    Acquisition strategy</basePractice>
    <basePractice id="AGR1A.BP5">Define
    selection criteria</basePractice>
    <basePractice
    id="AGR1A.BP6">Communicate the
    need</basePractice>

    <InProcessLink PID="AGR1A.BP2"
    dependantOn="AGR1A.BP1"/>
    <InProcessLink PID="AGR1A.BP3"
    dependantOn="AGR1A.BP2"/>
    <InProcessLink PID="AGR1A.BP4"
    dependantOn="AGR1A.BP3"/>
    <InProcessLink PID="AGR1A.BP5"
    dependantOn="AGR1A.BP4"/>
    <InProcessLink PID="AGR1A.BP6"
    dependantOn="AGR1A.BP5"/>
    <ExProcessLink PID="AGR1A.BP5"
    DependantOn="AGR1B.BP1"
    type="equivalent"/>
</process>
```

Each process is comprised of four tags; <Process/>, <BasePractice/>, <InProcessLink>, and <ExProcessLink>. The <Process> tag represents a process in the Medi SPICE framework and includes two attributes; the title of the process and the ID used to identify the process within the Medi SPICE framework. All other tags are nested within the <Process/> tag.

The <BasePractice/> tag is used to represent the base practices within the process. There are between 3 and 18 base practices within each process. Each base practice is comprised of an ID and the title of the practice.

The <InProcessLink/> tag represents a link between two practices within a single process. The tag contains two attributes; the first attribute identifies the practice which is dependent upon another practice and is given the attribute name *PID* while the second attribute identifies the practice which is depended upon, known as *dependantOn*.

The final tag is used to represent external links. This tag, titled *<ExProcessLink/>*, contains three attributes. The first two attributes are the same as those used within the *<InProcessLink/>; PID* and *dependantOn*, while the third attribute, titled *type*, denotes the class of link that exists between the practices. A detailed examination identified three types of links within the Medi SPICE framework: *breakdown*, *equivalent*, and *dependent*.

In some cases, a sub process is used to implement a base practice in another process. For example, *AGR1.BP3* defines the practice "Select Supplier" while sub process *ARG1B* defines the base practices that should be used to select a supplier, as illustrated in Fig. 3. This type of link is known as *breakdown*, as this type of link breaks down one practice into multiple base practices.

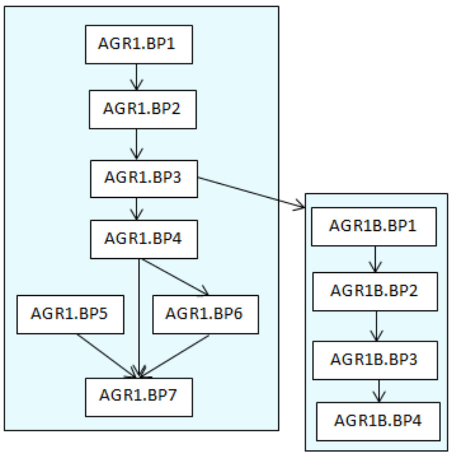


Figure 3. Between Process Link of type *breakdown*

In addition, some practices are semantically equivalent to practices in other process areas. For example, *AGR1A.BP5* is to "Define the selection criteria" while *ARG1B.BP1* states "Establish supplier selection criteria". Although the terminology is different between the two practices the underlying meaning is the same. This is depicted in Fig. 4. In this type of relationship, the type attribute is given a value of *equivalent.*

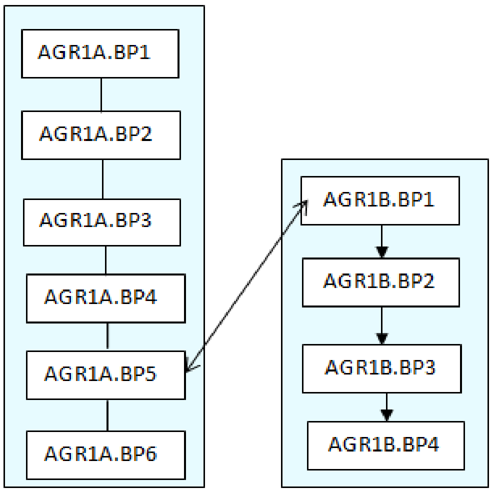


Figure 4. Between Process Link of type *equivalent*

The value given to the type attribute in the final class of relationship is *dependant*. In this case, a base practice must be performed before a subsequent base practice can be implemented. For example, the stakeholder requirements should be established *(ENG1.BP7)* before the establishment of the system requirements *(ENG2.BP2),* as illustrated in Fig. 1.

VI. Using the Dependency Graph

The next phase of this work will be to identify the base practices that are necessary to fulfil the requirements of multiple medical device software development regulations and standards such as ISO 13485 and ISO 14971. These standards define the requirements that are necessary to secure regulatory approval in order to sell medical devices.

Before a medical device can be marketed in the US, it may be required to first secure premarket approval from the FDA. To assist medical device organisations the FDA have produced a document entitled "Guidance for the content of Premarket Submissions for Software Contained in Medical Devices" in which they outline what is required in order to prepare a premarket submission.

In this document, the FDA state that an organisation must implement a QMS in order to sell their devices on the market in the US. This requirement is also necessary for sales within the EU. The International Organization for Standardisation (ISO) have produced the International standard *ISO 13485 – Medical devices - Quality management systems – Requirements for regulatory purposes* which details the requirements for a QMS. The FDA has produced a similar regulation, but have said, recently, that QMSs compliant with the ISO 13485 would also be acceptable.

In order to assist organisations to implement a quality management system, the base practices necessary to fulfil the requirements of the ISO 13485 will be identified through a thorough examination of the standard. Subsequently, through the use of the dependency graph, described above, all supporting base practices necessary to implement the identified base practices will be identified.

Using these base practices, a software process improvement roadmap will be developed that will guide an organisation through the implementation of a QMS. Each of the base practices will be grouped into one of three phases; Planning phase, SDLC phase and On-Going activities phase.

The planning phase occurs at the beginning of a medical device software development project. During this phase the organisation will define the lifecycle that will be used during the project and define strategies for a number of activities performed during the development of the medical device software. This phase will also include the definition of the quality objectives and the assignment of responsibility for the QMS to a member of the management team.

The second phase is performed during the development of the software. During this phase, the practices necessary to be compliant with the QMS are defined. The practices in this phase relate to the systems development lifecycle and include activities such as requirements analysis, design, and testing.

The ISO 13485 standard requires that a number of activities need to be performed at regular intervals during the development process. These activities do not belong to a single phase of the lifecycle but can occur during any phase of the development process. The practices belonging to these activities will be placed in the third phase of the roadmap, the On-Going activities phase. Examples of this type of activity are quality assurance activities, risk assessment activities and problem resolution activities.

In addition, this phase also includes an optional process that may be required during the development of a medical device software system, namely, Acquisition. It may be necessary for a medical device software organisation to acquire components that will be used in the produced medical device software. To assist these organisations the roadmap will include an optional process that will guide the organisation through the acquisition of the necessary components.

When the practices have been assigned to each of the three groups described above, they will then be sub-divided into steps that will allow the organisation to implement them in a sequential manner.

The dependency graphs described in this paper will play an important role in validating the proposed roadmaps. In addition to identifying necessary practices, the dependency graphs will also help to ensure that the activities are performed in the correct order. Using the machine readable format, each practice in a step will be validated to ensure that it does not depend on a step that is performed at a subsequent step.

VII. Conclusion

Medical device software is required to be developed and maintained through following high quality processes during the construction and distribution of the software. Depending upon the region in which the software is to be sold, local regulations must be adhered to in order to secure approval for sale. The Medi SPICE framework has been developed to assist medical device software organisations improve the quality of their processes.

This work complements the Medi SPICE framework through the development of a series of SPI roadmaps that medical device organisations can use to guide their software improvement activities. An important first step in this work has been the identification of the relationships that exist between base practices within the Medi SPICE framework. These relationships have been modelled in both a human and machine readable format allowing for quick analysis of these relationships during the creation of the roadmaps.

VIII. Acknowledgements

This research is supported by the Science Foundation Ireland (SFI) Stokes Lectureship Programme, grant number 07/SK/I1299, the SFI Principal Investigator Programme, grant number 08/IN.1/I2030 (the funding of this project was awarded by Science Foundation Ireland under a co-funding initiative by the Irish Government and European Regional Development Fund), and supported in part by Lero - the Irish Software Engineering Research Centre (http://www.lero.ie) grant 10/CE/I1855.

IX. References

[1] Burton, J., McCaffery, F., and Richardson, I., A risk management capability model for use in medical device companies. in International Workshop on Software quality (WoSQ '06). 2006. Shanghai, China: ACM

[2] US FDA Center for Devices and Radiological Health, Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices. 2005, CDRH: Rockville.

[3] Lee, I., Pappas, G., Cleaveland, R., Hatcliff, J., Krogh, B., Lee, P. , Rubin, H. , and Sha, L., High-Confidence Medical Device Software And Systems. Computer, 2006. 39(4): pp. 33 - 38.

[4] European Council, Council Directive 2007/47/EC (Amendment). 2007, Official Journal of The European Union: Luxembourg.

[5] Rakitin, R., Coping with defective software in medical devices. Computer, 2006. 39 (4): pp. 40 - 45.

[6] McCaffery, F., Dorling, A., "Medi SPICE Development", Software Process Maintenance and Evolution: Improvement and Practice Journal. Volume 22 Issue 4, pp. 255 – 268 http://www3.interscience.wiley.com/journal/122580316/abstrac

[7] Faris, T. H., (2006) "Safe and Sound Software: Creating an Efficient and Effective Quality System for Software Medical Device Organizations" ASQ Quality Press, 2006

[8] Mc Hugh, M., Mc Caffery, F. & Casey, V. 2011. Standalone Software as an Active Medical Device In: O'CONNOR ET AL (ed.) The 11th International SPICE Conference Process Improvement and Capability dEtermination. Dublin: Springer.

[9] Ge, X., Paige, R. F. & McDermid, J. A. 2010. An Iterative Approach for Development of Safety-Critical Software and Safety Arguments. Agile 2010.

[10] ISO 13485:2003 (2003) Medical devices — Quality management systems — Requirements for regulatory purposes. Second ed. Geneva, Switzerland, ISO.

[11] FDA 2007. Title 21--Food and Drugs Chapter I --Food and Drug Administration Department of Health and Human Services subchapter h--Medical Devices part 820 Quality System Regulation. U.S. Department of Health and Human Services.

[12] Denger, C., Feldmann, R., Host, M., Lindholm, C. & Shull, F. (2007) A Snapshot of the State of Practice in Software Development for Medical Devices. First International Symposium on Empirical Software Engineering and Measurement. Madrid, Spain.

[13] McCaffery, F., Burton, J., Casey, V., and Dorling, A., (2010) Software Process Improvement in the Medical Device Industry, in Encyclopedia of Software Engineering, P. Laplante, Editor. 2010, CRC Press Francis Taylor Group: New York. pp. 528 - 540.

[14] CMMI Product Team (2006) Capability Maturity Model® Integration for Development Version 1.2. Software Engineering Institute, Pittsburch PA.

[15] ISO/IEC 15504-5:2006 (2006) Information technology - Process Assessment - Part 5: An Exemplar Process Assessment Model. Geneva, Switzerland, ISO.

[16] Casey, V. (2010) Virtual Software Team Project Management. Journal of the Brazilian Computer Society, 16, pp 83 – 96.

[17] ISO/IEC 12207:2008 (2008) Systems and software engineering - Software life cycle processes. Geneva, Switzerland, ISO.

[18] ISO/IEC 15504-2 (2003) - Software engineering — Process assessment — Part 2: Performing an assessment. 2003: Geneva, Switzerland.

[19] Spice user Group, http://www.spiceusergroup.org/, accessed 31/09/2012

Mapping Between Service Designs Based on SoaML and Web Service Implementation Artifacts

Michael Gebhart
Gebhart Quality Analysis (QA) 82
Karlsruhe, Germany
michael.gebhart@qa82.de

Jaouad Bouras
ISB AG
Karlsruhe, Germany
jaouad.bouras@isb-ag.de

***Abstract*—Because of the increasing complexity of service landscapes and the requirement to fulfill quality attributes, such as loose coupling and autonomy, services have to be designed in detail before implementing them. For this purpose, the Object Management Group has standardized the Service oriented architecture Modeling Language, which enables the abstract formalization of service designs. However, existing mappings between abstract formalizations and web service implementation artifacts focus on certain modeling elements and do not consider service designs as self-contained development artifacts. In this article existing and applicable mapping rules for transforming service designs based on the Service oriented architecture Modeling Language into implementation artifacts are identified and missing rules are defined. For illustration purposes, service designs of a scenario in the context of a workshop organization system are transformed into web service implementation artifacts.**

Keywords-service design; SoaML; implementation; mapping; transformation.

I. INTRODUCTION

Today, services as methodology to integrate distributed systems become increasingly important. Software vendors enhance their products with service interfaces and enterprises organize their Information Technology (IT) according to service-oriented architecture (SOA) principles [18]. As result, services constitute the building blocks of today's IT and the complexity of the service landscape increases. Additionally, due to their influence, services are required to fulfill certain quality attributes [21], such as loose coupling and autonomy [20]. These attributes have been identified as important as they influence higher-value quality attributes of the entire architecture, such as its flexibility, maintenance, and cost-efficiency. In order to ensure the fulfillment of quality attributes with simultaneous handling of the service landscape complexity, a detailed planning of the services during their development and prior to their implementation is necessary. This phase is called service design phase. As part of a quality assurance process that analyses the quality of services based on abstract formalizations, a design of services has to be derived from implementation artifacts.

In order to enable a vendor-independent formalization of service designs with a common understanding and tool support, the Object Management Group (OMG) decided to work on a standardized meta model and profile for the Unified Modeling Language (UML) that enables the modeling of service-oriented architectures and their elements, the services. The result of this effort is the Service oriented architecture Modeling Language (SoaML), which is currently released in version 1.0. Compared to UML, SoaML adds several stereotypes necessary for the specifics of service-oriented architectures, such as service interfaces and participants. Today, SoaML gains increasing tool support, even IBM decided to replace their proprietary UML profile for software services with SoaML [24].

However, using SoaML in a service development process requires a systematic mapping between formalized service designs and implementation artifacts, such as web service artifacts. Otherwise, the service designs cannot act as development artifact within a model-driven approach as introduced by Hoyer et al. [22]. Furthermore, quality analyses that base on abstract SoaML models, such as introduced by Gebhart et al. in [1][2][16], cannot be applied on already implemented services as there is no way to derive an abstraction with guaranteed correct semantic. Mapping rules that are available today mostly focus on the mapping of certain selected modeling elements, such as the transformation of UML Classes into data types represented by XML Schema Definition (XSD). However, they do not consider a service design as a whole.

In this article, existing mapping rules based on UML and SoaML that are applicable for service designs are identified and summarized. Furthermore, additional mapping rules are defined if necessary. In this context, web services are assumed as implementation using in particular the Web Service Description Language (WSDL) and XSD to describe the service interface, Service Component Architecture (SCA) as component model, and the Business Process Execution Language (BPEL) for the implementation of composed services. As result, the mapping of service designs as self-contained development artifact is described which enables the modeling of service designs using SoaML within model-driven service development and quality assurance processes.

The article is organized as follows: Section II introduces the concept of service designs and analyzes mapping rules in existing work regarding their applicability for service designs. After introducing service designs of a workshop organization system in Section III, in Section IV, these service designs are mapped onto web services. Section V concludes this article and introduces future research work.

II. RELATED WORK

This section describes the fundamental terms and existing work in the context of specifying service designs and their mapping onto implementation artifacts.

A. *Service Design*

According to Gebhart et al. [15][16] and Erl [19], a service design consists of a service interface as external point of view and a service component fulfilling the functionality. In order to formalize service designs and to enable their transformation into implementation artifacts, Mayer et al. [19] introduce a UML profile for describing behavioral and structural aspects of service interactions. Similarly, within the SENSORIA project [13] a UML profile for the service interaction is specified. Also IBM [23] introduced a UML profile for modeling software services. Even though all of these UML profiles enable the modeling of services they lack in acceptance as they are not standardized. For that reason the OMG decided to work on a standardized UML profile [26] and a meta model to formalize service-oriented architectures and their services. As result, SoaML has been created [3].

According to Gebhart [4], in SoaML a service interface is described by a stereotyped UML Class that realizes a UML Interface describing the provided operations. A second UML Interface can be used for specifying callback operations the service consumer has to provide. An interaction protocol can be added as owned behavior. It is described by means of a UML Activity and determines the valid order of operation calls. The service component is represented by a UML Component stereotyped by a Participant. Ports with Service or Request stereotype constitute the access points to provided or required functionality and are typed by a certain service interface. An Activity as owned behavior visualized as UML activity diagram enables the specification of the internal logic. Figure 2 and Figure 6 illustrate a service interface and a service component.

B. *Mapping Rules*

In the context of mapping formalized service designs onto web service implementation artifacts based on XSD, WSDL, SCA, and BPEL approaches exist that consider either the derivation from SoaML-based models, UML models with own UML profiles applied, or standard UML models. This work is analyzed in order to identify applicable mapping rules to transform service designs.

For the generation of XSD, IBM [6] and Sparx Systems [7] provide adequate mapping rules that map UML class diagrams onto XSD artifacts and support both the transformation of classes and their relationships like aggregations, compositions, associations, and generalization. Both vendors integrate the mapping rules into their own tools, which enable a model-driven development with graphical tool support. The transformations are applicable to all UML models without any constraints. The applied rules can be used in our approach to map message types of service designs onto XSD.

Regarding WSDL, Grønmo et al. [9] discuss the advantages and disadvantages between using WSDL-independent and WSDL-dependent models. Their conclusion is that WSDL-dependent models obscure the behavior and content of modeled services and make service designs incomprehensible. WSDL-independent models in contract simplify building complex web services and integrating existing web services. For that reason they provide transformations based on UML class diagrams with custom WSDL-independent stereotypes. However, most of the presented transformations are based on standard UML elements and are thus applicable for service designs based on SoaML as it abstracts from WSDL details too. Also IBM[8] introduces mapping rules and an automatic transformation from UML to WSDL in [8]. These rules fully cover the transformation of standard UML elements into WSDL but are not described in detail. Only the relationships between source and target elements can be inferred and used in our work. In contradistinction to the previous related work the transformation generates also needed namespaces not bound to the source models but bound to the project structure used during the transformation. The project structure has the form of a file system containing source models and the relative paths will be used in order to generate namespaces for the target artifacts. This strategy may generate correct namespaces for a simple project. However, when merging the generated artifacts from many projects or changing the project structure during development the resulting namespace changes will make the WSDL files ambiguous.

Hahn et al. [12] present a transformation from a Platform Independent Model (PIM) to a Platform Specific Model (PSM), which converts SoaML to BPEL, WSDL, and XSD artifacts. Compared to our approach requiring a generation of BPEL processes from a UML activity diagrams, the authors use a BPMN processes as a source models for the generation of executable BPEL processes. Even though no detailed mapping rules are provided, a promising and consistent output is generated and the mapping is illustrated using a simple scenario. The approach can be considered as a proof for the possibility of producing web service artifacts from SoaML service designs. The authors restrict that a SoaML service interface is mapped onto one and only one WSDL document containing XSD types that represent the SoaML Messages. A new capability supported by the SoaML to WSDL transformation is the ability to generate Semantic Annotations for WSDL (SAWSDL).

For generating BPEL, Mayer et al. [5] discuss the difficulties when transforming a UML Activity illustrated by means of a UML activity diagram into an executable language, such as BPEL. They introduce two alternatives on generating BPEL constructs. The first alternative is to generate a BPEL process similar to the UML Activity where control nodes of the UML are replaced with edge and activity guards. The second alternative is to create a BPEL process with constructs in UML converted to their equivalent BPEL constructs. The first alternative is easy to be implemented and results in an unreadable and complex BPEL process whereas the second one results in a better structured orchestration. The approach presents a robust and promising transformation into BPEL. However, the WSDL artifacts are inferred from elements described by a custom

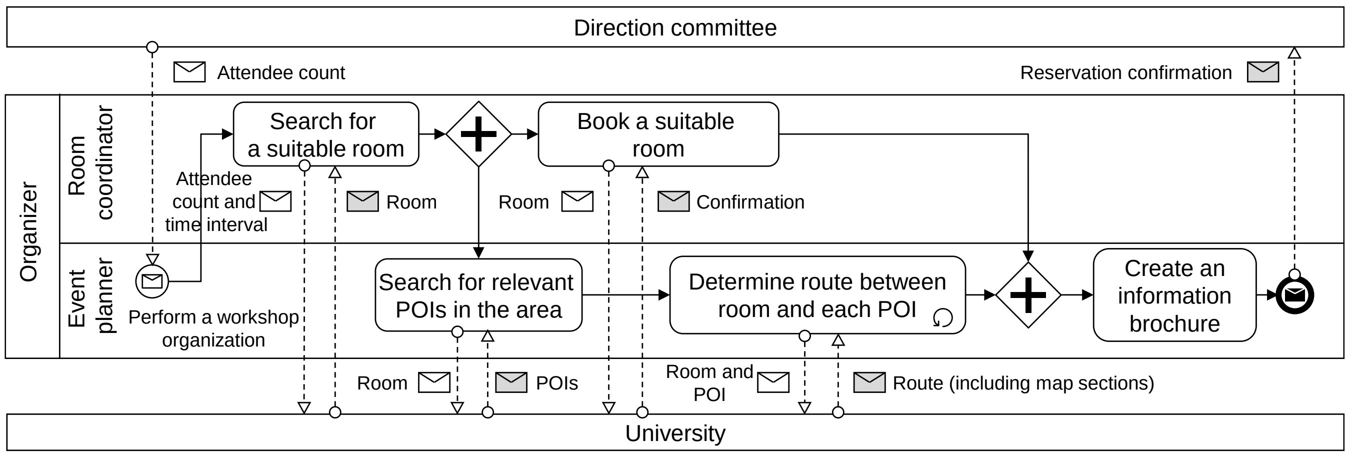

Direction committee
Attendee count
Reservation confirmation
Organizer
Room coordinator
Event planner
Search for a suitable room
Book a suitable room
Attendee count and time interval
Room
Room
Confirmation
Perform a workshop organization
Search for relevant POIs in the area
Determine route between room and each POI
Create an information brochure
Room
POIs
Room and POI
Route (including map sections)
University

Figure 1. Business process of the workshop organization scenario.

UML profile. Further mapping rules to transform workflows modeled using UML Activity elements onto BPEL artifacts are presented by IBM [14]. The approach handles some constraints of a UML Activity and provides adequate solutions. For example, to specify needed information, as for instance the partner links, the activity diagram should be extended with UML elements, such as input and output pins. Another constraint handled by the authors is how to model loop nodes in an Activity. Here, the authors propose a specific representation in UML to enable an easy and consistent generation of a BPEL loop element. These enhancements among others can be applied to consistently transform a UML Activity as the internal behavior of service components into an executable BPEL process.

SCA is a new technology for building applications and systems applying a service-oriented architecture paradigm. Combined with other technologies, such as WSDL and BPEL, SCA provides the underlying component model. In [10] Digre provides mapping rules for SoaML elements and SCA. The transformation is executed manually and the author mentions that ambiguities in the SoaML model may prevent from producing proper SCA models. This is exactly the reason, why a certain self-contained and well-understood design artifact, such as the service design in this article, has to be chosen when describing transformations. Another fully automated and tool-supported mapping of SoaML onto SCA artifacts is proposed by IBM [11]. The tool allows the application of SCA stereotypes to the source models in order to add more details specific to the SCA domain.

III. Scenario

In order to illustrate the transformation of service designs based on SoaML into web service implementation artifacts, in this section the scenario of a workshop organization at a university and the involved systems are introduced. Additionally, the development steps for creating the required service designs are explained.

A. *Business Requirements*

In a first step, the business requirements have to be formalized. For this purpose, beside business use-cases and the domain model as explained in [2] the business process expected to be supported by IT is described using the Business Process Model and Notation (BPMN) [25]. The process for the considered scenario is illustrated in Figure 1. The system based on this process helps visitors and members of the university in organizing a meeting or a workshop at a room located at the university campus. Two existing systems are involved in the realization of the business process namely the KITCampusGuide system and the facility management system. The KITCampusGuide system provides operations to manage Points of Interest (POI) and supports the determination of all relevant POIs (Parking, Cafeteria etc.) in the area surrounding the target and the provision of route guidance to all relevant POIs. The facility management system is concerned with room searches and enables the reservation of a room for a given number of attendees and at the desired time interval. Both systems are provided by the university.

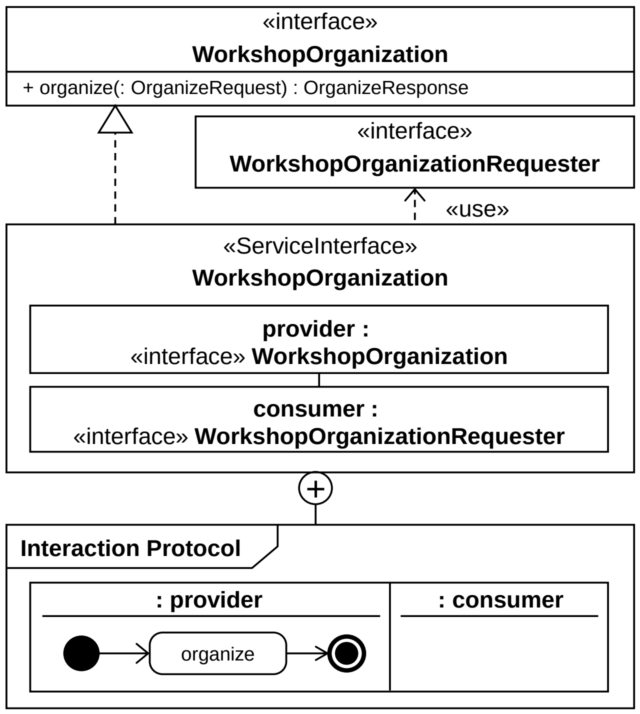

«interface»
WorkshopOrganization
+ organize(: OrganizeRequest) : OrganizeResponse
«interface»
WorkshopOrganizationRequester
«use»
«ServiceInterface»
WorkshopOrganization
provider : «interface» WorkshopOrganization
consumer : «interface» WorkshopOrganizationRequester
Interaction Protocol
: provider
: consumer
organize

Figure 2. Designed service interface.

B. Service Designs

In the second phase of the development process, the service design phase, a set of service designs has to be designed and modeled using SoaML. Each service design is built according to the understanding introduced in the Fundamentals section. In this article we demonstrate the approach using the "WorkshopOrganization" service that enables the orchestration of involved services. Figure 2 shows the designed service interface. The UML Interface realized by the ServiceInterface element lists the provided operation "organize" with its input and output parameters. The input and output parameters are defined using the message types "OrganizeRequest" and "OrganizeResponse" described in Figure 3. As the interface associated by means of the usage dependency does not contain any operation, the service consumer does not have to provide callback operations. This corresponds to the interaction protocol. Additionally, a service component is specified for this service representing the component that fulfills the functionality. The service component and its internal behavior are illustrated in Figure 4 and Figure 5.

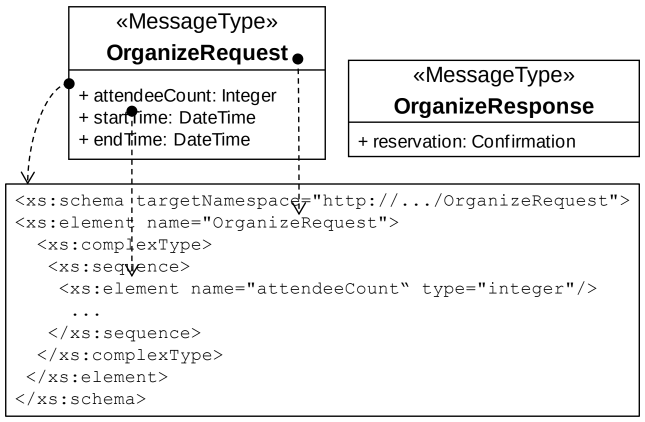
IV. MAPPING BETWEEN SERVICE DESIGNS BASED ON SOAML AND IMPLEMENTATION ARTIFACTS

In this section the steps necessary for mapping service design artifacts onto web service implementation artifacts are illustrated. Divided into four parts the first subsection targets the derivation of data types and their definitions using XSD. For the provided and required interfaces of the service interface, service interface descriptions based on WSDL with associated message types are generated. For realizing the orchestration of services, BPEL is derived from UML Activity elements added as owned behavior of the service component. Finally, a SCA component model describing the structure of the application is derived from the service component. For each step and for each transformation performed existing mapping rules are applied.

A. Derivation of Data Types

Data types contained within the SoaML service designs are expected to be mapped onto XSD to describe request and response messages used within WSDL operations.

The service interface in Figure 2 provides the operation "organize", which contains input and output messages in the form of UML DataTypes stereotyped by MessageType. They constitute containers for further data types described using attributes or UML Associations to other UML Classes. We follow the mapping rules provided by Sparx Systems [7]. Each input and output parameter is mapped onto an element with a complexType and a sequence of XML elements defining the attributes of the messages as demonstrated in Figure 3. The XSD descriptions are stored in separate files in order to allow other WSDL documents to reuse the data types. The separated XSD files are then imported into the WSDL document using an import statement with the corresponding namespace and schema location as shown in Source Code 1.



```
<xs:schema targetNamespace="http://.../OrganizeRequest">
<xs:element name="OrganizeRequest">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="attendeeCount" type="integer"/>
     ...
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 3. Derived XML Schema Definitions from SoaML messages.

```
<wsdl:types>
 <xs:import namespace="http://.../OrganizeRequest"
   schemaLocation="http://.../organize.xsd"/>
</wsdl:types>

<wsdl:message name="OrganizeRequestMessage">
 <wsdl:part name="body" element="OrganizeRequest"/>
</wsdl:message>
```

Source Code 1. Derived WSDL message types.

The following table summarizes the transformation, provides more details about the mapping rules, and lists the source and the target elements with necessary attribute configurations. Due to the lack of space, the following transformations are described in textually only.

TABLE I. SOAML ARTIFACTS TO XML SCHEMA DEFINITION

SoaML Artifact	XML Schema Definition
Package	A schema element with the "targetNamespace" attribute to identify and reference the XSD is generated.
Class (MessageType)	An element as a root element and a complexType definition containing a sequence of child elements are generated. The "name" attribute corresponds to the name of the class.
Attributes (ownedAttributes)	An attribute is mapped onto an element with the "name" and "type" attributes set to the same as in the source.
PrimitiveType, Datatype and MessageType	Are mapped onto the "type" attribute of an element generated while mapping the member attributes of a class. For each referenced data type an import element is used to add the corresponding external schema.
Association	An element is declared for each association owned by a class. The "name" attribute is set to the one of the association role. The "minOccurs" and "maxOccurs" reflect the cardinality of the association.
Generalization (Inheritance)	An extension element is generated for a single inheritance with the "base" attribute set to the base class name. The UML Attributes of the child class are then appended to an "all" group within the extension element.

B. *Derivation of Service Interfaces*

After generating data types, the operation definitions and their parameters can be derived from the SoaML service interface and its realized interface.

According to IBM [8], a port type acting as container for the operations is generated and each parameter is mapped onto a part element as shown in Source Code 1. The name of the port type is derived from the name of the realized interface in the SoaML service design and enhanced with the suffix "PortType". The WSDL operation element includes the attribute "name", which corresponds to the operation name within the service design. Additionally, the previously derived input and output messages are associated. In case of service inheritance the operations of the parent interface are copied into the same generated port type as stated by Hahn et al. [12]. This enables to overcome the not supported WSDL inheritance limitation.

```
<wsdl:portType name="WorkshopOrganizationPortType">
 <wsdl:operation name="organize">
  <wsdl:input message="OrganizeRequestMessage"/>
  <wsdl:output message="OrganizeResponseMessage"/>
 </wsdl:operation>
</wsdl:portType>
```

Source Code 2. Derived port type in WSDL.

Till now, the abstract part of a WSDL was generated. The concrete part encompasses deployment-specific details about how and where to access a service. A binding definition specifying the communication technology that can be used by the consumer is generated. The binding is named as a combination of the interface name and the suffix "SOAP". Additionally, it is associated with the prior defined port type by setting the attribute "type" to the name of the interface including the suffix "PortType". The messaging protocol binding and the transport protocol binding are set to Simple Object Access Protocol (SOAP) and Hypertext Transfer Protocol (HTTP). In this work we use SOAP as a default protocol. The final part focuses on the physical endpoint of the service. The endpoint is specified by a URL that has to be specified by the developer.

```
<wsdl:binding name="WorkshopOrganizationSOAP"
              type="WorkshopOrganizationPortType">
 <soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="organize"/>
</wsdl:binding>

<wsdl:service name="WorkshopOrganization">
 <wsdl:port binding="tns:WorkshopOrganizationSOAP"
            name="WorkshopOrganizationSOAP">
  <soap:address location="<server>:<port>"/>
 </wsdl:port>
</wsdl:service>
```

Source Code 3: Derived binding and service definition.

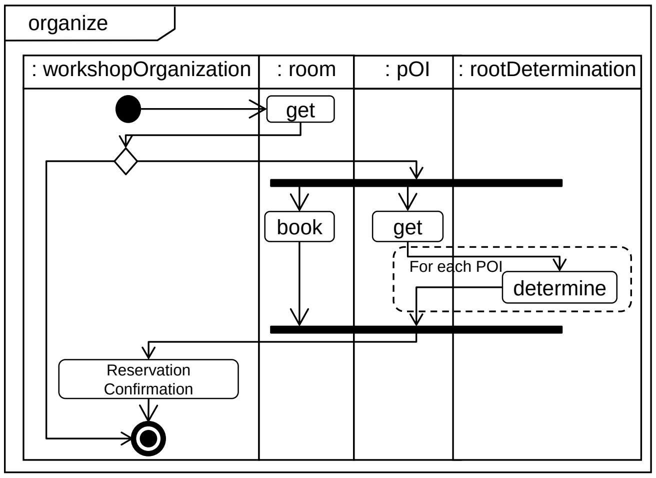


Figure 4. Interaction protocol for the operation "organize".

C. *Derivation of Executable Business Logic*

The mapping rules provided by IBM [14] cover all UML artifacts of a UML Activity involved in the derivation of control flow elements of a BPEL process. Additionally, new mapping rules to set attribute values were identified in this article and are also mentioned in the following transformation description.

The UML activity diagram in Figure 4 describes the internal behavior of a service operation "organize" and is considered to demonstrate the transformation for most often used control flow elements of a UML activity diagram. The first generated fragment for the BPEL process is the main scope, which exists only once and consists of a sequence of other activities. The first partition in the activity diagram contains an initial node which is mapped onto a receive activity with the attribute "partnerLink" set to the label of the partition namely "workshopOrganization". The attribute "operation" corresponds to the operation name in the interaction protocol. This activity is located at the top of the main scope and waits for an arriving message.

The involved web services are specified by separate WSDL definitions containing partnerLink definitions. In order to call these web services, the BPEL process sets a partnerLink for each invoke activity. The partnerLinks are derived from the label of the partitions, such as "room" or "rootDetermination".

```
<bpel:partnerLinks>
 <bpel:partnerLink name="client"
     partnerLinkType="WorkshopOrganization"
     myRole="WorkshopOrganizationProvider"/>
 <bpel:partnerLink name="room"
partnerLinkType="Room"
     partnerRole="RoomProcessProvider"/>
...
</bpel:partnerLinks>
```

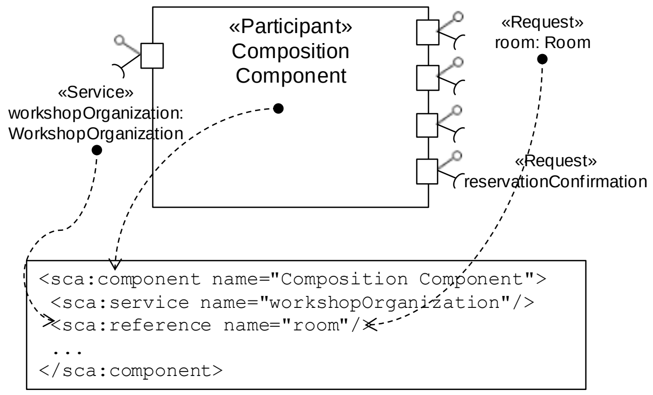
Source Code 4. Derived partnerLinks in the BPEL process.

The partition containing the initial node is mapped onto a partnerLink definition with the attribute “name” set to the value “client” representing the BPEL process itself. For the other partitions the attribute “name” is equal to the label of the respective partition. Moreover, the partnerLink defining the process itself has the attribute “myRole”, whereas other partnerLinks have an attribute “partnerRole” representing the role of an invoked web service. Source Code 4 shows the derived partnerLinks for the considered service operation and the invoked service “Room”.

After defining the partnerLinks, which belong to the abstract part of a BPEL process, the actions within the partitions are mapped onto invoke activities. Each activity has the attributes “name” and “operation” set to the name of the action. The attribute “partnerLink” is set to the corresponding partnerLink prior defined. The activities are located within the corresponding scopes of flow elements mapped later. The action “ReservationConfirmation” in the first partition is an opaque action executed by the BPEL process itself and thus is not mapped onto an invoke activity. After a skeleton for the BPEL process has been created, the control flow elements are derived from corresponding UML elements. The decision node is mapped onto a BPEL if-else construct. The condition of the node has to be added manually by the developer. The black bar representing a fork node and a parallel execution of the contained action is mapped onto a BPEL flow construct. The black bar representing a join node with incoming arrows is implicitly included in the earlier derived BPEL flow construct. The loop node is illustrated using a dashed area and is mapped onto a forEach construct with the attribute “parallel” set to the value “no”. If the loop node in UML contains a fork and a join node, the attribute “parallel” is set to “yes”.

D. Derivation of Component Models

In order to embed the already generated artifacts into an entire component model, SCA elements are derived from the service designs. Figure 5 illustrates the mapping between service components described by SoaML Participants and SCA elements, such as SCA Composites, Components, Services, References, and Wires, using mapping rules provided by Digre et al. in [10].



```
<sca:component name="Composition Component">
 <sca:service name="workshopOrganization"/>
 <sca:reference name="room"/>
 ...
</sca:component>
```

Figure 5. Derivation of SCA component model.

Regard naming conventions, each Participant is mapped onto a SCA component with name set to the label of the Participant. Since each SoaML Participant contains Services and Requests representing provided and required services, SCA Services and SCA References are generated. The names of these elements are set to the names of the ports within the SoaML Participant.

The SCA Composite is the basic unit of a composition in an SCA Domain and is an assembly of SCA Components, Services, References, and Wires. The service component presented earlier deals with the orchestration of external services and contains also a reference to an internal component for creating the reservation confirmation. These two components are to be grouped into an SCA Composite, whereas SoaML service channels wiring the Services to Requests are mapped onto SCA Wire elements. Additionally, if two Services or two Requests are wired together to delegate service calls, a promote element is added. Figure 6 illustrates the final SCA Composite in a graphical visualization as introduced by the standard.

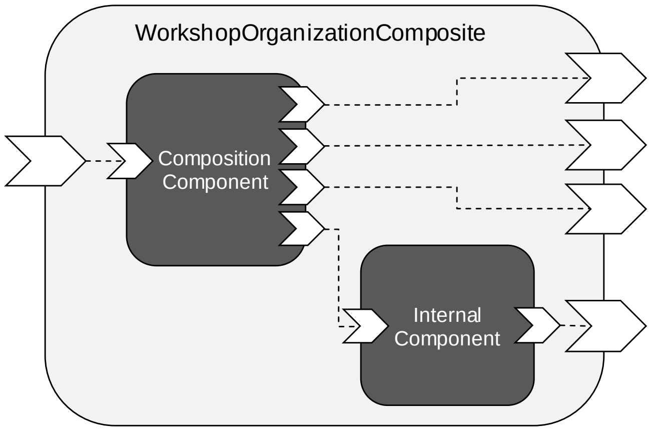


Figure 6. SCA Composite for the workshop organization process.

SCA requires that Service and Reference elements are compatible. The compatibility is assured by means of the assigned interfaces. The interfaces used in this context can be derived from service interfaces in SoaML as illustrated in section B. The resulting service interface descriptions based on WSDL can be embedded into the SCA Composite. For this purpose, based on the realized and used UML Interfaces representing provided and required interfaces within the service designs, a bidirectional service interface description using WSDL with a base and a callback interface is generated. An “interface.wsdl” element is added to the Service element with the attribute “interface” set to the URL of the WSDL service reprsenting the provided service interface “WorkshopOrganization”. The “callbackInterface” attribute of the Service element is set to the port type representing the “WorkshopOrganizationRequester”. For the corresponding SCA Reference, the assignment is reversed, i.e., the attribute “interface” of the interface element within the SCA Reference is set to the required interface and the attribute “callbackInterface” is set to the provided interface.

V. Conclusion and Outlook

In this article, we illustrated the mapping between service designs that are based on SoaML as standardized modeling language and web service implementation artifacts. As most mapping rules between abstract formalizations and implementation artifacts focus on UML or SoaML in general, we identified mapping rules that can be used for a transformation of service designs as self-contained service development artifacts.

The usage of the mapping rules was illustrated by means of a business process for organizing workshops at a university. In this context several service designs have been created and the most complex one was transformed into implementation artifacts. As implementation technologies web services based on XSD, WSDL, BPEL, and SCA have been chosen as they are most wide-spread today.

By identifying and describing necessary mapping rules, on the one hand this article enables IT architects and developers to systematically transform service designs into implementation artifacts. As result, this supports the usage of SoaML within a model-driven development processes for services as models based on this language can act as valuable development artifacts. On the other hand, the mapping rules help to derive abstract service designs from already implemented web services. As we work on automatic quality analyses of services designs based on SoaML as introduced in [1] and exemplified in [17], a systematic derivation of service designs from implementation artifacts is necessary. As result the mapping rules described in this article support quality analysis processes in the context of service-oriented architectures.

In order to leverage the mapping rules within our QA82 Architecture Analyzer tool [28] that enables the automatic quality analysis of service designs, the rules will be implemented by means of Query Views Transformation (QVT) [27]. This enables the automatic derivation of service designs from implemented web services that can be analyzed regarding wide-spread quality attributes, such as loose coupling and autonomy. As result, IT architects and developers will be able to automatically evaluate, whether developed web services support the flexibility, maintainability, and cost-efficiency of the IT.

References

[1] M. Gebhart and S. Abeck, “Metrics for evaluating service designs based on soaml”, International Journal on Advances in Software, 4(1&2), 2011, pp. 61-75.

[2] M. Gebhart and S. Abeck, “Quality-oriented design of services”, International Journal on Advances in Software, 4(1&2), 2011, pp. 144-157.

[3] OMG, “Service oriented architecture modeling language (SoaML) – specification for the uml profile and metamodel for services (UPMS)”, Version 1.0, 2012.

[4] M. Gebhart, “Service Identification and Specification with SoaML”, in Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments, Vol. I, A. D. Ionita, M. Litoiu, and G. Lewis, Eds. 2012. IGI Global. ISBN 978-1-46662488-7.

[5] Philip Mayer, Andreas Schroeder and Nora Koch, MDD4SOA Model-Driven Service Orchestration, 2008.

[6] IBM, Generating XSD Schemas from UML Models, Rational Systems Developer Information Center. http://publib.boulder.ibm.com/infocenter/rsdvhelp/v6r0m1/index.jsp. [accessed: July 11, 2012]

[7] Sparx Systems, XML Schema Generation, http://www.sparxsystems.com.au/resources/xml_schema_generation.html, 2011. [accessed: July 11, 2012]

[8] IBM, Transforming UML models into WSDL documents, Rational Software Architect. http://publib.boulder.ibm.com/infocenter/rsahelp/v7r0m0/index.jsp. [accessed: July 11, 2012]

[9] Roy Grønmo, David Skogan, Ida Solheim and Jon Oldevik, Model-driven Web Services Development, SINTEF Telecom and Informatics, 2004.

[10] Tom Digre, ModelDriven.org, http://lib.modeldriven.org/MDLibrary/trunk/Applications/ModelPro/docs/SoaML/SCA/SoaML to SCA.docx, May 2009. [accessed: July 11, 2012]

[11] IBM, Transforming UML models to Service Component Architecture artifacts, Rational Software Architect. http://publib.boulder.ibm.com/infocenter/rsahelp/v7r0m0/index.jsp. [accessed: July 11, 2012]

[12] Christian Hahn, David Cerri, Dima Panfilenko, Gorka Benguria, Andrey Sadovykh and Cyril Carrez, Model transformations and deployment, SHAPE 2010.

[13] SENSORIA, “D1.4a: UML for Service-Oriented Systems”, http://www.sensoria-ist.eu/, 2006. [accessed: July 11, 2012]

[14] IBM: Transforming UML models to BPEL artifacts, Rational Software Architect. http://publib.boulder.ibm.com/infocenter/rsahelp/v7r0m0/index.jsp, 2010. [accessed: July 11, 2012]

[15] M. Gebhart, M. Baumgartner, and S. Abeck, “Supporting service design decisions”, Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 76-81.

[16] M. Gebhart, M. Baumgartner, S. Oehlert, M. Blersch, and S. Abeck, “Evaluation of service designs based on soaml”, Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 7-13.

[17] M. Gebhart, S. Sejdovic, and S. Abeck, “Case study for a quality-oriented service design process”, Sixth International Conference on Software Engineering Advances (ICSEA 2011), Barcelona, Spain, October 2011, pp. 92-97.

[18] D. Krafzig, K. Banke, and D. Slama, Enterprise SOA – Service-Oriented Architecture Best Practices, 2005. ISBN 0-13-146575-9.

[19] T. Erl, Service-Oriented Architecture – Concepts, Technology, and Design, Pearson Education, 2006. ISBN 0-13-185858-0.

[20] T. Erl, SOA – Principles of Service Design, Prentice Hall, 2008. ISBN 978-0-13-234482-1.

[21] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, Addison-Wesley, 2003. ISBN 978-0321154958.

[22] P. Hoyer, M. Gebhart, I. Pansa, S. Link, A. Dikanski, and S. Abeck, “A model-driven development approach for service-oriented integration scenarios”, 2009.

[23] S. Johnston, “UML 2.0 profile for software services”, IBM Developer Works, http://www.ibm.com/developerworks/rational/library/05/419_soa/, 2005. [accessed: July 11, 2012]

[24] J. Amsden, “Modeling with soaml, the service-oriented architecture modeling language – part 1 – service identification”, IBM Developer Works, http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/index.html, 2010. [accessed: July 11, 2012]

[25] OMG, “Business process model and notation (BPMN)”, Version 2.0 Beta 1, 2009.

[26] OMG, “Unified modeling language (UML), superstructure”, Version 2.2, 2009.

[27] OMG, “Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification”, Version 1.1, 2011. [accessed: July 11, 2012]

[28] Gebhart Quality Analysis (QA) 82, QA82 Architecture Analyzer, http://www.qa82.de. [accessed: July 11, 2012]

Constructing Tool Chains Based on SPEM Process Models

Matthias Biehl, Martin Törngren
Embedded Control Systems
Royal Institute of Technology
Stockholm, Sweden
{*biehl,martin*}*@md.kth.se*

***Abstract*—The development of embedded systems requires a number of tools and it is widely believed that integrating the tools into an automated tool chain can improve the productivity of development. However, tool chains are not accepted by practitioners if they are not aligned with the established development culture, processes and standards. Process models exist for a variety of reasons, i.e., for documenting, planning or tracking progress in a development project and SPEM is the standardized formalism by the OMG for this purpose. We explore in how far a SPEM process models can be used for creating the skeleton of a tool chain, which is aligned with the process. We identify a number of relationship patterns between the development process and its supporting tool chain and show how the patterns can be used for constructing a tool chain. In two case studies, we examine the practical applicability of the patterns, when tailoring the design of a tool chain to a development process.**

***Keywords*-Generative Approach; Model Driven Development; Process Modeling; Tool Integration; Embedded Systems.**

I. Introduction

The engineering of an embedded system requires experts from a number of different engineering disciplines. Each engineering discipline prefers a different set of development tools that excel in that particular discipline [1]. The use of single, specialized tools has the potential to improve the development process, depending on the degree of automation they provide [2]. Since engineers need to exchange data and these tools do not interoperate well, a software external to the tools – a tool chain – is needed to facilitate the integration. Multiple tools have the potential to improve the productivity in the development process, depending on how well they are integrated with each other and their degree of automation [3]. Tool chains can provide different coverage of the development process; therefore, we distinguish between task-oriented tool chains with a small coverage and lifecycle-oriented tool chains with a larger coverage.

Many existing tool chains cover only one task in the development process, e.g., the tool chain between source code editor, compiler and linker. We call these tool chains task-oriented. The tools are used in a linear chain, so that the output of one tool is the input for the next tool. These tool chains have a relatively small scope and integrate a small number of tools from within one phase in the lifecycle. Characteristic for these traditional tool chains are their linear connections, using a pipes and filter design pattern [4].

In contrast, lifecycle-oriented tool chains have a larger scope, they focus on supporting the complete lifecycle from requirements engineering over verification and implementation to maintenance. In embedded systems development, these tool chains may span multiple disciplines such as software engineering, hardware engineering and mechanical engineering. These tool chains integrate a large number of different development and lifecycle management tools. In addition, modern development processes put new demands on the tool chain: processes might be agile, iterative or model-driven, which implies that the supporting tool chain cannot be linear.

When building a tool chain, it is thus important to study which development tools need to be connected. This information about the relationship of development tools is often already available in a formalized model. The Software & Systems Process Engineering Metamodel (SPEM) [5] can be used to describe the lifecycle. A SPEM model might already be available independently from a tool integration effort, e.g., as it is the case development with the Automotive Open Software Architecture (AUTOSAR) [6]. The information available in process models forms the skeleton of a tool chain, i.e., which tools are involved and how are they connected in the process. To construct an executable tool chain as a software solution, more detailed information is needed than is available in process models, e.g., information about the data of tools, how to access it, how to convert it and how to describe the relation between data of different tools. In this paper we evaluate to what extent information from existing SPEM models can be used for constructing a tool chain.

This paper is organized as follows: In Section II, we explain our approach for creating an initial design of a tool chain from a SPEM process model. We introduce SPEM for describing the processes and TIL to describe the architecture of a tool chain in Section II-A. This allows us to describe the relationship between process and tool chain as patterns in Section II-B and introduce ways of using the patterns in Section II-C. We apply the approach in two case studies in Section III. In the remaining sections, we discuss our approach, relate it to other work in the field, sketch future work and consider the implications of this work.

II. Approach

Tool chains are intended to increase the efficiency of development by providing connections between tools [3]. Ideally, connections for all tools used throughout the development process are provided; and in this case the tool chain supports the development process. The process provides constraints and requirements for the construction of the tool chain. While generic process models are available, e.g., the SPEM models for the Rational Unified Process (RUP) [7] or for AUTOSAR [6], companies also create individual process models for various purposes, e.g., to customize these generic models to their individual environments, to document the development process, to plan the development process, to track the progress in the development or to document their selection of tools.

If the process and the tool chain are described in a model, information from the process model can be reused for constructing a tool chain model. This approach ensures that the tool chain and the process are aligned. Process models only contain some, but not all information necessary for specifying tool chains. Especially the type of the connection between tools needs to be added later on.

A. *Formalized Description of Processes and Tool Chains*

In this section, we introduce modeling languages that are used for both the process and the design of the tool chain. We select two specific modeling languages, which on the one hand limits the scope of the work, on the other hand is a necessary preparation for formalizing and using the relationship between process and tool chain (cf. future work in Section VI).

1) Modeling the Product Development Process: There are both formal and informal processes in companies, documented to different degrees and there is an increasing trend to model processes. Several established languages exist for modeling processes or workflows. These languages have various purposes, BPMN [8] and BPEL [9] describe business processes and SPEM describes development processes. We apply SPEM, since it is a standardized and relatively widespread language for modeling development processes with mature and diverse tool support. A SPEM model describes both the product development process and the set of tools used and can thus be applied to describe the process requirements of a tool chain. An example model is provided in Figure 2. A number of concepts are defined in SPEM, we introduce here the core concepts that are relevant in the context of tool chains: a *Process* is composed of several *Activities*; an *Activity* is described by a set of linked *Tasks*, *WorkProducts* and *Roles*. A number of relationships, here represented by ≪.≫, are defined between the concepts of the metamodel: a *Role*, typically an engineer, can ≪*perform*≫ a *Task* and a *WorkProduct* can be marked as the ≪*input*≫ or ≪*output*≫ of a *Task*. A *WorkProduct* can be ≪*managed by*≫ a *Tool* and a *Task* can ≪*use*≫ a *Tool*.

2) Modeling the Design of the Tool Chain: We need an early design model that describes all important design decisions of a tool chain and chose to use the Tool Integration Language (TIL) [10], a domain specific modeling language for tool chains. TIL allows us not only to model a tool chain, but also to analyze it and generate code from it. The implementation of a tool chain can be partly synthesized from a TIL model, given that metamodels and model transformations are provided. Here we can only give a short overview of TIL, for an elaborated description of concrete graphical syntax, abstract syntax and semantics we refer to [10]. TIL has two basic types: *Components* and *Channels*, where *Components* are connected by *Channels*. The most important *Components* are *ToolAdapters*. For each tool, a *ToolAdapter* describes the set of data and functionality that is exposed by that tool in form of a tool adapter metamodel. Events can be triggered by *Users*. The relation between the tool adapters is realized as any of the following Channels: a *ControlChannel* describes a service call, a *DataChannel* describes data exchange by a model transformation and a *TraceChannel* describes the creation of trace links.

B. *Relationship Patterns between Process and Tool Chain*

If the process and tool chain are formalized as a model, we can also model the relationship between them more formally. A process described in SPEM might provide several opportunities for tool integration. Such an opportunity involves two tools and a direct or indirect connection between them. The tools and the connections found in SPEM are included into the tool chain architecture as ToolAdapters and Channels. The direction of the data flow can be determined by the involved work products, which have either the role of input or output of the task. Tasks connected to only one tool or tasks dealing with work products connected to the same tool do not require support from a tool chain; in these tasks engineers work directly with this tool, e.g., by using the GUI of the tool. To describe this relationship in more detail, we list patterns of both SPEM and TIL models and their correspondences.

Table I
Correspondences between SPEM and TIL Metaclasses

SPEM Metaclass	TIL Metaclass
RoleDefinition	User
ToolDefinition	ToolAdapter
TaskDefinition	Channel

The relationship patterns consist of a SPEM part, which matches a subgraph of a process model in SPEM, and a TIL part, which will become a new subgraph in the tool chain model in TIL. In the following, we show four SPEM patterns that describe tool integration related activities, they are illustrated in Figure 1, (1) - (4). The corresponding TIL pattern is the same for all SPEM integration patterns,

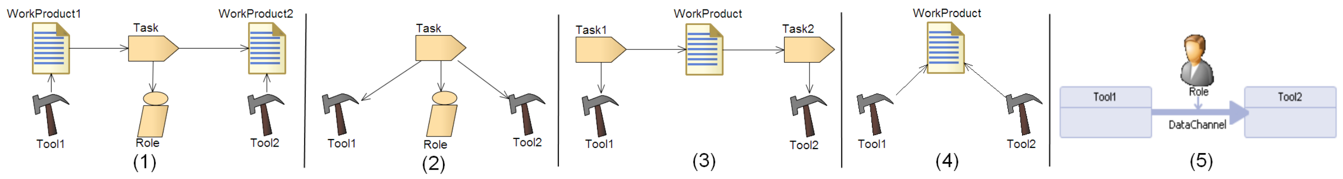


Figure 1. SPEM and TIL Patterns

visualized in (5). This mapping is established by pairs of model elements from both SPEM and TIL, whose name attribute is equivalent and whose types are of the metaclasses presented in Table I.

- *Task-centered Integration Pattern:* For each *TaskDefinition* in SPEM that has associated *WorkProducts* as input and output, where the input *WorkProduct* has a different associated *ToolDefinition* than the output *WorkProduct*, this pattern produces *ToolAdapters* and a *Channel* between them in the TIL model. The SPEM pattern is shown in (1) and can be observed in case study 1 in Figure 2 for the *Task TraceReq2UML* connecting the *WorkProduct RequirementsDatabase* and the *WorkProduct UMLFile*.
- *Multi-tool Task-centered Integration Pattern:* For each SPEM *TaskDefinition* with two SPEM *ToolDefinitions* associated with it, this pattern produces *ToolAdapters* and a *Channel* between them in the TIL model. The SPEM pattern is illustrated in (2).
- *WorkProduct-centered Integration Pattern:* For each SPEM *WorkProduct* that is both input and output of its associated *TaskDefinitions*, which have a different associated *ToolDefinition*, this pattern produces *ToolAdapters* and a *Channel* between them in the TIL model. The SPEM pattern is illustrated in (3) and can be observed in case study 2 in Figure 4 for the *WorkProduct ECUConfigurationDescription*, which is output of the *Task GenerateBaseECUConfiguration* and input to the *Task GenerateRTE*.
- *Multi-tool WorkProduct-centered Integration Pattern:* For each SPEM *WorkProduct* in SPEM that is associated to two different *ToolDefinitions*, this pattern produces *ToolAdapters* and a *Channel* between them in the TIL model. The SPEM pattern is illustrated in (4).

For all relationship patterns, the following constraints need to be fulfilled: For each *RoleDefinition* in SPEM that is connected to the *TaskDefinition*, we create a *User* model element in the TIL model. If a *ToolAdapter* corresponding to the *ToolDefinition* already exists in the TIL model, the existing *ToolAdapter* is connected, otherwise a new *ToolAdapter* is produced.

1) Implementation as Model Transformations: The implementation of the patterns offers possibilities for automation of the pattern usage. We implement the relationship patterns as model transformations, with SPEM as the source metamodel and TIL as the target metamodel. We chose the model-to-model transformation language in QVT-R, with the mediniQVT engine, and the Eclipse Modeling Framework (EMF) for realizing the metamodels. We use the SPEM metamodel, which is provided by the Eclipse Process Framework (EPF) under the name Unified Method Architecture (UMA), and for the visualization of SPEM models we use Enterprise Architect. For modeling and visualization of TIL, we use the TIL Workbench described in [10].

Patterns (1) to (5) are graphical representations of the relational QVT model transformation rules. Since QVT relational is a declarative language, the implementation describes the source patterns (1) - (4) and the target pattern (5) in the form of rules. Additionally, the attributes between source and target pattern are mapped, as described in Table I. Due to space constraints, the QVT rules are not included here.

C. Usage of Relationship Patterns

The relationship patterns can be used in different ways. Here, we apply the relationship patterns for constructing the initial design of a new tool chain starting from a process model. Other forms of using the relationship patterns are possible, but are not considered in depth here. We can use the patterns, e.g., for verification: based on a process model and a tool chain model we check if the requirements provided by the process are realized by the tool chain model.

The focus of this paper is the application of the relationship patterns to create an initial tool chain design in TIL from the process requirements expressed in the SPEM model. The patterns can be applied to a SPEM model that is complete and contains all necessary references to *ToolDefinitions*. The patterns ensure that the design of the tool chain is aligned with the process, a necessity for acceptance of the tool chain with practitioners. This design of the tool chain can be created in an automated way and might need to be iteratively refined by adding details.

The process model only provides the skeleton for the specification of a tool chain, such as the tools, which tools are connected and which user role is working with the tools. The process model does not provide the nature of the connections and the exact execution semantics of the automated tool chain. The nature of the connection can be data exchange, for creating trace links between tool data or for accessing specific functionality of the tool. This information needs to be added manually by configuring and choosing the right type of channel in TIL, a DataChannel, TraceChannel or ControlChannnel. Also, events need to be specified that trigger the data transfer or activate the tracing tool. For each ToolAdapter, a metamodel describing the data and functionality of the tool need to be added to the TIL model. For each DataChannel, a model transformation needs to be added.

To handle these cases, we add a refinement step, which complements the automated construction. Once this information is added, the TIL model can be used as input to a code generator for tool chains, as detailed in [11].

III. Case Studies

In this section, we apply the identified relationship patterns between a process model and a tool chain in two industrial case studies. This gives us the opportunity to study different ways of using the patterns and to explore the impact of different modeling styles.

A. Case Study 1: Construction of a Tool Chain Model for a Hardware-Software Co-Design Process

This case-study deals with an industrial development process of an embedded system that is characterized by tightly coupled hardware and software components. The development process for hardware-software co-design is textually described in the following:

- The requirements of the embedded system are captured in the IRQA[1] tool. The system architect designs a UML component diagram and creates trace links between UML components and the requirements.
- The UML model is refined and a fault tree analysis is performed by the safety engineer. When the results are satisfactory, the control engineer creates a Simulink[2] model for simulation and partitions the functionality for realization in software and hardware.
- The application engineer uses Simulink to generate C code, which is refined in the WindRiver[3] tool. The data from UML and Simulink is input to the IEC-61131-3 conform ControlBuilder tool. The data from ControlBuilder, Simulink and WindRiver is integrated in the Freescale development tool for compiling and linking to a binary for the target hardware.
- A hardware engineer generates code in the hardware description language VHDL from Simulink and refines it in the Xilinx ISE[4].

Based on the description of the process, we have created the corresponding SPEM model visualized in Figure 2.

We apply the model-to-model transformation that realizes the relationship patterns on the SPEM model in Figure 2.

[1]http://www.visuresolutions.com/irqa-web
[2]http://www.mathworks.com/products/simulink
[3]http://www.windriver.com
[4]http://www.xilinx.com/ise

This yields a tool chain model that is aligned with the process, as shown in Figure 3. By applying the task-centered integration pattern shown in (1), we identify integration tasks that are linked to two work products that in turn are linked to different development tools (e.g. the task *Trafo_UML2Safety*). Some other tasks are not concerned with integration, they are related to one tool only (e.g. the task *Use_UML*).

The TIL model resulting from application of the relationship patterns is internally consistent; this means that there are no conflicts, missing elements or duplications in the model. All tools mentioned in the SPEM model are also present in the TIL model as ToolAdapters and all ToolAdapters are connected. In addition, the approach ensures that the design of the tool chain matches the process.

Since the tool chain is modeled, we can easily change, extend and refine the initial model before any source code for the tool chain is developed. The TIL model is relatively small compared to the SPEM model, thus hinting at its effect to reduce complexity. When using the simple complexity metric of merely counting model elements and connections, we see that in the TIL model their number is reduced by $2/3$ compared to the SPEM model (cf. table II).

Table II
Size of the SPEM and TIL Model of Case Study 1

Count	Model Elements	Connections
SPEM Model	43	71
TIL Model	13	26

The important architectural design decisions of the tool chain (such as the adapters and their connections) can be expressed in TIL, while the complexity has been decreased compared to a SPEM model (cf. table II). The tool chain model can be analyzed and - after additional refinement with tool adapter metamodels and transformations - can be used for code generation, as detailed in [11], [10]. Moreover, the presented model-driven construction of the tool chain ensures that the tool chain is aligned with the process.

B. Case Study 2: Verification of a Tool Chain Model for AUTOSAR ECU Design

In this case study, we model a tool chain for AUTOSAR. AUTOSAR is developed by the automotive industry and defines an architectural concept, a middleware and in addition a methodology for creating products with AUTOSAR. The AUTOSAR methodology describes process fragments, so called capability patterns in SPEM. Generic AUTOSAR tool chains are implemented in both commercial tools and open frameworks, however, it is a challenge to set up tool chains consisting of tools from different vendors [12] and tool chains customized to the needs of a particular organization.

The SPEM process model is provided by the AUTOSAR consortium and is publicly available, which contributes to

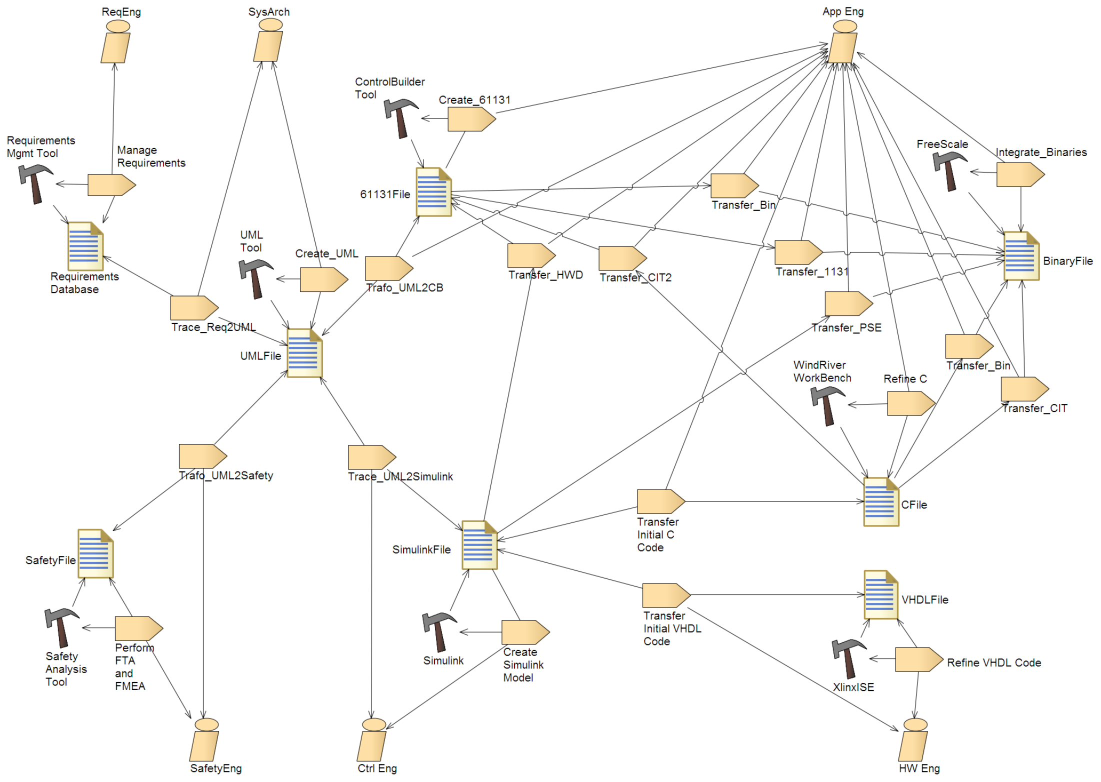


Figure 2. Case Study 1: Product Development Process of the Case Study as a SPEM Model

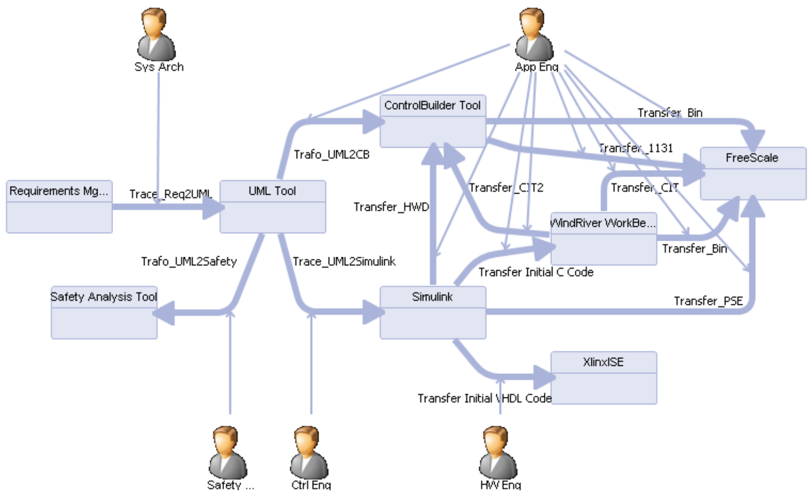


Figure 3. Case Study 1: Tool Chain of the Case Study as a TIL Model

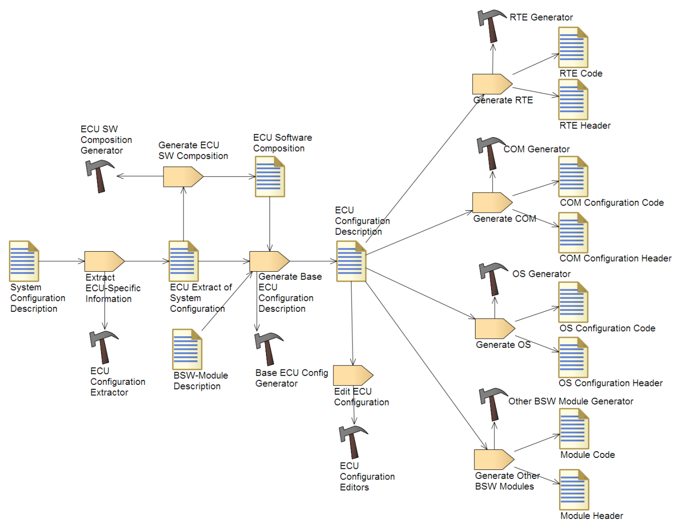

RTE Generator
RTE Code
Generate RTE
RTE Header
COM Generator
COM Configuration Code
Generate COM
COM Configuration Header
OS Generator
OS Configuration Code
Generate OS
OS Configuration Header
Other BSW Module Generator
Module Code
Generate Other BSW Modules
Module Header
ECU SW Composition Generator
Generate ECU SW Composition
ECU Software Composition
ECU Configuration Description
System Configuration Description
Extract ECU-Specific Information
ECU Extract of System Configuration
Generate Base ECU Configuration Description
ECU Configuration Extractor
BSW-Module Description
Base ECU Config Generator
Edit ECU Configuration
ECU Configuration Editors

Figure 4. Case Study 2: Excerpt of the AUTOSAR Methodology for Designing an ECU [6].

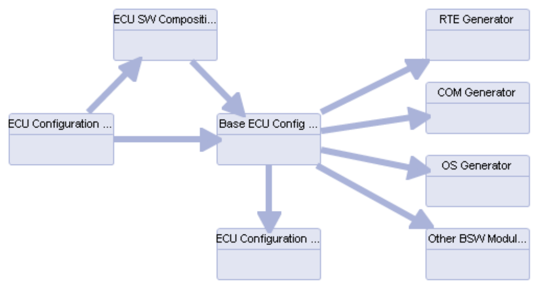


Figure 5. Case Study 2: AUTOSAR Tool Chain for Designing an ECU as a TIL Model

the transparency of this case study. An excerpt of this model that is relevant for the design of a ECU, is depicted in Figure 4. We use this excerpt of the SPEM model to initialize a tool chain. Applying the patterns creates the tool chain model in TIL, illustrated in Figure 5. Out of the four different SPEM parts of the relationship patterns (1) - (4), only the workproduct-centered integration pattern (3) matched several times in the SPEM model. This is due to the modeling style used in the AUTOSAR methodology, where *WorkProducts* are used as an interface for integrating tools.

The generated skeleton of the tool chain lays the foundation for ensuring that the AUTOSAR methodology can be realized by this tool chain. The skeleton can now be refined with metamodels, model transformations and the behavior.

IV. Discussion

This approach assumes that an appropriate process model for tool chains is available. We assume that the process model does not contain any integration related overhead, i.e., explicit representation of a model transformation tool and intermediate data model. We assume that tools have been assigned to process activities. The choice for certain tools is usually independent of automating the tool chain, the choice merely needs to be documented in the process model.

The use of the presented patterns is limited to processes represented in SPEM and tool chains modeled in TIL. However, the patterns could be adapted to similar process metamodels.

While it is possible to describe a part of the requirements for a tool chain with SPEM, SPEM models are not executable. We thus extract the relevant information from SPEM models to ensure that all tools and connections are represented in the tool chain. Based on this information, we generate a TIL models, which can be made executable by following a well-defined process, described in [10].

We have evaluated the approach in two case studies from the area of embedded systems. We do not see any reason why the patterns could not be applied for creating tool chain from process models in other application areas in the future.

V. Related Work

Related work can be found in the areas tool integration and process modeling. There are a number of approaches for tool integration, as documented in the annotated bibliographies [13], [14]. Most of the approaches do not take the process into account; in this section we focus on those approaches that do. We also take approaches from process modeling into account and classify them according to two dimensions: The first dimension comprises different execution mechanisms, which can be interpretation vs. compilation. The second dimension comprises different process modeling languages, which can be proprietary vs. standardized.

Interpretation-based approaches [15], [16], [17] use the process definition for tool integration. This technique is also known as enactment of process models. Since the description of the process is identical to the specification of the tool chain, no misalignment between process and tool chain is possible. There are two preconditions for this approach: the process model needs to be executable and the access to data and functionality of the development tools needs to be possible. The use of a proprietary process model for interpretation in tool chains is introduced in [18], as the process-flow pattern. Approaches that extend SPEM make the process model executable [15], [16]. The orchestration of tools by a process model is shown in [17]. However, the interpretation of integration related tasks is often not possible, since the interfaces to the development tools are not standardized. Thus, the use of process enactment to build tool chains is limited.

Compilation-based approaches transform the process model into another format, where the process model serves as a set of requirements. Proprietary process models provide great flexibility to adapt them to the specific needs of tool integration. An integration process model is developed in [19], where each process step can be linked to a dedicated activity in a tool. For execution, it is compiled into a low-level process model. The proprietary process model needs to be created specifically for constructing a tool chain. In this work, we use the standardized process metamodel SPEM [5], which allows us to reuse existing process models as a starting point for building tool chains and as a reference for verification for tool chains.

VI. Conclusion and Future Work

Process models exist for a variety of reasons, i.e., for documenting, planning or tracking progress in a development project and SPEM is the standardized formalism by the OMG for this purpose. In this paper, we recognize the development process modeled in SPEM as a set of requirements for the architecture of tool chains. We devise a number of patterns for creating the skeleton of a tool chain, which is aligned with the process.

In this work, we have selected specific languages to express the patterns; in the future, we would like to experiment with additional languages for describing the process model, such as BPMN. This might help us to further generalize the patterns.

Acknowledgement

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement 100203.

References

[1] J. El-khoury, O. Redell, and M. Törngren, “A Tool Integration Platform for Multi-Disciplinary Development,” in *31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 442–450, 2005.

[2] T. Bruckhaus, N. H. Madhavii, I. Janssen, and J. Henshaw, “The impact of tools on software productivity,” *Software, IEEE*, vol. 13, no. 5, pp. 29–38, Sep. 1996.

[3] M. Wicks and R. Dewar, “A new research agenda for tool integration,” *J. of Sys. and Sw.*, vol. 80, no. 9, pp. 1569–1585, Sep. 2007.

[4] M. Shaw and D. Garlan, *Software architecture*. Prentice Hall, 1996.

[5] OMG, “Software & Systems Process Engineering Metamodel Specification (SPEM),” ”OMG”, Tech. Rep., Apr. 2008.

[6] AUTOSAR Consortium. (2011, Apr.) Automotive open software architecture (AUTOSAR) 3.2. [Online]. Available: http://autosar.org/

[7] P. Kruchten, *The Rational Unified Process*. Addison-Wesley Pub (Sd), 1998.

[8] OMG, “Business Process Model And Notation (BPMN),” ”OMG”, Tech. Rep., Jan. 2011.

[9] OASIS, “OASIS Web Services Business Process Execution Language (WSBPEL) TC,” ”OASIS”, Tech. Rep., Apr. 2007.

[10] M. Biehl, J. El-Khoury, F. Loiret, and M. Törngren, “On the Modeling and Generation of Service-Oriented Tool Chains,” *Journal of Software and Systems Modeling*, vol. 275, 2012 [Online]. Available: http://dx.doi.org/10.1007/s10270-012-0275-7

[11] M. Biehl, J. El-Khoury, and M. Törngren, “High-Level Specification and Code Generation for Service-Oriented Tool Adapters,” in *ICCSA2012*, pp. 35–42, Jun. 2012.

[12] S. Voget, “AUTOSAR and the automotive tool chain,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE ’10, pp. 259–262, 2010.

[13] M. N. Wicks, “Tool Integration within Software Engineering Environments: An Annotated Bibliography,” Heriot-Watt University, Tech. Rep., 2006.

[14] A. W. Brown and M. H. Penedo, “An annotated bibliography on integration in software engineering environments,” *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 3, pp. 47–55, 1992.

[15] A. Koudri and J. Champeau, “MODAL: A SPEM Extension to Improve Co-design Process Models New Modeling Concepts for Today’s Software Processes, LNCS vol. 6195, ch. 22, pp. 248–259, 2010.

[16] R. Bendraou, B. Combemale, X. Cregut, and M. P. Gervais, “Definition of an Executable SPEM 2.0,” in *APSEC*, pp. 390–397, 2007.

[17] B. Polgar, I. Rath, Z. Szatmari, A. Horvath, and I. Majzik, “Model-based Integration, Execution and Certification of Development Tool-chains,” in *Workshop on model driven tool and process integration*, pp. 36–48, Jun. 2009.

[18] G. Karsai, A. Lang, and S. Neema, “Design patterns for open tool integration,” *Software and Systems Modeling*, vol. 4, no. 2, pp. 157–170, May 2005.

[19] A. Balogh, G. Bergmann, G. Csertán, L. Gönczy, Horváth, I. Majzik, A. Pataricza, B. Polgár, I. Ráth, D. Varró, and G. Varró, “Workflow-driven tool integration using model transformations,” in *Graph transformations and model-driven engineering*, pp. 224–248, 2010.

Tracing Requirements and Source Code during Software Development

Alexander Delater, Barbara Paech
Institute of Computer Science
University of Heidelberg
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
{delater, paech}@informatik.uni-heidelberg.de

Nitesh Narayan
Institute of Computer Science
Technical University of Munich
Boltzmannstrasse 3, 85748 Garching, Germany
narayan@in.tum.de

***Abstract*—Traceability links between requirements and source code are often created after development. This reduces the possibilities for developers to use these traceability links during the development process. Additionally, existing approaches applied after development do not consider artifacts from project management, which are used for planning and organizing a project. These artifacts can serve as a mediator between requirements and source code. In contrast to these existing approaches, we present an approach that creates traceability links between requirements and source code as the development progresses by incorporating artifacts from project management. In this paper, we make two key contributions. First, a Traceability Information Model integrating requirements, source code and artifacts from project management. Second, an approach for the (semi-) automatic creation of traceability links between artifacts from the Traceability Information Model achieving traceability between requirements and source code during the development process. We identified a catalog of information needs of developers from literature regarding requirements, source code that realizes these requirements, and work done by co-workers implementing these requirements. The presented approach satisfies the information needs of the developers during the development process, while keeping the traceability links up-to-date.**

Keywords*-*traceability; requirements; source code; software development; information needs.

I. Introduction

Traceability information supports the software development process in various ways, amongst others, program comprehension, change management, software maintenance, software reuse and prevention of misunderstandings [1]. Traceability between requirements and source code has been extensively researched in the past years and much progress has been made in this field. Because the manual creation of traceability links between requirements and source code is cumbersome, error-prone, time consuming and complex [2], a major focus in research is on (semi-) automatic approaches. Existing approaches use various techniques, e.g., information retrieval [3] [4], execution traces [5], static/dynamic analysis [6], subscription-based or rule-based link maintenance [7] or combinations of them [8]. However, all these approaches do not use artifacts from project management, but such artifacts, e.g., sprints and work items, are widely used in software development projects nowadays. Thus, the first key contribution of this paper is a Traceability Information Model (TIM) integrating requirements, source code and project management artifacts.

Traceability links between requirements and source code are often created after development [9] using the aforementioned approaches. This reduces the possibilities for developers not only to use their project knowledge to improve the quality of the traceability links, but also to use the traceability links during software development and maintenance. Therefore, we argue that traceability links between requirements and source code should also be created during the software development process and not only after development. Thus, the second key contribution of this paper is a (semi-) automatic approach for creating traceability links between artifacts from the TIM achieving traceability between requirements and source code during the development process.

Additionally, while creating traceability links between requirements and source code, the information needs of the developers during development should play a major role. The importance of such information needs is presented by Ko et al. [10] for collocated software development teams, and Sillito et al. [11] on questions raised during a program change task. We identified a catalog of information needs of developers from the contributions of Ko et al. and Sillito et al. regarding requirements, source code that realizes these requirements, and work done by co-workers implementing these requirements. The presented approach satisfies the information needs of the developers during development while keeping the traceability links up-to-date.

The paper is structured as follows: Section II provides background knowledge about a model unifying system development and project management that we built upon, and the subset of artifacts from this model that we focus on in this work. Section III defines a model for source code representation and introduces the TIM integrating artifacts from system development model, project management model and source code model. Section IV introduces an approach to (semi-) automatically create traceability links between artifacts in the TIM. Section V provides a fictional example project to highlight the benefits of the presented approach. Section VI introduces a catalog of information needs and how they are satisfied using the artifacts and relations from the TIM created for the example project. Section VII describes related work and Section VIII provides a discussion of the presented contributions. Finally, Section IX summarizes the contributions and discusses future work.

II. Background

This section provides background knowledge about a model unifying system development and project management, and the subset of artifacts from this model that we focus on in this work.

A. MUSE Model

In software development projects, two different types of models are used for abstraction: the *system model* and *project model* [12]. Artifacts from the system model describe the system under construction, such as requirements, components or design documents. Artifacts from the project model describe the on-going project, such as work items, developers, sprints or meetings. These two models have already been integrated within a model called MUSE: Management-based Unified Software Engineering [12].

While the MUSE model describes the system under development and its project management, it does not provide traceability to the source code. The MUSE model is implemented in the model-based CASE tool UNICASE [13], which is a plugin for the Eclipse integrated development environment (IDE) and is developed in an open source project [14]. For this work, we build upon the MUSE model and extend it with a new *code model* to support traceability to the source code. The code model is introduced in Section III.

B. Focus on Subset of Artifacts

The MUSE model supports a large amount of artifacts. Therefore, we focus on a subset of artifacts that are required by the information needs of the developers regarding requirements and co-workers implementing these requirements. From the system model, we focus on the artifacts of *feature* and *functional requirement* representing requirements at different levels of detail. A feature is an abstract description of a requirement, and it is detailed by one or more functional requirements. From the project model, we focus on the artifacts of *developers*, *work items* and *sprints*. Work items represent a unit of work and are the task descriptions used in software development projects (we use the term work item instead of *task* to avoid misunderstandings with the term task used in requirements engineering). They can describe work for new implementations and bug fixing. As they are the basis of the daily work, they are regularly kept up-to-date [15]. Developers are assigned to work items. Sprints are used to organize work items in work packages and they provide a time frame to realize the work items.

III. Modeling Code and Traceability Information

In this section, we define the representations of source code that extend the MUSE model by a new code model. Furthermore, we define a TIM integrating the artifacts we focus on from the MUSE model and the representations of source code from the code model.

A. Code Model

The code model contains file-based and change-based representations of source code. We chose to use these representations because they are widely used in software development projects and are independent of any programming language. This is supported by a comprehensive literature survey by Kagdi et al. [16]. For file-based representations, we focus on *code files* containing source code. For change-based representations that are supported by a version control system (VCS), we focus on *revisions*. Revisions themselves contain changed code files. Other representations would be possible, e.g., class, method or interface. However, not all programming languages support these artifacts, reducing the applicability of the code model. Table I shows the different representations of source code and their attributes.

Table I
Attributes of Representations of Source Code

Type	Attributes
Code File	fileName, projectName, pathInProject
Revision	date, author, number, repositoryUrl, pathInRepository, commitMessage, changedCodeFiles [added, modified, or deleted]

In the following, we describe the reasons for choosing these attributes. For code files, we require the attributes *fileName*, *projectName* and *pathInProject* to locate them in a project. For revisions, we require the attributes *date* and *author* to tell when and by whom the revision was created. We also need the attributes *number*, *repositoryUrl* and *pathInRepository* to reliably locate the revision in a VCS. Moreover, the attribute *commitMessage* is required to describe the changes contained in this new revision. This comment is usually written by the author of the revision and is optional. The most important information of a revision is stored in the list *changedCodeFiles* and each artifact in this list has the same attributes as the artifact code file. Moreover, each changed code file in the revision has a *state* [added, modified, or deleted] that shows if the code file was newly added, existed before and was only modified or was deleted in the revision.

B. Traceability Information Model

Traceability in a project should be documented in and driven by a Traceability Information Model [17]. A basic TIM consists of two types of entities: traceable artifacts and traceability links between these artifacts. It also defines which types of artifacts are intended to be traced to which related artifact types and by what type of traceability link. We define a TIM that realizes the artifacts from the MUSE model that we focus on and extends it by the two representations of source code. The TIM (see Figure 1) shows all artifacts that we want to connect: system model artifacts (feature, functional requirement), project model artifacts (sprint, work item, developer) and code model artifacts (code file, revision).

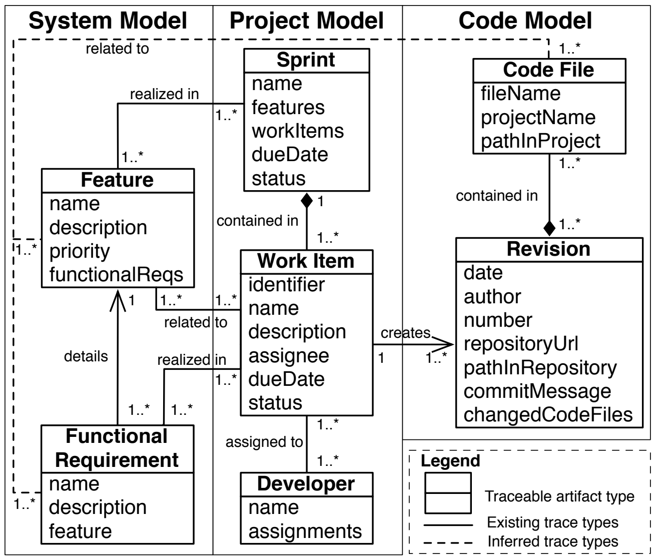


Figure 1. Traceability Information Model integrating system model, project model and code model

Figure 1 depicts the core traceable components. A feature is realized in a sprint and is detailed in one or more functional requirements. Functional requirements are realized by work items. A work item must have one or more linked functional requirements. A feature can be related to a work item, e.g. during bug fixing. Work items are contained in a sprint and are assigned to developers. One work item can create one or more revisions. A revision contains one or more changed code files. All these traceability links between the artifacts are represented as straight lines in Figure 1. All artifacts from the TIM can be found in common software development projects.

The central artifact is the work item, as it connects the artifacts from the system model to artifacts from the code model. Using work items, we can achieve traceability between requirements and source code by inferring traceability links. An inferred traceability link between two artifacts is derived from all artifacts in between these two artifacts. For example, a functional requirement is realized in one or more work items, and a work item creates one or more revisions containing code files. Thus, we can infer traceability links between the functional requirement and the code files. The inference process and the used algorithm is explained in detail in Section IV. The inferred traceability links are represented as dashed lines between features, functional requirements and code files in Figure 1.

IV. Tracing Requirements and Source Code during Software Development

This section presents a (semi-) automatic approach for creating traceability links between artifacts from the TIM during the software development process, especially between requirements and source code using work items.

We presume that the following situation is present in the software development project. First, there exists a list of features and detailing functional requirements. Second, a project manager has planned the realization of the features in sprints and s/he has broken down the realization of the functional requirements into work items for the developers in the software development project. Third, the work items are already assigned to developers, e.g. manually by the project manager or using an approach by Helming et al. for semi-automatic assignment of work items [15]. For the presented approach, we assume that all artifacts from the TIM are available in one integrated environment supporting traceability links between all artifacts. Such an integrated environment can be supported by the model-based CASE tool UNICASE [13].

A. Capturing Traceability Links

Figure 2 depicts the process of capturing traceability links and every activity is described in detail in the following. The core idea of creating traceability links between artifacts of the TIM is letting the developers create these links themselves. First, the developer selects a work item from his/her list of assigned work items and tells the system that s/he starts implementing source code. While working on the work item, all features or functional requirements the developer looks at during implementation are automatically captured by the system, meaning that the system logs these types of artifacts while a developer opens them during implementation. The developer can look at the linked features or functional requirements of the work item or look at other artifacts of these types to get a better understanding during implementation. After finishing the implementation of a work item in the source code, the developer tells the system that s/he has stopped implementation.

The developer does not immediately commit the changes to the VCS. Instead, before the commit, s/he has to validate two lists of artifacts: one list of all changed code files in the source code, and another list of all captured features or functional requirements that s/he looked at during implementation. While the former is standard in software development and already supported by any VCS, the latter represents additional work for the developers. This validation is necessary to only create relevant traceability links between the work

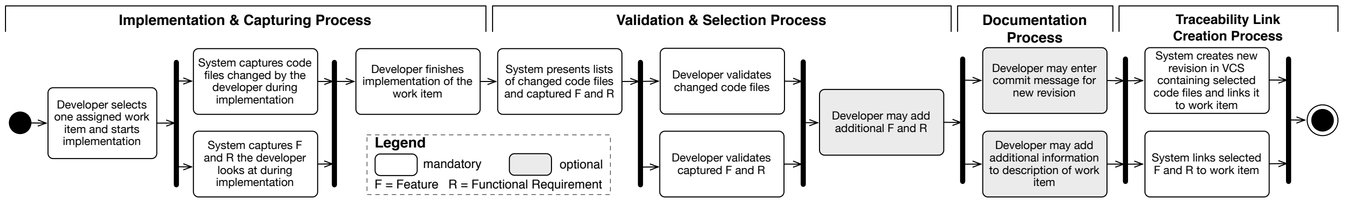

Implementation & Capturing Process
Validation & Selection Process
Documentation Process
Traceability Link Creation Process
Developer selects one assigned work item and starts implementation
System captures code files changed by the developer during implementation
System captures F and R the developer looks at during implementation
Developer finishes implementation of the work item
System presents lists of changed code files and captured F and R
Developer validates changed code files
Developer validates captured F and R
Developer may add additional F and R
Developer may enter commit message for new revision
Developer may add additional information to description of work item
System creates new revision in VCS containing selected code files and links it to work item
System links selected F and R to work item
Legend
mandatory
optional
F = Feature R = Functional Requirement

Figure 2. Process of Capturing, Validating and Creating Traceability Links (UML Activity Diagram)

item and these artifacts. For example, a developer can look at a functional requirement during development, which is not directly involved in the implementation, but related to the work item. During validation, a developer removes unrelated artifacts from the list. Furthermore, an optional activity for the developer is to select additional features or functional requirements that are related to the work item, but that s/he has not had a look at during implementation. Two other optional activities are to enter a commit message for the new revisions or add additional information to the description of the work item.

After validating all artifacts and optionally adding further features or functional requirements, the developer selects to commit all information to the VCS. The system then creates a new revision containing only the selected code files. The work item is linked to the newly created revision, and the attribute *identifier* of the work item is inserted at the end of the commit message of the revision to achieve bi-directional traceability between work item and revision. Moreover, the system links all validated and selected features and functional requirements to the work item.

B. Inferring Traceability Links between Requirements and Source Code

The created traceability links are used to infer links between requirements and code, specifically between features, functional requirements and code files. The Algorithm IV.1 for creating inferred traceability links is executed when the status of a work item is changed by the developer from *assigned* to *done*. The algorithm connects in a brute force manner all linked requirements (features, functional requirements) of a work item with all the code files in the linked revisions of the work item. The statement *workItem.getLinkedRequirements()* returns all features and functional requirements. The statement *workItem.getRevisions()* returns the linked revisions of a work item sorted by attribute *date* in ascending order. This is important because the algorithm applies change operations to the artifacts which need to be in the order they occurred. If the algorithm identifies already existing links, it does not create them again. If a code file was modified, its information consisting of the attributes *fileName*, *projectName* and *pathInProject* is updated for each linked requirement. If a code file is deleted, the link to the requirement is removed.

Algorithm IV.1: INFERTRACES($workItem$)

```
$allReqs = workItem.getLinkedRequirements()$
$allRevisions = workItem.getRevisions()$
for each $rev \in allRevisions$
  $allCodeFiles = rev.getAllChangedCodeFiles()$
    for each $cf \in allCodeFiles$
      $state = cf.getState()$
        if $state = ADDED$
          for each $req \in allReqs$
            if $req.isNotConnectedTo(cf)$
              $req.addLinkTo(cf)$
        if $state = MODIFIED$
          for each $req \in allReqs$
            if $req.isNotConnectedTo(cf)$
              $req.addLinkTo(cf)$
            $req.getLinkedCodeFile(cf)$
                $.update(fileName, projectName$
                $pathInProject)$
        if $state = DELETED$
          for each $req \in allReqs$
            $req.removeLinkTo(cf)$
return $(workItem)$
```

V. EXAMPLE

We use a fictional example project to highlight the benefits of the presented approach and to support discussion. The example project is a Java application called *Movie Manager* that one can use to manage his/her movie collection. Users can add, modify and delete movies as well as rate them. The application supports importing data about performers (actor/actress) of a movie from an Internet movie database. Presenting all information about the artifacts in the project is beyond the purpose of this paper. Therefore, we only provide a list of used artifacts with short descriptions to support basic understanding. There are two features (F) and six detailing functional requirements (R) (see Table III). The project is planned in two sprints with feature F1 developed in Sprint 1 and feature F2 developed in Sprint 2. Amy, Bill and Carl are members of a team collaborating to develop the application and they have eight work items (W) (see Table IV). Amy is mainly focusing on the data objects within the application, Bill is responsible for the user interface, and Carl is doing bug fixing.

Table II
CHANGED CODE FILES AND CAPTURED TRACEABILITY LINKS OVER TEN REVISIONS OF MOVIE MANAGER

C1	C2	C3	C4	C5	R1	R2	R3	R4	R5	R6	F1	F2	Work Item	Dev.	Rev.	Commit Message
c					LB						CL		W1	Amy	1	Created Data Object 'Movie' #W1
m	c				CL	LB	LB				CL		W2	Bill	2	Implemented basic UI for listing Movies #W2
	m					CL	LB				CL		W2	Bill	3	Display and change information of Movie #W2
	m	c						LB					W3	Bill	4	Added basic Rating Control and added it to Movies UI #W3
		m						LB			CL		W3	Bill	5	Completed 5 star Rating Control #W3
			c							LB		CL	W4	Amy	6	Created Data Object 'Performer' #W4
	m		m							LB			W5	Bill	7	Display information of Performers for Movie, currently showing dummy data as import needs to be implemented #W5
		m						LB			CL		W6	Carl	8	BugFix for Rating Control #W6
				c					LB			CL	W7	Amy	9	Performer Import #W7
				m					LB			CL	W8	Carl	10	Bugfix for Performer Import #W8

c = created m = modified LB = Linked Before CL = Captured Link

Table III
FEATURES AND FUNCTIONAL REQUIREMENTS

Artifact	Description	Detailing
F1	Movie Management: Add, modify and delete movies as well as rate them	-
F2	Performer Management: Import performers from Internet movie database	-
R1	Users should be able to add and remove a movie from the list	F1
R2	Users should be able to display and change the textual information about a selected movie	F1
R3	Users should be able to display a list of available movies and select one from the list	F1
R4	Users should be able to rate movies	F1
R5	Users should be able to import textual information about the performers of a movie from Internet movie database	F2
R6	Users should be able to display textual information about the performers of a movie	F2

F = Feature R = Functional Requirement

Table IV
DEVELOPERS AND WORK ITEMS

Artifact	Description	Assigned To
Amy	Database Expert	W1 W4 W7
Bill	UI Expert	W2 W3 W5
Carl	Bug Fixing	W6 W8

Artifact	Description	Realizing	Sprint
W1	Create Data Object for Movie	R1	S1
W2	UI for Movies	R2 R3	S1
W3	UI Control for Rating	R4	S1
W4	Create Data Object Performer	R6	S2
W5	UI for Performers	R6	S2
W6	Bugfix for Rating Control	R4	S2
W7	Performer Import	R5	S2
W8	Bugfix for Performer Import	R5	S2

W = Work Item R = Functional Requirement

A number of code files are developed to achieve Movie Manager: Movie.java (C1), MoviesUI.java (C2), RatingControl.java (C3), Performer.java (C4), PerformerImport.java (C5). Table II provides an overview about the ten created revisions, created (c) and modified (m) code files, used traceability links (LB) from the TIM and captured traceability links (CL) during the software development (in the small example, there are no code files that needed to be deleted). Furthermore, all three developers have entered commit messages for each revision that roughly describe how they have modified the source code.

The team used the presented process (see Figure 2) during development. In the following, the creation of revisions 1 and 2 is shortly explained. All other revisions were created in the same way. Work item W1 was assigned to Amy and she had to implement the data object for storing movies. First, she looked at the linked functional requirement R1 of her work item to get a better understanding of the attributes of the data object. She started implementing Movie.java (C1) and looked at F1 for the feature description. She finished implementation and validated and confirmed all captured links to F1 and R1. Next, she entered a commit message and the system created a new revision with the new code file C1.

Work item W2 was assigned to Bill and he was supposed to implement a user interface for listing the movies. Thus, he first looked at the linked functional requirements R2 and R3. Bill looked during implementation at feature F1 because it was already linked to R3. Furthermore, he looked at R1 because this requirement was also linked to F1. Bill used

Table V
INFORMATION NEEDS ON REQUIREMENTS DURING DEVELOPMENT ACTIVITIES WITH USED TRACEABILITY LINKS

Nr.	Information Need on Requirement	Development Activity	Used Traceability Links
1.	What is the program supposed to do?	Implementation, Program Comprehension	F-R, F-W, R-W, F-C*, R-C*
2.	Why was this code implemented this way?	Program Comprehension	C-Rev
3.	What have my co-workers been doing?	Change Awareness	F-W, R-W, W-D, W-S
4.	Which code is involved in the implementation of this feature?	Maintenance	F-C*, R-C*
5.	To move this feature into this code, what else needs to be moved?	Change Management	F-C*, R-C*
6.	What will be the impact of this change?	Change Management	F-R, F-W, R-W, F-C*, R-C*

F = Feature R = Functional Requirement W = Work Item S = Sprint D = Developer C = Code File Rev = Revision * = inferred

the inferred traceability link from F1 to Movie.java (C1) to change Movie.java because it missed an attribute that Amy forgot to implement, and created the code file MovieUI.java (C2). The inferred traceability link was created after Amy changed the status of her work item from *assigned* to *done*. He finished implementation and validated and confirmed all captured traceability links to R1, R2, R3 and F1. Finally, he entered a commit message and the system created a new revision with new code file C2 and modified code file C1.

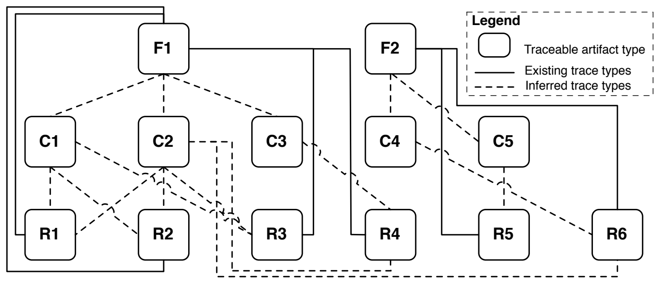


Figure 3. Existing and Inferred Traceability Links

After the completion of each work item, traceability links were inferred using the presented algorithm (see Algorithm IV.1). In revision 10, this resulted in the traceability links between features, functional requirements and code files shown in Figure 3. The straight lines show the traceability links that existed before. Furthermore, the inferred traceability links are shown as dashed lines.

VI. INFORMATION NEEDS ON REQUIREMENTS

Developers have various information needs during the software development process. Ko et al. [10] identified 21 and Sillito et al. [11] identified 44 information needs, respectively. From these 65 information needs represented as questions, we have identified those which are asked by developers during software development focusing on requirements, code that implements these requirements, and work done by co-workers related to these requirements. We looked through all information needs and used the following criteria for identification: a) mentioning terms that are related to requirements, e.g., *feature*, *concern*, *behavior* or expressions like *supposed to*, b) mentioning the term *impact* in conjunction with a changing requirement, and c) mentioning terms like *developer* or *co-worker*. We have identified six information needs (see Table V, Nr. 1-3 from Ko et al. and Nr. 4-6 from Sillito et al.) that met these criteria. All other information needs are rather specific for implementation and do not focus on requirements, e.g. reproducing a failure during bug fixing or understanding execution behaviour. We defined in Table V for each information need on requirements during what development activity the information need occurs and the used traceability links.

A. Identified Information Needs on Requirements

In the following, we explain how these information needs of developers can be satisfied by employing the TIM (see Figure 1), the captured traceability links from the process (see Figure 2) and the inferred traceability links (see Algorithm IV.1) for the example project mentioned in Section V.

1) What is the program supposed to do?: The features and functional requirements define what the program is supposed to do. As a work item needs to have a relation to functional requirements and can be related to features, an assigned developer can use the linked artifacts during implementation and program comprehension. For example, Amy knows during implementation what attributes the data object for movies requires since the functional requirement R1 is linked to her work item W1. However, she forgot to implement one attribute in revision 1; so, Bill had to change the data object again in revision 2. Furthermore, if a developer is interested in the purpose of a code file during program comprehension, s/he can use the inferred traceability links from the code file to the features and functional requirements. For example, if Carl is interested in the purpose of C3 (RatingControl.java), he can use the inferred traceability links to F1 and R4 that were created when Bill finished the work item W3 in revision 5.

2) Why was this code implemented this way?: Starting from the code files, a developer can look at the linked revisions. The commit messages may contain information concerning why the code was implemented this way. For example, Bill decided to implement the Rating Control with a 5 star rating and documented his decision in the commit message of revision 5. Documenting these decisions as artifacts of type rationale would be part of future work.

3) What have my co-workers been doing?: Since all work items are contained in a sprint and assigned to developers, a developer is able to see on what features or functional requirements his/her co-workers will be working on or have been working on in the past, which is supporting change awareness. Furthermore, a developer is able to see the co-workers that have previously worked on the same feature or functional requirement. Using this information, s/he can seek further knowledge from these co-workers. For example, Carl can see that Bill has worked previously on the Rating Control and he can ask him for advice during bug fixing.

4) Which code is involved in the implementation of this feature?: A feature is detailed in functional requirements. A developer can use the inferred traceability links from features and functional requirements to code files to quickly identify code that is involved in the implementation of a feature. For example, Carl can see that code files C4 (Performer.java) and C5 (PerformerImport.java) are involved in feature F2 (Performer Management) during bug fixing described in W8. This enables to identify not realized features and functional requirements as well as the progress of their implementation.

5) To move this feature into this code, what else needs to be moved?: 'Moving a feature' means that an entire feature with all its detailing functional requirements and realizing code can be moved from one development project to another project. As one feature is connected to detailing functional requirements, and these artifacts are connected by inferred traceability links to code files, related code files can be identified during change management. For example, the code files C1, C2 and C3 are related to feature F1 through their relations to the requirements R1, R2, R3 and R4 (see Figure 3). Therefore, if a feature needs to be moved, all its related functional requirements and the realizing code files can be easily identified. However, this may require additional code files to be moved that are required by the to-be-moved code files. Additional work on integrating the moved code files in the new environment may be necessary, as well.

6) What will be the impact of this change?: If a feature or a functional requirement need to be changed to reflect changed customer demands, all related artifacts maybe affected by this change can be identified easily during change management. For example, suppose R4 is changed to support a different rating, it can be identified that W3, W6 and C3 are maybe affected by this change. Affected work items can be identified, e.g. if a change in a feature or functional requirement is comprehensive, the planning of the realization in the work items needs to be adapted. An initial set of code files can be identified potentially affected by this change. The changes in the code files can result in additional changes in other code files. The initial set of code files can be a starting point for detailed change impact analysis.

B. Frequently Unsatisfied Information Needs

Many of the frequent information needs are problematic, because the searches for this information are often unsatisfied and have long search times. It is of particular interest that the most difficult information needs to satisfy are questions regarding requirements and co-workers working on these requirements [10]. Ko et al. have identified seven most frequently unsatisfied information needs, from which three are exactly the same information needs 1-3 from Table V that met our criteria. For example, searches for the information need *1. Why was the code implemented this way?* resulted in 44% of unsatisfied searches and a maximum of 21 minutes of observed search time.

One of the most frequently sought and acquired information by a developer includes what co-workers have been doing, which corresponds to the information need *3. What have my co-workers been doing?*. To determine who to ask, developers often identify co-workers by inspecting commit logs, but such information is not always accurate [10]. Our approach helps developers determining co-workers who have worked on the same requirements as themselves in the past to seek further information.

VII. Related Work

Approaches related to our work can be divided into two groups: approaches achieving traceability between requirements and source code after development and approaches capturing traceability links as we do during development.

A. Traceability between Requirements and Source Code

In [18], a general overview about requirements traceability is provided. As the manual creation of traceability links between requirements and source code is error-prone, time consuming and complex [2], research focuses on (semi-) automatic approaches. Existing approaches create traceability links between requirements and source code using various techniques, e.g., information retrieval [3], [4], [19], [20], [21], execution-trace analysis [5], [22], [23], static/dynamic analysis [6], subscription-based or rule-based link maintenance [7] or combinations of them [8], or only create links between work items and code [24]. However, no approach uses artifacts from project management to create traceability links between requirements and source code, as we do with our approach using work items.

B. Capturing Traceability Links

An approach similar to ours for the automatic capturing of links was presented by Omoronyia et al. [25]. They have achieved traceability between use cases and source code. In contrast, our approach supports features and functional requirements. Their approach is based on tracing the operations carried out by a developer called navigation trails. However, this approach requires an elaborate model with rankings of navigation trails to derive the most relevant links. Rankings of links are currently not supported by our approach. Thus, in future work we want to analyze whether the availability of work items can support this ranking.

Their approach is also able to identify which developer is involved in the realization of a specific use case, which is also supported by our approach. The contribution of Omoronyia et al. shows that tracking changes displays some advantages over the other approaches. For example, relating a developer to the source code and requirements is almost impossible with the other approaches, but very easy if changes/operations are tracked, like in our approach. Furthermore, their approach does not support work items from project management and revisions in a VCS. However, such artifacts are widely used in software development projects nowadays. Therefore, our approach is more easily applicable in practice compared to the approach by Omoronyia et al.

Omoronyia et al. [25] also claim to satisfy certain information needs. However, they did not use a structured method to identify these information needs like we did and only proposed those that were satisfied by their approach. Furthermore, their information needs are not based on project management and co-workers within the project. The benefit of our approach is that we can satisfy these information needs of developers during the development process.

VIII. Discussion

Egyed et al. [26] investigated the effort of recovering traceability links between requirements and code after development. In general, these traceability links were recovered by project members who were not directly involved in the realization of a particular requirement, but knew the code base. Our approach distributes the effort of creating traceability links over all developers actively participating in the project while they perform their implementation work. Using our approach, the developers are now involved in the traceability process, they can use their expertise and project knowledge to create reliable traceability links and these links also help them to satisfy their information needs during development. As a developer benefits not only from these traceability links himself/herself, but also his/her co-workers, we expect that they are better motivated to create and validate traceability links during software development.

Additionally, one might ask: ”Why is (manually) creating links between requirements and work items, and between work items and code files less complex compared to existing work on linking requirements to source code directly?”. We argue that our approach is less cumbersome and error-prone than manually creating direct links between requirements and code, because the only manual work is to establish initial links between work items and requirements (which is typical for issue management) and to validate the automatically captured links (which should be easy as the links refer to the work just finished). Creating direct links manually requires the developer to keep every relationship in mind.

In the current approach, developers might make mistakes when adding non-related features or functional requirements to a work item. However, this risk is reduced since we let the developer validate all traceability links before they are created. It has been shown that humans were better at validating links as opposed to searching for missing links [27]. This strengthens our approach of letting the developers validate the links going to be created instead of recovering links or searching for missing links. The additional work of the developers introduced by validating traceability links and manually adding additional ones is considered as small, compared to the effort to establish traceability links after development using various approaches mentioned before.

Currently we are developing tool support based on UNICASE, which is a plugin for the Eclipse IDE. The Eclipse IDE supports various programming languages through additional plugins, e.g. Java, C++, Python etc. By integrating UNICASE and Eclipse with plugins for VCSs like Subversion or Git, a comprehensive tool environment can be provided supporting developers while they perform various development activities. By using these plugins, file-based as well as change-based representations of source code can be accessed. We looked at various research tools, e.g., TagSEA [28], and commercial tools, e.g., IBM Rational Team Concert [29]. Some of these tools do support all the elements that we have (requirements, work items, code). However, our tool would provide, unlike all other tools, complete traceability between all these elements as well as (semi-) automatic linkage of requirements and code.

IX. Conclusion and Future Work

In this paper, we presented an approach for tracing requirements and source code during software development to satisfy information needs of developers regarding requirements during development. We defined a TIM that integrates requirements, source code and artifacts from project management. We also presented an approach for the (semi-) automatic creation of links between artifacts from the TIM.

In this work, we only focused on information needs of developers. However, we are aware that also information needs of other project participants can be satisfied with the created links, e.g., of project managers or requirements engineers, which is subject to future work. Furthermore, we are aware that the algorithm for inferring links is very basic and might create a lot of links. Therefore, we will investigate possibilities for more advanced inference algorithms, e.g. an algorithm providing a relevance ranking for each link based on the change history of the artifacts connected by the link, to identify relevant links from the large set of inferred links. Currently we develop tool support based on our approach. Once the tool is finished, we will empirically evaluate the approach in the UNICASE project itself and apply it in various case studies. We will compare our approach to existing baseline approaches w.r.t. precision and recall.

REFERENCES

[1] Egyed, A. and Grünbacher, P. Supporting software understanding with automated requirements traceability. International Journal of Software Engineering and Knowledge Engineering, vol. 15, no. 5, pp. 783-810 (2005)

[2] Spanoudakis, G. and Zisman, A. Software traceability: A roadmap. Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing, pp. 395-428 (2004)

[3] Hayes, J.H., Dekhtyar, A., and Osborne, J. Improving requirements tracing via information retrieval. International Conference on Requirements Engineering, pp. 138-147 (2003)

[4] De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G. Recovering traceability links in software artifact management systems using information retrieval methods. Transactions on Software Engineering Methodology, vol. 16, no. 4, art. 13, ACM (2007)

[5] Eisenberg, A.D. and De Volder, K. Dynamic feature traces: Finding features in unfamiliar code. In ICSM 05: Proceedings of the 21st IEEE International Conference on Software Maintenance, pp. 337-346 (2005)

[6] Antoniol, G. and Gueheneuc, Y.G. Feature identification: A novel approach and a case study. In ICSM 05: Proceedings of the 21st IEEE International Conference on Software Maintenance, pp. 357-366 (2005)

[7] Maeder, P. and Gotel, O. Towards Automated Traceability Maintenance. Journal of Systems and Software, vol. 85, no. 10, pp. 2205-2227 (2011)

[8] Eaddy, M., Aho, A.V., Antoniol G., et al. CERBERUS: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In ICPC 08: Proceedings of the 16th IEEE International Conference on Program Comprehension, pp. 53-62 (2008)

[9] Cleland-Huang, J., Heimdahl, M., Huffman Hayes, J., Lutz, R., and Maeder, P. Trace queries for safety requirements in high assurance systems. In REFSQ 12: Proceedings of the 18th International Conference on Requirements Engineering: Foundation for Software Quality, pp. 179-193 (2012)

[10] Ko, A.J., DeLine, R., and Venolia, G. Information needs in collocated software development teams. In ICSE 07: Proceedings of the 29th International Conference on Software Engineering, pp. 344-353 (2007)

[11] Sillito, J., Murphy, G.C., and Volder, K.D. Asking and answering questions during a programming change task. IEEE Trans. Softw. Eng., vol. 34, no. 4, pp. 434-451 (2008)

[12] Helming, J., Koegel, M., and Naughton, H. Towards traceability from project management to system models. In TEFSE 09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering, pp. 11-15. IEEE Computer Society (2009)

[13] Bruegge, B., Creighton, O., Helming, J., and Koegel, M. Unicase - an Ecosystem for Unified Software, In ICGSE 08: Distributed software development: methods and tools for risk management, pp. 12-17 (2008)

[14] UNICASE Open Source Project. http://www.unicase.org/ [retrieved: September, 2012]

[15] Helming, J., Arndt, H., Hodaie, Z., Koegel, M., and Narayan, N. Automatic Assignment of Work Items. In ENASE 10: Evaluation of Novel Approaches to Software Engineering, Communications in Computer and Information Science, vol. 230, pp. 236-250 (2011)

[16] Kagdi, H., Collard, M.L., and Maletic, J.I. A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution, Journal of Software Maintenance and Evolution, vol. 19, pp. 77-131 (2007)

[17] Maeder, P., Gotel, O., and Philippow, I. Getting Back to Basics: Promoting the Use of a Traceability Information Model in Practice. In TEFSE 09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering, pp. 21-25. IEEE Computer Society (2009)

[18] Dahlstedt, A. and Persson, A. Requirements interdependencies: State of the art and future challenges. In Engineering and Managing Software Requirements, Aurum and Wohlin (eds.) Springer, pp. 95-116 (2005)

[19] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E. Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering, pp. 970-983 (2002)

[20] Marcus, A. and Maletic, J.I. Recovering documentation-to-source-code traceability links using latent semantic indexing. In ICSE 03: Proceedings of the 25th International Conference on Software Engineering, pp. 125-135. IEEE Computer Society (2003)

[21] Marcus, A., Maletic, J.I., and Sergeyev, A. Recovery of traceability links between software documentation and source code. International Journal of Software Engineering and Knowledge Engineering, vol. 15, no. 5, pp. 811-836 (2005)

[22] Egyed, A. A Scenario-Driven Approach to Trace Dependency Analysis. Transactions on Software Engineering, vol. 29, no. 2, pp. 116-132, IEEE (2003)

[23] Burgstaller, B. and Egyed, A. Understanding where requirements are implemented. In ICSM 10: Proceedings of the 26th IEEE International Conference on Software Maintenance, pp. 1-5 (2010)

[24] Anvik, J. and Storey, M.A. Task articulation in software maintenance: Integrating source code annotations with an issue tracking system. In ICSM 08: Proceedings of the 24th IEEE International Conference on Software Maintenance, pp. 460-461 (2008)

[25] Omoronyia, I., Sindre, G., Roper M., Ferguson J., and Wood, M. Use case to source code traceability: The developer navigation viewpoint. In RE 09: Proceedings of the 17th IEEE International Requirements Engineering Conference, pp. 237-242 (2009)

[26] Egyed, A., Graf, F., and Grünbacher, P. Effort and quality of recovering requirements-to-code traces: Two exploratory experiments. In RE 10: Proceedings of the 18th International IEEE Requirements Engineering Conference, pp. 221-230 (2010)

[27] Kong, W.-K., Huffman Hayes, J., Dekhtyar, A., and Holden, J. How do we trace requirements: an initial study of analyst behavior in trace validation tasks. In Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, In conjunction with CHASE 11, pp. 32-39 (2011)

[28] TagSEA. http://tagsea.sourceforge.net/ [retrieved: September, 2012]

[29] IBM Rational Team Concert. http://www.ibm.com/software/rational/products/rtc/ [retrieved: September, 2012]

An Empirical Study Identifying High Perceived Value Requirements Engineering Practices in Global Software Development Projects

Mahmood Niazi [a, b, c], Mohamed El-Attar [a],

[a] Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Saudi Arabia
[b] Faculty of Computing, Riphah International University Islamabad, Pakistan
[c] Keele University, Keele, ST5 5BG, UK
{mkniazi, mettar}}@kfupm.edu.sa

Muhammad Usman [d] and Naveed Ikram [c]

[c] Faculty of Computing, Riphah International University Islamabad, Pakistan
[d] International Islamic University Islamabad, Pakistan
naveed.ikram@riu.edu.pk, m.usman@iiu.edu.pk

Abstract—Requirements-related problems are reported to be the main reason in failures of global software development (GSD) projects. There is not much work done to improve requirements practices for GSD projects. In this paper, we report results of a study conducted in an ongoing project whose aim is to develop a framework for the requirements engineering process of global software development projects. The objective of this paper is to report a recent empirical study which was aimed in identifying high perceived value RE practices in the GSD projects. We used an online survey questionnaire to collect data from 39 RE practitioners of GSD organizations. We have identified 11 frequently cited high value RE practices that should be planned and implemented in GSD projects to avoid frequently occurring requirements related problems.

Keywords - ***Global Software Engineering; Requirements Engineering.***

I. INTRODUCTION

In global software development (GSD), software or part of software is developed by geographically dispersed teams and companies whereby the vendor company in one country provides its services at low costs to the client company in another country. GSD has been growing steadily and an 18-fold increase in the outsourcing of IT-enabled business processes is projected [21]. Over the last decade, many firms in Europe have outsourced software development projects to other countries such as India, China and Russia. Low cost, time saving, access to global IT talent are main reasons for software development outsourcing for the client companies [2]. Moreover, offshore vendors improve their skills and service quality with the experience of offshore outsourcing projects, learning new ways to satisfy the clients' needs.

GSD, however, has many challenges and risks [9]. Significant failure rates have also been reported in GSD projects [4]. Nam et al. [12] investigated 93 client companies and found that 36 did not intend to continue their relationships with vendors. King [10] reports that JP Morgan decided to perform many software activities that it previously outsourced, and did not renew its $5 billion contract with IBM. At the root of many failures is the increased complexity that outsourcing brings to development projects. This complexity results in: high coordination costs, information security problems, lack of direct communication [19], perceived loss of expertise in the outsourced activity [5], cultural misunderstandings [11] and infrastructure problems [1].

Although a variety of software development tasks are outsourced, previous work suggests that most of the factors contributing to the failure of outsourcing are related to requirements [17]. This is not surprising given that the requirements engineering (RE) process has a huge impact on the effectiveness of all software development processes [20]. A previous UK study of non-outsourced projects found that out of 268 documented development problems, requirements problems accounted for 48% of all software problems [6]. In another study of GSD projects, RE problems in multi-site software development organisations were identified [3]. The evidence is clear: problems in the requirements phase have a wide impact on the success of software development projects [6, 20] and an even greater impact on the success of GSD projects [3].

In order to improve the RE process Sommerville and Ransom [20] have suggested 66 RE practices. All of these RE practices were designed for non-GSD projects and it is hard to know if these practices can also be used in the GSD projects. Despite the importance of RE in the GSD projects, no empirical study has been conducted to observe the perceived benefit RE practices in the GSD projects.

We propose to adapt and customize these 66 RE practices specifically for GSD projects. The objective of this paper is to report a recent empirical study which was aimed in identifying high perceived value RE practices in the GSD projects. To achieve this we address the following Research Question (RQ):

RQ: Which RE practices can be effectively used in the GSD projects?

In order to address this RQ we will:

- Determine what the most important of all the RE practices advocated by Sommerville and Ransom are for GSD projects.
- Identify any additional RE practices important for GSD projects that may lack from the list of Sommerville and Ransom [20] practices.

In earlier research works [15], we discussed preliminary results based on a pilot study with five GSD organisations. In this paper, we discuss the findings of a broader survey of 39 RE experts of GSD projects.

This paper is organised as follows. In Section II background is provided. Section III describes the research methodology. In Section IV the findings of this work are presented and analysed with some discussion. Section V describes the summary of results. Section VI provides the conclusion and future work.

II. BACKGROUND

A. RE practices designed by Sommerville and Ransom

Sommerville et al. [20] suggested a requirements framework that includes 66 requirements practices that can lead to RE process improvement and ultimately business benefits [20]. The 66 requirements practices are classified as basic, intermediate and advanced. There are 36 *basic* practices concerned with the fundamental activities required to gain control of the RE process; 21 *intermediate* practices mostly concerned with the use of methodical approaches and tools; and 9 *advanced* practices concerned with methods such as formal specification used typically for critical systems development. The 66 RE practices are grouped into 8 major categories [20].

Thus far, no research has considered using these 66 practices in global software development projects. Few studies have used these practices for non-outsourced projects such as [14] and [20]. It is important to identify best RE practices for GSD projects as previous work suggests that half of the companies that have tried GSD have failed to realize the anticipated outcomes, the root cause of which is often related to RE problems [17, 18].

B. Perceived Benefits.

We assert that “perceived benefits” of a particular RE practice can be used as a judgement criterion for determining the degree of importance (value) placed on a RE practice by requirements stakeholders. This is because where requirements stakeholders from different organisations perceive a RE practice as having a high-perceived or medium-perceived benefits then that practice should be considered for its importance in a RE process of GSD projects. The information about relative “perceived benefits” can help researchers and requirements stakeholders to better understand various RE practices within the GSD initiatives.

III. RESEARCH METHODOLOGY

A. Data Collection

Based on the available resources and nature of this research, we set up an online survey questionnaire to collect data from RE practitioners of GSD organizations. A pilot study was conducted with five RE experts to validate the questionnaire. We invited 150 RE experts from different software companies involved in GSD projects via emails. Basic information about the software companies was available on their respective websites. Besides request for participation, emails contained the link of the survey site. Out of 150 invitations, only 39 RE experts agreed to participate so the response rate is 26%.

Each requirements stakeholders was asked to choose and rank 66 RE practices against four types of assessments that have been developed previously [13, 20]. These assessments were:

- *High Perceived Benefits (H)*: A practice has a documented standard and is always followed as part of the organisation’s GSD process i.e., it is mandatory.
- *Medium Perceived Benefits (M):* This means that the practice is widely followed in the organisation’s GSD process but is not mandatory.
- *Low Perceived Benefits (L):* Some GSD projects may have introduced the practice only for that project. This practice is described as ‘low’ perceived benefit.
- *Zero Perceived Benefits (Z):* The practice is never or rarely applied to any GSD projects.

From this list, we have the ‘perceived benefit’ associated with each RE practice, i.e. the degree of importance placed on a RE practice by requirements stakeholders based on their experience from previous GSD projects.

We received responses from 39 RE practitioners working in software companies involved in GSD. Participants were RE practitioners with experience ranging from 1 year (minimum) to 13 years (maximum) with average experience equal to 6 years. 70% of participants were from multinational companies. Most of the participants’ companies develop business applications and data processing applications. Few participants also work in the domain of real time, safety critical and embedded systems. The majority of the participants work in large sized companies having staff sizes greater than 200.

B. Data Analysis

In order to analyse the perceived benefit of each identified RE practice, the occurrence of a perceived benefit (high, medium, low, zero) in each response was counted. By comparing the occurrences of one RE practice’s perceived benefits obtained against the occurrences of other RE practices’ perceived benefits, the relative benefit of each RE practice was identified. We have also used this approach to identify high and low valued RE practices and software process improvement de-motivators in our previous research [13]. For most of the data analysis, we have used statistical analysis. We believe that the presentation of data using statistical analysis is an effective mechanism for comparing and contrasting within, or across groups of variables.

IV. RESULTS AND ANALYSIS

This section is divided into 8 sub-sections, each of which corresponds to one of the eight Sommerville and Ransom categories of RE practices.

A. Requirements Document Practice

This category has eight practices which are labelled RD1 to RD8. Table 1 shows the participant responses for this category. RD1 'define a standard document structure' is frequently cited high value requirements document practice. Because of temporal, geographical and cultural barriers in GSD, communication and coordination becomes challenging [8]. This leads to problems caused by a lack of shared understanding and other knowledge management issues. Use of predefined document structure improves shared understanding and helps the readers in reading, following and updating the document. Our results indicate that using agreed upon standard documents can assist in reducing many issues in GSD projects. RD4 'include a summary of the requirements' is also one of the frequently cited high value practices in this category. This practice can help readers of the requirements documents to get the overview of the whole requirements document.

TABLE I. REQUIREMENTS DOCUMENT PRACTICES

ID	**Requirements Documents Practice**	**Type of Assessment (n=39)**			
		H	M	L	Z
RD1	Define a standard document structure	20	13	5	1
RD2	Explain how to use the document	15	11	11	2
RD3	Include summary of the requirements	19	13	5	2
RD4	Make a business case for the system	15	18	4	2
RD5	Define specialized terms	14	15	9	1
RD6	Make document layout readable	17	16	5	1
RD7	Help readers find information	11	20	5	3
RD8	Make the document easy to change	11	21	4	3

B. Requirements Elicitation Practices

There are 13 practices in this category which are labelled RE1 to RE13 as shown in Table 2. The results show that the most common 'high' value requirements elicitation practice is RE3 'Identify and Consult System Stakeholders'. It is an interesting result as GSD system stakeholders are usually not directly available to the vendor team. They are available on the client side. To overcome this problem different tools are used ranging from asynchronous communication media (email, blogs etc.) to synchronous media (instant messaging, video conferencing etc.). System stakeholders are the domain experts and if they are not available for consultation because of different GSD barriers then the requirements related issues will not be resolved. Other most common (19 out of 39) high value practice is RE5 'define the proposed system's operating environment'. Many GSD organisations define the proposed system's operating environment during or before requirements elicitation. It is critical to determine the scope of the environment where the system will be deployed [7].

It is interesting to see in Table 2 that 12 out of 39 participants consider RE10 'prototype poorly understood requirements' as a low and zero value practice. This is not in line with the general view in RE literature about prototyping misunderstood requirements. One probable reason can be the cost and effort associated with prototyping of misunderstood requirements.

TABLE II. REQUIREMENTS ELICITATION PRACTICES

ID	**Requirements Elicitation Practices**	**Type of Assessment (n=39)**			
		H	M	L	Z
RE1	Assess System Feasibility	17	17	4	1
RE2	Be sensitive to organizational /political consideration	16	13	8	2
RE3	Identify, consult system stakeholders	27	11	0	1
RE4	Record requirements sources	16	17	5	1
RE5	Define system's operating environment	19	15	3	2
RE6	Use business concerns to drive elicitation	14	18	5	2
RE7	Look for domain constraints	18	14	5	2
RE8	Record requirements rationale	11	13	3	2
RE9	Collect reqs. from multiple viewpoints	14	19	4	2
RE10	Prototype poorly understood requirements	8	19	9	3
RE11	Use scenarios to elicit requirements	12	17	8	2
RE12	Define operational processes	15	15	6	2
RE13	Reuse requirements	9	16	11	3

C. Requirements Analysis and Negotiation

This category contains 8 practices; labelled as RA1 to RA8. Table 3 presents the participants' responses for the practices in this category. RA1 'Define system boundaries' is rated as most common high value practice. Clear definition of the system boundary early in the project helps clarify the system scope and identifying the interfaces and dependencies with other systems. Early definition of the system boundary also helps in effort and size estimation. RA3 'Use software to support negotiation' and RA7 'Use interaction matrices' are frequently cited low value practices. It seems as if they are not commonly understood and used practices in GSD projects.

TABLE III. REQUIREMENTS ANALYSIS AND NEGOTIATION PRACTICES

ID	**Requirements Analysis and Negotiation Practices**	**Type of Assessment (n=39)**			
		H	M	L	Z
RA1	Define system boundaries	20	14	3	2
RA2	Use checklists for analysis	13	12	12	2
RA3	Use software to support negotiation	6	11	17	5
RA4	Plan for conflict resolution	14	12	9	4
RA5	Prioritise requirements	19	17	2	1
RA6	Classify requirements using a multi-dimensional approach	11	12	13	3
RA7	Use interaction matrices to find conflicts and overlaps	5	10	17	7
RA8	Assess requirements risks	15	12	10	2

D. Describing Requirements Practices

This category has five practices which are named as DR1 to DR5 as shown in Table 4. The most common high value practices are DR1 'Define standard templates for describing requirements' and DR3 'Use diagrams appropriately'. The high value of DR1 corresponds well with the high value of RD1 'Define a standard document structure' described in section IV (A). Use of standard templates for describing requirements in GSD facilitates shared understanding and handles differences in the organizational culture. Templates help reduce the ambiguities of natural language as it poses a control and structure on the form the requirements are expressed. RE practitioners also see high value in using diagrams for describing requirements. Diagrams and models help visualize and understand requirements quickly. The high value for DR3 is indicative of the desire of RE practitioners to have a clear, common and unambiguous view of system requirements before the design and development of system commences. DR5 'Specify requirements quantitatively' and DR4 'Supplement natural language with other description of requirement' are the most common medium value practices in this category.

TABLE IV. DESCRIBING REQUIREMENTS PRACTICES

ID	**Describing Requirements Practices**	**Type of Assessment (n=39)**			
		H	M	L	Z
DR1	Define standard templates for describing requirements	19	12	7	1
DR2	Use languages simply and concisely	17	15	5	2
DR3	Use diagrams appropriately	19	10	4	5
DR4	Supplement natural language with other description of requirement	13	19	5	2
DR5	Specify requirements quantitatively	9	19	10	1

E. System Modelling Practices

This category has six practices labelled SM1 to SM6. The results show that the most common high value requirements modelling practices are SM5 'Use a data dictionary' and SM3 'Model the system architecture'. Appropriate use of data dictionary helps in removing the ambiguities and inconsistencies in the requirements document. It is also an important practice in building the same understanding of different terms, concepts and definitions. Modelling the system architecture (SM3) early is important for the realization of non-functional requirements of a system. Non-functional requirements are vital for the success of a system. We see SM3 as an important practice as it prompts the consideration of non-functional requirements on board as early as the requirements phase.

TABLE V. SYSTEM MODELLING PRACTICES

ID	**System Modelling Practices**	**Type of Assessment (n=39)**			
		H	M	L	Z
SM1	Develop complementary system models	9	15	11	4
SM2	Model the system's environment	9	15	13	2
SM3	Model the system architecture	14	17	6	2
SM4	Use structured methods for system modelling	13	14	9	3
SM5	Use a data dictionary	15	12	10	2
SM6	Document links between stakeholder requirements and system models	11	13	12	3

F. Requirements Validation Practices

There are eight practices in this category (Table 6). RV1 'Check that requirements document meets your standards' is the most frequently cited high value practice. This practice corresponds well with other similar practices in our study, i.e. RD1 and DR1. GSD practitioners believe that the compliance with standard document structure or template is important and should be ensured. RV7 'Propose requirements test cases' is another common high value practice. One of the criteria of writing better requirements is that they should be testable and verifiable. If test cases are written for requirements then this will improve the quality of the requirements. Prototyping is generally considered to be an important and useful validation technique. However, prototyping was a low value practice in the requirements elicitation section. The cost and effort associated with prototyping seem be the inhibitors of using it as an elicitation or validation technique.

TABLE VI. REQUIREMENTS VALIDATION PRACTICES

ID	**Requirements Validation Practices**	**Type of Assessment (n=39)**			
		H	M	L	Z
RV1	Check that requirements document meets your standards	21	10	5	3
RV2	Organise formal reqs. inspections	13	11	12	3
RV3	Use multi-disciplinary teams to review requirements	16	11	8	4
RV4	Define validation checklists	14	14	6	5
RV5	Prototyping to animate requirements	7	15	13	4
RV6	Write a draft user manual	13	13	10	3
RV7	Propose requirements test cases	17	12	7	3
RV8	Paraphrase system models	5	14	12	5

G. Requirements Management Practices

There are nine practices in this category as shown in Table 7. RM1 'Uniquely identify each requirement' and RM7 'Identify global system requirements' are most frequently cited high value practices. Unique identification of each requirement helps in traceability and management of all requirements. Identification of global system requirements establishes the required system level properties. They are concerned with the system architecture and should be appropriately handled in early stages of the requirements phase.

TABLE VII. REQUIREMENTS MANAGEMENT PRACTICES

ID	**Requirements Management Practices**	**Type of Assessment (n=39)**			
		H	M	L	Z
RM1	Uniquely identify each requirement	20	12	5	2
RM2	Define policies for reqs. management	16	11	8	4
RM3	Define traceability policies	16	12	5	6
RM4	Maintain a traceability manual	13	10	8	8
RM5	Use database to manage requirements	14	9	7	9
RM6	Define change management policies	16	12	7	4
RM7	Identify global system requirements	17	10	8	4
RM8	Identify volatile requirements	9	14	10	6
RM9	Record rejected requirements	7	8	16	8

H. RE for Critical Systems Practices

There are nine practices in this category that are labelled CS1 to CS9. Most of the practices in this category have similar frequency. This category of practices seems less relevant to GSD as critical systems are usually not developed in GSD or outsource setting.

TABLE VIII. RE FOR CRITICAL SYSTEMS PRACTICES

ID	**RE for Critical Systems Practices**	**Type of Assessment (n=39)**			
		H	M	L	Z
CS1	Create safety requirement checklists	14	10	8	7
CS2	Involve external reviewers in the validation process	9	14	9	7
CS3	Identify and analyse hazards	14	10	9	6
CS4	Derive safety requirements from hazard analysis	14	12	7	6
CS5	Cross-check operational and functional reqs. against safety reqs.	16	7	11	5
CS6	Specify systems using a formal specification	11	14	10	4
CS7	Collect incident experience	12	13	9	5
CS8	Learn from incident experience	16	11	8	4
CS9	Establish an organizational safety culture.	11	15	7	6

V. SUMMARY OF RESULTS

In this section, we summarise our results and also give recommendations to RE practitioners about high perceived value RE practices which can be used in GSD projects. These high valued practices should be planned and implemented in GSD projects to avoid frequently occurring requirements related problems. Table 9 lists the most commonly cited high value practices in each category. Implementation of RD1, DR1 and RV1 will help reduce inconsistencies and ambiguities in the requirements document. It will also help to build a shared understanding of system requirements. Client and vendor organisations, being in different countries, usually have different nomenclatures. Promoting the use of standard templates will help dealing with the issues related to cultural differences of client and vendor sides. RE3 has the highest frequency count in elicitation. In GSD, vendor teams usually do not have direct access to the system stakeholders. We recommend that managers should plan to compensate for this limitation in GSD projects. GSD managers can use technology (e.g. video conferencing) or occasional visits to allow development team representative(s) to have direct communication with system stakeholders. The system boundaries (RA1) should be defined as it will help in clear definition of scope, identification of system interfaces with other systems and in project estimation and scheduling. When it comes to the requirements descriptions, we recommend that the diagrams should be used appropriately (DR3) as many concepts or processes are better explained with diagrams. This also increases the understanding of a problem domain and facilitates knowledge transfer. Use of data dictionary (SM5) is also a recommended practice for system modelling as appropriate use of data dictionary helps the readers in better understand different documents. It also avoids the inconsistent use of terms and concepts.

TABLE IX. MOST COMMONLY CITED HIGH VALUE RE PRACTICES

ID	Practice	Category
RD1 RD3	Define a standard document structure Include summary of the requirements	Documentation
RE3 RE5	Identify and consult system stakeholders Define system's operating environment	Elicitation
RA1 RA5	Define system boundaries Prioritise requirements	Analysis & Negotiation
DR1	Define standard templates for describing requirements	Description
DR3	Use diagrams appropriately	Description
SM5	Use a data dictionary	Modelling
RV1	Check that the requirements document meets your standards	Validation
RM1	Uniquely identify each requirement	Management

VI. CONCLUSION AND FUTURE WORK

To the best of our knowledge, this is the first attempt to investigate the most relevant RE practices for GSD organizations. We have identified frequently cited high value RE practices which should be used in GSD projects to avoid frequently occurring requirements related problems. We have observed that not all 66 RE best practices are perceived as high value practices for GSD projects

Our ultimate goal is to develop a framework for improving RE in GSD projects (GlobReq). The proposed GlobReq will be an easy to use framework which will be accompanied by a website and tool support to facilitate its use in industry. The aim is to help companies avoid randomly implementing promising new models and frameworks just to see them discarded.

The following two research questions are our future work in this project:

RQ1. How can we develop GlobReq?

The basis of the GlobReq framework will be Sommerville and Ransom's framework of requirements practice, empirical study with GSD organisations and our questionnaire based survey. We will collect detailed empirical data from GSD organisations and practitioners to construct and validate the GlobReq frameworks. The following initial criteria will be used for the development of the GlobReq framework. We have used this approach successfully in previous empirical research with software development organisations [13; 16].

- User satisfaction: stakeholders need to be satisfied with the results of the GlobReq framework. Stakeholders (e.g. requirements engineers, systems analysts, outsourcing project staff) should be able to use the GlobReq to achieve specified goals according to their needs and expectations without confusion or ambiguity.
- Ease of use: complex models and standards are rarely adopted by organisations as they require resources, training and effort. The structure and contents of GlobReq should be simple, flexible and easy to follow.

- Better requirements: GlobReq should aid the development of high quality requirements (e.g. less ambiguous, more comprehensive, consistent and feasible).

Based on the empirical data, Sommerville's Requirements Framework will be rationalised to GSD environments. GlobReq framework will be developed from the rationalised Sommerville Framework together with additional empirical data collected from GSD collaborators. The frequently cited RE practices with 'high' and 'medium' perceived value' will be the basis of the GlobReq framework.

RQ2. How can we evaluate the effectiveness of GlobReq?

The evaluation of the end product is important in order to show up areas where the end product has deficits. The evaluation assists in future planning and decision making. In the evaluation process the lessons learned and results are used to enlighten future projects. We will use an expert panel review to seek the opinions of software outsourcing experts about the GlobReq framework. The criteria described in "GlobReq framework development" will be used, i.e. ease of use, user satisfaction and better requirements as the basis of this evaluation.

We will identify GSD experts for the evaluation of the GlobReq framework. These experts will be selected on the basis of their practical and/or academic experience of GSD projects. These experts will be from other organisations (i.e. not from organisations who participated in the data collection process). We have made preliminary discussions with these experts and we are in the process of explaining their role in this project.

ACKNOWLEDGEMENT

We are thankful to King Fahd University of Petroleum and Minerals, Saudi Arabia for supporting this research project via a project number IN111030.

REFERENCES

[1] Barthelemy, J. The hidden cost of IT outsourcing, *Sloan Management Review* 42 (3). 60-69. 2001

[2] Bush, Ashley A. , Tiwana, Amrit and Tsuji, Hiroshi. An Empirical Investigation of the Drivers of Software Outsourcing Decisions in Japanese Organizations, *Information and Software Technology Journal, 50(6), 499-510, 2008*

[3] Damian, D., E. and Zowghi, D. Requirements Engineering challenges in multi-site software development organizations, *Requirements Engineering Journal, published by Springer Verlag.* 8 (3). 149-160. 2003

[4] Foote, D. Recipe for offshore outsourcing failure: Ignore organization, people issues, *ABA Banking Journal* 96 (9). 56-59. 2004

[5] Gonzalez, R., Gasco, J. and Llopis, J. Information systems outsourcing risks: a study of large firms, *Industrial management and data systems* 105 (1). 45–62. 2005

[6] Hall, Tracy, Beecham, Sarah and Rainer, Austen. Requirements Problems in Twelve Software Companies: An Empirical Analysis, *IEE Proceedings - Software* (August). 153-160. 2002

[7] Jackson, M. *Software Requirements and Specifications.* Addison-Wesley / ACM Press. 1995

[8] Khan, Siffat Ullah, Niazi, M. and Rashid, Ahmad. Barriers in the selection of offshore software development outsourcing vendors: an exploratory study using a systematic literature review, *Journal of Information and Software Technology* 53 (7). 693-706. 2011

[9] Khan, Siffatullah, Niazi, Mahmood and Rashid, Ahmad. Critical Barriers for Offshore Software Development Outsourcing Vendors: A Systematic Literature Review. *16th IEEE Asia-Pacific Software Engineering Conference, APSEC09. Penang, Malaysia.* 2009

[10] King, W. Outsourcing becomes more complex, *IT Strategy and Innovation - ISM Journal* 22 (2). 89 - 90. 2005

[11] Kobitzsch, Werner, Rombach, Dieter and Feldmann, Raimund, L. Outsourcing in India, *IEEE Software* March/ April 2001 78-86. 2001

[12] Nam, K., Chaudhury, A., Rao, Raghav and H., Rajagopalan, S. A Two-Level Investigation of Information Systems Outsourcing, *Communications of ACM* 39 (7). 36-44. 1996

[13] Niazi, M, Cox, K. and Verner, J. An empirical study identifying high perceived value requirements engineering practices. *Fourteenth International Conference on Information Systems Development (ISD´2005) Karlstad University, Sweden August 15-17.* 2005

[14] Niazi, M and Shastry, Sudha. Role of Requirements Engineering in Software development Process: An empirical study. *IEEE International Multi-Topic Conference (INMIC03).* 402-407. 2003

[15] Niazi, M., El-Attar, Mohamed, Usman, Muhammad and Ikram, Naveed. GlobReq: A Framework for Improving Requirements Engineering in Global Software Development Projects: Preliminary Results. *International Conference on Evaluation & Assessment in Software Engineering (EASE 2012) Spain.* 2012

[16] Niazi, Mahmood, Cox, Karl and Verner, June. A Measurement Framework for Assessing the Maturity of Requirements Engineering Process, *Software Quality Journal: in press for publication* 16 (2). 157-298. 2008

[17] Oza, Nilay V. and Hall, Tracy. Difficulties in managing offshore outsourcing relationships: An empirical analysis of 18 high maturity Indian software companies, *Journal of Information Technology Case and Application Research* 7 (3). 25-41. 2005

[18] Oza, Nilay V. and Hall, Tracy. Difficulties in managing offshore outsourcing relationships: An empirical analysis of 18 high maturity Indian software companies. *4th International Outsourcing Conference, Washington DC.* 2005

[19] Pyysiainen, J. Building trust in global inter-organizational software development projects: problems and practices. *International Conference on Software Engineering: Global Software Development Workshop.* 2001

[20] Sommerville, Ian and Ransom, Jane. An empirical study of industrial requirements engineering process assessment and improvement, *ACM Transactions on Software Engineering and Methodology* 14 (1). 85-117. 2005

[21] United-Nations. *World Investment Report. The shift towards services, New York and Geneva*. 2004

Towards Automated Process Assessment in Software Engineering

Gregor Grambow and Roy Oberhauser
Computer Science Dept.
Aalen University
Aalen, Germany
{gregor.grambow, roy.oberhauser}@htw-aalen.de

Manfred Reichert
Institute for Databases and Information Systems
Ulm University
Ulm, Germany
manfred.reichert@uni-ulm.de

Abstract—**When assessing software engineering processes, current reference models approaches typically rely on manual techniques for acquiring evidence of practices, which is then correlated with expected model attributes to assess compliance. This is costly, error-prone, and assessment feedback is infrequent and detached from the original context. Automated data acquisition could improve this situation, but must overcome various challenges. This paper presents an automated approach for process assessment that relies on semantic extensions to a process-aware information system to provide an in-the-loop automated process assessment capability. This can reduce the effort required to determine process compliance, maturity, or improvement, and can provide more timely and precise feedback compared to current manual process assessment methods and tools. The evaluation showed the approach's technical feasibility, model diversifiability across various process assessment models (CMMI, ISO/IEC 15504, ISO 9001), and suitable performance and scalability. All in all, this paper contributes a practical, variable approach for automating parts of the assessment of executed processes.**

Keywords-software engineering process assessment; semantic technology; Capability Maturity Model Integration; ISO/IEC 15504; ISO 9000

I. Introduction

Processes - be they technical, managerial, or quality processes, are an inherent part of software engineering (SE), and subsequently so is process assessment and process improvement [1]. Software process improvement typically involves some assessment, and common reference model assessment standards utilize external audits (CMMI [2], ISO 15504 [3], and ISO 9001 [4]) that are performed manually to gather compliance evidence. Often the maturity of software organizations is assessed based primarily on their process-orientation and correlation of processes to a reference model.

If SE processes were supported or enacted by process-aware information systems (PAIS), then the efficiency of data acquisition and analysis for process assessment could also be improved. One necessary prerequisite - the adoption and use of automated process enactment support is relatively rare in SE projects. This can be attributed to a number of factors, some of which are that: while all projects are unique, software development projects face a high degree of new and changing technological dependencies, which typically impact project tool environments, knowledge management, process integration, and process data acquisition; significant process modeling effort is necessary and PAIS usage has been somewhat restrictive [5]; SE processes are knowledge processes [6], and thus, the exact operational determination and sequencing of tasks and activities is not readily foreknown, while process models are too inflexible to mirror such operational dynamics.

We developed the Context-aware Software Engineering Environment Event-driven frameworK (CoSEEEK) [7] to improve SE process support and guidance in an automated fashion. That way, enhanced support features are possible, such as automatically gathering information from the environment and users, uniting it with information from a knowledge base, and utilizing this information for on-the-fly process optimization (Section IIIC provides more information on CoSEEEK). Given such a context-aware event-driven automated process guidance system, we investigated the feasibility of enabling in-the-loop automated process assessment support. Our ontology-based approach semantically enhances a PAIS for SE operational process enactment and assessment support.

The paper is organized as follows: Section II describes the attributes of three common process models. Section III describes our automated process assessment solution approach. An evaluation of this approach is described in Section IV. Section V discusses related work. Section VI concludes the paper.

II. Process Assessment Models

Three of the most mature and prevalent process assessment approaches used in software projects (CMMI, ISO/IEC 15504 / SPICE, and ISO 9001) are described in order to later show how automation was achieved. Despite the differences, with ISO 9000 being more of a requirement model and CMMI and SPICE meta-process models, they are similarly used for assessing process compliance or maturity. All three models have several basic concepts in common: They define basic activities to be executed in a project such as 'Identify Configuration Items' for configuration management. (These will be mapped by a concept called *base practice* in our approach.) These activities are grouped together (e.g., 'Establish Baselines' in the configuration management example, with these groupings being mapped by a concept called *process* in our approach.) In turn, the latter are further grouped (e.g., 'Configuration Management') to allow further structuring. (This will be mapped by a concept called *process category* in our approach.) To be able to rate these practices and processes, the assessment models feature a *performance scale* to quantify the assessment. Finally, most models use the quantified assessments to assign *capability levels* to processes.

A. CMMI

CMMI (Capability Maturity Model Integration) [2] is one of the most widely used assessment models. It exists in different constellations, from which CMMI-DEV (CMMI for Development) is utilized in our context. The CMMI staged representation model comprises five *maturity levels* (1-'Initial', 2-'Managed', 3-'Defined', 4-'Quantitatively Managed', 5-'Optimizing'). The levels indicate 'Degree of process improvement across a predefined set of process areas, in which all goals within the set are attained' (cf. [2]). To implement this, each of the levels has subordinate activities that are organized as follows: A maturity level (e.g., '2') has *process categories* (e.g., 'Support') that have *process areas* (e.g., 'Configuration Management') that have *specific goals* (e.g., 'Establish Baselines') that finally have specific practices (e.g., 'Identify Configuration Items'). To quantify the assessment, CMMI has a performance scale (1-'unrated', 2-'not applicable', 3-'unsatisfied', 4-'satisified'). Using these concepts, process assessment is applied as follows:

- Rate each generic and specific goal of a process area using the introduced performance scale.
- A maturity level is achieved if all process areas within the level and within each lower level are either 2 or 4 (cf. the performance scale introduced).

In addition to these concrete activities and maturity levels, CMMI features *generic goals* (e.g., 'Institutionalize a Managed Process') with *generic practices* (e.g., 'Control Work Products'). These are subordinate to capability levels (0-'Incomplete', 1-'Performed', 2-'Managed', 3-'Defined', 4-' Quantitatively Managed', 5-' Optimizing'). The latter indicate 'Achievement of process improvement within an individual process area' (cf. [2]).

SCAMPI (Standard CMMI Appraisal Method for Process Improvement) [8] is the official CMMI appraisal method. It collects and characterizes findings in a Practice Implementation Indicator Database.

B. ISO/IEC 15504 (SPICE)

The SPICE (Software Process Improvement and Capability Determination) [3][9] model is an international standard for measuring process performance. It originated from the process lifecycle standard ISO/IEC 12207 [10] and maturity models such as CMM (the predecessor of CMMI). SPICE comprises six *capability levels* (0-'Incomplete process', 1-'Performed process', 2-'Managed process', 3-'Established process', 4-'Predictable process', 5-'Optimizing process'). Each of the latter has one or multiple *process attributes* (e.g., '2.1 Performance Management'). A process reference model was included in the initial version of the standard. This was later removed to support different process models (or the ISO/IEC 12207). Thus, mappings to various process models are possible. In this paper, the examples use the initial process model specifications for illustration. These comprised *process categories* (e.g., 'Organization') with *processes* (e.g., 'Improve the process') that contained *base practices* (e.g., 'Identify reusable components'). SPICEs measurement model applies the following performance scale for assessment: 1-'not achieved' (0-15%), 2-'partially achieved' (16% - 50%), 3-'largely achieved' (51% - 85%), and 4-'fully achieved' (86% - 100%).

As opposed to the CMMI, SPICE does not use assessments of practices to directly determine whether an overall capability level is achieved, but uses them to assign to each process one or more capability levels and to use them to recursively calculate assessments for projects and organizations. The assessment comprises the following steps:

- Assess every base practice with respect to each of the process attributes.
- Determine the percentage of base practices of one process that have the same performance scale with respect to one process attribute.
- Assessment of the processes: Assign the capability level for process attributes where all base practices of the process have performance scale 3 or 4 and for all lower capability levels the same applies with performance scale 4.
- Assessment of a project is done by using the mathematical mean of the ratings of all of its processes.
- Assessment of an organization is done by using the mathematical mean of the ratings of all of its projects.

C. ISO 9001

ISO 9000 comprises a family of standards relating to quality management systems. ISO 9001 [4] deals with the requirements organizations must fulfill to meet the standard. Formal ISO 9001 certifications have gained great importance for organizations worldwide. The ISO 9001 assessment model uses no capability scale; it only determines whether or not a certain practice is in place. Therefore, a simple performance scale suffices: 0-'not satisfied', 1-'satisfied'. The assessed practices are structured by *process sub-systems* (e.g., 'Organization Management') that contain *main topic areas* (e.g., 'Management responsibility'). In turn, the latter contain *management issues* (e.g., 'Define organization structure'). Based on these concepts, a recursive assessment can be applied rating an organization by its *process sub-systems* and the contained *management issues* with a pass threshold of 100%. Our approach is targeted at creating more quality awareness in companies, not at replacing or conducting formal reviews. Therefore, the standard ISO 19001:2011 (Guidelines for auditing management systems) [11] is not taken into account here.

D. Summary

As shown by these three assessment models, the approaches to process assessment differ significantly. This applies for the concepts utilized as well as for the applied procedures: For example, CMMI knows two different types of levels that have subordinate activities. For ISO/IEC 15504, the levels have certain attributes that serve to assess all existing practices. As opposed to the two other models, ISO 9001 does not apply levels or different performance scales. These differences hamper convergence to a unified model or approach and present the primary technical challenge.

III. AUTOMATED PROCESS ASSESSMENT

This section describes the approach taken to provide automated process assessment including the conceptual framework and procedure applied. The approach extends and annotates process management concepts, enhancing them with additional information required for assessment. The aim of our approach is not to replace manual ratings of processes conducted by humans or to be used in formal process audits. It shall rather contribute to the quality awareness of a company and provide information on the current state of the process as it is executed. Therefore, our approach, despite adding automated rating facilities, still integrates and relies on manual ratings or confirmations for ratings. The newly introduced *Context Management* component actively uses this information and interacts with the process execution component. The latter is tightly integrated with a *Process Management* component that executes workflows to operationally support a SE process.

A. Conceptual Framework

To achieve extended assessment functionality, process management concepts were enhanced. These are defined in the *Context Management* component and are associated with a *Process Management* component that manages process execution. This is illustrated by Figure 1.

Figure 1 shows a simple workflow in the *Process Management* component: This workflow is defined by 'Workflow Template 1' that contains four activity templates. Both of these concepts are mirrored in the *Context Management* component by the *Work Unit Container Template* that contains *Work Unit Templates*. When the workflow is to be executed, it is instantiated in the *Process Management* component and then represented by a workflow instance ('Workflow Instance 1') containing the activities to be processed. These two concepts are again mirrored in the *Context Management* component by the *Work Unit Container* that contains *Work Units*. These have explicitly defined states that are automatically synchronized with the states in the *Process Management* component. That way, the *Context Management* component is aware of the current execution state of workflows and activities.

Similar to the *Work Unit Containers* and their templates, the concepts for process assessment are separated into template concepts for definition and individual concepts holding the actual values of one execution. The *Assessment Process Template* defines one process assessment model. In alignment to the aforementioned assessment approaches, it features templates for *Process Categories*, *Processes*, and *Base Practices* as well as *Capability Levels*. The latter are general level concepts used to model various capability or maturity levels that can be calculated for other concepts such as *Base Practices* or *Assessment Processes*. To explicitly configure how the capability level achievement will be determined, *Capability Determinator Templates* are used. The *Assessment Process Template* also defines a number of *Performance Scales* that are used for the assessment later. For all these concepts, there are individual counterparts used for each concrete assessment that are based on the template concepts. Table 1 depicts their relevant properties including a short description.

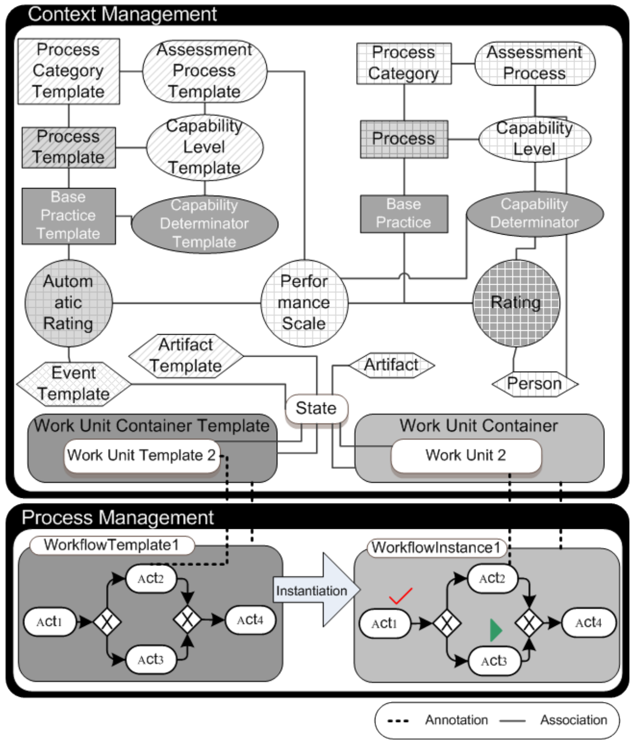


Figure 1. Conceptual framework for automating process assessment.

TABLE I. CONCEPTS PROPERTIES

Property	Description
Assessment Process Template	
capabilityLevels	all defined capbility levels templates
procCatTempls	all defined process category templates
Capability Level Template	
calcFor	concept, for which the level is calculated
capDet	attached capability determinator templates
perfScale	required performance scale for achievement
scaleRatio	ratio of capability determinators that must meet required performance scale
subCL	subordinate capability template template
subCLPerfScale	required performance scale of subordinate level
Level	number indicating the level
Capability Determinator Template	
Source	base practice to be assessed
Target	capability level, for which this determinator is used

For flexibility in the assessment calculation, the *Capability Level Templates* have a property ‘calcFor’ that is used to attach them to the target concept to be calculated (e.g., the whole assessment process when calculated for a project of a single process). As proposed by the three introduced models, level achievement calculation can rely on the assessment of the practices or subordinate levels. Therefore, the achievement of a capability level is determined by the following properties: ‘perfScale’ defines which *Performance Scale* the attached *Capability Determinators* has, and via ‘scaleRatio’ a ratio of *Capabiltiy Determinators* can be defined as required for the *Performance Scale*. Additionally, as the *Capability Levels* are connected to other subordinate levels, the *Performance Scale* of their determinators can also be used (cf. SPICE, required by the ‘subCLPerfScale’ property).

The assessment of the concrete individual concepts is then applied via the explicit *Rating* concept, which connects a *Performance Scale* with a *Base Practice* and a *Capability Determinator*. It can also be connected to a concrete *Person* who will then be asked to do the assessment. To support automation in the assessment procedure and unburden the users, it is also possible to automate ratings with *Automated Rating*. It can be connected to an *Event Template* concept that, in turn, is connected to the *States* of *Artifacts* or *Work Unit Containers*. That way, it can be configured so that when the *Concept Management* component receives certain status change events, a certain *Performance Scale* is assigned to a certain rating. Examples of such a definition include: ‘Assign *Performance Scale* 1 if workflow x is present (created)’ or ‘Assign *Performance Scale* 2 if workflow x is completed’ or ‘Assign *Performance Scale* 3 if *Artifact* y is in state z’.

B. *Assessment Procedure*

The concrete assessment procedure applied to rate process performance is shown in Listing 1. The following algorithm describes how a concrete *Assessment Process* is created from its template, how the ratings are applied to the different *Base Practices* contained in the process, and how achievement of maturity/capability levels is determined.

Listing 1. The Rate Process Performance algorithm in pseudocode.

```
Require: Project P, AssessmentProcessTemplate APT,
Person Pers
01: AssessmentProcess AP ← createConcept(APT)
02: linkConcepts(P, AP)
03: for all APT.processCategoryTemplates PCT do
04:  ProcessCategory PC ← createConcept(PCT)
05:  linkConcepts(AP, PC)
06:  for all PCT.processTemplates PT do
07:    Process PR ← createConcept(PRT)
08:    linkConcepts(PC, PR)
09:    for all PRT.basePracticesTemplates BPT do
10:      BasePractice BP ← createConcept(BPT)
11:      linkConcepts(PR, BP)
12:    end for
13:  end for
14: end for
15: for all APT.capabilityLevelTemplates CLT do
16:  CapabilityLevel CL ← createConcept(CLT)
17:  linkConcepts(AP, CL)
18:  linkConcepts(CL, CLT.calculatedFor)
19:  for all CLT.capabilityDeterminatorTemplates
          CDT do
20:    CapabilityDeterminator CD ←
              createConcept(CDT)
21:    linkConcepts(CL, CD)
22:    List relatedBPs ← getRelatedBasePracts(CD,
                                    AP)
23:    for all relatedBPs BP do
24:      new rating(CD, BP,
           AP.getStandardPerformanceScale,Pers)
25:    end for
26:  end for
27: end for
28: automatedRating(AP)
29: manualRating(AP)
30: for all AP.capabilityLevels CL do
31:  checkAchievement(CL)
32: end for
```

The algorithm requires a concrete project and an *Assessment Process Template* to be used for that project. The first part of the algorithm (lines 01-14) then creates a structure comprising *Process Categories*, *Processes* and *Base Practices* for the new *Assessment Process*. For this paper, the following two functions are used: ‘createConcept’ creates an individual concept from a given template and ‘linkConcepts’ links two individual concepts together.

The second part of the algorithm (line 15-27) creates the *Capability Level* structure. Therefore, the *Capability Levels* and their attached *Determinators* are created first. Thereafter the *Determinators* are linked to the *Base Practices* they use for determining capability. This is done using the function ‘getRelatedBasePractices’ that gets all *Base Practices* in the current *Assessment Process* that are configured to be connected to a certain *Capability Determinator* via their templates. For each of these *Base Practices*, a new *Rating* is created linking them to the *Capability Determinator*. To this *Rating*, a standard *Performance Scale* (usually the one equal to ‘not achieved’) and a responsible person are attached.

The third part of the algorithm (lines 28-32) deals with the concrete assessment. During the whole project, an automated rating is applied whenever a matching event or status change happens. At the end of a project (or anytime an assessment is desired), the manual rating is applied, distributing the rating information to the responsible person, who can then check the automated rating, rate practices that have not yet been rated, or distribute certain parts of the assessment to others who can provide the missing information needed to rate the practices. The final action applied is to check the achievement for each *Capability Level* of an *Assessment Process*.

C. *Technical Realization*

The aforementioned conceptual framework was technically realized via integration in CoSEEEK [7], a framework whose purpose is to provide holistic support for SE projects and processes. This contains the Context Management component with semantic web technology (i.e., an OWL DL ontology [12] and the reasoner Pellet [13]), enabling better knowledge reusability and logical classification capabilities regarding the contained knowledge. This knowledge is extended by contextual information automatically received by sensors using the Hackystat framework [14]. In turn, the *Process Management* component integrates the dynamic PAIS AristaFlow [15], which enables the correct dynamic adaptation of running workflows. For further information on these components and their application for integrating areas such as knowledge management or quality management, see [16][17].

IV. EVALUATION

This section evaluates our approach by applying it to the three different process assessment models introduced in Section II, and further elucidates technical realization details. A selection of the applied concepts is shown in Figure 2 for all of the three models.

A. CMMI

An excerpt of the implementation of the CMMI model is shown in Figure 2(a). On the upper half, the templates for defining the CMMI concepts are shown: The structure of the process is built by the *Process Category Template* (used for the process areas CMMI), the *Process Template* (used for the specific goals CMMI), and the *Base Practice Template* (used for the specific practice of CMMI). Connected to the 'CMMI Template' (implemented by the *Assessment Process Template*) are also the 'Maturity Levels' (implemented by the *Capability Level Template* concept). In addition to this structure with the specific goals and maturity levels, the applied concepts can also be used to implement the generic goals of CMMI with their generic practices and the relating capability levels as illustrated. For the *Assessment Process Template*, the maturity levels are connected to the *Capability Determinators* of all specific practices that belong to the relating maturity level. The *Capability Determinators* also realize connections to *Base Practices* that implement CMMIs generic practices applied to the respective process area (implemented by a connection from the Base Practice, the Process, and the Process Category, cf. 'Establish an Organizational Policy', 'Institutionalize a Managed Process', and 'Configuration Management' in Figure 2). Similar connections can be established for the capability levels, so that the staged or the continuous representation of CMMI to assess respectively the maturity of a whole organization or its capabilities concerning the different process areas. For the capability determination, the *Assessment Process Template* is also connected to the *Performance Scales* that will be used for it. The figure shows one example of them (4 – Satisfied).

On the lower part of Figure 2(a), the individual concepts for the assessment of one concrete project with CMMI are illustrated. It shows one exemplary maturity level and one process area with one specific goal with one specific practice. The *Capability Determinators* of the maturity level are connected to the specific practices that shall be rated via the *Rating* that has an assigned *Performance Scale*. A similar excerpt of the structure is shown for the capability levels and generic goals in the figure.

The achievement calculation for the maturity levels is done with the 'perfScale' and 'scaleRatio' properties of the *Capability Level Template*: That way it can be defined that 100% of the *Capability Determinators* must have the Performance Scale '4' or '2' as defined in the CMMI model. If calculations for all of the projects of an organization were in place, maturity indicators for the entire organization could use the lowest maturity level achieved by all projects.

B. ISO/IEC 15504 (SPICE)

An excerpt of the implementation of the SPICE model is shown in Figure 2(b). In this case the names of the concepts match with the names used in SPICE (e.g., for capability levels or base practices). The *Performance Scales* are defined for the *Assessment Process Template* similar to the CMMI implementation, e.g., 4 – Fully Achieved (86%-100%) as shown in the figure. The process areas that are subordinate to the capability levels in SPICE are implemented using the *Capability Determinator Templates*. Each of the latter is connected to all *Base Practice Templates* to enable their rating concerning all process attributes as required by SPICE.

The lower part of Figure 2(b) again shows an excerpt of the individual concepts used for the assessment of a concrete project. It comprises an exemplary capability level with its two process attributes and an exemplary process category with one process and one base practice.

The SPICE assessment works as follows: All base practices are rated according to all process attributes, and capability levels are determined for the processes. A level is achieved if all its related process areas have only ratings with *Performance Scales* '3' or '4', and the process areas of the subordinate levels all have *Performance Scale* '4'. The assessment of the project is the mathematical mean of the assessments of the processes, and can thus be easily computed without explicit modeling. The same applies to the assessment of a whole organization.

C. ISO 9001

As ISO 9001 is a requirement and not a process model, it must be mapped to the organization's process. This can be applied by connecting *automated ratings* to *events* occurring in the execution of *work unit containers* representing the real execution of a related workflow or be applied manually by a person doing a manual rating. An excerpt of the implementation of the ISO 9001 assessment model is shown in Figure 2(c). In this case, the upper part of the figure again shows the template concepts for defining the model. Compared to the other two models, ISO 9001 is simpler: It knows no capability levels and only two performance scales (as shown in the figure). Therefore, there is only one *Capability Level Template* defined that is used to determine achievement for the whole ISO 9001 assessment. That template has one *Capability Determinator Template* for each management issue.

The lower part of Figure 2(c) again shows the individual concepts used for a concrete assessment using a concrete example for a process subsystem, a main topic area, and a management issue. The assessment is applied by the 'perfScale' and 'scaleRatio' properties of the single *Capability Level*, specifying that all *Capability Determinators* must have the *Performance Scale* '1'. As ISO

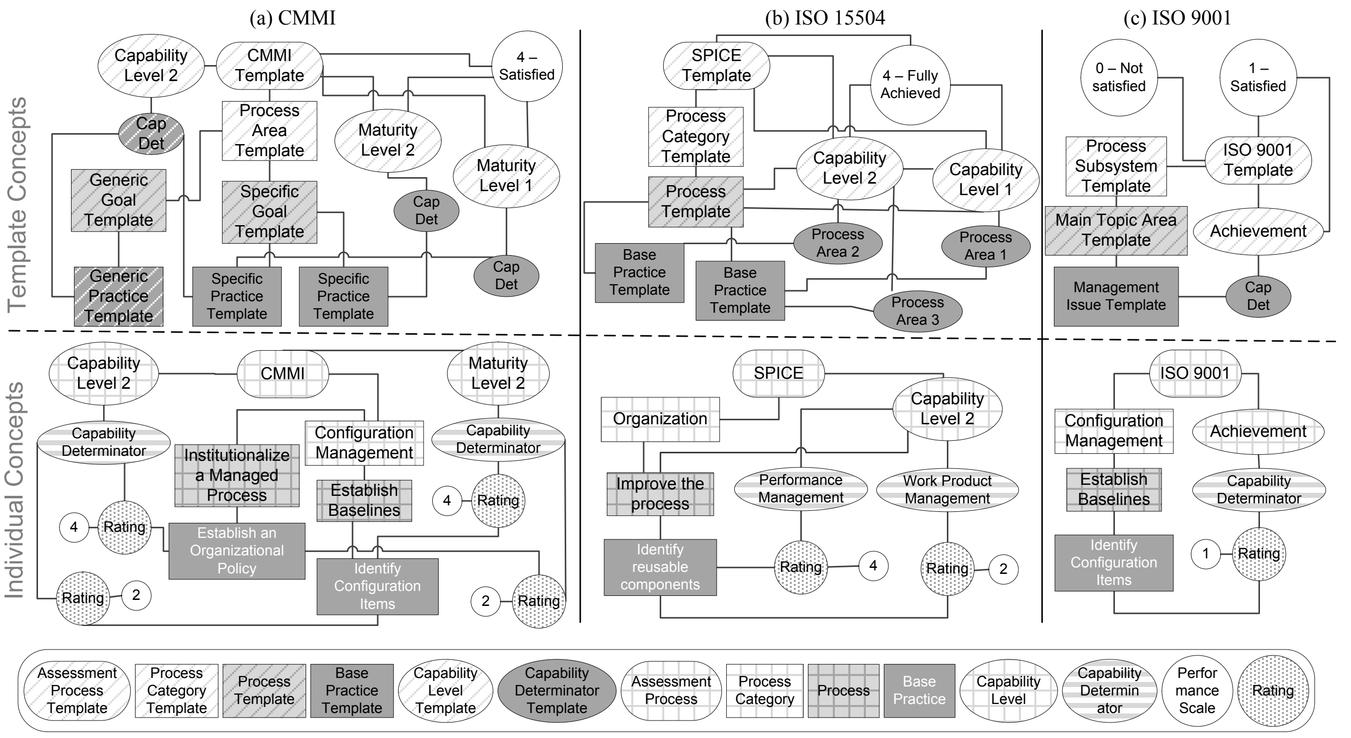


Figure 2. Realization for specific reference models: (a) CMMI (b) ISO 15504 (c) ISO 9001.

9001 knows no project level, this can be added by using a separate Assessment Process for each project, and cumulating the assessment over the whole organization (if all projects have achieved, the whole organization has achieved).

D. *Performance and Scalability*

Process assessment approaches often comprise dozens or even hundreds of concepts (e.g., SPICE has over 200 base practices), which implies the creation of an even higher number of concepts in the ontology to enable automated assessment. Therefore, the utilization of a separate ontology for process assessment is considered to keep the operational ontology of the CoSEEEK framework clean. Furthermore, to support stability and performance, the CoSEEEK ontologies are not managed as plain files but stored in a database (using Protégé functionality). The test configuration consisted of a PC with an AMD Dual Core Opteron 2.4 GHz processor and 3.2GB RAM with Windows XP Pro (SP3) and the Java Runtime Environment 1.5.0_20, on which CoSEEEK was running networked via Gigabit Ethernet to a virtual machine (cluster with VMware ESX server 4.0, 2 GB RAM allocated to the VM, dynamic CPU power allocation) where the AristaFlow process server is installed.

The approach supports model diversity, and thus the ontology size can vary based on various reference models. Scalability of the approach was assessed, since a large number of concepts can be required with complicated models such as SPICE - which has over 200 *Base Practices* that require linking to all process areas and calculation of all *Capability Levels* for the *Processes*. The most resource intensive point is when the entire *Assessment Process* for a project is created, thus performance and scalability tests were conducted for the automatic creation of linked ontology concepts, scaling the number of concepts to account for smaller to larger models.

The results obtained were: 1.7 seconds for the creation and linking of 100 concepts, 14.2 seconds for the creation and linking of 1000 concepts, and 131.4 seconds for the creation and linking of 10000 concepts. The results show that the computation time is acceptable with approximately linear scaling. The slight reduction in average creation time for a single concept is perhaps explainable by reduced initialization percentages and caching effects. At this stage, the performance of the Rate Process Performance algorithm (Listing 1) was not assessed since it is fragmented across a project timescale (at the beginning the concepts are created and later the ratings are applied), it is dependent on human responses (manual ratings), and live project data has not as yet been collected.

V. Related Work

As to related work, a multi-agent system approach is presented in [18] to enable automatic measurements for the SW-CMM (Software Capability Maturity Model). The latter is combined with the GQM (Goal-Question-Metric) [19] method, where Goals of the SW-CMM are used as first step for GQM.

An OWL ontology and reasoner approach for CMMI-SW (CMMI for Software) is presented in [20]. In contrast to our approach, the size of the ontology caused issues for the reasoner. A software process ontology in [21] enables the capturing of software processes on a conceptual level. An extension includes specific models such as SPICE or CMMI. Ontological modeling of both CMMI and ISO 9001 as well as certain process interoperability features is shown in [22]. The authors identify issues in consistently implementing both models simultaneously. This problem was addressed in our approach by including concepts abstracted from a single model. In [23], a Process-Centered Software Engineering Environment supports process implementation focused on CMMI and a Brazilian process improvement model. For CMMI-specific appraisals, multiple supportive tools are available such as the Appraisal Assistant [24]. However, these focus only on CMMI / SCAMPI support.

We provide a more general and flexible approach, since the applied concepts are abstracted from a single model. In contrast to above related work that focused on one or two specific models, ours is capable of assessment model diversity as shown in Section IV. Furthermore, it integrates automated SE process enactment support and supports a combination of automated and manual ratings. That way, the assessment is tightly and automatically integrated with SE process execution support, providing the option of automatic on-the-fly assessments while preserving the ability for humans to manually rate practices and processes. This can support quality awareness.

VI. Conclusion and Future Work

This paper has described an approach for automating the assessment of software engineering processes, first elucidating the differences between three common SE process reference models, and thereafter presenting our conceptual framework with semantic extensions to a process-aware information system. It was shown how process reference models such as CMMI, ISO 15504, and ISO 9001 were unified in the ontology and the algorithm that performs the assessment was described. The evaluation demonstrated the technical feasibility, model diversity, and that performance with current technology for expected application scenarios is sufficient.

Our approach is not meant to replace manual ratings or formal appraisals. In our opinion, this is not possible in an automated fashion due to the many factors influencing such ratings in real world process execution. However, our approach can support data collection, contribute to the quality awareness of an organization, and highlight areas for process optimization. Furthermore, it can help prepare an organization for a formal appraisal.

Future work involves empirical studies to evaluate the effectiveness of the approach in industrial settings with a variety of software organizations, with various SE process lifecycle models in various projects, at various process capability levels and utilizing different process assessment standards simultaneously.

Acknowledgment

This work was sponsored by the BMBF (Federal Ministry of Education and Research) of the Federal Republic of Germany under Contract No. 17N4809.

References

[1] P. Bourque and R. Dupuis, (ed.), “Guide to the Software Engineering Body of Knowledge”, IEEE Computer Society, 2004.

[2] CMMI Product Team, “CMMI for Development, Version 1.3,” Software Engineering Institute, Carnegie Mellon University, 2010.

[3] ISO, “ISO/IEC 15504-2 -- Part 2: Performing an assessment,” 2003.

[4] R. Bamford, and W. J. Deibler, “ISO 9001: 2000 for software and systems providers: an engineering approach,” CRC-Press, 2004.

[5] M. Reichert and B. Weber, “Enabling Flexibility in Process-aware Information Systems – Challenges, Methods, Technologies,” Springer, 2012.

[6] G. Grambow, R. Oberhauser, and M. Reichert, “Towards Dynamic Knowledge Support in Software Engineering Processes,” 6th Int’l Workshop Applications of Semantic Technologies, 2011, pp. 149.

[7] R. Oberhauser and R. Schmidt, “Towards a Holistic Integration of Software Lifecycle Processes using the Semantic Web,” Proc. 2nd Int. Conf. on Software and Data Technologies, 3, 2007, pp. 137-144.

[8] SCAMPI Upgrade Team, "Standard CMMI Appraisal Method for Process Improvement (SCAMPI) A, v. 1.3," Software Engineering Institute, 2011.

[9] ISO, “ISO/IEC 15504-5:2012 -- Part 5: An exemplar software life cycle process assessment model,” 2012.

[10] ISO, “ISO/IEC 12207:2008 -- Software life cycle processes,” 2008.

[11] “ISO 19011 - Guidelines for auditing management systems,” 2011.

[12] World Wide Web Consortium, “OWL Web Ontology Language Semantics and Abstract Syntax,” 2004.

[13] E. Sirin, , B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, “Pellet: A practical owl-dl reasoner,” Web Semantics: Science, Services and Agents on the World Wide Web, 5(2), 2007, pp. 51-53.

[14] P.M. Johnson, “Requirement and design trade-offs in Hackystat: An in-process software engineering measurement and analysis system,” Proc. 1st Int. Symp. on Empirical Software Engineering and Measurement, 2007, pp. 81-90.

[15] P. Dadam and M. Reichert, “The ADEPT project: a decade of research and development for robust and flexible process support,” Computer Science-Research & Development, 23(2), 2009, pp. 81-97.

[16] G. Grambow, R. Oberhauser, and M. Reichert, “Knowledge Provisioning: A Context-Sensitive Process-Oriented Approach Applied to Software Engineering Environments,” Proc. 7th Int’l Conf. on Software and Data Technologies, 2012.

[17] G. Grambow, R. Oberhauser, and M. Reichert, “Contextual Injection of Quality Measures into Software Engineering Processes,” Int'l Journal on Advances in Software, 4(1 & 2), 2011, pp. 76-99.

[18] M.A. Seyyedi, M. Teshnehlab, and F. Shams, “Measuring software processes performance based on the fuzzy multi agent measurements,” Proc. Intl Conf. on Information Technology: Coding and Computing (ITCC'05) – Vol. II, IEEE CS, 2005, pp. 410-415.

[19] V.R. Basili, V.R.B.G. Caldiera, and H.D. Rombach, “The goal question metric approach,” Encycl. of SW Eng., 2, 1994, pp. 528-532.

[20] G.H. Soydan and M. Kokar, “An OWL ontology for representing the CMMI-SW model,” Proc. 2nd Int'l Workshop on Semantic Web Enabled Software Engineering, 2006, pp. 1-14.

[21] L. Liao, Y. Qu, and H. Leung, “A software process ontology and its application,” Proc. ISWC2005 Workshop on Semantic Web Enabled Software Engineering, 2005, pp. 6–10.

[22] A. Ferchichi, M. Bigand, and H. Lefebvre, “An ontology for quality standards integration in software collaborative projects,” Proc. 1st Int'l Workshop on Model Driven Interoperability for Sustainable Information Systems, 2008, pp. 17-30.

[23] M. Montoni et al., “Taba workstation: Supporting software process deployment based on CMMI and MR-MPS,” Proc. 7th Int’l Conf. on Product-Focused Software Process Improvement, 2006, pp. 249-262.

[24] Appraisal Assistant, http://www.sqi.gu.edu.au/AppraisalAssistant/about.html [July 2012]

Specification of Formalized Software Patterns for the Development of User Interfaces

Danny Ammon, Stefan Wendler, Teodora Kikova, Ilka Philippow
Software Systems / Process Informatics Department
Ilmenau University of Technology
Ilmenau, Germany
{danny.ammon, stefan.wendler, teodora.kikova, ilka.philippow}@tu-ilmenau.de

Abstract — **The aim of this paper is the development of specifications for a general analysis model for user interface patterns that can be applied in a model-based user interface development process. To accomplish this, we compile a detailed definition of what user interface patterns are and how they can be classified. Furthermore, we analyze how available methods and notations can be used for a pattern application in user interface development, based on two exemplary applications of the pattern "Advanced Search" in the formal notations UIML and UsiXML. From the resulting possibilities and limitations in identification, selection, instantiation and integration of user interface patterns, we derive specifications for a sufficient pattern description and development integration method: an exact definition, a metamodel, a specialized language, and, in practice, a repository or pattern management software.**

Keywords — user interface patterns; user interface development; pattern specifications; UIML; UsiXML.

I. Introduction

A. Motivation

The design and implementation of user interfaces is still a complex and resource-consuming task. In general, pattern-based software development is a means to more efficient implementation by applying reusable solutions for miscellaneous software design problem classes. In this regard, the use of software patterns in user interface development would offer generic solutions for recurring components of a user interface, depending on a certain interface paradigm. Navigation through tabs, for example, would be a feasible solution for the need to switch between complex sets of documents, websites, forms, etc. in graphical and touch user interfaces.

Currently, the application of such user interface patterns is situated only on an informal level with textual descriptions of common design solutions [1]. There is only limited research into generative, formalized user interface patterns, which can be applied for the automation of re-use of design solutions [1]. In this regard, methods for the development of user interfaces were introduced, starting at the stage of task or system models and matching user interface patterns with parts of these models [2][3].

However, we found no consistent suggestion of a pattern-based user interface design and implementation process, which combines a sufficient pattern repository, consisting of formalized user interface patterns, and an end-to-end solution of model-based pattern matching, selection, instantiation, and code generation. In addition, a generally accepted notation for user interface patterns is missing, which allows an abstract formulation of human-computer interaction components. Being transferable into concrete user interface-part descriptions and, finally, instantiable into source code, these abstract components could be deployed to form elements of real user interface patterns and thus facilitate reuse in GUI development.

B. Objectives

The aim of this paper is the development of a specification for a general analysis model that describes generative user interface patterns so that their common aspects can be identified and captured. This basic specification and its understanding are needed for the integration of methods that enable the matching and code generation based on the application of these patterns. We explicate how available methods and notations could be used for a user interface pattern repository or pattern manager. Moreover, we analyze the strengths and weaknesses of these existing assets and point out what better suitable methods and formats would have to be capable of. A sufficient solution for pattern-based user interface development should particularly meet the following criteria:

- reusability and variability of stored user interface patterns
- ability of user interface patterns to be composed in order to form a hierarchy of GUI components
- instantiation of user interface patterns into varying interface paradigms and types

Based on these criteria, we review the state of the art and describe a perspective on user interface patterns that paves the way for the specification of a sophisticated metamodel needed in model-based user interface development environments.

C. Structure of the Paper

In Section II, we analyze existing methods of user interface development and independent interface description languages. We also outline the current status of the application of user interface patterns in the development process. In Section III, we propose a definition and characterization of user interface patterns, their inclusion criteria and dimensions. We use this definition to establish and utilize a formalized pattern, advanced search, for the application of current methods and notations in Sections IV and V. We show the results and weaknesses of our work and derive requirements for a fully applicable formal pattern description language in Section VI. Finally, in Section VII we conclude with specifications for formalized user interface patterns, which will meet the three criteria mentioned in our objectives.

II. Related Work

A. User Interface Development and Description

Today, the design and implementation of a software user interface mainly concentrates on the basic conditions and abilities of the before-chosen programming language and used software frameworks or libraries. After the general design of a user interface, the implementation is in focus, whether it is in Java Swing, HTML and CSS or C# and the XAML, to name only a few examples. While there has been a lot of research conducted on model-based user interface development, only a limited number of generic model concepts for a methodic interface design exist. One of these can be found in [4], where common steps of a user interface development process are explicated. Four model layers and corresponding transformations to derive user interface specifications from requirement models are proposed by Ludolph.

Another approach relies on a UML-based design of user interface software architecture [5]. Chlebek describes a comprehensive process and provides several perspectives onto the user interface development. Also, a special description language for the development process, which is independent from target source code, is used by him.

A greater number of platform-independent user interface description languages do exist. These languages are often XML-based and thus markup languages. Some of them have been developed for certain software projects or company-specific programming tools, such as XUL [6] and XAML [7]. Others, like UIML [8][9] and UsiXML [10], are results of research projects, but are rarely used in practice.

None of those generic concepts for interface development processes we found enabled the application of user interface patterns. Neither do independent user interface description languages have sufficient capabilities to store user interface patterns in their format. The GUI aspects described by these languages tend to be invariant and too concrete in specification [11] so that they do not provide any means to adapt the user interface to varying contexts. However, several special approaches for an integration of patterns into user interface development exist, which are outlined in the following subsection.

B. Pattern-based User Interface Development

Currently, there is no generally accepted definition of software patterns for user interface development. Instead, different concepts and terms exist, such as user interface patterns, user interface design patterns, or human-computer interaction patterns. Most of them refer to textual and graphical descriptions as solutions of a user interface design problem concerning mostly visual aspects and interaction concepts. These are termed descriptive user interface patterns [1]. Several libraries of descriptive patterns exist, such as [12][13][14]. Rarely do such descriptive pattern collections provide implementation details [15].

For a direct integration of reusable patterns for the user interface into a development environment, formal models or notations are needed, which enable a certain functionality and can be instantiated into certain model stages or source code, like design patterns. This variant is called generative pattern [1].

Generative patterns can be applied in a pattern-based user interface development process. One example of such a development process has been created by the University of Rostock in Germany [3][16][17][18][19][20]. Therein, model-based and pattern-driven design has been integrated by using several model layers (task, dialog, presentation and layout) to perform an identification and a selection, an instantiation and an integration of user interface patterns during the generation of the used models [3]. A tool has been developed, which supports this integrated development [3]. The user interface patterns are stored as fragments of the used models (“patterns in modeling”). They are used for more efficient modeling steps (“accelerating the design”) [18]. In a similar approach, an enhanced CASE tool was suggested, where user interface patterns are stored as class diagrams [2]. The static description of classes is then matched with the existing patterns, enabling a high level design of systems and their user interface. Identified classes can be replaced by the corresponding stored pattern, which is again a contribution to efficiency of the interface development process at the modeling level.

Other approaches go further to the generation of formal user interface description or source code [16][17]. Here, XUL is used to store formalized patterns, or a combination of PLML [21], UsiXML and additional components [22].

However, due to the used description languages, only one interaction paradigm is supported — the so-called WIMP (windows, icons, menus and pointer) interface typical for modern desktop computer and notebook operating systems. Furthermore, a major issue of the suggested integration of patterns in the development process is the need for manual retouching work. In this respect, the pattern instances have to be created manually by adapting them to their application context. In addition, not all kinds of patterns are supported. The occurrence of sub-patterns is the only relation between user interface patterns, which is dealt with in detail.

Starting with the definition itself, currently there is no consent to the arrangement of software patterns for the development of user interfaces, their structure and characteristics, as well as their relations among each other and to other software patterns.

III. User Interface Patterns: Definition and Characteristics

A generally acceptable definition of software patterns in user interface development should not only describe precisely what a pattern is and how it can be reused and adapted. Additionally, it should combine the several dimensions these patterns can be classified with. Thus, we propose the following definition:

In general, **user interface patterns** are software patterns, which can be applied for the specification, description and development of user interfaces.

As there is no common basis in literature for user interface pattern characteristics, the definition above is to be refined by our findings and arguments focused on the compositional view on user interface patterns we gathered during our observations in an industry project. The argumentative perspective presented here leads to requirements for a formal definition of user interface patterns that can be implemented by a metamodel in future work. To establish a more detailed clarification we describe the aspects a user interface pattern basically consists of in the following sub-section.

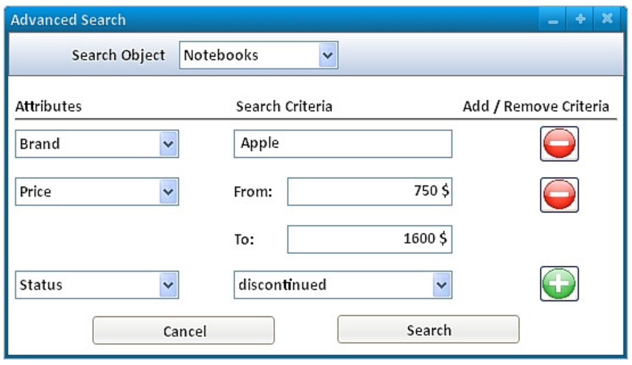
A. *User Interface Pattern Aspects*

Firstly**,** a user interface pattern incorporates a stereotype but abstract **view**. This aspect defines the selection, arrangement and types of user interface controls. Regarding this aspect, the user interface pattern does not refer to certain GUI frameworks so that the view can be implemented using different languages and technologies. In addition, the view is abstract in order to allow its application in various contexts. The abstract manner of view is backed by other user interface pattern specification language sources. For instance, the “facets” and “Abstract Interaction Objects” of the “abstract UI model” in [1] imply a view that has to be refined and transformed to certain platforms and renderings [10]. Besides UsiXML, UIML [8] specifications can be used to define a view composed of abstract elements in its structure section, which will be refined by a peer section to translate the view elements to certain GUI framework components or user interface controls.

Many user interface pattern libraries like [12][13] only focus on the view aspect. Metaphors [4][11] like trash bins and shopping carts may represent the foundation for the views of user interface patterns, but they also drive the aspect of interaction.

Secondly, a user interface pattern embodies a stereotype **interaction**. An interaction between a user and several user interface pattern instances of a certain type is always perceived and performed in the same way by the user. For example, each time a user interacts with a “Search Box” [12], he inputs the search string, selects the search category using the list box and finally triggers the actual search with the button. The options and sequences of interaction along with related behavior are defined independently from the context the pattern is being used in. Another example underlines that: A set of checkboxes is used to select only two options out of many available. The user interface pattern has to enable this constraint in its definition, regardless of the actual number of checkboxes within the possible pattern instances. Forming a unit of general purpose and applicability together with the view aspect, the interaction aspect adds essential value to the user interface pattern definition, which is reusable in many contexts, accordingly. The interaction strongly relies on and refers to the view aspect. This unity of view and interaction primarily forms the reusable entity and distinguishes the user interface pattern from ordinary GUI framework components and composite user interface controls.

Thirdly, besides the first two mandatory aspects, a user interface pattern may define an optional context dependent **control**. This aspect is primarily needed for user interface patterns that are composed of several user interface controls or even other user interface patterns. These composite patterns react on the context they are applied to by selecting, instantiating and configuring their child elements. An example for such a pattern is given by the “Advanced Search” [12], which enables the user to select search criteria depending on the object to be searched. This particular pattern offers “a special function with extended term matching, scoping and output options”, when “users need to find a specific item in a large collection of items” [12]. A possible interface of an advanced search pattern instance is drawn in Figure 1.



Search Object: Notebooks

Attributes	Search Criteria	
Brand	Apple	
Price	From: 750 $	
	To: 1600 $	
Status	discontinued	

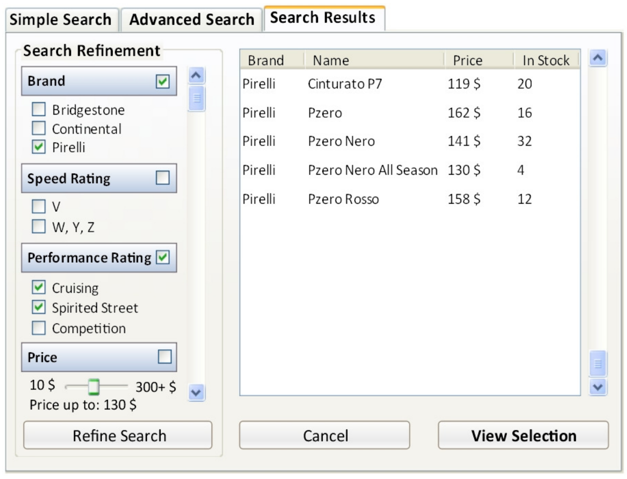
Figure 1. Interface example for an advanced search dialog

Each search criterion line refers to one of the object attributes’ data type and thus can be regarded as a smaller user interface pattern that is instantiated on demand. For all money types, as shown in Figure 1, two values can be entered as search parameters. Each time a money type occurs, the same view and behavior are to be instantiated, hence this type of search criterion line is defined as a user interface pattern.

Another example is depicted in Figure 2. This dialog is composed of several user interface patterns working together. A “Data Table” [13], which is configured according to the object to be displayed, is arranged on the right hand side. On the left hand side, a search refinement can be specified using the given criteria, which are derived from the objects’ attributes and their data. The main user interface pattern defines the entire “Search Results” tab, configures and instantiates its child patterns depending on the object and its attributes to be searched. Eventually, the interaction aspect of the dialog is distributed along the pattern instances. The controlling aspect of the main pattern handles the lifecycle of each child pattern instance and queries their interaction events in order to complete its own interaction sequence. For example, only the activated search criteria in Figure 2 are considered for compiling search data, when the button “Refine Search” is activated. Thus, the second aspect of user interface patterns provides the input for the controlling mechanism of more sophisticated or hierarchical user interface patterns. The need for a controlling aspect depends on the structure and purpose of the pattern itself. The simple search box does not need the third aspect, since it always features the same visuals, configured data and output events or data. Its behavior is limited to states that can be determined at design time easily. In contrast, the states of the “Advanced Search” or “Search Refinement” can be determined only at runtime, with

)

knowledge about the application context, and finally, user inputs.



Brand	Name	Price	In Stock
Pirelli	Cinturato P7	119 $	20
Pirelli	Pzero	162 $	16
Pirelli	Pzero Nero	141 $	32
Pirelli	Pzero Nero All Season	130 $	4
Pirelli	Pzero Rosso	158 $	12

Figure 2. Interface example for a search results dialog

According to the need for the controlling aspect, more examples of user interface patterns are “Wizard” [12] or reusable dialog types like the “Search Refinement” that may act as templates for several dialog instances. The context the controlling aspect of the pattern relies on can be embodied by a static artifact, e.g., an object and its attributes, or dynamic artifact, e.g., a state machine or task model. By referring to the latter, an implicit connection to the dialog controller of the software architecture can be established.

B. *Variability Perspective on User Interface Patterns*

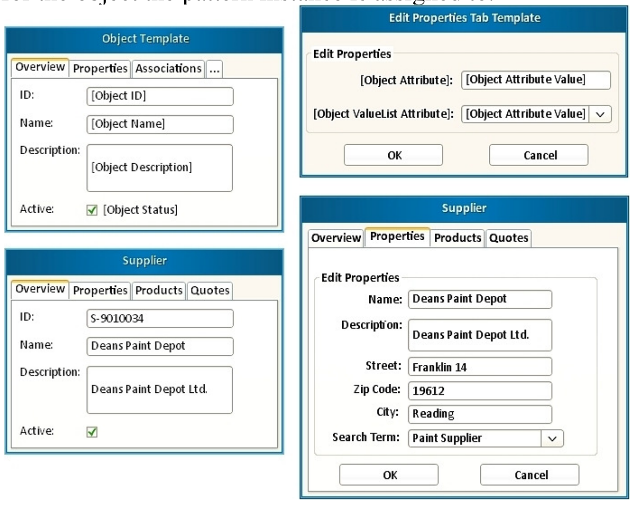
The instantiation of a user interface pattern for varying contexts will result in implementations of given architectural components that differ in certain aspects. That is why we refer to the common architectural pattern MVC [23] as a perspective for discussing the adaptability, variability and reuse of user interface patterns in different contexts.

Firstly, the easiest way of applying a user interface pattern in various contexts can be established by adapting it to accept a range of **data types** for its defined view aspect elements. For instance, a “Data Table” [13] being part of a user interface pattern view aspect will be instantiated for displaying a variety of business objects with different data types in their attributes. Another example is given by “Event Calendar” [12] or similar patterns, which interpret the given values within the model by proving an appropriate display of data. This kind of reuse would only affect the model part of the architecture.

Secondly, user interface patterns need to be adapted to the actual **dialog layout**. A “Double Tab Navigation” [12] needs to be shaped to the actual menu contents and layout to be displayed, for example. This results in a change of the presentation (view) component and related events, where the number, ordering and layout of required user interface controls have to be determined and implemented accordingly. Consequently, user interface patterns act as templates for the static and dynamic aspects of the view component and its presentation control. Therefore, the view and its related controller have to be adapted.

Thirdly, besides the prior concerns, composite user interface patterns have to feature variability regarding their **controlling aspect**. The controller of an MVC triad can be considered to be acting on two different levels. One part of the controller is responsible for the visual event handling only and is closely related to the view aspect of a user interface pattern instance. Due to cohesion and coupling concerns, the scope of this controller should be limited to one visual design unit, meaning one user interface pattern instance and its specified behavior at a time. The other part of the controller should handle the application related or logical behavior. Since user interface patterns can be composite, controllers should follow the same structure and be assigned to the individual pattern instances. With this compositional structure of the patterns and the controllers accordingly, the reuse of certain combinations of patterns will be facilitated.

An example depicting the variability of user interface patterns is given in Figure 3. On each side of the upper half a visual representation of a user interface pattern specification is shown. The first dialog sketch defines the view used for a business object and the tabs, which establish the navigation structure a user might interact with. The second dialog sketch above visualizes a sub-pattern that is used for the “Properties” tab. Therefore, the example consists of a composite user interface pattern. Possible instances of the two patterns are shown below. Concerning variability of model data and presentation (view), the specific dialog on the lower right hand side shows that displayed data and corresponding user interface controls are chosen dynamically for the object the pattern instance is assigned to.


Object Template

Overview | Properties | Associations | ...

ID: [Object ID]
Name: [Object Name]
Description: [Object Description]
Active: ✓ [Object Status]

Edit Properties Tab Template

Edit Properties
[Object Attribute]: [Object Attribute Value]
[Object ValueList Attribute]: [Object Attribute Value]
OK | Cancel

Supplier

Overview | Properties | Products | Quotes

ID: S-9010034
Name: Deans Paint Depot
Description: Deans Paint Depot Ltd.
Active: ✓

Supplier

Overview | Properties | Products | Quotes

Edit Properties
Name: Deans Paint Depot
Description: Deans Paint Depot Ltd.
Street: Franklin 14
Zip Code: 19612
City: Reading
Search Term: Paint Supplier
OK | Cancel

Figure 3. User interface pattern templates (above) and instances (below)

Especially the “Search Term” attribute is to be mentioned, as there is a distinction between text fields and list boxes regarding the data type. The lower left hand side dialog sketch has fixed visuals and data assignments, but it is variable, as it considers the actual type and number of associations an object may possess. For each association, an assignment dialog is presented that can be accessed by the dynamically instantiated tab. In the example, “Products” and “Quotes” tabs refer to the associations of the object “Supplier”.

To conclude, a user interface pattern specification has to enable the definition and distinction of all three aspects in order to provide the preconditions for effective reuse and variability. User interface patterns are meant to be adapted to different data types to be displayed. In addition, they need to be aware of the number and layout of their view components. Lastly, user interface patterns do not only need to adapt to their own variable interaction, which depends on actual view component instances, furthermore they need to define a variable control to enable the collaboration of and interaction with their child elements.

C. Classifying Dimensions for User Interface Patterns

We refer to the following dimensions to classify user interface patterns:

The **degree of formalization** distinguishes between descriptive and generative patterns [1]. The argument has been raised that a user interface pattern needs a rich human understandable specification. The latter resembles merely a description in prose and represents descriptive patterns, which cannot be processed by generators and other tools of the development environment. Thus, a machine-readable form amends the user interface pattern entity to a generative pattern [1].

The **user paradigm** reflects how the users' tasks will be supported by the entire user interface. Ludolph [4] mentions the design of object-oriented user interfaces, which enable the user to manipulate only one object in a dialog at once, as well as the procedural paradigm, which allows the user to accomplish a complete process consisting of several steps in a defined sequence. These options are complemented by the function-oriented paradigm, which provides a dialog for completing a certain step or complex task out of a process working with more than one object. The user interface patterns vary in their capability to support the three paradigms. For instance, the "Wizard" [12] is intended to build a procedural user interface. Other user interface patterns can be compiled to display the data of several business objects and form a collaboration to support the user concerning a certain function.

The **variability** of the user interface pattern can also serve as a dimension. There are patterns, which hardly feature any variability between their instances. For instance, "Breadcrumbs" [13], an "Event Calendar" [12], or a "Date Selector" [12] always feature the same abstract visuals and interactive behavior. So these patterns are called static or invariant patterns, with respect to the visual and interaction aspects. The other patterns with true variability in view, interaction and even control can be called dynamic user interface patterns.

A final dimension can be proposed with the **application area**: Firstly, user interface patterns can be interface-specific (graphical user interfaces — GUI, text-based interface or spoken dialog systems, etc.). Secondly, paradigm-specific (WIMP or touch-based interface, etc.) patterns can be differed. Thirdly, some system-specific patterns (Windows, MacOS, Android or iOS, etc.) have emerged from the appropriate GUI specification guidelines. Finally, user interface patterns can be closely associated with a certain domain (eBusiness, simulation systems, etc.). Remarkable reuse across different systems in similar use cases of a domain may be driven by a stable set of user interface patterns.

Finally, user interface patterns within the given dimension can be related to each other. For example, interface-specific user interface patterns often do have different system-specific appearances. Particularly for descriptive and generative descriptions of one pattern, we suggest that they should be made available in a linked form in future user interface pattern libraries in order to facilitate the understandability of both human and machine involved in the same development process. In this context, current (descriptive) patterns libraries also have to be checked if all containing pattern descriptions fulfill the aforementioned definition and criteria of a user interface pattern.

IV. FORMAL DESCRIPTION OF GUI PATTERNS

In the following two sections, we describe an assessment of the capabilities of pattern-based user interface development with respect to the application of current methods and notations. To accomplish this, we outline two practical examples of formalizing and utilizing patterns from general description to their application in source code. Since the state of research in generative user interface patterns mainly focuses on the WIMP paradigm, we also concentrate on that area.

GUI patterns are generative user interface patterns with an application area in WIMP software. Formal notations are necessary to implement generative GUI patterns. Since there is no generally used pattern language, independent user interface description languages are widely applied for formalizing GUI patterns (see Section II). In our prior work, we conducted an extensive investigation on formal graphical user interface specification languages and their applicability for GUI patterns. Such languages offer elements like templates (UIML) and abstract as well as concrete models (UsiXML). Both have been developed further by extensive work of research and have reached a high level of maturity. Therefore, we used UIML and UsiXML for the formalization of exemplary GUI patterns. For our analysis, we focus on the GUI pattern "Advanced Search" (Figure 1).

For a formalization of the advanced search pattern in UIML and UsiXML, at first we analyzed the components and dynamics of the pattern and found the following contents:

Advanced search view aspect:

- User interface controls: text field, dropdown list, checkbox, button
- Possible sub-pattern: „Date Selector" [12] for date data types within the objects
- Layout: four-column grid with a dynamically varying number of rows (search attribute, search criterion or value, logical conjunction, add or remove function)

Advanced search interaction aspect:

- Input parameters consisting of attributes and their values of searchable objects
- Output result: logical conjunction of search clauses

Advanced search control aspect:
- 1. Selection of search criteria from dropdown list determines input form of search value,
- 2. Click on plus button adds another search clause,
- 3. Click on minus button deletes last search clause,
- 4. Click on search button sends finished search clauses

These results can be used as specifications for a formal notation of the advanced search pattern. In UIML, a static interface part (view aspect) is described in structure tags, while changes in this part during runtime, which are triggered by user interaction, can be described in behavior tags (Figure 4). By implementing certain rules of changing structural code depending on input, the interface can be manipulated in various ways. These rules contain the condition they are triggered by and the specific action, which is performed. Through the application of parameter-driven templates, parts of structural code, and thus portions of the view aspect, can also be reused. By implementing these UIML concepts, the view and interaction aspects of an advanced search can be represented.

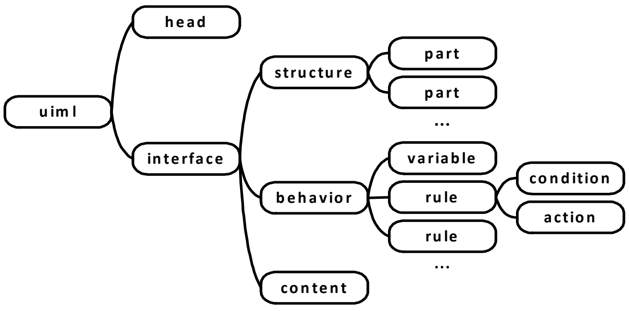


Figure 4. UIML structure for advanced search description

The UsiXML language relies on more complex and methodic specifications. Here, different kinds of models in a model-based interface development process are proposed. The most important are the following: The abstract user interface model (AUI), where a user interface can be described independently from the type of interface, paradigm, system or software (see Section III, application area of user interface patterns). Such abstract models can be concretized in the concrete user interface model (CUI), which relies on GUI description, much like UIML. Other models describe the processes of interaction with the planned interface (task model), static data and functions of it (domain model) or show connections between the different models (mapping model) [10]. For our description of an advanced search GUI pattern we focused on the CUI, where a GUI part can be differentiated into several windows with their own user interface controls and behaviors (Figure 5). However, UsiXML does not allow the use of variables or dynamic manipulations of already described window contents, like UIML does. Therefore, a complete advanced search with a potentially unlimited number of search clauses could not be implemented.

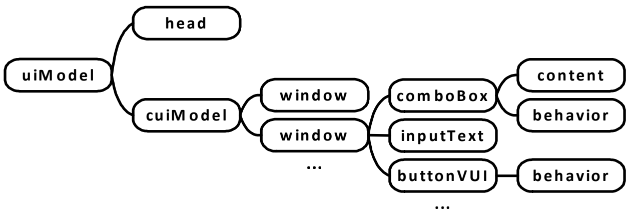


Figure 5. UsiXML structure for advanced search description

While XML is a good format for the view aspect of machine and human readable user interface patterns and therefore, in a way, generative as well as descriptive patterns, major problems of the use of interface description languages arise from the nature of patterns: Those languages are not created for the storage of incomplete, template-like interface descriptions, which are missing all concrete specifications, e.g., of user interface controls. This incomplete description often cannot be fully linked to the interaction or control aspects of the pattern. Certain limitations of the description languages, especially in UsiXML, also prevent the complete implementation of the interaction or control aspect. Furthermore, most of the languages are adapted to graphical user interfaces under the WIMP paradigm and do not allow the description of other interface types (an exception is the UsiXML AUI model). To achieve a full variability, which supports all mentioned aspects and dimensions outlined in Section III, the option to describe other interface types would be necessary. Finally, code in independent user interface description languages is built to be rendered in the user interface programming language, once the development is nearing completion. Here, several renderers for UIML and UsiXML already exist. The integration of user interface patterns into the code generation process, however, is not comparable to a rendering, since these patterns need to be instantiated first. The following subsection deals with a concept for these necessary development steps.

V. INSTANTIATION AND CODE GENERATION

Necessary for the application of existing generative user interface patterns is their procedural and technical integration into user interface development. This includes the steps of identification, selection, instantiation and integration of user interface patterns [3].

An **identification** of patterns in a planned user interface can take place at the modeling stage. The occurrence of user interface patterns can be identified in dynamic descriptions of a desired interaction process, namely in task models, or in static model components, like class diagrams. This part of pattern-based development is well-researched (see Subsection II.B for references to examples).

Also, the **selection** of patterns can be accomplished easily. Formalized and generative user interface patterns have to be stored in a pattern repository. Upon identification of patterns in a model, a list of suggestions with identified patterns should be displayed and desired patterns can be selected. Again, suggestions for pattern storage and selection have been made in the references in Subsection II.B.

Identification and selection of patterns are part of system interface modeling. Thus, the described technical solutions can be plugins or special applications for the integration of patterns in this process.

The next necessary step is the **instantiation** of patterns. Since software patterns are general descriptions, which are independent from a concrete modeling or implementation scenario, specific details are missing. For this reason, the user interface descriptions outlined in Section IV are incomplete, template-like. For example, in the advanced search pattern, as described above, the content and layout of the dropdown list a user selects attributes of the searchable objects with is missing, since these objects and their attributes vary in each specific implementation of an advanced search (searching in emails, products, pictures, etc.).

The instantiation fills these gaps in a user interface pattern with specific values. Thus, the result of an instantiated user interface pattern is a complete description of this special part of the user interface. For the use of independent interface descriptions, like UIML and UsiXML, that means a complete description and a valid XML-based document is achieved only after instantiation. An instantiated user interface pattern in UIML or UsiXML can be rendered in the final interface language. Therefore, the general process of generating source code from user interface patterns will be as depicted in Figure 6.

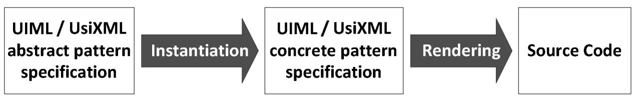

UIML / UsiXML abstract pattern specification
Instantiation
UIML / UsiXML concrete pattern specification
Rendering
Source Code

Figure 6. User interface pattern instantiantion and rendering

The next step is the **integration** of instantiated patterns in the interface implementation. Besides rendering, the main task of the integration is the establishment and application of a relation between selected patterns or patterns and other parts of the source code. The key to these relations is the existence of defined input parameters and output results for each pattern. For the advanced search pattern as an example, input parameters and output results are defined in Section IV. The output result of a user interface pattern, e.g., a set of found objects from an advanced search, can serve as an input for another pattern, in this case a search results pattern [12]. Or, input parameters and output results of interactions in user interface patterns can be used to connect the integration of patterns in components of finished source code manually.

For XML-based user interface description languages, renderers can be applied to get source code from instantiated patterns. Since for our example, advanced search, no sufficient renderer was available, we implemented XSLT scripts for the transformation of UIML and UsiXML patterns into JavaScript and HTML code.

VI. Results and Discussion

Through the exemplary formalization, instantiation and code generation of the user interface pattern “Advanced Search”, we could assess the possibilities and limitations of current methods for pattern-based user interface development. Basically, a formal description of GUI patterns is possible, and after that, they can be instantiated and transformed into source code.

The application of UsiXML shows that, while it supports abstract user interface models, it does not allow dynamic creation and manipulation of interface parts in a UsiXML document yet. Therefore, UIML is better suited to store user interface patterns in an existing XML-based interface description language.

However, user interface description languages are not exactly suitable for the storage of user interface patterns, as shown in the previous sections. They are missing options of template-like interface descriptions without layout or content specifications, so that only after instantiation, valid descriptions are established. Thus, the first of our criteria for the analysis of current pattern-based interface development methods from Section I.B, the variability and reusability of stored patterns, is not met through the use of general XML-based description languages.

A composition of user interface patterns and their integration into the source code is also possible through the steps outlined in Section V. A full composition ability of user interface patterns to form a hierarchy of GUI components, however, fails with current established methods because there is no standardized functionality of pattern storage, instantiation and code integration. Such a part of development tools could be called pattern manager and should be able to suggest, instantiate, connect and generate source code of user interface patterns, which are stored in a pattern repository. A standard exchange format of communication between patterns is also missing, since our definition of input parameters and output results is applicable, but arbitrary and not further developed. Thus, the second criterion of our objectives is also not met.

A pattern instantiation into varying interface paradigms and types as named in our last criterion is not possible with the application of GUI-specific description languages. UsiXML supports an abstract user interface model, but only as a component of a GUI description, not as a separately usable model. The degree of abstraction of the AUI is too high; it does not contain a complete interaction or communication model, so that it is not sufficient for the storage of a complete user interface pattern.

An implementation of variability, hierarchy and interaction of a composition of user interface patterns with the application of current notations is a very difficult task. Moreover, a pattern lifecycle with independent formalization, instantiation and code generation is very extensive and could be less complex.

In our conclusion we will use the developed criteria and found shortcomings of current pattern-based interface development to define first specifications of improved methods and notations.

VII. Conclusion and Future Work

In this paper, we have shown that current solutions for pattern-based user interface development do not meet the criteria of a complete and efficient design method for any kind of user interface.

Based on the results of our practical formalization, instantiation and code generation of the advanced search user interface pattern, we propose the following specifications of a sufficient pattern description and development integration method:

- An exact **definition** of generative user interface patterns: inclusion and exclusion criteria, characteristics, adaptability and composition ability should be established to describe them as artifacts in the development process. We proposed a first version in Section III.
- A **metamodel** for the structure and behavior of user interface patterns, which reflects the defined aspects. It would serve as a guideline for a standardized pattern implementation, as well as a method for traceability of certain aspects between different phases of the interface development.
- A specialized **pattern language**, which allows an exact and formal representation of patterns according to the definition and their metamodel.

And, as a practical addition:

- A development tool or module for user interface development, which contains the **pattern repository** and the **pattern manager** and offers the functionality described in Section VI. Here, it should be resorted to implementation and storage standards to assure the availability of such a tool in different development environments. The full lifecycle of user interface patterns, from their creation to their application and further development, should be supported.

Based upon these specifications, a practical solution can be approached. In our further research, we plan to concentrate on proposing a metamodel for generative user interface patterns as well as a first draft for a special description language for user interface patterns.

REFERENCES

[1] J. Vanderdonckt and F.M. Simarro, “Generative pattern-based Design of User Interfaces,” Proc. 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems (PEICS '10), ACM, June 2012, pp. 12-19, doi: 10.1145/1824749.1824753.

[2] R. Beale and B. Bordbar, “Pattern Tool Support to Guide Interface Design,” Human-Computer Interaction (INTERACT 2011), LNCS Vol. 6947, 2011, pp. 359-375.

[3] F. Radeke, P. Forbrig, A. Seffah, and D. Sinnig, “PIM Tool: Support for Pattern-driven and Model-based UI development,” Proc. the 5th International Conference on Task Models and Diagrams for Users Interface Design (TAMODIA'06), LNCS Vol. 4385, 2006, pp. 82-96.

[4] M. Ludolph, “Model-based User Interface Design: Successive Transformations of a Task/Object Model,” in User Interface Design: Bridging the Gap from User Requirements to Design, CRC Press, Boca Raton, Ed.: L.E. Wood, 1998, pp. 81-108.

[5] P. Chlebek, “User Interface-orientierte Softwarearchitektur,” Mainz: Vieweg, 2006.

[6] Mozilla Developer Network, “XUL,” https://developer.mozilla.org/en/XUL 25.06.2012.

[7] Microsoft, “XAML in WPF,” http://msdn.microsoft.com/en-us/library/ms747122.aspx 25.06.2012.

[8] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster, “UIML: An Appliance-Independent XML User Interface Language,” Proc. Eighth International World Wide Web Conference (WWW’8), Elsevier Science Pub., May 1999.

[9] UIML 4.0 specification, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uiml 10.05.2012.

[10] J. Vanderdonckt, Q. Limbourg, B. Michotte, L. Bouillon, D. Trevisan, and M. Florins, “UsiXML: a User Interface Description Language for Specifying multimodal User Interfaces,” Proc. W3C Workshop on Multimodal Interaction (WMI'2004), 19-20 July 2004.

[11] S. Wendler, D. Ammon, T. Kikova, and I. Philippow, “Development of Graphical User Interfaces based on User Interface Patterns,” Proc. PATTERNS 2012, 22-27 July 2012.

[12] M. van Welie, “A pattern library for interaction design,” http://www.welie.com 10.05.2012.

[13] Open UI Pattern Library, http://www.patternry.com 10.05.2012.

[14] A. Toxboe, “User Interface Design Pattern Library,” http://www.ui-patterns.com 10.05.2012.

[15] J. Engel, C. Herdin, and C. Maertin, “Exploiting HCI Pattern Collections for User Interface Generation,” Proc. PATTERNS 2012, 22-27 July 2012.

[16] A. Wolff, P. Forbrig, and D. Reichart, “Tool Support for Model-Based Generation of Advanced User Interfaces,” Proc. MoDELS'05 Workshop on Model Driven Development of Advanced User Interfaces, Montego Bay, Jamaica, 2 October 2005.

[17] A. Wolff, P. Forbrig, A. Dittmar, and D. Reichart, “Tool Support for an Evolutionary Design Process using Patterns,” Proc. Workshop on Multi-channel Adaptive Context-sensitive (MAC) Systems: Building Links Between Research Communities, Glasgow, 15 May 2006.

[18] M. Wurdel, P. Forbrig, T. Radhakrishnan, and D. Sinnig, “Patterns for Task- and Dialog-Modeling,” J.A. Jacko (ed.) HCI International 2007, Beijing, 22-27 July 2007, pp. 1226-1235.

[19] D. Reichart, A. Dittmar, P. Forbrig, and M. Wurdel, “Tool Support for Representing Task Models, Dialog Models and User-Interface Specifications,” Interactive Systems, Design, Specification, and Verification (DSVIS'2008), LNCS Vol. 4323, 2008, pp. 92-95.

[20] A. Wolff and P. Forbrig, “Deriving User Interfaces from Task Models,” Proc. the 4th International Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI 2009). Sanibel Island, USA, 8 February 2009.

[21] S. Fincher, “PLML: Pattern Language Markup Language,” http://www.cs.kent.ac.uk/people/staff/saf/patterns/plml.html 25.06.2012

[22] F. Radeke and P. Forbrig, “Patterns in Task-based Modeling of User Interfaces,” M. Winckler, H. Johnson, P. Palanque (Eds.): Proc. 6th International Workshop on Task Models and Diagrams for User Interface Design (TAMODIA'07), Toulouse, France, 7-9 November 2007.

[23] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stahl, A System of Patterns, New York: Wiley, 1996.

Lowering Visual Clutter of Clusters in Component Diagrams

Lukas Holy, Jaroslav Snajberk, and Premek Brada
Department of Computer Science and Engineering, Faculty of Applied Sciences,
University of West Bohemia, Pilsen, Czech Republic
{lholy, snajberk, brada}@kiv.zcu.cz

***Abstract*—Nowadays, component applications can easily consist of hundreds or thousands of components and it is thus difficult to understand their structure. Diagram visualization does not help much because of visual clutter caused by big amount of elements and connections. This paper describes an approach of removing a large part of connections from the diagram while preserving the information about component interconnections. It also describes a viewport technique for showing all information about interfaces for selected group of components right in the diagram area. After that it presents novel integration of above mentioned techniques which maps a group of components to the content of a viewport. These techniques are among other benefits useful in the reverse engineering process. The main idea of this technique can be used in a similar way to reduce the clutter in the node-link graphs. To show the effect of this technique, example reduction of lines is discussed. So the better understanding of a diagram is also shown on preliminary results.**

Keywords*-*software visualization; component; visual clutter.

I. Introduction

Software applications become more and more complex and although there are lots of tools, which help the development process, they are still limited in helping human understanding of the application structure. Software components are one of the ways to handle this complexity as they encapsulate parts of functionality to unified components. Even with the usage of components, applications can easily consist of hundreds or thousands of them. It is therefore difficult to explore the structure of the application and create a mental model of the whole system.

One of the ways how to get an insight into a component application structure can be UML (Unified Modeling Language) component diagram. When the diagram becomes large there are many problems with exploring it. One is the contradictory need of providing enough details and showing the complete diagram (application structure) at the same time. Another question is how to reduce visual clutter caused by the large number of elements and connections between them. This visual clutter makes tracing of dependencies difficult and hinders orientation in the diagram. Current tools do not offer features designed for work with such large diagrams, as we have shown in our previous paper [1]. In this work, we present several techniques to reduce visual clutter in UML component diagrams and help user to form clusters of components.

A. Structure of the Paper

In the following section, a problem of visual clutter is defined first. After that, a related work is described in Section III. Then, in Sections IV and V a SeCo (Separated Components) technique and its implementation is presented. This technique helps to reduce the visual clutter in large graphs. Also, another technique called viewport is shown in Section VI ,which helps to form component clusters. After that, the novel integration of SeCo and viewport techniques is proposed in Section VII. Finally, our contribution is discussed in Section VIII and summarized in Section IX.

II. Problem Definition

Developers face multiple challenges in large diagrams visualization such as difficult orientation, limited amount of visible elements on the screen while showing its details, insufficient details when showing overview or the visual clutter [2].

This paper focuses on the problem with highly connected components and the clutter caused by their connection visualization as well as the component clusters visualization.

Very often, only a small amount of components is connected to a large number of other components. Such components are often, among developers, informally called “God Objects”. It is difficult to trace the connections in the surrounding area of these objects. Another problem in visualization is forming and working with clusters of components, which usually represent one feature or logical unit of the system. These problems cause exhausting space, which is one of the essential resources in the visualization and can be used for easing the work with large component diagrams.

III. Related Work

Visual clutter can be reduced by many techniques. The whole taxonomy of these techniques has been described by Ellis and Dix in [3]. A short description of those techniques related to our work is provided.

The clutter caused by the lines is often reduced by edge bundling [4]. Although this approach reduces the clutter, it can be difficult to trace the dependencies between connected nodes leading through the edge bundles.

The visual clutter can be also lowered by using node clustering, where one cluster usually represents multiple nodes. The overview of clustering algorithms can be found in [5].

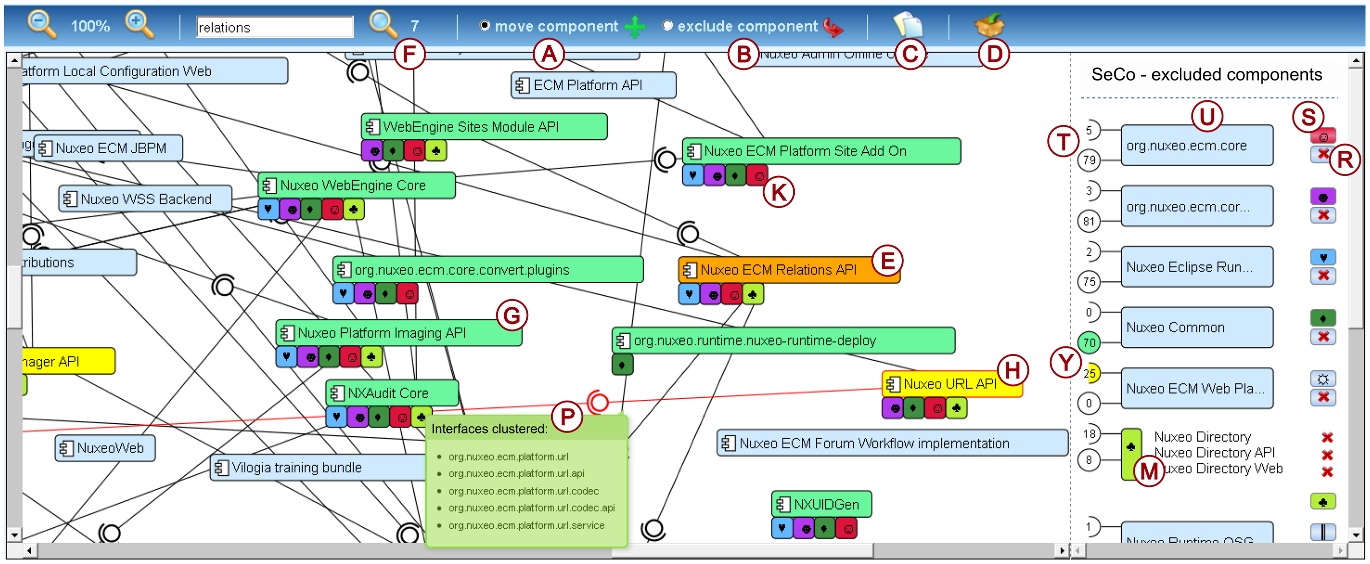


Fig. 1. Complex Component Application Explorer tool demonstration

Another influencing factor is the chosen layout algorithm, which can ease orientation in both clustered graphs [6] and a non-clustered ones [7]. In the following section, our visual clutter reduction approach is described.

IV. The Separated Components Area Technique

The technique proposed in [8] reduces the visual clutter by removing the components with a large number of connections from the main diagram into a so called *separated components area* (abbreviated to SeCo) placed on the border of a window (right sidebar in Figure 1).

When a user moves components from the main diagram to this area, the lines between these components and remaining components are elided. SeCo consists of a list of items. Each item consists of clustered interfaces (indicated with mark (T) in Figure 1), components (U) and one corresponding symbol (S). Interfaces are clustered into two sets (T): all provided interfaces (displayed as “lollipops”) and all required interfaces (displayed as “sockets”). Numbers inside the clustered interfaces represent the number of elements clustered in the given symbol.

The purpose of symbols is to create clear and easily recognizable key, which uniquely identifies one item within SeCo. Then, these symbols can be used in the diagram area to represent connection between a given component and the corresponding item placed in SeCo. They are shown as small rectangles neighboring the displayed components (K) and containing the symbol, which corresponds to the connected item (S).

It is possible that a particular functionality of the system is implemented by several components. When this functionality is used by a large number of other components in the system, it is beneficial to represent them as a group in SeCo (M).

V. The Separated Components Area Technique’s Implementation

SeCo technique implamentation is called CoCA-Ex (Complex Component Applications Explorer). CoCA-Ex works on the ComAV platform (Component Application Visualizer) [9], so it could use ComAV’s reverse-engineering and management features. ComAV can automatically reverse-engineer the whole component-based application of a supported component model. Further component models can be easily added using an extension mechanism offered by RCP (Rich Client Platform). ComAV is also able to add other visualization styles. It means that CoCA-Ex is only one way how structures of applications can be visualized on ComAV platform.

The CoCA-Ex tool can be used via desktop application interface or web interface. In a desktop interface version, structure of all analyzed applications is saved for future visualization. Such structure is handled as a project by the ComAV – it is shown in a project view with other projects (structures), it can be renamed, deleted or updated. Such project oriented approach is known from Eclipse IDE (Integrated Development Environment). In a web interface version the user starts the visualization process by picking desired components from the local machine and uploading them to the server. The ComAV platform creates the model of the application and the CoCA-Ex shows the application diagram in the web page. The demonstration of the CoCA-Ex’s interface is shown in Figure 1.

CoCA-Ex use servlets from the JEE technology, as the back-end technology. Servlets are used mainly because of the Java implementation of the ComAV tool. HTML5, JavaScript, jQuery framework and CSS3 (Cascading Style Sheets) were used for the front-end. Canvas and SVG (Scalable Vector Graphics) elements from HTML5 (Hypertext Markup Language) are used to represent the nodes of the diagram. Although the HTML5 technology is still not fully supported by all main browsers, its current state is sufficient for CoCA-Ex purposes. Also desired features such as SVG support or Canvas are likely to be stable in the near future.

The tool provides standard features such as panning and zooming. There are two modes of manipulating the components with appropriate icons in the toolbar. First mode is for moving components (A) where the user can manually adjust the layout of the diagram. Second mode (B) serves for removing components from the diagram area to the SeCo area simply by clicking on the desired components, which should be removed. Last two icons in the toolbar serve for the automatic removal of a configured amount of components from the diagram to the SeCo area. The tool is currently configured to remove 15% of most connected components. The icon (C) is used for removing these components and adding them to SeCo area as individual items. The next icon (D) creates one group for all of them.

CoCA-Ex offers a fulltext search in components' names. In Figure 2, one can see the search for a word "relations". Seven components in the diagram contain this word as indicated by the number seven (F). Matching components are highlighted by orange color (E).

If one clicks on the provided interfaces of a component in SeCo, these interfaces and connected components become highlighted by green color. An example is shown on dependency between the *Nuxeo Common* component's provided interfaces (Y) and *Nuxeo Platform Imaging API* component (G). Similarly, for interfaces required by components in SeCo highlighting by yellow color is used. It is demonstrated on dependency between *Nuxeo URL API* component (H) and *Nuxeo ECM Web Platform UI* component's required interfaces (Y).

For several components from the SeCo area (those with symbols' background highlighted by different colors (S)) there are delegates shown in the diagram area, e.g., (K). For inspecting interfaces, the tool offers highlighting of a connection by a red color and showing the interfaces involved in the connection (P), as shown in the green tooltip. Each individual component shown in SeCo has its own button (R) to remove it back to its original position in the diagram area.

VI. Viewport for Component Diagrams

The viewport technique shows the diagram zoomed-out to provide the appropriate overview of the complete architecture, with elements displayed without details. Besides that it shows selected components in detail inside a *viewport area* plus all their relations with other components in the diagram in an interactive border area (see gray area marked with (11.) in Figure 2). These relations are clustered into two sets for each component: all provided interfaces (displayed as "lollipops") and all required interfaces (displayed as "sockets").

These interfaces are then connected to clustered proxy components, visually represented as rectangles with rounded corners. Each rectangle represents one or more components. Numbers inside the clustered interfaces and proxy components represent the number of elements clustered in a given symbol.

VII. Viewport for Groups of Components

This section presents the novel integration of viewport and SeCo technique. For showing groups of components (as described in Section IV) in the diagram area a viewport can be used. A group of components shown in the SeCo can be moved to the diagram area and shown as a viewport. Similarly the viewport and its content can be moved from the diagram area to the SeCo.

According to level of details of a viewport, it is possible to show:

1) a viewport as a symbol belonging to a group only,
2) a viewport with all details for all components and their relations in given group.

These possibilities are described in following sections.

A. Viewport with Details

For moving a group from SeCo to the diagram area there is an icon (indicated with mark (1.) in Figure 2). After a user clicks on this icon the group will disappear from the SeCo and will be shown in the diagram area as a viewport.

Each viewport has its small toolbar, which contains a symbol representing a group (2.) and icons for important actions. The symbol has similar meaning as symbols used in SeCo. In Figure 2 there is the icon for canceling the viewport (4.). It releases the components from the viewport to the diagram area and deletes the viewport itself. Also there is the icon (3.) for moving a whole viewport to SeCo, which removes the viewport from diagram area and shows its contents in the SeCo as a group. Finally one can see icon (6.) for minimizing viewport to be represented as a icon only, which is described in following section.

B. Viewport as a Symbol

One of the viewport's important features is its ability to be collapsed into an symbol (7.). It is very important part of visible elements reduction process as well as visual clutter reduction. Viewport symbol represents the whole viewport and its content. It means that components included in the viewport are not visible at the time the viewport is collapsed into the symbol. When a user hovers a mouse over the this symbol a small toolbar appears. There are icons for following actions:

- showing viewport in full details (8.), which shows viewport in a way described in previous section,
- moving viewport from diagram area into a SeCo (5.), which creates a group in SeCo from components contained in viewport,
- releasing components from given viewport and removing the viewport itself (9.).

VIII. Discussion and Examples

In a lot of situations one can use the SeCo features to form groups of components. These groups can serve as named categories according to, which the user can classify the rest of the components in the diagram area and thus form a logical units of an investigated system.

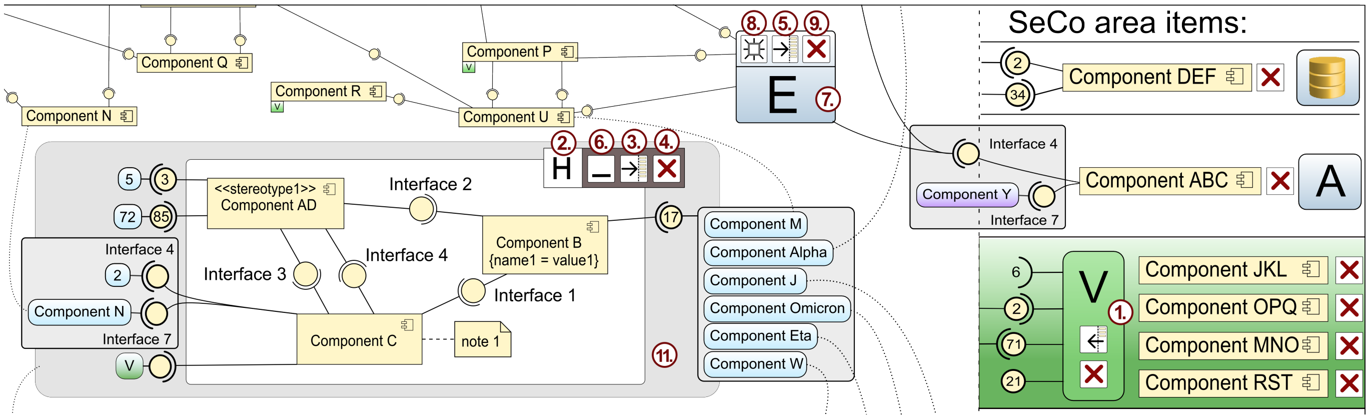


Fig. 2. Viewport with SeCo

A viewport gives an alternative to form groups in the diagram area. Its benefit is also the ability to be moved to the place in the diagram where a user forms the group. It should enable to explore and understand the dependencies in large diagrams by showing the context of a selected diagram subset. The proxy elements should reduce the need for the disorienting pan&zoom otherwise necessary while exploring dependencies and provide user relevant information in one place. The viewport is placed on a given position in the diagram, thus there can be more viewports in a diagram. At the moment when a group inside a viewport is not important, it can be collapsed into a viewport symbol. It gives a user a possibility of showing several groups and still have enough space in diagram area to work with rest of the components.

Several experiments using the proposed technique were performed. In one of them only 7 Nuxeo components have been removed from the diagram area into SeCo leading to 241 of 698 interface connection lines remaining in the graph. Therefore, the graph was reduced of about 65% of lines.

It shows that by using the proposed technique, significant visual clutter reduction may be achieved.

IX. Conclusion and Future Work

In this paper, an advanced technique was described. This technique helps to reduce the amount of lines in UML component diagram of large applications, by removing the selected components from the diagram area. It uses SeCo where the selected components are shown, and symbolic delegates, which represent the connections instead of lines. A viewport technique was also described. This technique is used for showing all the information about interfaces for selected group of components right in the diagram area. The novel integration of above mentioned techniques was proposed. These techniques maps a group of components to the content of a viewport. Viewport symbols for graphical representation of groups were also described. These symbols saves a space in the diagram area. Appropriate interactions were proposed for all these techniques.

These techniques are, among other benefits, useful in the reverse engineering process when the user is interactively getting familiar with the whole diagram. It helps with creating the mental model of the application by easing the process of clusters creation. Which is the reason why these techniques will be part of a ComAV platform, that already supports reverse-engineering of applications of various component models.

Preliminary evaluation shows that the presented ideas are helpful in large graph visualization, where one suffers from visual clutter caused by the large number of connection lines.

Implementation of viewport technique is scheduled for integration into CoCA-Ex application to enable users to form relevant clusters comfortably and validate the ideas on concrete tasks. We also plan to evaluate above mentioned ideas by users or case study.

Acknowledgment

The work was supported by the UWB grant SGS-2010-028 Advanced Computer and Information Systems. Authors would like to thank Jindra Pavlikova for the work on the implementation.

References

[1] L. Holy, J. Snajberk, and P. Brada, “Evaluating component architecture visualization tools - criteria and case study,” 2012.
[2] R. Rosenholtz, Y. Li, and L. Nakano, “Measuring visual clutter,” *Journal of Vision*, vol. 7, no. 2, August 2007.
[3] G. Ellis and A. Dix, “A taxonomy of clutter reduction for information visualisation,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, no. 6, pp. 1216 –1223, nov.-dec. 2007.
[4] D. Holten, “Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 741–748, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/TVCG.2006.147
[5] S. Schaeffer, “Graph clustering,” *Computer Science Review*, vol. 1, no. 1, pp. 27–64, 2007.
[6] Q. Feng, “Algorithms for drawing clustered graphs,” 1997.
[7] S. Hachul and M. Jnger, “Large-graph layout algorithms at work: An experimental study,” http://jgaa.info/ vol. 11, no. 2, pp. 345369, 2007.
[8] L. Holý, K. Ježek, J. Snajberk, and P. Brada, “Lowering visual clutter in large component diagrams,” in *16th International Conference Information Visualisation*, 2012.
[9] J. Snajberk, L. Holy, and P. Brada, “Comav - a component application visualisation tool,” in *Proceedings of International Conference on Information Visualization Theory and Applications*. SciTePress, 2012.

A Framework for Characterizing Usability Requirements Elicitation and Analysis Methodologies (UREAM)

Jos J.M. Trienekens
IE&IS
TUE
Eindhoven, The Netherlands
j.j.m.trienekens@tue.nl

Rob J. Kusters
Management Sciences
Open University
Heerlen, The Netherlands
rob.kusters@ou.nl

***Abstract*—Dedicated methodologies for the elicitation and analysis of usability requirements have been proposed in literature, usually developed by usability experts. The usability of these approaches by non-expert software engineers is not obvious. In this paper, the objective is to support developers and managers in a software development project in deciding on which methodology to select, taking into account local strengths and weaknesses. We define a framework based on a set of criteria that allow for the comparison of methodologies.**

Keywords-usability; usability requirements.

I. Introduction

In the development of interactive systems, usability is increasingly considered to be a crucial factor for the success of a software system [13]. However, identifying and specifying usability requirements are not trivial tasks. It is even further complicated by the existence of multiple, different definitions of usability. Multiple approaches have been proposed on how to elicit and analyze usability requirements. Therefore, a need arises to compare the available methodologies in order to make a well-founded decision about which can be used in a project, based on the specific characteristics of the project. In this paper, we present a structured comparison of usability elicitation and analysis approaches that is designed to help the stakeholders of a project, e.g., project coordinators, managers, and developers, decide on a methodology to use for usability requirements elicitation and analysis. We define a framework for extracting specific properties of a methodology so as to allow for a direct comparison of different approaches presented in literature. The selected methodologies represent a selection of what we believe are the most important approaches to usability requirements elicitation and analysis.

In Section 2, we give definitions of terms required to compare usability requirements elicitation and analysis approaches. Section 3 describes the aforementioned framework, and, in sections 4 to 7, this framework is applied to each methodology. Section 8 gives a comparison of the results obtained for each of the methodologies and section 9 concludes with an overview of the most relevant findings from this comparison.

II. Definitions

The following section gives definitions for the most relevant terms used throughout this paper:

- Usability: the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use [7].
- UREAM: usability requirements elicitation and analysis methodology.
- Methodology: a coherent and structured set of procedures to carry out usability requirements elicitation and analysis in a step-wise and well-defined way.
- Method: a coherent set of steps in a methodology is defined as a method.
- Technique: a systematic way to carry out a particular procedure (for example: a survey and a questionnaire are techniques for a method that focuses on an analysis of user tasks.
- HCI: Human-Computer Interaction is a research area that studies of how people interact with computers and to what extent computers are or are not developed for successful interaction with human beings.

III. Towards a Framework for UREAM comparison

As it is stated in the introduction, there are many different methodologies to elicit and analyze usability requirements. In order to support the developers or managers to compare the methodologies and to provide them with the criteria needed to select one to deploy, we propose the following framework to compare the different methodologies. The framework consists initially of three steps. First, each methodology will be decomposed into methods. Methods are coherent elements of a methodology. They describe a single function resulting in a sub-deliverable of the methodology. Examples are ‘pre-study’, ‘user profiling’, ‘task analysis’ and ‘usability specification’. In second step each method will be assessed using a set of criteria. Finally, the results of the assessment of each method will be combined to obtain the result for a methodology. This combination can be done in several ways according to the type of criterion. Some, such as required effort, can be added across methods but if, e.g., a single method requires the availability of an HCI expert, this requirement translates to the methodology as a whole.

The reason we decompose the methodology into methods is to achieve a more accurate and concrete comparison. The methods employed are as will be shown easier to identify, describe and therefore easier to assess while the methodology as a whole will tend to be a fairly complex amalgamation of these constituent parts, which makes direct characterization of this combination much more difficult and dubious. Using the criteria to assess the methods first and then combining the result for each methodology will focus each discussion on a manageable level, thus helping developers to understand what the differences are and why there is a difference between the methodologies. Reasons may be, among others, that different methods or different techniques are used. Methodologies containing different methods will have different properties and therefore different results, but methodologies that include the same methods might also have different properties because the methods use different techniques.

The proposed framework consists of a set of criteria that can be used to assess the methodologies. This set of criteria is divided into four categories, namely the external factors (Section 3.1), the characteristics of the methodology (Section 3.2), the effort (Section 3.3) and the quality and effectiveness (Section 3.4). In the following, a short description of each criterion is given, and the arguments for selecting the criteria are described. Moreover, it is explained how the score is calculated and how the scores are combined for a particular methodology.

A. External factors

The first category concerns external factors. Information about the requirements of a methodology about the external environment in which it is to function is crucial for developers and managers to decide whether or not to apply this methodology in a particular context (also mentioned by Davis [8] as a first step of choosing a strategy for requirement elicitation). The external factors category consists of three criteria:

C1.1 Does a methodology / method need a human computer interaction (HCI) expert?

This criterion answers whether an HCI expert is needed to do this method or methodology properly. It is included in the framework because there are projects that do not allow for the involvement of an HCI-expert, e.g., due to budget reasons or a lack of qualified personnel. This criterion is mentioned in all four assessed methodologies [1], [2], [4], [3]. Each method and the methodology can be given either a plus or a minus for this criterion. A score of ‘+’ indicates that the method/methodology needs an HCI-expert and a score of ‘-‘ indicates that it does not need one. If one of the methods needs an expert then the methodology needs an expert as well.

C1.2 Does a methodology / method need access to representative users?

This criterion indicates whether the methodology / method requires access to representative users. Involving users in the project increases the dependencies on external factors. Having access to the representative users and working with them is not a simple task. This property is also mentioned in all four assessed methodologies. Each method gets a plus or minus for this criterion to indicate whether or not it involves representative users. If the methodology does not involve users, a score of ‘-‘ is assigned. If there is some user involvement in a methodology, ‘+’ is the result. A methodology that very strongly relies on user involvement gets a score of ‘++’.

C1.3 Does a methodology / method work with non-experienced users?

Some methods/methodologies require a certain level of knowledge or experience of the users to ensure an efficient communication and collaboration with them [8]. Inexperienced users might have difficulties with articulating their requirements [9], [10]. If this criterion is applicable for the method, then a score between 1 and 5 is assigned. If a method does not involve the user, this criterion is not applicable. For the methodology, a combined score on the same scale is calculated. However, this is not necessarily the arithmetic mean of the results for the methods because some methods may have greater influence on the overall score than others.

B. Characteristics

The second category focuses on the characteristics of the methodology. The characteristics provide the developers with insight whether a methodology is appropriate.

C2.1 Does a methodology / method give strict guidance to help the developers to carry it out?

The methods of the given methodology are assessed on how accurately they are described. Or in other words, whether a non-experienced developer can execute it well based on the description. A scale of ‘- -‘ to ‘++’ is the range of the evaluation for this criterion. A combined score for the methodology (also‘- -‘ to ‘++’) is assigned.

C2.2 Does a methodology / method take the user feedback into account for further improvement?

It is very important to take the user feedback into account for further improvement with respect to usability of the system design [11]. A score of ‘+’ or ‘-‘ is assigned to indicate whether feedback from users is taken into account. We consider user feedback as the input from the user that is based on a proposal made to the user or a prototype presented to them. If the methodology contains a certain number of methods which take the user feedback for further improvement, then it is argued that the methodology will also get a plus for this criterion.

C. Maintaining the Integrity of the Specifications

The third category is the effort, i.e., the time and the cost that is needed for a methodology. This helps the developer to make tradeoffs.

C3.1 Is a methodology / method time consuming?

This criterion indicates how time-consuming the methods are. A score between ‘- -‘ and ‘++’ is the result of this criterion applied on the methods. A score of ‘++’ indicates the method is very time consuming, while a ‘- -‘ indicates that executing the method can be completed in a very short time. A cumulative score of each method is assigned to the methodology. If a project has a time constraint within which it needs to be finished, the cumulative score will help the developers to decide on a methodology.

C3.2 Is a methodology / method common in the software development process?

Time consumption is not an absolute value. It is also related to the degree of integration in common software engineering methods. Integration means less additional work and also will promote more experience with the approach among software engineers, impacting positively on the amount of effort required. The methods that are used in the elicitation and analysis process of the usability requirements might already be included or commonly used in the development process of the product for other reasons. Then the methods might be easily adapted such that it would not take any additional time. A list of commonly used functional requirements elicitation techniques indicate the answer to this criterion [12]. A value between '–' and '+' is assigned to each method to assess whether the method is common or not for software development processes. Of course, this provides only a guideline. Actual fit with a local process will still need to be determined when actually adopting an approach.

D. Quality and effectiveness

The last category is the quality and effectiveness of each method and the methodology. This will also help the developers to make the trade-offs. The objective of this set of criteria is to indicate the level of detail that is elicited.

C4.1 Does a methodology / method elicit enough information to help the developer specify the fit criterion?

Because it is hard to measure the non-functional requirements, eliciting information to specify the fit criterion of the usability requirement might be a crucial factor for selecting a certain methodology [12]. Juristo et al. argued that some proposed methodologies in the literature did not derive enough information to help the developers design and implement the elicited requirements [4]. The methods get a '-', a neutral or '+' for this criterion depending on whether they do not elicit enough information, it depends or it does (explicitly) elicit enough information, respectively. An average within the same range is given to the corresponding methodology.

C4.2 Does a methodology / method elicit the dependencies between the usability requirements and other functional and non-functional requirements?

Usability requirements are sometimes related to specific functional requirements [12]. Knowing the interdependencies between requirements is important for the system design and change management. Therefore, this criterion can be an important factor when selecting a methodology. A scale including '+', neutral, and '-' is used to indicate that dependencies are completely, partially, or not elicited, respectively. The proposed framework is applied to four selected methodologies of respectively Nielsen [1], Carlshamre et al. [2], Seffah et al. [3] and Juristo et al. [4].

IV. METHODOLOGY 1: THE USABILITY ENGINEERING LIFECYCLE (EUL)

The methodology, Usability Engineering Lifecycle (UEL) [1], was proposed in 1992 as one of the first approaches to usability engineering. It presents a practical usability engineering process that can be incorporated into the product development process. This methodology provides a very comprehensive set of methods that can be applied to elicit and analyze the usability requirements. Some of the other methodologies select a subset of the methods that are presented in this methodology. Therefore, this methodology is chosen to be assessed first and the framework is applied.

Ten methods are included in this methodology. Each method will be described shortly.

1. "Know the user". This is used to analyze the individual user's characteristics (e.g., work experience, knowledge level, work environment and social context), the user's current task (e.g., the overall goals, how they approach the task, the needed information, the way of dealing with exceptional circumstances or emergencies), to do functional analysis (e.g., the underlying functional reason for the task) and to have the evolution of the user (e.g., an educated guess about future users and uses).
2. "Doing competitive analysis". This analyzes the existing products heuristically according to established usability guide lines (e.g., usability goals and levels) and performs empirical user tests with these products.
3. "Setting usability goals". This is specified according to the five main usability characteristics (i.e., learnability, efficiency, ability of infrequent users to return to the system without having to learn it all over, frequency and seriousness of user errors and user satisfaction).
4. "Participatory design". This involves users in the design process through regular meetings to help the designer by asking questions and reacting to the designs that they do not like.
5. "Coordinated design of the total interface". This is to achieve consistency of the total interface. This can be approached by using interface standards and the product identity statement (a high-level description of what kind of product it is).
6. "Doing guidelines and heuristic analysis". A list of well-known principles of guidelines for user interface design should be followed. And a heuristic evaluation can be performed on the basis of the guidelines. Prototyping and empirical testing should be combined into iterative design to capture the design rationale, analyze the trade-offs, make the right decision and evolve the design. This combination will be considered as a method.
7. "Prototyping". This is commonly known and often deployed in software engineering.
8. "Empirical testing".
9. "Collect feedback from field use". This method is similar to empirical testing.

Each method is first analyzed separately. The result of applying the framework for all methods and methodologies can be found in Table 1.

TABLE I. RESULTS PER METHOD AND METHODOLOGY

Methods / Methodologies	**External factors**			**Characteristics**		**Effort**		**Qual. and Effectiv.**	
	C1.1 HCI expert	***C1.2 User access***	***C1.3 Non experts***	***C2.1 Strict guidance***	***C2.2 User feedback***	***C3.1 Effort***	***C3.2 Common in SRM***	***C4.1 Info for fit***	***C4.2 Depend-dencies***
Know the user	-	+	5	+	-	0	+	0	-
Competitive analysis	-	+	5	0	+	0	-	0	-
Setting usability goals	-	-	n/a	+	+	0	-	+	-
Participatory design	-	++	3	+	+	-	-	0	-
Coordinated design	-	-	n/a	-	-	+	0	+	-
Guidelines and heuristic analysis	+	-	n/a	0	-	+	-	-	0
Prototyping	-	-	n/a	-	0	0/++	+	-	+
Empirical testing	-	++	5	+	+	+/++	0	+	+
Collect feedback from field use	-	++	5	0	+	+	0	+	+
Total for UEL	+	+/++	5	0	+	++	-	+	+
Pre-study	-	-	n/a	-	-	-	+	-	-
User profiling	-	+	4	+	-	-	+	-	-
Task analysis	-	+	2	+	+	++	0	-	-
Usability specification	-	-	n/a	+	-	-	+	+	-
Prototype and usability testing	-	+	2	-	+	++	-	+	-
Total for Delta	-	+	3	+	+	-	+	+	-
System summary form	-	+	4	+	-	+	+	-	-
Compile system summary form	+	-	n/a	-	-	+	-	-	-
Context of use portfolio	+	-	n/a	--	-	+	-	-	-
Frs portfolio	-	-	n/a	--	-	+/-	+	-	+
Review and validate integrated picture	+	+	3	--	+	?	-	-	-
Total for ACUDUC	+	+	4	--	+	+	+	-	+
Apply the patterns	-	-	n/a	++	-	0	-	+	-
IFR table	-	+	2	++	+	+	-	+	++
Total for GEUF	-	+	2	++	0	0	-	+	+

Combining the results from the individual methods allows us to judge the methodology as a whole. In order to deploy this methodology, the following criteria for the external factors have to be fulfilled: The developers need to have access to an HCI expert to do the guidelines and heuristic analysis properly (C1.1: +). The methodology needs frequent and reliable access to the representative users in order to perform some of the methods (C1.2: +/++), but it does not require the users to be experienced (C1.3: 5). The methodology does not give very strict guidance to help the developers (C2.1: 0). It suggests a set of techniques to do some of the methods. And the methodology includes methods such as participatory design and empirical testing to elicit the user feedback and take it into account to improve the usability (or the specification of requirements) (C2.2: +). The effort that needs to be put into the methodology is high. Because of the comprehensive set of methods, the methodology is very time consuming. And only a small part of methods are a part of the regular software engineering process (C3.2: -). The rest needs to be added (C3.1: ++). But, the quality of the methodology is fair. It gives enough information about the quantities of usability requirement to specify the fit criterion and it gives an indication about the dependencies between the requirements (C4.1, C4.2: +).

V. METHODOLOGY 2: THE DELTA METHOD

The Delta method [2] is a task-based and usability-oriented approach to requirement engineering. This method was applied in a project to improve the overall usability of the systems delivered. The results prove that the delta method rendered usable systems and helped in eliciting functional requirements in a natural way. This methodology derives its method from the usability definitions in ISO 25062 [7]. Each method corresponds to the users, goals and context of use in the usability definition. This methodology consists of five methods.

1. "Pre-study". Here the scope of the prospective system, the customer categories, and the fundamental services of the system are identified.
2. "User profiling". This provides an overview of the prospective users by means of questionnaire.
3. "Task Analysis". This captures the work tasks of the users through interviews.
4. "Usability specification". This defines an agreed level of usability that the system should supply.
5. "Prototype and usability testing". This tests the results of the last method.

The results for all methods can be found in Table 1. Combining the results from the individual methods delivers the following results. This methodology does not involve usability experts in any of its methods (C1.1: -). Representative users are accessed in most of the methods and this methodology works well with users of moderate level of experience (C1.2: +; C1.3: 3). Guidance is provided by means of questionnaire, activity graph and usability levels (C2.1: +), but most of the methods do not give quantitative results and users feedback is considered only in prototype testing method (C4.1: +). This methodology proves not to be very time consuming (C3.1: -).

VI. Methodology 3: Approach Centered on Usability and Driven by Use Cases (ACUDUC)

Seffah, Djouab and Antunes [3] present a method that combines usability-centered requirements engineering processes with those based on use cases. They compare similar approaches in both processes and define their own method, called ACUDUC, Approach Centered on Usability and Driven by Use Cases. This is based on the Unified Software development Process [5] as a representative of use-case-driven software engineering methodologies and the RESPECT framework (Requirements Specification in Telematics) [6], a user-centered requirements process. They use the definitions of usability given by ISO 9241 and ISO 9126. The methodology involves five methods.

1. "System Summary form". Here, the stakeholders fill in a form about general characteristics of the system.
2. "Compile System Summary forms". A usability expert combines the stakeholder's forms into a summary form.
3. "Creating the context of use portfolio". This results in a document, which describes all the aspects that have an important impact on the system usability [3].
4. "Creating the functional requirements portfolio". This document consists of use cases and system functionalities as well as characteristics and constraints of the system and UI-prototypes.
5. "Review". Here, all artifacts are being reviewed to ensure integrity and consistency among them.

The results can be found in Table 1. Combining the results from the individual methods delivers the following results. As some of the methods involve usability experts and need representative users, the overall methodology does so as well (C1.1: + and C1.2: +). However, most of the methods do not directly involve users. Those methods that do involve users, can deal with inexperienced users and therefore the overall methodology can be considered to work with inexperienced users rather well (C1.3: 4). The description of the methodology is rather vague in parts and therefore it does not provide the stakeholders of the software engineering project with strict guidance (C2.1: '- -'). There is only one method that takes the user feedback into account. However, we consider this sufficient to conclude that the methodology takes the user feedback into account (C2.2: +). Many of the methods can be assumed to be rather time consuming and so is the complete methodology (C3.1: +). However, it combines, as stated earlier, the elicitation and analysis of functional and usability requirements in a single methodology. Therefore, the methodology overall can be considered to be common for requirements elicitation and analysis because no work is done exclusively for elicitation and analysis of usability requirements (C3.2: +). From the methods as they are described by the methodology's authors, it can be doubted that the non-functional requirements are quantified very precisely as there is no method that does so. Therefore, the whole methodology is judged as not eliciting enough information about quantity (C4.1: -). Because of its integrative (functional and nonfunctional) approach, the methodology can elicit the relation between functional and non-functional requirements rather well (C4.2: +).

VII. Methodology 4: Guidelines for Eliciting Usability Functionalities (GEUF)

The methodology Guidelines for Eliciting Usability Functionalities (GEUF) was proposed in 2007 [4]. It refines the method guidelines and heuristic analysis of the first methodology (UEL). The methodology addresses usability requirements as functional requirements during the requirements engineering stage. Based on the guidelines that are provided in the usability literature, the authors have listed a list of functional usability features as a starting point for identifying usability features with an impact on software system functionality. Based on the HCI literature about each feature (if enough is found), the subtypes are listed for each of the features (called usability mechanisms). For each mechanism, the elicitation and specification guides are defined from a development perspective. A set of issues is derived from the elicitation process and needs to be discussed with stakeholders. An initial common vision of system functionality is built before the developers and the users can discuss whether and how specific usability mechanisms affect the software. Two methods are used.

1. "Apply the patterns". Here a template derived from the research is applied to the specific situation.
2. "Applying the Issue/Functionality/Requirement (IFR) table to the issues".

The results for all methods can be found in Table 1. Combining the results delivers the following results.

The result of the methodology is combined as follows. If the developers select this methodology, there is no requirement for having access to an HCI expert (C1.1: -). The methodology does require the involvement of users to discuss the issues (C1.2: +). Therefore, the users should have a high level of knowledge and/or experience to help the developers find correct answers to the issues (C1.3: 2). It was already indicated that the methodology does take the user feedback into account. But this happens only once, there is no iterative design and continuous involvement of the users, therefore it only gets a neutral (C2.2: 0). The methodology is well explained and makes it easy to systematically apply the templates and the table. Hence, it does give a very strict guidance (C2.1: ++). It is not time consuming as it is a one-time task and only considered the proposed mechanisms (C3.1: 0). It does not elicit other functional requirements nor analyze other aspects (e.g., task analysis). But it does take some effort to learn it because it is a new methodology and needs patience to apply it. The template helps the developers to specify the fit criterion using standardized sentences and using the results of IFR table to fill in the specification (C4.1: +). It also explicitly captures the dependencies between requirements in the table (C4.2: +).

VIII. Discussion

For the external factors, there are only few differences. Both methodologies Delta Method and GEUF can be used without access to an HCI expert or with only little involvement of such an expert. Unsurprisingly, all methodologies rely on having access to representative users. This can be attributed to the fact that usability requirements are very individual and therefore cannot be properly identified and analyzed without contacting the prospective users. The biggest difference within this category can be identified for criterion C1.3. While the UEL methodology works especially well with inexperienced users, the GEUF methodology should not be used with inexperienced users because it is likely that their needs would not be captured correctly within this methodology. Within the category characteristics, the most noticeable difference lies in C2.1. While Delta method and GEUF give good guidance and UEL earns a neutral score, the ACUDUC methodology scores poorly. The only methodology that does not sufficiently take user feedback into account is GEUF. The third category, time and effort, shows notable differences in results for both of the criteria. The results for criterion C3.1 have great variance. While we consider the Delta method methodology to be least time consuming, the UEL methodology is considered most time consuming. This can be attributed to its comprehensive set of methods. Overall, ACUDUC and Delta method are considered to mainly consist of methods that are common in software development. For ACUDUC, this can at least partially outweigh the relatively high effort in time needed for this methodology. For Delta method, it points to a comparatively small overall effort. Within the category of effectiveness, only the Delta method cannot elicit dependencies between functional requirements and usability requirements. All methodologies except ACUDUC have the potential to analyze the requirements in enough detail to be able to specify a fit criterion.

IX. Conclusions and Future Work

By comparing the four different methodologies for usability requirement elicitation and analysis on the basis of our UREAM framework we could reach the following conclusions. In terms of external factors (like the need of usability experts, access to representative users and their experience) the Delta method and GEUF are probably most cost-efficient as they can be executed without the help of an HCI expert. All methodologies need access to representative users. However, the Delta method and GEUF can be applied well even with non-experienced users. In terms of characteristics of the methodology, the internal factors like taking user feedback into account is considered in all methodologies except GEUF. Both the Delta method and GEUF provide a strict guidance to the developer for executing the methods. So the Delta method obtains a better score in characteristics compared to the other methodologies. In terms of effort, the Delta method is probably more effective as it can handle a tight project schedule and most of the methods are in common to functional elicitation and analysis, so that the effort to capture them is minimized. In terms of quality and effectiveness, GEUF scores well as the developers can do a quantitative analysis with respect to most of the methods, and the methods support the developers to understand the dependencies between other functional requirements. We suggest that regarding UREAM selection for a concrete project, first the individual characteristics of the project have to be considered, and subsequently the framework-based tables can be used. We are convinced that the (initial) UREAM framework has been validated in this research project. However, further research is needed to elaborate the UREAM framework further so that it (i.e., dimensions and criteria) can also offer a structured basis for the development of new and advanced usability requirements elicitation and analysis methodologies.

Acknowledgment

We would like to acknowledge the support of X. Lu, P. Meenakshy, and T. Milde in preparing this paper.

References

[1] J. Nielsen, “The usability engineering life cycle”’ IEEE Computer vol. 25, nr. 3 (March) 1992, pp. 12-22.

[2] P. Carlshamre and J. Karlsson, “Usability-oriented approach to requirements engineering”, Proc. ICRE96, IEEE Computer Society Press, Los Alamitos, CA, pp. 145-152., 1996.

[3] A. Seffah, R. Djouab, and H. Antunes, “Comparing and Reconciling Usability-Centered and Use Case-Driven Requirements Engineering Processes”, Australian Computer Science Communications, Vol. 23, nr. 5, 2001, pp. 132 – 139.

[4] N. Juristo, A.M. Moreno, and M. Sanchez-Segura, “Guidelines for eliciting usability functionalities”, IEEE Trans Softw Eng 2007; Vol. 33, nr. 11, pp. 744-758.

[5] Jacobson I., Booch G., Rumbaugh J. The Unified Software Development Process. Object Technology Series. Addison Wesley, 1999.

[6] M.C. Maguire, User-centred requirements handbook. EC Telematics Applications Programme, Project TE 2010 RESPECT (Requirements Engineering and Specification in Telematics), WP4 Deliverable D4.2, version 3.3, May 1998.

[7] ISO IEC 25062 Software engineering — Software productQuality Requirements and Evaluation(SQuaRE) — Common Industry Format(CIF) for usability test reports, Geneva, International Organization for Standardization

[8] G.B. Davis, “Strategies for information requirements determination”, IBM Systems Journal, Vol. 21, nr. 1, pp. 4-30, 1982.

[9] B. Nuseibeh, and S. Easterbrook, “Requirements Engineering: A Roadmap”, The Future of Software Engineering, Special Issue 22nd International Conference on Software Engineering, ACMIEEE, pp. 37-46, 2000.

[10] S. Adikari, C. McDonald, and N. Lynch, “Usability in Requirements Engineering”, Proc. ACIS 2006, Adelaide, Paper 76.

[11] K. Van de Berg, J.M. Conejero, and R. Chitchyan, AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation, Report UTwente, 2005.

[12] H. van Vliet, Software engineering: principles and practice. Wiley, 2008.

[13] A.E. Bayraktaroglu, F. Calisir, and C.A. Gumussoy, “Usability and functionality: A comparison of project managers' and potential users' evaluations”, Procs. IEEE IEEM, 8-11 Dec. 2009, Hongkong, pp. 2019 – 2023.

A Multilevel Contract Model for Quality-Driven Service Component Architecture

Maryem Rhanoui
IMS Team, SIME Laboratory
ENSIAS
Rabat, Morocco
mrhanoui@gmail.com

Bouchra El Asri
IMS Team, SIME Laboratory
ENSIAS
Rabat, Morocco
elasri@ensias.ma

***Abstract*—Service Component Architecture (SCA) is a recent approach and an industry standard for developing complex and distributed systems. Despite the growing research work it still lacks a formal basis for handling trust and reliability of quality-driven systems. In this paper, we present main techniques and models for assuring quality and trustworthiness of component-based systems in general, and then we present our contract-aware service component meta model. We propose a multilevel contract model that aims to address reliability and quality issues for service component oriented systems by expressing a set of its properties and constraints.**

Keywords-Service Component; Service Component Architecture; Quality-Driven System; Contract; Aspects.

I. Introduction

Service Oriented Architecture (SOA) is a promising paradigm for developing complex systems that utilizes services as fundamental elements for developing applications. In this perspective, Service Component Architecture (SCA) is a new concept that offers a component model for building SOA architecture.

In the context of a growing interest in reuse of business components, the development of critical and complex systems is confronted with limitations and challenges as service assembly difficulties and the complexity related to numerous SOA standards, therefore SCA emerged as a unifying response.

Official SCA specification document includes SCA assembly model specification [1] and SCA policy framework [2]. However, as an expanding approach, it still needs more formal models and frameworks for modeling and verifying systems.

In spite that the main purpose of software engineering is to find ways of building quality software [3], our literature review shows that most research efforts have focused on technical aspects of Service Component Architecture, leaving aside the treatment of quality issues and extra-functional properties of service component.

In this scope, our fields of research focus on the design and development of complex and safety-critical systems. Critical systems [4] are systems whose failure could cause loss of human lives, cause property damage, or damage to the environment, such as aviation, nuclear, medical applications, etc.

As a matter of fact, dependability [5], which is the property that allows placing a justified confidence in the quality of the delivered service, is becoming increasingly important in complex systems design.

In this paper, we remind the definition of Service Component Architecture and present main techniques and models for handling quality and trustworthiness of component-based systems.

Among the presented approaches, we are interested by the contract-based approach [6], which is a light-weight formal method for designing quality-driven systems by specifying its non-functional and quality properties. Despite the fact that the concept of component contracts was formerly proposed, it still not commonly used in software development.

Our contribution is as follow: we propose a multilevel contract model for modeling both functional and non-functional / quality properties of service components, this model covers different levels of systems, that is the component, composite and final system. Furthermore, it will allow the verification and validation of the constraints outlined in the contract.

In this article, we propose a meta-model for multilevel contracts for service component architecture.

The remainder of this paper is organized as follows: Section II will be dedicated to the presentation of the concept of service component and a survey of main techniques and models for assuring trustworthiness and quality of component-based systems.

Section III will present and justify the choice of our proposed multilevel contract model. In Section IV we will present a meta-model for contract-aware service component architecture. Finally in Section V we illustrate our approach with a case study.

II. Quality-Driven Service Component Architecture

Service-oriented computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications [7]. Service Component Architecture (SCA) proposes a programming model for building components based applications following the SOA paradigm.

The purpose of SCA is to provide a model for creating service-oriented component independent of any specific programming language and to unify the methods of encapsulation and communication in service-oriented architectures by providing a component model.

In this section, we describe, at a glance, the SCA architecture, present the component model and survey the main quality approaches for component-based systems.

A. Service Component Architecture

1) Architecture

An SCA application consists of one or more components that can be implemented in different languages.

A component is a software entity and the basic element of a business function that contains zero or more services and / or reference. A component may have properties and can be either an implementation itself, or a composite. Fig. 1 shows an example of SCA component.

2) Benefits

SCA had emerged as a new architecture for addressing complexity issues of developing SOA solutions. Its offers many advantages:

- Simplify the development of business component and assembly and deployment of business solutions built as networks of services;
- Increase agility and flexibility and protects business logic assets by shielding from low-level technology change and improves testability.

3) Component Model

Various component models of Service Component Architecture were proposed in literature.

For Ding [8] proposed component model, a service component provide and require services. A service can be described by operation activities as by well-defined business function. A component provides and consumes services via ports.

A port p is a tuple (*M,t, c*), where M is a finite set of methods, t is the port type and c is the communication type.

A component *Com* is a tuple (*Pp, Pr,G,W*), in which *Pp* is a finite set of provided ports, *Pr* is a finite set of required ports, *G* is a finite sub component set.

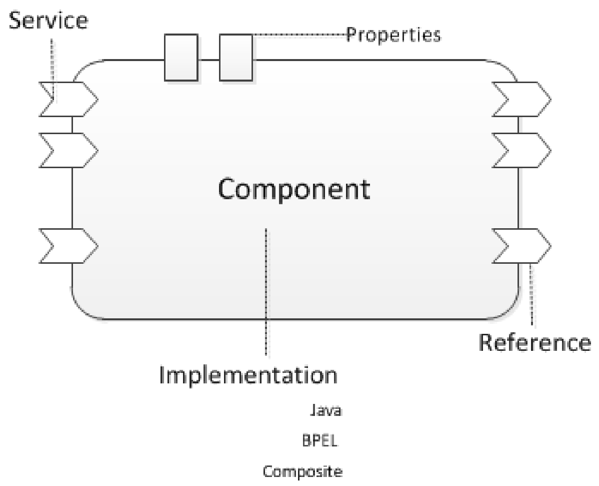


Figure 1 - SCA Component [10]

Moreover, Du et al [9] included contract concept in the Service Component meta-model.

A **contract** *Ctr* is a quadruple (*P, Init, Spec, Prot*) where

- *P* is a port;
- *Spec* maps each operation *m* of *P* to its specification (*am,, gm , pm*) where:
  - *am* contains the resource names of the port *P* and the input and output parameters of *m*.
  - *gm* is the firing condition of operation m, specifying the environments under which *m* can be activated.
  - *pm* is a reactive design, describing the behaviour of *m*.
- *Init* identifies the initial states.

Prot is a set of operations or service calling events.

B. Quality Approaches

There are a wide variety of works and techniques to ensure systems quality, we have identified the main techniques used for component-based systems during all phases of the system's life-cycle as shown in Fig. 2.

Hence, in design phase, functional and extra-functional requirements (as reliability, availability…) are defined and expressed. For this, Design by Contract [6] is an approach and method of software design. It is based on the legal definition of contracts which binds both parties and highlights the interest to precisely specify the interfaces behavior of a software component in terms of pre-conditions, post conditions and invariants.

Subsequently, the reliability of the components and the composite system is evaluated and predicted.

The evaluation and prediction of reliability is to predict the failure rate of components and overall system reliability. They can be used in the operational phase and the early stages of system design software.

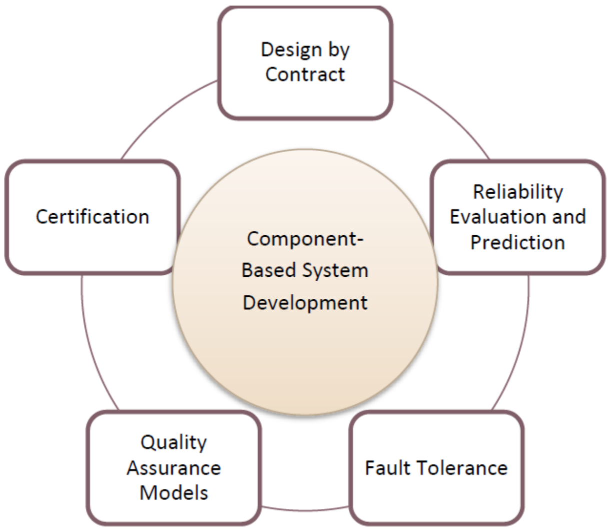

Design by Contract
Certification
Component-Based System Development
Reliability Evaluation and Prediction
Quality Assurance Models
Fault Tolerance

Figure 2 - Quality efforts in CBSs

In addition, the system should continue to operate even in the presence of a failure of one of its components; fault tolerance is the techniques and mechanisms that allow a system to be reliable, available and secure despite the presence of failures.

Furthermore, the development and build process should conform to quality standards; quality assurance is a planned and systematic pattern of all necessary actions to ensure that the item or project conforms to technical requirements [11].

Finally, the achieved quality and trustworthiness is certificated and asserted. Third-party certification is a method to ensure software components are conform to the defined standards; based on this certification, trusted assemblies of components can be constructed [12].

III. Multilevel Contracts

A. Design by Contract

The contract-based approach provides proofs of non-functional and quality properties without requiring the full formality of proof-directed and mathematical development.

The requirements can be specified as preconditions, post conditions and invariants.

- **A precondition** is a constraint that must meet a client when calling a service.
- **A postcondition** is a constraint that must be met by the supplier after use of the service.
- **Invariants** are constraints that must meet all entities that fold to the contract.

This approach is particularly appropriate in the component-based context. In fact, a pre-condition on the parameters of an operation or a service defines a contract that the required/given component agrees to respect. Conversely, post-conditions on the return types of a required component define the customer's expectation from the service provider. Any violation of the contract is the manifestation of a software bug; a pre-condition violation is a bug in the client side and a post condition violation a bug in the supplier side.

It is important that quality is considered during all stages of the development lifecycle of the software. In fact, the contract-based approach allows both defining the desired quality properties and verifying and validating their accuracy.

B. Why Contracts?

Dependability is a major requirement of modern systems which consists of the system's ability to offer a trusted service. It is important to be able to affirm the respect of quality assertions of these systems.

To meet these requirements, we choose a contractual approach [13]. Indeed, within the component and service paradigms, contracts have become an integral part of their definition [14]:

A software component is a unit of composition with ***contractually*** *specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*

A contract defines the constraints between components that is to say, the rights and obligations between the service provider and the client. It has the advantage of expressing the conditions of use of a service by clarifying the obligations and benefits of stakeholders.

We believe that design by contracts can address some of the quality problems of large and complex systems development by explicitly specifying functional and non-functional properties of its components.

Unlike mathematical evaluation and prediction techniques, the contract-based approach is a light-weight formal method for specifying and designing quality-driven systems, it can be introduced in an early stage during the design phase.

To our knowledge, there is still no research work for introducing the concept of contract in service component based systems in order to manage and handle quality requirements.

C. Contracts Levels

Beugnard [15] proposed a classification of contracts into four categories**:**

- **Basic contracts** that ensure the possibility of running the system properly;
- **Behavioral contracts** that improve trust in the system functionalities;
- **Synchronization contracts** that specify synchronization strategies and policies;
- **QoS contracts** which is the highest level and specify quality of service attributes.

This classification has a good coverage of functional and qualitative aspects of components, nevertheless, they don’t handle trustworthiness of composition operations and composite components, yet we have defined three levels of component contracts:

- **Intra-component contracts** concerns the good operations of the component and the respect of its quality requirements;
- **Inter-component / Compositional contracts** ensure the safe composition and trusted assembly of components;
- **System contracts** concerns properties and requirement of the whole system.

D. Contracts by Aspects

Separation of concerns is the process of dividing software into distinct features that overlap in functionality as little as possible, Aspect-Oriented Programming (AOP) [16] aims at providing a means to identify and modularize crosscutting concerns, by encapsulating them in a new unit called aspect.

It was already stated that the design by contract methodology is an aspect of the software system [17]. As such, a contract can be expressed in AOP terminology.

Lorenz [18] classifies aspects for design-by-contract in three types:

- **Agnostic aspects** that don't affect a method's assertions,
- **Obedient aspects** where the input and output states remains unchanged
- **Rebellious aspects** which changes the behavior of existing methods.

Our proposed solution is based on the aspect oriented programming (AOP) for building contract-aware service component based systems. The essential advantage of AOP is the externalization and isolation of crosscutting concerns so that requirement contracts will be expressed outside of the business code of the system.

As AspectJ [19] is one of the first and best known Aspect-Oriented Programming tools, we choose it to implement our approach.

E. *Constraint Specification Language*

In addition, we formalize contracts in UML's Object Constraint Language (OCL) [20], which is a concrete specification language that will help improving the expressiveness of the contracts.

IV. META MODEL

As part of the Model Driven Architecture [21], the Object Management Group (OMG) has defined a meta-metamodel called MOF (Meta Object Facility) [22]. MOF is a specification that defines the concepts to be used to define meta-models.

We propose a MOF compliant meta-model of quality-driven service component architecture; we introduce contract concept and a support of quality requirements as shown in Fig 3.

The meta-model can be divided into two parts: the service component meta-model and its extension with multilevel contract.

A. *Service Component*

Component: A *component* is the unit of construction of Service Component Architecture and an instance of an implementation; it is characterized by *services* executing operations, *properties* and *references* to other services.

Components can be combined into a *composite*.

Service: Services provided by the *component* for other components. A Composite Service can promote a Component Service.

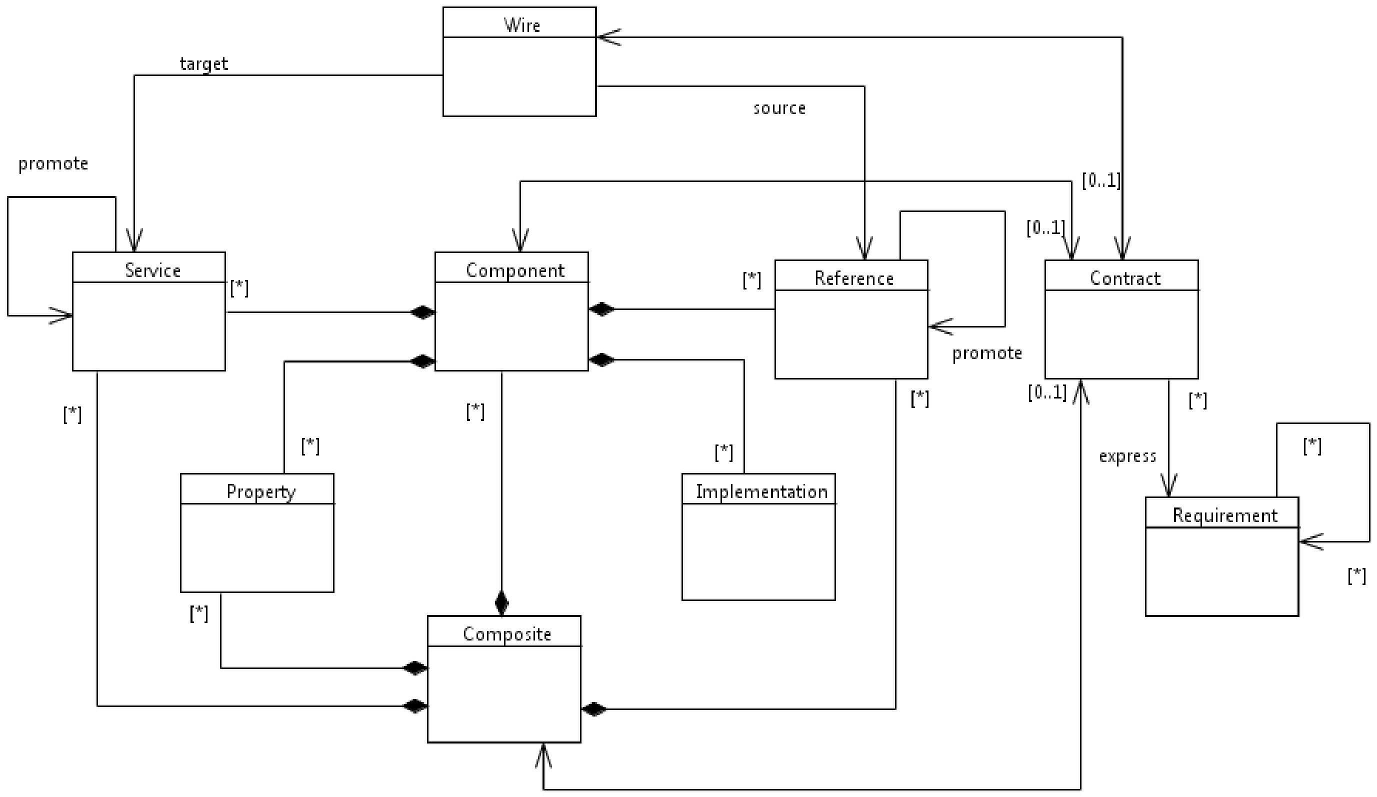


Figure 3 - Service Component Meta Model

Reference: Services required by the *component* from other components. A Component Reference can promote a Composite Service.

Property: External set values or attributes of a *component* or *composite*.

Implementation: Implementation is a program code implementing business functions; a *component* can implement different implementation technologies such as Java, C++, BPEL, etc.

Wire: Wiring that describes the connections between *services* (source) and *references* (target) of *components*.

Composite: A composite contains assemblies of service *components*. Composites also contain *services*, *references* and *properties*.

B. Contract

Constraint: The expressed constraints of the system, defines the obligations that must be verified by the software components.

Contract: A contract specifies the interfaces behavior of a *component* in terms of a configuration of pre-conditions, postconditions and invariants.

A contract is associated to:

- Component element, to ensure the good operations of the component and the respect of its quality requirements.
- Composite element, to ensure the good operations of the composite and the respect of its quality requirements.
- Wire element, to define a contract on the binding of two components.

A contract can express both functional and nonfunctional *requirements.*

Requirement: Functional and non-functional requirements expressed by the *contract*. The requirements are described by a structure of boolean expressions and can be constituted of a set of other requirements. A functional requirement is a property related to the functionality of a the service component. A non-functional requirement is the quality or characteristics of a service component that determines how and under which conditions the service will be delivered. These requirements are not directly related to the functionality provided by the component.

V. CASE STUDY

In this section, we present a simplified case study to illustrate our multilevel contract approach and apply the enunciated concepts. We consider an Airport Management System. The airport has high reliability and dependability requirements. Our system is composed of five components as shown in Fig. 4.

BoardingComponent manage the boarding operations in the airport, it has one service which is promoted by the composite and has two references toward **CheckInComponent** and **SecurityInformationComponent.**

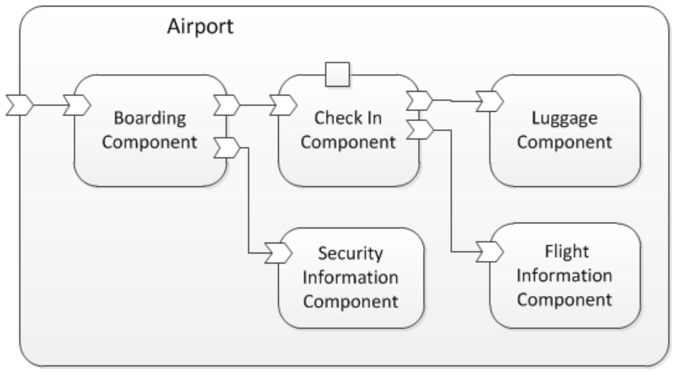


Figure 4 - Airport Management System

CheckInComponent manage the check in process of the passengers, it has one service wired with **BoardingComponent** and has a reference toward **LuggageComponent**.

SecurityInformationComponent manage the security information of the passengers. It has one service wired with **BoardingComponent.**

LuggageComponent manage the luggage check and control of the passengers.

FlightInformationComponent gives the necessary informations of the flights in the airport. It has one service wired with **CheckInComponent.**

To ensure the reliability of our system, we first identify its requirements and then we define the corresponding contracts.

To check in, the passengers must respect the check-in deadline, that is to say, the time beyond which they cannot not register or leave their luggage. Depending on the destination and departure airport, the check-in deadline varies from 15 to 90 minutes before departure time. Moreover, the check in service has to be available 7d/7 at any time of the day and respond within an acceptable period of time. This represents the functional and non-functional requirements of the check in component, then we define a component contract.

Furthermore, the component has two references towards luggage and flight information components; its good operation depends on the correct assembly of these components. Then we define an inter-component or assembly contract.

Finally the airport management system should be reliable and available, which correspond to a system quality contract.

The functional and quality contracts are defined in the design phase of the system development lifecycle, are implemented in the construction phase and are verified during the execution of the components, which allows us to monitor and confirm the compliance with the formerly defined requirements.

VI. Related Works

Some research works related to implementing contracts by using aspects were proposed in the literature, as Contract4J [23] and ContractJava [24].

Contract4J [23] is an open source tool that uses Java 5 annotations and AspectJ. Contract4J offers three annotation types: pre-, post-conditions, and class-invariants. However, although it is still functional Contract4J is no longer maintained since 2007.

ContractJava [24] is a Java extension in which contracts are specified in interfaces. However, class invariants and "result" or "old" variables are not supported.

Handshake [25] is a Java extension that can be enabled where contracts are specified in a separate file with special syntax. However it is not compatible with recent JVM releases.

CONA [26] is a tool that extends the Java and AspectJ syntax with support for Design by Contract and enforces their runtime validation. However its architecture is very complex.

Besides they are not suitable for component-based systems they are mostly limited and academic tools and do not offer a complete and available framework.

Furthermore, our approach is more generic; it handles both functional and non-functional properties of service component, and covers the single component and the composite levels.

VII. Conclusion

This work presented our proposed meta-model of multilevel contract for service component architecture.

Based on a review of main techniques and models for modeling and verifying quality-driven systems; we concluded that contract-based approach is very suitable for component-based systems in general and consequently for service component based systems.

Contracts is a design approach for describing both functional and non-functional properties of complex and quality-driven systems, it also involves synchronization and Quality of Service (QoS) aspects. We will implement it using aspect oriented programming.

We propose a multilevel contract model for expressing and verifying functional and non-functional properties in all levels of service component based systems.

As a continuation of this work, our objective is to propose a modeling framework with a tooling environment and adapt it to Service Component Architecture for safety-critical and quality sensitive systems.

References

[1] M. Beisiegel, "Service Component Architecture Specification." 2007.

[2] G. Barber, "SCA Policy Framework Specification", 2011, Available: http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1.html. [Sep. 15, 2012].

[3] B. Meyer, *Object-Oriented Software Construction*. 1997.

[4] U. Isaksen, J. P. Bowen, and N. Nissanke, "System and Software Safety in Critical Systems," 1996.

[5] J.-C. Laprie, *Guide de la sûreté de fonctionnement*. 1995.

[6] B. Meyer, "Applying 'Design by Contract'," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[7] M. P. Papazoglou and D. Georgakopoulos, "Introduction: Service-Oriented Computing," *Communcications of the ACM - Service-Oriented Computing*, vol. 46, no. 10, pp. 24–28, 2003.

[8] Z. Ding, Z. Chen, and J. Liu, "A Rigorous Model of Service Component Architecture," *Electronic Notes in Theoretical Computer Science*, vol. 207, pp. 33–48, 2008.

[9] D. Du, J. Liu, and H. Cao, "A Rigorous Model of Contract-based Service Component Architecture," *IEEE Computer Society*, vol. 2, pp. 409–412, 2008.

[10] SCA Consortium, "Service Component Architecture - Building Systems using a Service Oriented Architecture." 2005.

[11] ANSI/IEEE Std. 730-1981, IEEE Standard for Software Quality Assurance Plans, 1981.

[12] B. Councill, "Third-Party Certification and Its Required Elements," in *Proceedings of the 4th Workshop on Component-Based Software Engineering*, 2001.

[13] M. Rhanoui and B. E. Asri, "Toward a Quality-Driven Service Component Architecture, Techniques and Models," in *Proceedings of the 14th International Conference on Enterprise Information System*, pp. 192–196, 2012.

[14] C. Szyperski, *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley, 2002.

[15] A. Beugnard, J.-M. Jezequel, N. Plouzeau, and D. Watkins, "Making Components Contract Aware," *Computer*, vol. 32, pp. 38–45, 1999.

[16] R. E. Filman, T. Elrad, S. Clarke, and M. Aksit, *Aspect-Oriented Software Development*. Addison-Wesley, 2004.

[17] F. Diotalevi, *Contract Enforcement With AOP*. IBM, 2004.

[18] D. H. Lorenz and T. Skotiniotis, "Extending Design by Contract for Aspect-Oriented Programming," 2005.

[19] The AspectJ Team, "The AspectJ Programming Guide." 2003.

[20] Object Management Group, "Unified Modeling Language (UML) 2.0 OCL Convenience Document." 2005.

[21] Object Management Group, "Model Driven Architecture (MDA)," 2003.

[22] Object Management Group, "Meta Object Facility (MOF) V2.4.1." 2011.

[23] D. Wampler, "Contract4J for Design by Contract in Java: Design Pattern-Like Protocols and Aspect Interfaces," in *Proceedings of the Fifth AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*, 2006.

[24] R. B. Findler and M. Felleisen, "Contract Soudness for Object Oriented Languages," in *Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 2001.

[25] A. Duncan and U. Hoelzle, "Adding Contracts to Java with Handshake," 1998.

[26] T. Skotiniotis and D. H. Lorenz, "Conaj: Generating Contracts as Aspects," 2004.

Call for Software Tenders: Features and Research Problems

Jorge Hochstetter, Carlos Cares
Departamento de Ingeniería de Sistemas
Universidad de La Frontera
Temuco, Chile
jhoch@ufro.cl, carlos.cares@ceisufro.cl

Abstract— **A relevant part of software industry deals with ad hoc software solutions, which are externalized by software consumers. The process of acquirment follows a public procedure of requesting proposals. This procedure is based on a call-for-tenders document that contains, at least, a software specification and project constraints, such as budget and time. The general assumption is that the requirements stage happens prior to the call-for-tender process. However, public documents of software tendering processes support the contrary assumption. The aim of this paper is to sustain that call-for-tender processes require additional study from an empirical point of view in order to solve problems derived from current industry practices. We have analyzed call for tenders in relation to requirements engineering proposals and also under a procedural approach. In order to sustain the inclusion of call for tenders in the scope of software engineering a set of different open problems is identified.**

Keywords: call for tenders; software tenders; requirements engineering; tendering process; software projects.

I. Introduction

Currently, organizations are increasingly becoming acquirers of needed capabilities by obtaining products and services from suppliers and the development of these capabilities are diminishing in-house [1]. This scenario also holds true for public institutions. In particular, organizations in the public sector who are normally required by law to make public bids for their purchases in order to achieve a transparent process. In several countries, the government is the main buyer of goods and services [2]. For this reason, these countries have invested a great deal of money and effort in defining purchasing policies [4],[5]. The Software industry is a particular case where this has happened.

Therefore, organizations stimulate the competition in the software industry by way of performing calls for tenders (C4T). A call for tenders is the process where a company invites the providers to satisfy particular needs, goods or services. In this paper, we are considering the case of software tendering processes that imply software development, i.e., a software project.

The bidding process associated with a software product is constituted by a set of common practices related to Software Engineering (SE), which have been explored mainly from the Requirements Engineering (RE) perspective. Indeed, within the various problems and challenges of the Requirements Engineering, there is improvement of the quality associated with the specification process, as part of the software process [6].

The literature about calls for tenders is contradictory to the practice. On the one hand, Requirements Engineering promotes a complete and consistent software requirements specification (SRS) before starting the software process. Thus, the theoretical suggestions and corresponding assumptions indicate that the call to public notice phase occurs after the ER stages. Therefore, an SRS becomes part of the call-for-tenders document [6],[7],[8],[9]. On the other hand, on the practice side, we have found that it is very difficult to find a SRS as part of the call-for-tenders document. What we find is that just some of these are included as part of the bid, i.e., the need of including a requirements elicitation and specification phase [10]. Other sources confirm that call-for- tenders documents just include a first approximation of Requirements Engineering’s stages and products [9],[11],[12].

With this information, we can conclude that software industry generates its offers for software development projects in a scenario with uncertain and incomplete information. Moreover, if we are aware that this phase is the seed of a software project, then, it is obvious that uncertain and incomplete information at the level of a call-for-tenders document can imply a high frequency of two deviations: (a) under the given budget, time and project conditions, the software can be well developed, which probably also means that projects conditions are relaxed, hence the customer is losing efficiency, or (b) under the given scenario, the project cannot be developed, which implies that the provider will not accomplish the project goals, or should spend extra money to achieve them. The worst case would reach a state where the project results are aborted. These cases have been illustrated by both theoretical simulations [13],[14] and empirical findings [15],[16].

As a position paper, the goal of this essay is to sustain that call-for-tenders processes require additional study and research with respect to what is happens in industry today. To cover this problem, we analyze the call-for-tenders process stages comparing them to Requirements Engineering stages, recognizing different research problems and proposing research approaches to these sets of problems.

Furthermore, in Section II, we present the basics for differentiating call-for-tenders documents and processes from SRS documents and RE stages. In Section III, we present a generic call-for-tenders process from both customers’ and software providers´ perspectives, and we argue about research problems associated to call-for-tenders activities. To conclude, we summarize our point of view of differentiating but, at the same time, integrating

Requirements Engineering body of knowledge to call-for-tenders processes, as a first step to covering call-for- tenders problems.

II. DIFFERENTIATING REQUIREMENTS ENGINEERING AND CALL FOR TENDERS

To our knowledge, public tender processes for software products have been scarcely investigated in technical literature. Normally, the problem is put in the context of RE activities.

However, in Software Engineering, the acquisition problem has already been recognized as an independent process having its own stages and deliverable products. A clear example is CMMI for acquisitions [1]. In this case, the process view point is the customer´s; therefore, it includes engaging and managing suppliers. Although it is not common, the problem of Call for Tenders has been separately analyzed. For example, Costa et al. [12] address the problem of systematically performing the bidding phase of the public tender process, and propose a multicriteria socio-technical approach to increase the transparency and efficiency of the process.

TABLE I. CALLS FOR TENDERS AND REQUIREMENTS ENGINEERING DIFFERENCES

Requirements Engineering Stage	Call-for-tenders Process
The product is a software requirements specification.	The product is a call-for-tenders document.
The software requirements specification includes software requirements.	The call-for-tenders document includes managerial, economic, budget and software requirements.
It is assumed that the list of software requirements in the software requirements specification is complete.	It is assumed that the list of software requirements in the call-for-tenders document may be incomplete.
Normally it occurs after a contract.	Normally it occurs before a contract.
The software developer may be selected.	The software developer should be selected.
Most of the time a software requirements specification constitutes a specific technical solution.	Most of the time a call-for-tenders document looks for a specific technical solution.
Most of the time it´s focuses are software requirements.	Most of the time it´s focuses are business goals.
There are a set of modelling languages for representing Requirements Engineering deliverables.	There are not modelling languages for representing call for tenders's deliverables.
The involved actors are requirements engineers and stakeholders.	The involved actors are customers and providers.

On the other hand, when practice and theory are confronted, there is evidence that call-for-tenders processes are not following literature recommendations [10],[17],[18]. Moreover, by analyzing different call-for-tenders documents from public distributions [19],[20],[21] we can see that many call-for-tenders documents include, as part of the project, the requisite of formulating a Requirements Engineering stage and a software requirements specification as a deliverable product of the outsourced project.

We sustain that, from the perspective of Software Engineering, i.e., from software production point of view, the generation of a Call for Tenders is a completely different stage from requirements engineering. For example, we have detected a high dispersion of extensions, level of detail, technical specifications, and business goals descriptions among other differences. Moreover, a software requirements specification is only part of the call-for-tenders document, if at all, and there is a low and controlled interaction between suppliers and customers. The detected differences are presented in Table I.

III. CALL FOR TENDERS' RELATED WORK

In spite of the already recognized differences, it is worth mentioning that RE appears to be similar to the call-for-tenders process and, most of the time, they even appear as one phase in the software process. In order to describe current proposals regarding Requirements Engineering and Call for Tenders we summarize them.

Renault et al. [9] present the case of the elicitation and selection of software components. These processes are driven by public tender processes and the use of a Requirement Patterns Catalogue is proposed to save time and reduce errors. Carvallo & Franch [7] present a similar case study where the activities undertaken to obtain, analyze and structure the requirements to be included in a public tender process for an ERP are analyzed.

Lauesen [17] provides a set of guidelines for the creation of the call-for-tenders documents, i.e. the point of view is from the customer, although it is also recognized that these guidelines could also support the suppliers. In a practical sense he affirms that customers do not normally apply these guidelines.

Paech et al. [18] report a set of experiences of a supplier company analyzing call-for-tenders documents for generating technical proposals. They found that existing challenges can be confronted by using requirements engineering techniques.

Additionally, we have started a research line to analyze call-for-tenders documents from the perspective of requirements engineering metrics and it´s comparison to software providers' decision variables. Thus, on one hand [10], we present an approach for analyzing call-for-tenders documents adapting an ontology of speech from goal-oriented requirements engineering to generate metrics. On the other, [3] we describe the application of a focus group technique that provides some clues to understand providers' analyses for applying, or not, for a call-for-tenders process. An initial set of variables are identified in order to describe critical decision points.

From this review, our opinion is that the topic of Call for Tenders is almost unmentioned in Software Engineering literature, but, at the same time, is an underlying topic, i.e., it seems to be present, however, it is not referred to by it´s name (or similar names). This way, several results from Software Engineering research could be applied to different stages of a call-for-tenders process.

In order to make these relationship explicit, i.e., between call-for-tenders process and existing research approaches, we offer an analysis which is based on call-for-tenders stages. We consider the stages starting from the internal requirements collection, to the software development stage made by an external provider. In each stage we have looked for existing software engineering approaches which support it. In addition, we also identify specific problems at each stage to which we have not found any approaches. Therefore, we call them open problems.

IV. Calls for Tenders Support and Open Problems

In this section, we present the general stages of a call-for-tenders process. In the left part of Table 1 we have summarized these stages from both customer´s and provider´s perspectives. Further on, we describe each stage and we analyze how existing software engineering approaches support the technical task at each stage. However, it is necessary to consider that different referred to authors, most of the time, do not distinguish if they are assuming the existence of a call-for-tenders process.

1) Gather organizational needs. We assume that an organization can have a set of planned systems to develop. Requirements Engineering’s proposals adopt a top-down focus, i.e. they can conceive software systems from Business strategies [22],[23] or from Business goals [6],[24]. These systems should accomplish specific functionalities and quality attributes in order to deliver business goals. However, there are two processes to which we have not found approaches. Firstly, computational units collect requirements as part of a day-to-day routine. Therefore, the process is not exclusively top-down. Thus, here is an open problem: given a set of already gathered requirements, how could the parameters of future computational systems be established in order to get convenient bids? Although the “natural” answer seems to be hybrid approaches (top-down and bottom-up together) we have found neither hybrid approaches considering bottom-up requirements, nor the way of combining requirements to conceive a set of future software systems.

2) Estimate budget and time for conceiving systems. To trigger a system development process it is necessary to have a good estimation of both the business value of the system and the cost of the system (including software and organizational adoption). Regarding the business value there are several approaches [13],[25]. In the case of estimating the software development effort, in spite of it being a familiar topic in software engineering [26],[30] the methods require a previous estimation of software size (measured by lines of code, function points or use case points), most of these methods consider particular features of software teams. Therefore, from the perspective of a call-for-tenders process, an early estimation method is required for a generic team which considers the current context. Thus, these methods consider special requirements such as interoperability of existing systems and the reuse of existing libraries or software components, but overall, an incomplete requirements specification. Thus, an open problem here is how budget can be better estimated to incentivize providers to apply and, at the same time, to pay the “fair” price.

3) Specify a call-for-tenders document. At this stage, the customer should produce the document which will be published. Although components of a call for tenders could be well-established, [1] it is necessary to reach an equllibrium between collecting enough information for providers, in order that they formulate good solutions to the problem, and the cost of collecting this information. A related problem is how much detail should the specified solution have (if any) because, on one hand, an open perspective should enable creative and unexpected technical solutions, and at the same time present new organizational challenges to adopt them. On the other hand, a closed perspective allows for the presentation of tailored solutions but, at the same time, reduces the amount of providers, hurting the competitiveness of call-for-tenders process.

4) Search and select requests for proposals. Now, from the perspective of providers, several calls could be available in a specific time period. The providers must decide from a set of call-for-tenders documents which of them they will apply for. Normally, just to study one call-for-tenders document could take too much time. We have not found approaches to make the decision, about how to select calls to evaluate nor how to evaluate them. Derived problems are: how to promise proper software functionality if this has been (normally) poorly specified. A particular problem detected by Hochstetter et al. [3] is how to match the specified problem to existing software assets in order to apply with a competitive offer.

5) Questions and answers. Normally, a public call-for-tenders process considers a public questions and answers process where providers can resolve doubts emerging from the interpretation of call-for-tenders documents. Questions and answers are published, thus, new variables are added to the process. For example, the number of questions should be an indicator of providers’ interest in applying and hence an indicator of process` competitiveness which could imply that more than one potential provider may abandon the process. At the same time a resolved doubt, also resolves any doubt regarding competition. Therefore, the open problem for providers is to achieve an equilibrium between resolving uncertainness to the public and resolving uncertainness over competitiveness. From the perspective of a public-sector customer, answers should show an efficient use of public resources and a transparent way of spending them.

6) Prepare a proposal. Once a provider has decided to apply, the next problem is how to build a “good” proposal. Some work has been done about how to present requirements in a clear and structured way [9],[26] However, the problems seem wider. We have detected that multiple applications from the same organization is normal practise. Therefore, we can even consider the problem about how many proposals can be made and then answer how to make them. Therefore, “good” proposals should consider not only technical content, but available time, assets and team among internal resources and existing systems, standards, assets and stakeholders availability among the external ones. Also, it is necessary to achieve an equilibrium between the cost of preparing the offer and the probability of losing it. We formulate the problem of how to build an offer considering socio-technical constraints from the customer and provider.

7) Select an offer (provider). Following providers' applications, a selection process is initiated. Normally, this process has an administrative stage where legal constraints are verified. Only then is a technical evaluation begun. It seems to be a stage where different approaches from Software Engineering have been proposed: quality models including non-technical factors [7], framework for selecting COTS [28], particular suggestions from acquisition standards [1], and other contemporary approaches such as trust relationships [29]. In order to acknowledge this stage as an already addressed topic, we do not add open problems here.

8) Negotiate the contract. The main goal of this stage is to sign the final contract. In the case of call for tenders from the private sector, it could imply an opportunity to detail some "risky sentences" from both, call for tenders and the bid. However, in the case of government calls, bureaucracy should seriously delay the starting of the process, which would mean changes of socio-technical conditions under which the offer was formulated. To start, or not to start, the project without a contract could be a relevant decision to make. Therefore, trust is a variable key here. There are no studies showing a complete set of variables and how to deal with them. Thus, intermediate payment, engineering deliverables, intellectual property and exploitation of it, stakeholders' availability, post-end availability from the provider, and associated penalties are variables to consider and balance in order to reach a final agreement.

9) Develop Software. At this stage, the traditional software life cycle is activated. As we have sustained, most of the time it involves requirements elicitation and specification. Therefore, traditional approaches can be applied regarding additional aspects, such as imposed project milestones, external monitoring, programming practices and technology imposed by the contract and other considerations derived from a supervised project.

TABLE II. CALL FOR SOFTWARE TENDERS, EXISTING APPROACHES AND OPEN PROBLEMS

Customer	Supplier	Approaches	Open Problems
Gather organizational needs		Top-down proposals from Requirements Engineering e.g. goal-oriented requirements engineering [23],[31] and strategic IT alignment [32] produce systems to-be.	How to combine incomplete requirements to conceive systems which could be, moreover, separately developed even by different external teams.
Estimate budget and time		Traditional software engineering economics considers software size metrics and team features as part of estimation models [25].	How to calculate early software projects estimations under incomplete and uncertain information.
Specify a call-for-tenders document		There are some proposed standards for software acquisition processes including topics to include in call-for-tenders documents [1]. Additionally, other Requirements Engineering approaches results are useful for call-for-tenders specifications [9],[10],[11].	How to aid the writing of call-for-tenders documents balancing detailed information and enough space for creative solutions.
	Search and select call for tenders	No accessible approaches	How to select calls for tenders and how to efficiently evaluate them in order to build competitive bids with low effort.
	Ask for doubts in tender process	No accessible approaches	How to select what doubts to address or what kind of questions or answers could lose other competitors.
Answer questions		No accessible approaches	How to show, in answers, high consistency to call-for-tenders document, a transparent process and efficient use of public resources (if it corresponds).
	Prepare a proposal	The challenges can be confronted by using requirements engineering techniques for generating technical proposals [14],[18].	How to generate a bid regarding the cost of prepare it, the probability of losing it and the involved risks coming from incomplete information.
Select an offer (provider)		There are studies for applying techniques to select software components and providers [12], [26], [27].	.
Negotiate the contract	Negotiate the contract	No accessible approaches	How to handle and balance variables such as changing requirements, stakeholders' availability, penalties, post-sale service at contract time.
	Develop Software	It corresponds to the classical scope of Software Engineering; therefore, its results are applicable to this stage.	How to develop software under a scenario of external supervising, monitoring and even, sometimes, under different developing conditions.

V. Conclusions and Further Lines of Research

In this paper, we have presented the problem of producing and applying to a public call for software tenders as a Software Engineering problem which requires additional approaches. Because it appears to be closely related to Requirements Engineering Stages, we have presented a set of differences among them. Moreover, we have sustained that the common theoretical assumption regarding to call for tenders happening after the requirements stage is not necessary true, therefore, a set of new problems are derived from actual practices. In order to analyze this situation we have followed call-for-tenders phases under the perspective of a customer-supplier interaction. In each phase, we have identified current Software Engineering approaches but we have also obtained a list of open problems corresponding to the different call-for-tenders phases.

Therefore, we have provided enough arguments to sustain that the call-for-tenders process is a different Software Engineering stage and deserves additional research attention, especially in the way that it takes place in industry. Particularly, we propose to focus on the set of problems partially treated which seem to have a high impact on software projects. After all, the call-for-tenders process is the seed of software process.

Acknowledgments

This work has been sponsored by the Investigation Direction, University of La Frontera, Temuco, Chile, DIUFRO DI11-0015 Project.

References

[1] C. Team, "CMMI for acquisition, version 1.3," Technical Report, Carnegie Mellon University, Pitts burgh, 2010.

[2] T. P. Hill, "On goods and services," The Review of Income and Wealth, vol. 23, no. 4, pp. 314-339, 1977.

[3] J.Hochstetter, C. Cachero, C. Cares, and S. Sepúlveda, "Call for Tender Challenges in Practice: a Field Study," in Proc. XV Congreso Iberoamericano en Software Engineering, Buenos Aires, Argentina, 2012.

[4] C. C. Pimenta, "Gestión de compras y contrataciones gubernamentales," RAE-electrónica, vol. 1, no. 1, pp. 1-12, 2002.

[5] S. Lauesen and J. P. Vium, "Communication gaps in a tender process," vol. 10, no. 4, pp. 247-261, Nov. 2005.

[6] A. v. Lamsweerde, "Requirements engineering: from craft to discipline," in Proc. of the 16th ACM SIGSOFT Int. Symposium on Foundations of software engineering, Atlanta, Georgia, 2008, pp. 238-249.

[7] J. P. Carvallo and X. Franch, "On the Use of Requirements for Driving Call-for-Tender Processes for Procuring Coarse-grained OTS Components," in Proc. Requirements Engineering Conference, 2009. RE '09. 17th IEEE International, pp. 287 - 292, 2009.

[8] S. Biffl, D. Winkler, R. Höhn, and H. Wetzel, "Software process improvement in Europe: potential of the new V-modell XT and research issues," Software Process Improvement Practice, Wiley 2006, vol. 11, no. 3, pp. 229–238.

[9] S. Renault, Ó. Ménez-Bonilla, X. Franch, and C. Quer, "A Pattern-based Method for building Requirements Documents in Call-for-tender Processes," International Journal of Computer Science and Applications, vol. 6, no.5 pp. 175 -202, 2009.

[10] J. Hochstetter, C. Díaz, and C. Cares, "Licitaciones Públicas de Software: Métricas Basadas en Actos de Habla," in Proc. Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on, Madrid, España, 2012, pp. 451-456.

[11] J. Brender and P. McNair, "User requirements specifications: a hierarchical structure covering strategical, tactical and operational requirements," International Journal of Medical Informatics, vol. 64, pp. 83-98, 2001.

[12] C. B. e. Costa, E. Corrêa, J.-M. D. Corted, and J.-C. Vansnickd, " Facilitating bid evaluation in public call for tenders: A socio-technical approach," Omega, vol. 30, no.30, pp. 227-242, 2002.

[13] B. A. Aubert, S. Rivard, and M. Patry, "A transaction cost approach to outsourcing behavior: some empirical evidence," Information and Management, vol. 30, no. 2, pp. 51-64, 1996.

[14] S. Whang, "Contracting for software development," Management Science, vol. 38, no.3, pp. 307-324, 1992.

[15] D. Gefen, S. Wyss, and Y. Lichtenstein, "Business familiarity as risk mitigation in software development outsourcing contracts," MIS Quarterly, vol.32, no.3, pp. 531-551, 2008.

[16] R. T. Nakatsu and C. L. Iacovou, "A comparative study of important risk factors involved in offshore and domestic outsourcing of software development projects: A two-panel Delphi study," Information & Management, vol. 46, no. 1, pp. 57-68.

[17] S. Lauesen, "COTS tenders and integration requirements," In Proc. of the IEEE Int. Req. Eng. Conf. (RE), 2004, pp. 166-175.

[18] B. Paech, R. Heinrich, G. Zorn-Pauli, A. Jung, S. Tadjiky, B. Regnell, and D. Damian, "Answering a Request for Proposal – Challenges and Proposed Solutions Requirements Engineering: Foundation for Software Quality," in Proc. *REFSQ*, Essen, Germany, 2012, pp.16-29.

[19] ChileCompras, "Website. http://www.mercadopublico.cl (04 2012)."

[20] Comprasnet, "Website. http://www.comprasnet.gov.br (04 2012)."

[21] Compranet, "Website. http://www.compranet.gob.mx (04 2012)."

[22] E. J. Braude and M. E. Bernstein, "Software engineering: Modern Approaches," J. Wiley & Sons. Second Edition, 2011. ISBN 978-0-471-69208-9.

[23] E. Yu, "Modelling Strategic Relationships for Process Reengineering," Ph.D. thesis, also Tech. Report DKBS-TR-94-6, Dept. of Computer Science, University of Toronto, 1995.

[24] P. Keil, "Principal agent theory and its application to analyze outsourcing of software development," in ACM SIGSOFT Software Engineering Notes, vol. 30, pp. 1-5, 2005.

[25] B. W. Boehm, "Software Engineering Economics," Software Engineering, IEEE Transactions on, vol. 10, no. 1, pp. 4-21, 1984.

[26] N. Aissaoui, M. Haouari, and E. Hassini, "Supplier selection and order lot sizing modeling: A review," Computers \& operations research, vol. 34, no. 34, pp. 3516-3540, 2007.

[27] D. Lowe, "A framework for defining acceptance criteria for web development projects," Web Engineering, pp. 279-294, 2001.

[28] H. C. Esfahani, E. Yu, and M. C. Annosi, "Strategically balanced process adoption," in Proc. of the 2011 International Conference on Software and Systems Process, Hawaii, USA, pp. 169-178, 2011.

[29] A. Heiskanen, M. Newman, and M. Eklin, "Control, trust, power, and the dynamics of information system outsourcing relationships: A process study of contractual software development," The Journal of Strategic Information Systems, vol. 17, no. 4, pp. 268-286, 2008.

[30] H. Erdogmus, B. W. Boehm, W. Harrison, D. J. Reifer, and K. J. Sullivan, "Software engineering economics: background, current practices, and future directions," in Proc. of the 24th International Conference on Software Engineering, 2002, pp. 683-684.

[31] A. v. Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on: IEEE, 2001, pp. 249-262.

[32] S. J. Bleistein, K. Cox, J. Verner, and K. T. Phalp, "B-SCP: A requirements analysis framework for validating strategic alignment of organizational IT based on strategy, context, and process," Information and Software Technology, vol. 48, no. 9, pp. 846-868, 2006.

BPEL-RF Tool: An Automatic Translation from WS-BPEL/WSRF Specifications to Petri Nets

María Díaz, Valentín Valero, Hermenegilda Macià, Jose Antonio Mateo, Gregorio Díaz
Informatics Research Institute of Albacete (I3A) Albacete, Spain
Email: {Maria.DiazTello, Valentin.Valero, Hermenegilda.Macia, JoseAntonio.Mateo, Gregorio.Diaz}@uclm.es.

***Abstract*—Composite Web services technologies are widely used due to their ability to provide interoperability among services from different companies. Thus, orchestration languages like WS-BPEL have recently appeared to manage the interactions of multiple services in order to achieve a global aim. Web services are usually *stateless*, which means that no state is stored from the clients viewpoint. However, some new applications and services have emerged, which require to capture the state of some resources. Therefore, new standards to model Web services states have arisen, such as Web Services Resource Framework (WSRF). In this paper, we present a tool, which takes as input a specification in BPEL-RF (a language defined on the basis of both standards), and transforms it into a prioritised-timed coloured Petri net (PTCPN). These PTCPNs can be verified and validated with the well-known tool, CPNTools.**

***Keywords*-Web Service compositions; WS-BPEL; WSRF; Coloured Petri nets; Tool support; Stateful workflows**

I. Introduction

The development of software systems is becoming more complex with the appearance of new computational paradigms such as Service-Oriented Computing (SOC), Grid Computing and Cloud Computing. In January of 2004, several members of the *Globus Alliance* organization and the computer multinational *IBM* with the help of experts from companies such as *HP, SAP, Akamai, etc.* defined the basis architecture and the initial specification documents of a new standard to describe distributed resources, Web Services Resource Framework (WSRF) [10]. The WSRF elements that are considered in the language BPEL-RF are:

- **WS-ResourceProperties**: There is a precise specification to define WS-Resource properties, based on a Resource Properties Document (RPD), which represents the properties of the associated resource (disk size, processor capacity, etc.). Nevertheless, for simplicity, we only consider a single property for each resource, which is an integer value. Resources are identified by their EPRs (End-Point References); so, we will also use this mechanism for identification purposes, but, for simplicity, we will consider these references as static, instead of assuming a dynamic mechanism to assign them. As a shorthand notation, EPRs will also be used to denote the resource property values. Among the operations allowed by the standard are *GetResourceProperty* and *SetResourceProperty*, which are used to manipulate the resource property values.
- **WS-ResourceLifetime**: The WSRF specification does not provide a standard way to create resources. However, resources have an associated lifetime, which means that once this time has elapsed, the resource is considered to be destroyed. We have then included, for completeness, an operation to create resources, *createResource*, in which the initial value of the resource, its lifetime and the activity that must be launched upon its destruction are indicated. We also have an operation in order to modify the current resource lifetime, *setTimeout*.
- **WS-Notification**: Clients can subscribe to WSRF resources in order to be notified about some topics (resource conditions). We therefore include the *subscribe* operator, indicating the condition under which the subscriber must be notified, and the activity that must be executed upon that event.

WS-BPEL [3], for short BPEL, is an OASIS orchestration language for specifying actions within Web service business processes. BPEL is an orchestration language in the sense that it is used to define the composition of services from a local viewpoint, describing the individual behaviour of each participant. BPEL processes use *variables* to temporarily store data. Variables are, therefore, declared on a process or on a scope within that process. In our case, there will be a single scope (*root*); so, no nesting is considered here. Besides, for simplicity again, we will only deal with integer variables.

An orchestrator consists of a main activity, representing the normal behaviour of this participant. There are also event and fault activities, which are executed upon the occurrence of some events, or due to some execution failures, respectively. BPEL activities can be *basic* or *structured*. *Basic activities* are those which describe the elemental steps of the process behaviour, such as the assignment of variables (*assign*), empty action (*empty*), time delay (*wait*), invoke a service (*invoke*) and receive a message (*receive*), reply to a client (*reply*), and throw an exception (*throw*). We also have an action to *terminate* the process execution at any moment (*exit*). For technical reasons, we have also included a barred form of *reply* action, which is used when a service invocation expects a reply, in order to implement the synchronization with the *reply* action from the server. *Structured activities* encode control-flow logic in a nested way. The considered structured activities are the following: a *sequence* of activities, separated by a semicolon, the parallel composition, represented by two parallel bars (‖), the conditional repetitive behaviour (*while*), and a timed extension of the receive activity, which allows to receive different types of messages with a time-out associated (*pick*).

The main motivation of this work is to provide a formal semantics for WS-BPEL+WSRF to manage stateful Web services workflows by using the existing machinery in distributed systems, and specifically a well-known formalism, such as prioritised-timed coloured Petri nets (PTCPN), which are a graphical model that also provide us with the ability to simulate and analyse the modelled system. In order to deal with the integration of BPEL plus WSRF in a proper way, we have realised that it is more convenient to introduce a specific semantic model, which covers properly all the relevant aspects of WSRF such as notifications and resource time-outs. The integration of both standards is not new; in the literature, there are a bundle of works defining this integration, but none of these works define a formal semantics in terms of Petri nets.

In [16], the integration of BPEL in Grid environments is considered, and the author discusses the benefits and challenges of extensibility in the particular case of OGSI workflows combined with WSRF-based Grids. Other two works centred around Grid environments are [8] and [11]. The first one justifies the use of BPEL extensibility to allow the combination of different GRIDs, whereas Ezenwoye et al. [8] share their experience on BPEL to create and manage WS-Resources that implement the factory/instance pattern in bioinformatics. On the other hand, Ouyang et al. [15] define the necessary elements for translating BPEL processes into Petri nets. Thus, they cover all the important aspects in the standard such as exception handling, dead path elimination, and so on. The model they consider differs from ours in that we formalise the whole system as a composition of orchestrators with resources associated, whereas they describe the system as a general scope with nested sub-scopes leaving aside the possibility of administering resources. Besides, we have also formalized the event handling and notification mechanisms. Following this translation, in [14], Ouyang et al present the tool WofBPEL and a companion tool, BPEL2PNML. The idea behind is to provide tool support for the analysis of BPEL processes. Related to π-calculus semantics, Dragoni and Mazzara [6] propose a theoretical scheme focused on dependable composition for the WS-BPEL recovery framework. In this approach, the recovery framework is simplified and analysed via a conservative extension of π-calculus. The aim of this approach clearly differs from ours, but it helps us to have a better understanding of the WS-BPEL recovery framework. In addition, we also consider time constraints. Moreover, we would like to highlight the work of Farahbod et al. [9] and Busi et al. [4]. In the first one, the authors extract an abstract operational semantics for BPEL based on abstract state machines (ASM) defining the framework BPEL$_{AM}$ to manage the agents who perform the workflow activities. In this approach, time constraints are considered, but they do not formalize the timed model. In the second one, they also define a π-calculus operational semantics for BPEL and describe a conformance notion. They present all the machinery to model web service compositions (choreographies and orchestrations). The main difference with our work is that we deal with distributed resources.

Finally, in the literature one can find several tools performing the opposite translation, i.e., from Petri nets into BPEL. In [2], van der Aalst and Lassen present the theory and implementation of a translation between WF-nets and BPEL. The implementation is performed via the tool WorkflowNet2BPEL4WS. This tool automatically translates coloured Petri nets, CPNs, into BPEL code. These CPNs are specified using CPN Tools [5]. Other similar proposal, WoPeD [7], is a Java-based tool that provides an easy-to-use software for modelling, simulating and analysing workflow processes and resource descriptions. WoPeD supports the CPN notation and the standard file format of WoPeD is PNML, allowing model exchange with other Petri net tools. After this introduction, Section II shows briefly the language BPEL-RF, whereas Section III presents indeed the tool. Section IV contains a case study so as to illustrate how the tool works. Finally, Section V finishes the paper with some conclusions and possible future work.

II. BPEL-RF LANGUAGE

In this section, we are going to present briefly the main characteristics of the language called BPEL-RF (Business Process Execution language for the Resource Framework). An operational semantics for this language was presented in our previous work [12], and the corresponding translation to prioritised-timed coloured Petri nets in [13]. Due to the lack of space, we omit here these transformations, so the interested reader can refer to [12], [13] for them.
We use the following notation: *ORCH* is the set of orchestrators in the system, *Var* is the set of integer variable names, *PL* is the set of necessary partnerlinks, *OPS* is the set of operations names that can be performed, *EPRS* is the set of resource identifiers, and *A* is the set of basic or structured activities that can form the body of a process. Note that each orchestrator uses its own variables despite we have not separated *Var* in its corresponding subsets.

The specific algebraic language, then, that we use for the activities is defined by the following BNF-notation:

$$\begin{aligned}
&A ::= \mathit{throw} \mid \underline{\mathit{receive}}(pl, op, v) \mid \mathit{invoke}(pl, op, v_1) \mid \mathit{exit} \mid \\
&\quad \mathit{reply}(pl, v) \mid \overline{\mathit{reply}}(pl, op, v_2) \mid \mathit{assign}(expr, v_1) \mid \mathit{empty} \mid \\
&\quad A\,;A \mid A \parallel A \mid \mathit{while}(cond, A) \mid \mathit{wait}(timeout) \mid \\
&\quad \mathit{pick}(\{(pl_i, op_i, v_i, A_i)\}_{i=1}^{n}, A, timeout) \mid \\
&\quad \mathit{createResource}(EPR, val, timeout, O, A) \mid \\
&\quad \mathit{getProp}(EPR, v) \mid \mathit{setProp}(EPR, expr) \mid \\
&\quad \mathit{setTimeout}(EPR, timeout) \mid \\
&\quad \mathit{subscribe}(O, EPR, cond', A)
\end{aligned}$$

where $O \in ORCH, EPR \in EPRS, pl, pl_i \in PL, op, op_i \in OPS, timeout \in \mathbb{N}, expr$ is an arithmetic expression constructed by using the variables in *Var* and integers; v, v_1, v_2, v_i range over *Var*, and $val \in \mathbb{Z}$. A condition $cond$ is a predicate constructed by using conjunctions, disjunctions, and negations over the set of variables Var and integers, whereas $cond'$ is a predicate constructed by using the corresponding EPR (as the resource value) and integers. Notice that *setProp* and *getProp* do not contain the property name since, for simplicity, we are only considering a single property for each resource. We therefore use its EPR as representative of this property, as we already observed in the introduction. Note that we do not take into consideration correlation sets, dynamic partnerlinks or instance creation, since we only deal with the static aspects of WS-BPEL. We plan as part of our future work an extension of this operational semantics enriched with these additional constructions, as well as with the inclusion of structured variables, instead of just considering all variables as integers. An orchestration is now defined as a tuple $O = (PL, Var, A, A_f, \mathcal{A}_e)$, where A and A_f are activities defined by the previous syntax and $\mathcal{A}_e$ is a set of activities. Specifically, A represents the normal workflow, A_f is the orchestrator fault handling activity and $\mathcal{A}_e = \{A_{e_i}\}_{i=0}^{m}$ are the event handling activities.

III. BPEL-RF TOOL

As WS-BPEL and WSRF are XML-based languages, and the PTCPNs supported by CPNTools are also represented by XML files, we have used XSLT stylesheets to transform the BPEL-RF document into another XML document representing the PTCPN in a format supported by CPNTools. These XSL stylesheets are created using a XSLT editor. The obtained XML document can be visualized, simulated and verified with CPNTools. As the tool has been developed in Java, it is multi-platform, i.e., runs on Windows/Linux/Mac systems under the Java virtual machine® (the tool is available at [1]). The XSLT transformation sheets (*eXtensible Stylesheets Language/Transform*) are a W3C declarative language to transform XML documents into other XML documents or to some other kind of documents. The XSLT stylesheets are widely used, as an easy way to apply transformation rules to a source document in order to obtain the corresponding output documents. Nowadays, XSLT is widely recommended in web edition area, due to its ability to generate HTML or XHTML sheets.

For making that transformation, XSLT allows to convert the input in two ways: On the one hand, the programmer can manipulate the contents of the document to organize them without changing the document format, whereas, on the other hand, the programmer can use XSLT sheets to transform the contents into other different formats.

We have then defined a number of rules to extract the PTCPN elements from the choreography defined as a composition of WS-BPEL documents. Thus, our tool, BPEL-RF, is used to achieve this transformation in an automatic way, presenting to the user a *.cpn file*, which can be opened with CPNTools. After doing this, the user can analyse and verify the model by using the features of CPNTools.

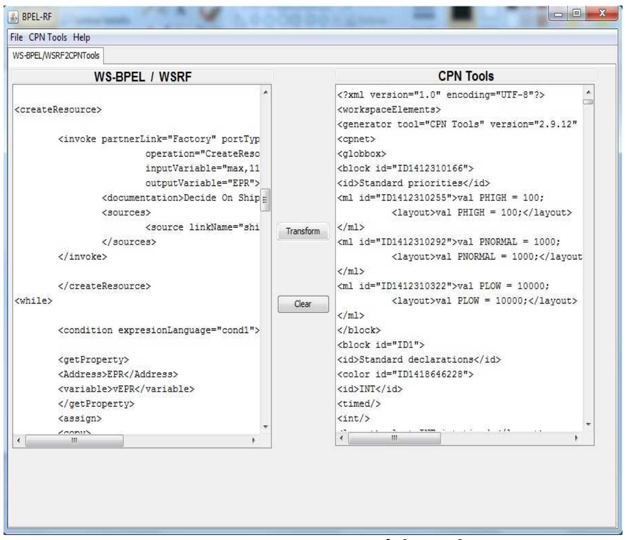
The XSLT stylesheet document starts with the instruction $\langle\ ?xml\ version =' 1.0'?\rangle$. The element root is a stylesheet, which contains all other elements. In an XSLT stylesheet, the name of reserved elements by the specification comes from the same namespace, so they must be written preceded by the appropriate alias that must point to the URL: http://www.w3c.org/1999/XSL/Transform.
In Fig. 1, we show a piece of the structure of the XSLT document.
Once we have located the initial and final mark of the root element "xsl:stylesheet", we define the transformation rules:

- Each rule is defined by an "xsl:template".
- In the rules, we indicate those elements of the XML document that will be transformed.
- The rules also indicate how each element must be transformed.
- Each rule is applied to all elements of the XML document.
- In the XSLT rules, between their initial and final marks, one can include:
  - Text to be written literally in the output document.
  - Marks that are added to the XML output document.
  - Reserved elements to perform an action such as retrieving the value of an item, sorting results, calling other rules of the stylesheet, etc.

```
<?xml version="1.0" ?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform
version="1.0">
<xsl:output indent="yes" />
<xsl:template match="/">
<workspaceElements>
<generator tool="CPN Tools" version="3.2.2" format="6" />
<cpnet>
...
<page id="ID6">
<template>
<xsl:for-each select="//process">
<xsl:for-each select="child::*">
<xsl:if test="(name()='pick')">
<xsl:call-template name="pick" />
<xsl:call-template name="picktrans" />
</xsl:if>
....
</template>
</page>
...
</cpnet>
</workspaceElements>
</template>
</stylesheet>
```

Figure 1. Illustration of an XSLT template

For the sake of simplicity, BPEL-RF Tool has a very simple and intuitive interface shown in Fig. 2. It consists of a main frame with separated elements such as a file menu and the transformation panel. The file menu has three different submenus, namely: *File*, *CPN Tools* and *Help*. The *File* submenu offers two options. The first one, *Open WS-BPEL WSRF File*, opens a BPEL-RF document previously edited and saved with the tool; whereas the second one, *Exit*, exits the program. The *CPN Tools* submenu only offers one option, *Save Coloured Petri Net*, which saves the translated XML code to a .cpn file. Finally, the last submenu, *Help*, consists of two options *Help* and *About*. The option *About* only informs users about the tool version, the option *Help* offers users a wide user manual with the possibility of searching through the information using either a table of contents or a search option.



```
<createResource>

        <invoke partnerLink="Factory" portTyp
                operation="CreateReso
                inputVariable="max,11
                outputVariable="EPR">
            <documentation>Decide On Ship
            <sources>
                <source linkName="shi
            </sources>
        </invoke>

        </createResource>
<while>

        <condition expresionLanguage="cond1">

        <getProperty>
        <Address>EPR</Address>
        <variable>vEPR</variable>
        </getProperty>
        <assign>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<workspaceElements>
<generator tool="CPN Tools" version="2.9.12"
<cpnet>
<globbox>
<block id="ID1412310166">
<id>Standard priorities</id>
<ml id="ID1412310255">val PHIGH = 100;
        <layout>val PHIGH = 100;</layout>
</ml>
<ml id="ID1412310292">val PNORMAL = 1000;
        <layout>val PNORMAL = 1000;</layout
</ml>
<ml id="ID1412310322">val PLOW = 10000;
        <layout>val PLOW = 10000;</layout>
</ml>
</block>
<block id="ID1">
<id>Standard declarations</id>
<color id="ID1418646228">
<id>INT</id>
<timed/>
<int/>
```

Figure 2. Main screen of the tool.

The main elements of the interface are:

- The **WS-BPEL / WSRF Textbox** permits users to introduce XML code following the specification given by WS-BPEL and WSRF. This XML is used as the source code to be translated into PTCPN. This code can be introduced in two ways; either by writing the XML code by hand or by loading a previously saved document using the *Open WS-BPEL WSRF File* submenu mentioned above. A dialog window will be shown to the user asking him to select the document to be opened. If the file is not valid, an error message will be displayed on the screen.
- In the **CPNTools Textbox**, after clicking on the button "Transform", the corresponding Petri Net XML specification is shown. To save this specification, the user must click on the *Save Colored Petri Net File* option in the CPN Tools menu. A dialog window will be shown to the user to choose the destination folder.

Moreover, we have another two buttons on the screen:

- **The Transform button** generates the corresponding PTCPN. The result will be automatically displayed in the CPN Tools Textbox after a few seconds. If the WS-BPEL WSRF Textbox is empty, pressing the Transform button will have no effect.
- **The Clear button** is used to clean the contents of both text boxes. If both are empty, pressing on this button will have no effect.

IV. Case Study: Automatic management system for stock market investments

The case study concerns a typical automatic management system for stock market investments, which consists of *n+1* participants: the online stock market system and *n* investors, A$_i$, $i = 1, \ldots, n$. Here, the resource will be the stocks of a company that the investors want to buy just in case the price falls below an established limit, which the investors fix previously by means of subscriptions, i.e., an investor subscribes to the resource (the stocks) with a certain guard (the value of the stocks he/she want to pay for it). The lifetime *lft* will be determined by the stock market system and the resource price will be fluctuating to simulate the rises/drops of the stock. Notice that we do not take into account the stock buy process since our aim is to model an investors' information system. Thus, the participants will be notified when their bids hold or the resource lifetime expires. Let us consider the choreography $C = (O_{sys}, O_1, \ldots, O_n)$, where $O_k = (PL_k, Var_k, A_k, A_{f_k}, \mathcal{A}_{e_k})$, k=sys, 1,..., n; $Var_{sys} = \{at, vEPR\}$, $Var_i = \{v_i\}$, $A_{f_k} = exit$. Variable $vEPR$ serves to temporarily store the value of the resource property before being sent; v_i is the variable used for the interaction among participants, and, finally, at controls the period of time in which the auction is active. Note that the value *x* indicates the resource value at the beginning, *at0* is the time that the "auction" is active, and, finally, x_i is the value of the stocks that he/she wants to pay for. Suppose that all the variables are initially 0:

$$
\begin{aligned}
&Asys = assign(x+1, vEPR); assign(at0, at);\\
&\qquad CreateResource(EPR, lft, x, empty);\\
&\qquad while(actualTime() <= at, Abid)\\
&Abid = getProp(EPR, vEPR); assign(vEPR + bid(), vEPR);\\
&\qquad setProp(EPR, vEPR); wait(1, 2)\\
&A_i = wait(1, 2); subscribe(O_i, EPR, EPR < x_i, Acond_i);\\
&\qquad pick((pl_i, buy, v_i, empty), empty, at0)\\
&Acond_i = getProp(EPR, vEPR); invoke(pl_i, buy, vEPR)
\end{aligned}
$$

Here, the function *bid* is used to increase/decrease the stocks value simulating the fluctuation of the stocks price.

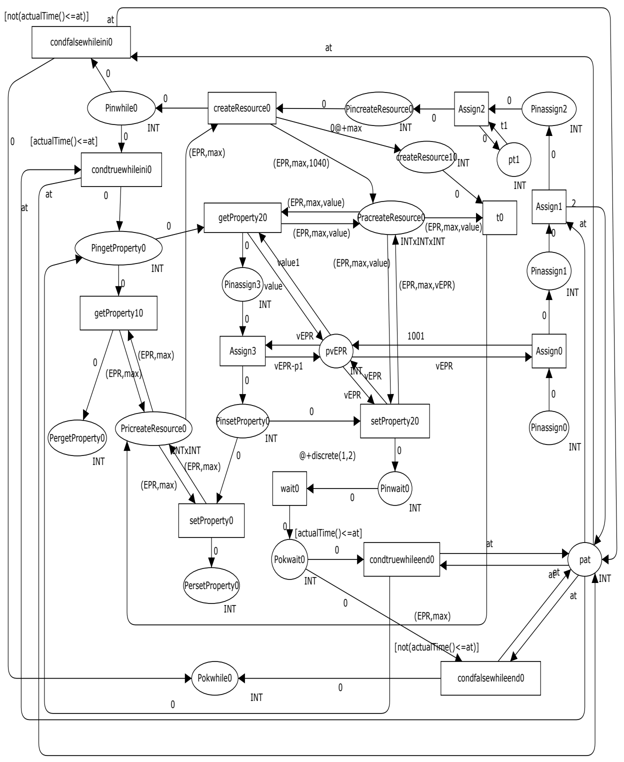


Figure 3. PTCPN of the online stock market.

In Figs. 3 and 4, the PTCPNs for one buyer and for the system are depicted. These figures have been obtained automatically by using our tool.

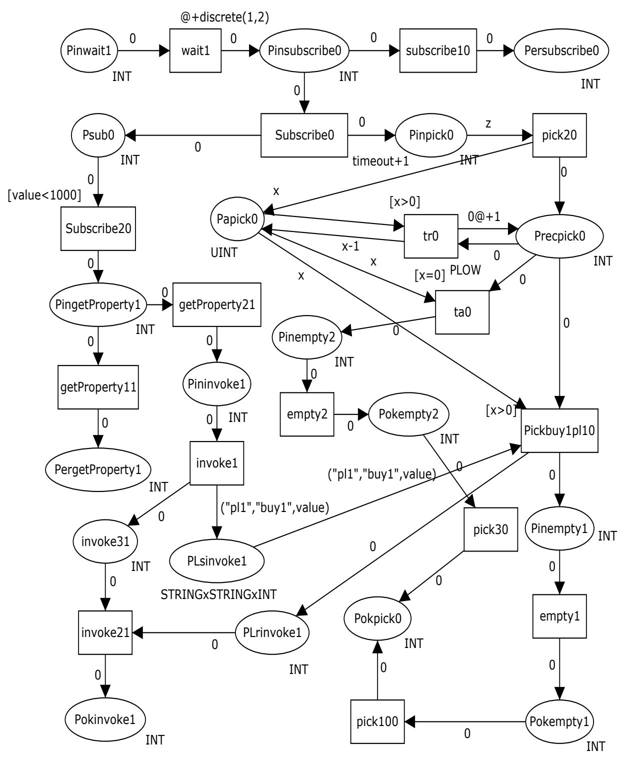


Figure 4. PTCPN of one buyer.

A. Analysis

CPNTools offers us two forms to check the correctness of our system: formal verification and simulation. First, the simulation helps designers to understand how the system exactly works and it is a mean to detect possible errors in early stages of the development process in order to refine the model according the clients' requirements. Besides, formal verification through state space analysis could be done in order to ensure that our system achieves some formal properties such as liveness, deadlock-freeness, and so on. In this way, Table I shows the results obtained considering 1, 2, 3, 4 or 5 investors. Note that we have considered the following assumptions:

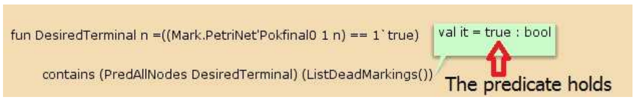
- The "auction" time *at0* is limited to 10 time units.
- The resource is active during 15 time units (*lft=15*).
- The resource value *x* is 100 money units.
- The value of subscription of each investor *i*, x_i, is $x-(9+i)$, that is, if the system has only one investor its subscription guard will be $x < 90$, whereas with 5 investors, the last investor will have a subscription guard of $x < 86$.
- The function *bid* will fluctuate the stocks price between -2 and 1 in order to simulate that the price only can rise 1 and drop 2 at most each time unit.

We will focus on deadlock-freeness to ensure that the system never gets stuck while the participants have activities to do in their workflow. We have leveraged the functions offered by CPNTools to demonstrate that in all dead markings of the system the final place is marked, which leads us to conclude the system has finished correctly. This final place, *Pokfinal0*, is marked by a transition when all the participants have finished their workflow. For the sake of clarity, we have not drawn this place in each figure. Thus, the next SML code checks when this situation occurs:
fun DesiredTerminal n =((Mark.PetriNet'Pokfinal0 1 n) == 1'true),
which returns *true* if the place *Pokfinal0* is marked. In addition, it is needed to evaluate the following predicate:
PredAllNodes DesiredTerminal=ListDeadMarkings(), to check
that the list of dead marking contains the marking of the *Pokfinal0* place.

Properties	Number of investors				
	1	2	3	4	5
State Space Nodes	3561	7569	16983	50350	89879
State Space Arcs	5203	12843	33271	112101	262215
Time (s)	2	7	23	146	1140
Dead Markings	124	244	454	1108	874

Table I
STATE SPACE ANALYSIS RESULTS

In Fig. 5, we show the results offered by CPNTools to our queries for the case of **three** investors. Here, it can be appreciated that all dead markings hold the predicate *DesiredTerminal*, and, therefore, when the system reaches a dead marking is because system has terminated, which demonstrates the absence of deadlocks in our case study.


```
fun DesiredTerminal n =((Mark.PetriNet'Pokfinal0 1 n) == 1`true)      val it = true : bool

      contains (PredAllNodes DesiredTerminal) (ListDeadMarkings())
```

Figure 5. Result of the queries in CPNTools.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, a tool which permits the automatic translation between BPEL-RF specifications and PTCPNs supported by CPNTools has been presented. This is a great advantage with respect to our previous works in such a way the user only needs to provide the XML code for the orchestration and the tool will extract automatically the corresponding translation in order to effectuate the formal analysis of the system. This analysis can be done by simulation or by formal verification. In the case study, we have centred on formal verification looking for the absence of deadlocks in the model. Finally, as future work, we plan to extend our work with additional features of both WS-BPEL and WSRF, as the discovery of existing resources. We are also working on the demonstration of the equivalence between the operational semantics of [12] and the Petri nets semantics of [13].

ACKNOWLEDGMENT

This work has been partially supported by CICYT project TIN2009-14312-C02-02, and JCCM project PEII09-0232-7745.

REFERENCES

[1] [retrieved:September,2012] BPEL-RF tool web site, http://www.dsi.uclm.es/retics/BPELRF/

[2] W. M. P van der Aalst and K. B. Lassen. Translating unstructured workflow processes to readable BPEL: Theory and implementation, Journal of Information Software Technology, vol. 50, number 3, pp. 131-159, 2008.

[3] [retrieved:September,2012] Alexandre Alves, Assaf Arkin, Charlton Barreto, Ben Bloch,

Francisco Curbera, and Rania Khalaf. Web Services Business Process Execution Language, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[4] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G.Zavattaro, Choreography and Orchestration: A Synergic Approach for System Design. In International Conference of Service Oriented Computing (ICSOC), Lecture Notes in Computer Science, vol. 3826, pp. 228-240, 2005.

[5] [retrieved:September,2012] CPNTools official web site, http://cpntools.org.

[6] N. Dragoni and M. Mazzara, A formal Semantics for the WS-BPEL Recovery Framework - The pi-Calculus Way. In International Workshop on Web Services and Formal Methods (WS-FM). Lecture Notes in Computer Science, vol. 6194, pp. 92-109, 2009.

[7] A. Eckleder and T. Freytag, WoPeD 2.0 goes BPEL 2.0. In 15th German Workshop on Algorithms and Tools for Petri Nets, Algorithmen und Werkzeuge für Petrinetze (AWPN 2008). CEUR Workshop Proceedings, vol. 380, pp. 75-80, 2008.

[8] O. Ezenwoye, S.M. Sadjadi, A. Cary, and M. Robinson, Grid Service Composition in BPEL for Scientific Applications. In OTM Conferences, pp. 1304-1312, 2007.

[9] R. Farahbod, U. Glässer, and M. Vajihollahi, A Formal Semantics for the Business Process Execution Language for Web Services. In Joint Workshop on Web Services and Model-Driven Enterprise Information Services (WSMDEIS), pp. 122-133, 2005.

[10] [retrieved:September,2012] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, T. Storey, and S. Weerawaranna, Modeling Stateful Resources with Web Services, http://www.globus.org/wsrf/.

[11] F. Leymann, Choreography for the Grid: towards fitting BPEL to the resource framework. Journal of Concurrency and Computation : Practice & Experience, vol. 18, issue 10, pp. 1201-1217, 2006.

[12] J.A. Mateo, V. Valero, and G. Diaz, An Operational Semantics of BPEL Orchestrations Integrating Web Services Resource Framework. In International Workshop on Web Services and Formal Methods (WS-FM), 2011.

[13] [retrieved:September,2012] J.A. Mateo, V. Valero, H. Macià, and G. Diaz. A Coloured Petri Net Approach to Model and Analyse Stateful Workflows Based on WS-BPEL and WSRF. Technical Report DIAB-12-04-2, University of Castilla-La Mancha. Available at: http://www.dsi.uclm.es/trep.php?codtrep=DIAB-12-04-2

[14] C. Ouyang, E. Verbeek, W. M. P. van der Aalst, S. Breutel, M. Dumas, and A. ter Hofstede, WofBPEL: A Tool for Automated Analysis of BPEL Processes., In Third International Conference on Service-Oriented Computing (ICSOC 2005). Lecture Notes in Computer Science, vol. 3826, pp. 484-489, 2005.

[15] C. Ouyang, E. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas, and A.H.M. ter Hofstede. Formal semantics and analysis of control flow in WS-BPEL. Science of Computing Programming, vol. 67, issue 2-3, pp. 162-198, 2007.

[16] A. Slomiski. On using BPEL extensibility to implement OGSI and WSRF Grid workflows. Journal of Concurrency and Computation : Practice & Experience, vol. 18, pp. 1229-1241, 2006.

Automated Reuse of Software Reuse Activities in an Industrial Environment – Case Study Results

Marcus Zinn
University of Plymouth
Plymouth, UK
marcus.zinn@plymouth.ac.uk

Klaus-Peter Fischer-Hellmann
University of Applied Science
Darmstadt, Darmstadt, Germany
k.p.fischer-hellmann@digamma.de

Ronald Schoop
Schneider Electric Automation
Seligenstadt, Germany
ronald.schoop@schneider-electric.com

Abstract - **The reuse of prefabricated software units, such as classes, components and services is one of the central topics of software engineering and requires lot of knowledge and experience. Instead of focusing on the knowledge management processes and a resulting lifelong learning process of individuals, this paper shows an experimental study based on an approach of automation of knowledge based reuse activities. This is done by employing a unified view of software construction activities and software units used by these activities in an industrial environment. It concludes that software engineers of different industrial business units and knowledge levels can be supported by performing different software construction activities with only one approach, the result of which avoids a long learning process for software engineers.**

Keywords-Automated software unit reuse; software reuse activities; industrial environment; case study.

I. INTRODUCTION

The reuse of software units (like classes, components, or services) requires professional knowledge or expertise. A software unit is a technical unit, and can, therefore, be defined like a software component in the context of this paper:

"*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition of third parties*". [1]

Typically, software engineers have to acquire this knowledge. In industrial environments, the knowledge depends not only on the technical properties of a software unit but also on the technical environment, technical topic (e.g., embedded devices) and the business topics (e.g., Automation, Datacenters, Mines & Minerals). Today knowledge about software units in a reuse context is a broad field. As adequate description of knowledge in the context of this paper following definition is used:

"*... the capability of a man (or an intelligent machine) to use information for problem-solving*" [2]

Starting from this point of view a software engineer has to have different kinds of information to perform software reuse, as for example: (1) Information about technical properties such as programming language, necessary technical environment, and dependencies. A software engineer has to know this information. [3]

(2) Information about interfaces and business context. A software unit solves at least one problem. Typically, the interfaces and provided data types are related to this fact. By handling such a software unit a software engineer have to be aware about this information. [3] (3) Information about the reusable artefact. Today a reusable software unit is more than a single binary file. Related information like test cases, documentation, and versioning are also reusable and sometimes implied. A software engineer has to deal with this related information. [4] (4) Information about related reuse concepts and processes. Software unit reuse is not undertaken if a software engineer decides to perform reuse. Many activities such as search, validation, integration, transformation, and testing are part of a reuse process. A software engineer must be aware of the existence of different reuse processes and technologies.

As a result of these perspectives, reusing a software unit may define as the use of different information about a software unit and a given environment to perform a number of reuse activities. The result is a reused software unit in a software development project.

Based on the high number of different technologies, business context, reuse artefact information and possible reuse concepts or technologies, the amount of necessary knowledge is high. This results in a problem for software engineers. Each time they wish to reuse a software unit they have to know about the relevant activities, and the related knowledge and information. If this knowledge is missing the reuse cannot be carried out successfully.

A solution may be the automation of reuse activities. As shown in the automation industry, this requires the development of supporting systems that are able to perform activities for a user. By automating software reuse activities, software engineers are able to perform these activities without having acquired the complete knowledge. Such an approach would reduce the problem of missing knowledge and was discussed in the past [5] and [6] under the name of "Service based Software Construction Process (SSCP)". However, the experimental proof of this concept is still missing.

This paper describes the setup and the results of the first phase of an experiment validating the concept of SSCP, which is described by the following hypothesis:

"*Automated Software reuse activities will reduce the problem of missing knowledge in software unit reuse*"

This work forms part of the research on a Service-based Software Construction Process (SSCP) incorporating the field of Software Unit Reuse. The goal of this research is to identify a semantic model (about finding, adapting, integrating, and deploying of software units) combined with service technology that supports software engineers by performing software reuse (finding, adapting, integrating, and deploying) without having all needed information. The paper contributes to the research area by demonstrating the positive effect of automated software reuse activities, based on software reuse knowledge on the problem of missing knowledge in software unit reuse, in a real world experiment.

After the problem statement in the next section, the Section 3 shows the focused solution of this paper. This is used in Section 4 to describe the experiment setup and execution. Section 5 discusses the experiment results followed by the conclusion section (Section 6)

II. The Problem of Reuse in Multiple Industrial Software Development Teams

Typical aims of software reuse are to reduce costs and time in development projects [6]. These are two reasons why reuse of software units is an important part of software development in industrial areas [5]. However, the use of reuse in industrial projects does not guarantee a successful project, a fact, which has been demonstrated by several project studies in the past [6]. Typical problems are [6], e.g. ,: Misconceptions (reuse == repository, reuse == OO), No non-reuse specific processes modified, No reuse specific processes installed, No training/awareness actions, Reusable assets produced but then not used, Multi contractor / Multi company project, and No production of assets.

The last problem 'No production of assets' differs from the others. This problem deals with the fact that a software unit must be developed in order to be reusable [7]. If this is not the case, the amount of required resources is decreased by reuse [6][7]. Based on this statement, the effort to reuse increases after the creation of a software unit and should remain at the same value continuously for each reuse.

An internal study conducted by Schneider Electric [8] indicates a complex but interesting picture. A set of around 50 software units (so-called 'bricks' in industry area) has been created and widely reused. The average reuse number is between 9 and 10. The distribution of reuse for different bricks is shown in Figure 1. It starts with a minimum of 3 reuses (the point where typically a cost breakeven would start compared to a non reuse approach) and spans up to 36 reuses.

Relating to the above mentioned fact 'No production of

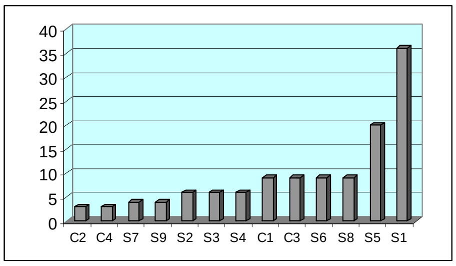


Figure 1. Distribution of reusable bricks [8]

assets' the study of Schneider Electric shows a dilemma of reuse in industrial environments. A reusable software unit creates additional reuse effort during the creation phase and in reuse phases of each development team which reuses this unit.

Creation Phase Dilemma (CPD): The creation of reusable software includes different phases, which focus the reusability. Typical examples are given by Software Product Line approaches [7]: (1) Generalisation – The interfaces and functions of a software unit must be generalised to increase the reuse probability. (2) Integration – The software unit must be built in a way that it can be integrate in the development projects of other teams. (3) Support – The software unit must be 'equipped' with additional reuse artefacts, which support the reuse, e.g., reuses documentation. Additionally, such a unit have to be installed in a system, which provides access to it.

All of these steps require knowledge from an expert user.

Reuse Phase Dilemma (RPD): Each development team has now different challenges for reusing such a software unit. Typically, each team has to find and download the software unit [8]. In the next steps, they have to understand and integrate the unit into their development projects [7]. Sometimes software units must be adapted (transformed) for that specific application [9]. Figure 2 shows also the typical support and maintenance effort, which is created during these steps. This effort is the results from the problem that the development teams have not enough knowledge to perform the described reuse steps.

CPD and RPD are typical theoretical examples discus-

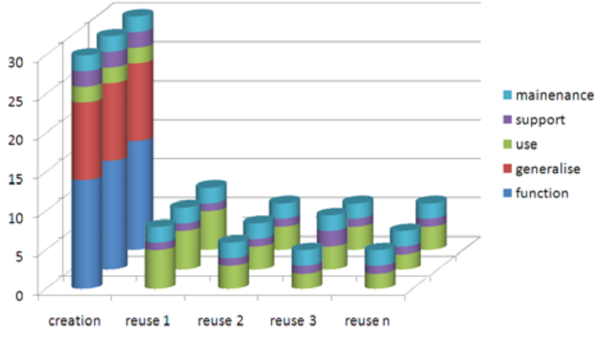


Figure 2. Support and maintenance effort [8]

sions of problems. The reality creates two additional dilemmas in the context of CPD and RPD. (1) **Creator dilemma (CD)**: The creation team is not available for support at the time of reuse (people are loaded with other projects or change team or organization) (2) **Reuser dilemma (RD)**: The reuse teams are different for each development projects, and therefore the exchange of a 'learning curve' between the teams is not possible.

Figure 2 shows that each development team has nearly the same problems and need nearly the same amount of resources. The challenge of reuse based software development in industrial areas is to reduce the sketched dilemmas. The purpose of this study is to show that reuse of a single software unit in multiple teams does not need this amount of resource on both sites: creator and reuser.

III. CONTEMPORARY SOLUTIONS

Nowadays, there are different approaches for the above-mentioned problems. The first approach is so called information systems, which, in general, enable the storage of information. This enables a user to search for information. However, such systems are not designed specifically to address the issue of transformation, but treat the subject of information generally [10]. Generally, such systems can be used to save information about an area of knowledge in textual form, but without the context of knowledge (see [10]). Each software construction activity may be described in this form and may be stored in an information system. The user is now faced with the problem of obtaining this information and interpreting it correctly in order to perform a successful transformation. Usually, information systems are not intended to apply their stored information automatically. But they can be extended for this task [10].

Despite this lack of functionality, information systems comprise a part of this article's advocated solution. Extensions of information systems are so-called Knowledge Base System (KBS) [10]. Such systems are defined as:

"*… a method that simplifies the process of sharing, distributing, creating, capturing, and understanding a company's knowledge*." [11]

Knowledge systems are not fundamentally designed for the subject of software construction activities. Furthermore, the authors of this article believe knowledge systems are missing a fundamental property: the automated application of stored knowledge for specific tasks. However, there is a lack of systems that have asserted themselves and are not focused on the typical software construction activities of software units. The latter property 'application of knowledge', is also a part of the solution discussed in this article. Basically, the knowledge that is necessary for perform an reuse activity can be stored in knowledge systems.

The area of software development has currently seen a number of interesting approaches dealing with specific subjects of a software reuse activity. Most of them are specific for one reuse activity type. For example there are two existing approaches for the activity of software unit transformation which are of interest: Model transformation [12] and generative programming [13]. Both approaches have existed for some time and form the basis for approaches that are being used today. Both support software engineers in generating reusable transformation models or rules. However, additional knowledge is necessary to make use of both approaches. This can be found in other activity areas like deployment [14] and Integration [15]. For the integration of software units into Integrated Development Environments (IDE) very specialised solutions exits e.g., Packaging for Eclipse or Packaging for Visual Studio. But these products are too specialised and require different kinds of specialised knowledge from the user.

The above mentioned solutions have one common problem. They assume a high learning curve. But learning how to implement every existing technology or solution for knowledge based problems cost too much time. It is necessary to identify a solution, which is able to support software engineers by performing software reuse activities without a lifelong learning process.

IV. FOCUSED SCENARIO

The basic idea of the targeted solution is that an expert applies knowledge (knowledge extraction) about the software reuse activity of a specific software unit to a system, which is able to perform the activity automatically with a minimum of human interaction based on knowledge. Users who do not have the necessary knowledge are now able to perform this activity (knowledge injection). A learning process for this specific activity and the specific software unit is not necessary. Figure 3 shows this scenario.

The idea was presented in previous [5][6] where its advantage was demonstrated for two reuse activity examples: Integration of software units into integrated development environments (IDEs) [15], and deployment of software units into embedded devices [14].

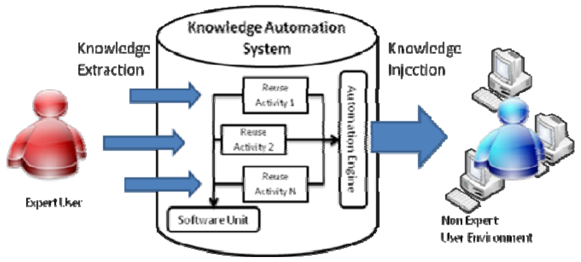


Figure 3. Concept of the focused solution

For the experiment demonstrated in this publication the software construction activities 'Integration' and 'Transformation' were chosen.

V. THE EXPERIMENTAL SETUP

A. *Technical structure and infrastructure*

The following section utilises this theoretical description to create the basis for the activities of integration, transformation and deployment of real models. The experiment is performed by software engineers using these models. Theses engineers try to perform different transformation and integration software construction activities with and without the support of the proposed solution. The second step comprises a description of the design and implementation of the experiment. These descriptions are intended for the replication of the experiment, and to ensure the sustainability of the experiment for the study's results. The setup of the experiment is divided into three distinct areas:

(1) Description of the environment,
(2) Description of the technical structure of the experiment, the necessary elements, and
(3) Description of the measurement process.

Description of the environment: The experiment was conducted at a German location of the company Schneider Electric (Address Steinheimer Strasse 116, 63500 Seligenstadt, Germany). The company has participated by means of employees at this site and from other international locations using the company intranet. The experiment itself was conducted in normal offices, which provide a connection to this intranet source.

Description of the technical structure of the experiment, the necessary elements: The technical design of the experiment is mainly a hardware and software infrastructure. Figure 4 shows this structure in the environment of the Schneider Electric intranet. Six important elements are involved. The first element is the intranet (1), which is used to connect the various other elements of the technical structure. The second elements (2) are the connected databases, including the software units and complete information about the re-use activities. Four databases are important for the experiment:

1) SOA4D: This is an open source repository software unit with further information about device profiles, including four web services. This repository is based on the Forge technology and offers a web interface.
2) Prometheus SQL: this is a specially developed Repository. It belongs to the approach and uses a Microsoft SQL database and Microsoft SQL database interface.

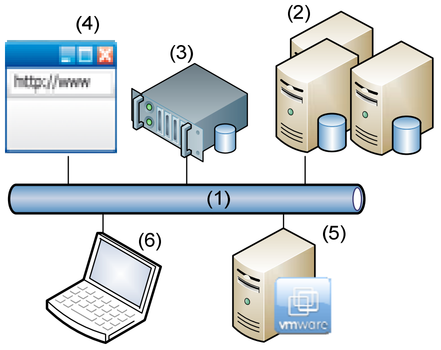


Figure 4. Experimental environment and setup

3) DDXML repos: This is a Schneider Electric internal repository that contains XML elements describing embedded devices. Communication with this repository will be achieved via a Web service.
4) Brick Catalogue: This is, Schneider Electric internal repository used by all Schneider Electric business units containing software unit.

The third element (3) in the experiment's design is the Prometheus Server. This comprises the core of the technical structure. The server maintains information about software units and software construction activities in the connected databases and makes this information available to the user. Finally, the Prometheus Server performs requested activities and presents the available results to users. The fourth element (4) is a website through, which the user can communicate with the Prometheus Server. The website runs on a further server and contains a web application giving the user the ability to query information from the server or to perform reuse activities on the server. This web application is named 'Ecostruxure repository' and for this experiment the 4.1 version was used. The basic technology of the Web application is Microsoft Silverlight version 4.0. The website used the endpoint '/RepositorySearch.html' and was available within the company's intranet. The fifth element (5) of the structure is a VM-Ware server. This server is used to fulfil the experiment's required operating system environment and runs as a virtual machine (VM) to make this available. For the connection to the server VM-Ware Workstation software with version 8.0 was installed on a laptop (6). These elements are common office laptops used within the company Schneider Electric. The laptops were used with the VM-Ware Workstation software with version 8.0. In addition to the computer network environment, there is the possibility to use telephone, internet, voice, conversation, or literature. This is also reflected in the working environment within the company's sites.

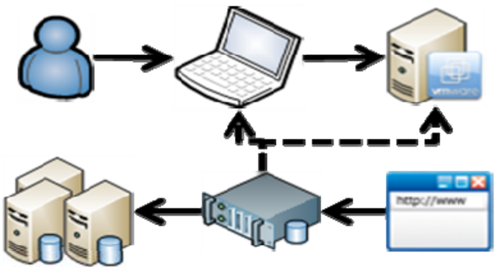


Figure 5. - Basic experiment scenario

Figure 4 shows the scenario based on the experimental setup. Users are able to view the test environment (operating system, the virtual machine) from element (5) (VM-Ware server) by using element (6) (office laptop). Within this test environment, all necessary software applications are found by means searching for information on the Internet, or performing activities on the intranet, as well as various means of communication usually employed by Schneider Electric (FTP, Skype, TELNET). Furthermore, users can now click element (4) (the website) to access and use the Web application, which allows communication with element (3) (Prometheus server). The Prometheus server communicates with the databases that are marked as element (2). Also the Prometheus server interacts with the elements (5) (VM-Ware server) by using element (6) (office laptop) (see Section III). Figure 5 shows this interaction scenario.

Figure 6 shows the different measurement variants in the experimental setup. This can be accomplished by three different (technical) variants. The first is the purely visual recognition of the user's actions and does not require any technical measure (called 'Observer'). The second is to record the user's interactions with the virtual machine as video recording (called 'Recording'). For this, the installed VM Ware Workstation software with version 8.0 is used, which already includes the feature of video recording. The third variant is to log the information (called 'Logging'). This is done in three elements of the experiment's design:

- Create the user data in virtual machines. These data can be analysed after the experiment.
- The Prometheus Server attracts all incoming server requests and performed activities. This information can also be queried after the end of the experiment and used for analysis.
- The data and information are generated and stored in the databases through the interaction of the user.

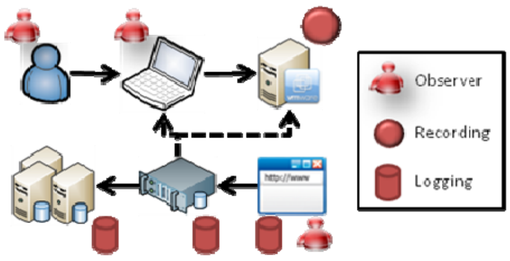


Figure 6. Overview measurement utilities

Description of the technical setup for the measurement and the measurement process itself:

(1) Experimental groups and scenarios: There are a total of three experimental groups: the first group (1) consists of experts for one particular software unit. These individuals receive expert status either because they have created this software unit or are well acquainted with its use. The selection of experts is performed via the Internet from public data of Schneider Electric software units. These data also contain the contact person responsible for this software unit. These people are also asked directly whether they have created the software unit and / or have used it frequently. Altogether the study requires 5 experts. The second experimental group (2) consists of 10 software engineers with the following characteristics: first, the people should actively participate in the software development of a project at the time the experiment takes place. On the other hand, it is important that these people do not have the same expert status as the previously selected 5. The last criterion is that these people are neither expert in the software unit nor in the technology standard development platform for this unit.

The third group (3) is similar to the second experimental group and consist of 10 participants. Therefore, the same rules used for selection of the second experimental group apply.

Note: In this the next phase of the experiment, the total number of participants will be increased up to 30 per group.

Procedure: In principle, there are 3 different experimental groups required to perform seven scenarios. Table 1 shows the different scenarios related to the different groups.

TABLE I. SCENARIOS OF THE EXPERIMENT

Scenario	**Description / (GroupID)**
(1) Observation of experts	The experts from experimental group (1) performs transformation and / or integration activities (manually). / **(1)**
(2) Collection of software units and activities	Collection of software units and activities: In this scenario, each of the selected experts from experimental group (1) insert the knowledge about the unit and the specific transformation and, or integration activity into the Prometheus Server./ **(1)**
(3) Prometheus Validation	The experts perform the same activities as in scenario (1) but now with Prometheus Server support. The expert validates the results. / **(1)**
(4) Reuse activities with Prometheus	Participants from the group (2) are asked to take over one transformation and integration task. They have to use the Prometheus Server for this purpose. / **(2)**
(5) Reuse activities without Prometheus	In this scenario, the people placed in the experimental group (3) are asked to take over a transformation or integration task. Activities are repeated so they correspond to those of the experts from scenario (1), The Prometheus Server is not

	used / **(3)**
(6)/(7) Validation of the results	Validation of the results: This scenario will test the results of the experimental group (2) and (3) by the experts for the respective software unit from experimental group (1) and (2). / **(1)**

(2) Measurement

In the following section, the methodology of measurement of the experiment will be explained. This includes the definition of the measurable variables and the process of measuring.

Definition of variables: The results of the measurement procedures are stored in the form of variables. In addition, each variable is assigned a unique name within the experiment. In this section, all variables are named and briefly presented. Table 2 shows the different measurable variables in the different scenarios.

TABLE II. OVERVIEW OF VARIABLES

Sc. ID / ID	**Name: Description**
(1,3,4, 5)/ (A)	**ActivityDuration**: How long does it take an expert/user to perform an activity? This variable contains a value that expresses how long the expert takes for the preservation of the task.
(1,3,4, 5)/(B)	**TaskAnalysisActivityDuration**: How long did it take the expert/user to analyse the task initials? This variable describes the time between being presented with the task and the start of work on the computer.
(1,3,4, 5)/(C)	**TaskActivityDuration**: How much time does expert/user spend working on the computer in order to perform the activity? This variable describes the time between the start and completion of work on the computer activity.
(1,3,4, 5)/(D)	**ActivityCarriedOutSuccessfully**: Has the expert/user completed the activity successfully? This variable represents whether an activity was successful or not.
(1,3,4, 5)/(E)	**UseKnowledgeSources**: What kind of knowledge sources did the expert/user use to perform the activity? This variable describes the sources consulted to perform the activity such as the Google phone or contacting another expert for information.
(1,3,4, 5)/(F)	**MadeSubTasks**: What sub tasks did the expert undertake in order to perform an activity?
(2)/(G)	**EnterUnitDuration**: How long does it take the user to enter all necessary information about a software unit into the Prometheus system? This variable contains a value of the expert testimony of how much time was needed from commencing work on the computer to enter the information of its software unit.
(2)/(H)	**EnterActivityDuration**: How long does the expert take to enter an activity for a software unit in the Prometheus system? This variable contains a value of the experts' statement of how long since commencing work on the computer it took to input the specific activity of entering the activities information.
(2)/(I)	**TotalInputDuration**: How long does it take the expert to enter all the information into the Prometheus system? This variable contains a value of expert testimony on how long the whole process of entering all their data took.
(2)/(J)	**SuccessfulEntry**: Could the expert enter all the important information? This variable tells us whether an expert could enter all the information about a software module and complete activities in the system.
(2)/(K)	**MadeSubTasks**: What sub tasks did the expert undertake in order to perform an activity?
(3,6,7)/ (L)	**ResultIsValid**: Is the result of an activity conducted by Prometheus or without equivalent to the result of the same activity conducted by an expert? This variable indicates whether the expert considers the result of activities performed by Prometheus or without it as good as the result, which was achieved through manual execution of the same activity.

Measurement Execution Process: In Figure 6, three variants of measurement used to measure the variables were introduced. The following section shows, which of these techniques are used for the different variables.

In Scenarios (1), (3), (4), and (5), seven measurements are raised per cycle: (A) The variable 'ActivityDuration' is measured by the observer (measurement variant 1). Here, the observer measures from the time, which he assigns the task to the expert/user up to the time the expert says the task was completed. The time is recorded in whole minutes. (B) The variable 'TaskAnalysisActivityDuration' is determined by the interaction of measurement variant (1) and (2). Here, the observer notes the time at which the task is assigned to the expert/user (see variable 'ActivityDuration'). The end of this phase can be measured at the time when the expert commences an activity on the virtual machine. The time is recorded in whole minutes. (C) The variable activity of 'TaskActivityDuration' determines the interaction of the measurement variants (2) and (1). The point in time at which the activity is started on the virtual machine is measured. The endpoint is the time the expert/user tells the observer that the task was completed. The time is recorded in whole minutes. (E) The variable 'UseKnowledgeSources' is determined by the measurement variants (1) and (2). The observer notes all information coming from the expert's behaviour that cannot be measured by measurement variant (2). The type of measurement (2) also used to analyse, which sources of information accessed through the use of the virtual machine. Typically such sources can be classified by using source names and the type of resource, e.g., (1) co-worker, telephone, and (2) website, Google (Web browser). (D) The variable 'ActivityCarriedOutSuccessfully' is measured by measurement variant (1). The expert/user is asked after the completion of the activity if he has done this successfully. The variable can only be set to yes or no. (F) The variable 'MadeSubTasks' is determined by the measurement variants (1) and (2). Here, the observer notes the progress of the entire task. This can be done based on the recording of the activities in the virtual machine itself, which is operated by the observer both on the external (outside the virtual machine) and internal (within the virtual machine) view. The observer here notes, which activities were measurable, including their start and end time, e.g., starts 10:41 expert uses web browser.

In scenario (2), five measurements are made: (G) The variable input 'EnterUnitDuration' determines the measurement variants (1) and (3). The website (see Figure

4) logs every activity of the user. Accordingly, the entry of the website is the start time and represents the initial value used for the measurement. To avoid error, the observer compares measured time with the automatically measured time. The end time is determined by the expert's signal indicating that he/she has to finish the task. The observer notes down this time. Time is measured in whole minutes. (H) The variable 'EnterActivityduration' is measured by the measurement variant (3) on the Prometheus Server (see Figure 4) and the website (see Figure 4). The server and the website recognize the time of a user's request. Each measurement contains the time and the names of tasks, e.g., 10:00:00 user creates a new software unit. (I) The variable 'EnterActivityDuration' is measured by the measurement variant (1). The observer records the start time point at which he/she hands over the task to the experts. The end time is determined by the expert's signal that he/she has finished the task. The observers take note of this point in time. Time is measured in whole minutes. (J) The variable 'SuccessfulEntry' is measured with the measured variants (1) and (3). Firstly, the expert must inform the observer that he/she was able to enter all information into the system. Secondly, the Prometheus server writes all values into the database. The variable can only be set to yes or no. (K) The variable 'MadeSubTasks' is measured in the same way than in Scenario (1,3,4,5)/(F).

In scenarios (4), (6), and (7) one measurement is made: (L) The variable 'ResultIsValid' is captured by the measurement variant (1). The expert examined the results of the performed activity from the scenarios (3), (4), and (5) with the same activity carried out in scenario (1). It tells the observer whether the result has the same value and is usable. The variable can only be set to 'yes' or 'no'.

Definition of Software units and reuse activities: The different scenarios 1-7 are performed in this experiment with the software units shown in Table 3.

TABLE III. USED SOFTWARE UNITS

Name / ID	Description	Tec/ Unit Type / Repository	Integration effort / Transformation effort
DPWS / SU1	Enable devices for WS* profiles	Java / Component / SOA4D	Advanced into Eclipse/Advanced using IKVM
DPWS / SU2	Enable devices for WS* profiles	C++ / Component / SOA4D	Advanced into Visual Studio / None
CWS / SU3	Webservice for data exchange of business units	Soap-C# / Webservice / Prometheus	Normal into Visual Studio / Advanced using SVCUtil
CWS / SU4	Webservice for data exchange of BUs	Java-Android / Class / Prometheus	Advanced into Eclipse / Advanced using Java2SOAP
Code Signing / SU5	Webservice for Code signing	Soap-C# / Webservice / Brick Repos.	Normal into Visual Studio / Normal using SVCUtil

Table 3 shows that five integration and four transformation activities are connected with the five software units. The integration activities typically focus integration of software units on the most common IDEs (Visual Studio and Eclipse). The transformation activities include the transformation of software units on three different transformation tools (IKVM [16], SVCUtil [17] and WSDL2Soap [18]

VI. EXPERIMENT RESULT DISCUSSION

A. Experiment Results

The experiment's results were collected in the way described in the previous section. The next step is to discuss these results. First of all, the result of one software unit with a transformation activity will be discussed in more detail. After this analysis, the results of all software units will be summarised and compared. For this purpose, two perspectives were used for analysing the summarised results: Comparing different groups from the perspectives of (1) activity execution and (2) use of knowledge.

1) Detailed result example

One of the measured software unit is the 'Device Profile for WebServices' Java stack, which enables Java based embedded devices to handle mutable WS* Protocols like WebService discovery. The transformation task for this software unit was to use IKVM transformation tool to transform the complete DPWS Java Stack into a C# Stack. This task requires knowledge about the DPWS Java Stack (especially the references of the 20 different JAR Files), the .NET Platform and experience in using IKVM. This scenario was taken from a real development scenario of Schneider Electric in the European research project for industrial automation SOCRADES [19].

Expert scenarios (1-3): Scenario 1: In the first scenario, the Expert was measured by performing this task manually. The main result is that the experts needs 14:23 min.. In Scenario 2 it was measured how long the expert needs to insert the software unit and the transformation activity. The initial creation of the software unit into Prometheus needs 12:06 min. and the transformation needs 38:03 min.. In Scenario 3, the expert was observed by using the Prometheus Server to perform this task. He needs 2:04 min. to perform the task and received a 2:56 min. training into the system (this training will only be necessary once per expert). The expert validated the result as a correct transformation.

Non-expert scenarios (4-5): In Scenario 4, five non-experienced software engineers of the industrial areas of Building, Power and Industry (Automation) did the task without support of the Prometheus Server.

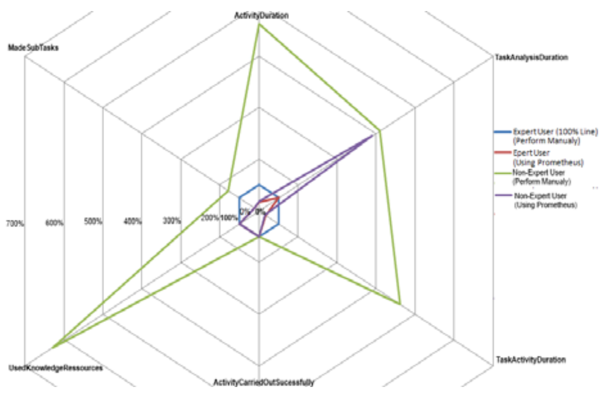


Figure 7. Results of the different groups for DPWS transformation ctivities

The different participants need 42 min., 90 min., 77 min., 69 min., and 104 min. (rounded off). Thus, the average time was 76 min. (rounded off). The expert validates all final results as valid. In Scenario 5 the participants of group (3) use Prometheus to perform the task. The measured introduction task performing times (in minutes) were (3:03/2:23), (2:56/2:10), (2:33/1:59), (2:45/2:22), and (2:43/2:23). The average time was (2:48/2:18). The expert validates the results as correct results. Figure 7 summaries the results. The validation in Scenario 6 and 7 are not shown in Figure 7 because of all results were valid. Additionally to the measured time the kind of used knowledge resources were measured. Only online websites, downloaded documentation, and the expert were used as knowledge resource. The expert in scenario 1 uses only one knowledge resource (an older development project) 4 times. By adding the necessary information into the Prometheus system of Scenario 2 the expert only uses one knowledge resource (the introduction). In Scenario 3, the experts need only the introduction to perform the activity. The non-expert group (2) of scenario 4 needs multiple resources multiple times. Figure 8 shows the used number of knowledge resources in each scenario (average values).

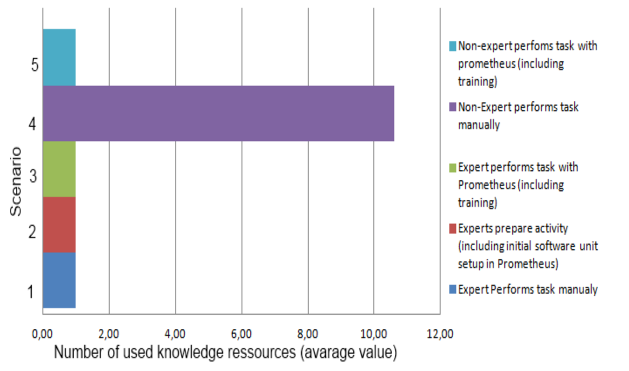


Figure 8. Overview of number of used knowledge resources

The non-expert group (3) of scenario 5 needs only one knowledge resource (the introduction).

2) Comparing of different groups from the perspective of activity execution

Figure 9 and Figure 10 show the results of the three groups in transformation and integration activities measured in the Scenarios 1, 3, 4, and 5. The different results of the software units are summarised by using this type of view. In the context of transformation, Figure 9 demonstrates a clear separation of the different groups. Starting with the Expert Users without Prometheus support (Expert, Scenario 1) as the 100% comparison line, the

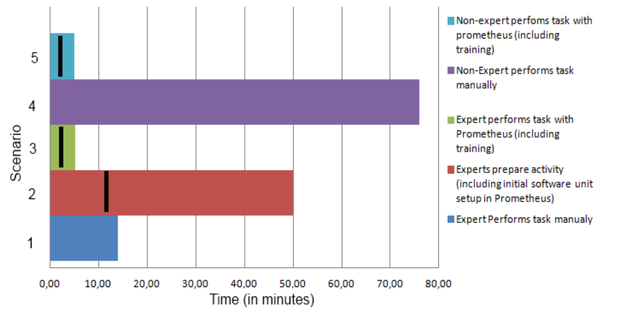


Figure 9. Results of the different groups for transformation activities (5 software units)

measured values of the second group (User with Prometheus support – User (P)) are significantly decreased. This fact is mentioned especially in the variable ‘ActivityDuration’ (1). On the other hand, the Variable ‘TaskAnalysis-ActivityDuration’ (2) is much closer to the comparison line. As a result, Prometheus Users are able to perform a specific activity much faster than an expert user or a Non-Expert user. In comparing the two variables of the comparison line with user (without Prometheus support User) Figure 9 shows a further significant difference. Both variables of the user are decreased. The normal user needed much more time to fulfill the given tasks. But this difference changes by analyzing the results of users (with Prometheus support). Compared to the expert with Prometheus support this group has no significant differences, but compared to the expert group without Prometheus support the measured values decrease significantly. In Figure 9, the two lines of Prometheus supported users are more or less congruent.

As a result of this consideration, it is clear that the Prometheus approach creates a positive effect for Non-Expert User and even for expert users.

Figure 10 shows the measured values for the integration activity. The first interesting point is the general comparison to the results shown in Figure 9. Both pictures show nearly the same result, but the positive characteristics are not so distinct. Only the users (without Prometheus support) performing the integration activity need less time (compared to the 100% comparison line) then the same group was performing the transformation activity. That both results a nearly the same indicates that the used approach supports software engineers by performing these kinds of activities.

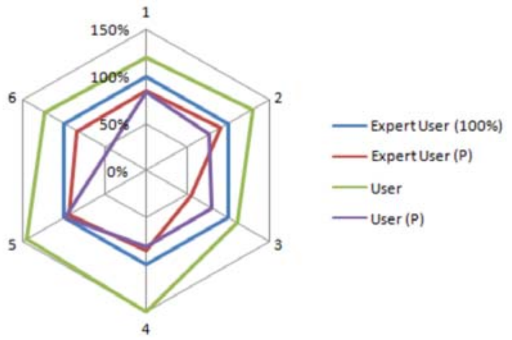


Figure 10. Results of the different groups for integration activities

All users (experts and non-expert user) were able to perform the given activities correctly and needed less time than the expert user (without Prometheus support).

3) Comparing of different groups from the perspective of the use of knowledge

In Figure 9 and Figure 10, it is also mention that most of the expert users (80%) (without Prometheus support) did not use a measurable knowledge base. The other 20% used exactly one knowledge base. All experts or users (with Prometheus support) only used the knowledge base that was the documentation of the Prometheus system. The users (without Prometheus support) performing both the transformation and the integration activity used much more knowledge bases. The most used knowledge base was the internet.

B. Impacts on industrial reuse

In applying the aforementioned approach to industrial environments faced with both creator and reuse phase dilemmas, and therefore no knowledge transfer, leads to the following effect, shown in Figure 11: The effort for the creation team increases by adding the software unit information into the Prometheus system. The theoretical very useful but missing support effort is mostly replaced by the effort for this ‘knowledge injection’.

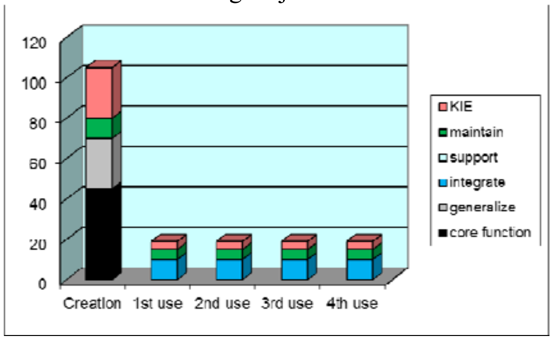


Figure 11. Effects on MTwKIE

The major effect is visible at the reuse site. Even without or just less support, the effort for reuse for single users or team is significantly reduced. In the case of this experiment the reduction of the measured variable are ~38,5% in the transformation activity case compared to the expert user (perform manually) (see Figure 9), ~ 73,21% in the transformation activity case compared to the non-expert user (perform manually) (see Figure 9), ~38,5% in the integration activity case compared to the expert user (perform manually) (see Figure 10), and ~ 73,21% in the integration activity case compared to the non-expert user (perform manually) (see Figure 10). This is mainly based on the fact, that expert and non-expert Prometheus users do not spend much time in searching a software unit and preparation/execute a specific reuse task. The same positive effect is expected in the reuse of a software unit multiple teams of different business units. The approach detailed in this paper has two positive effects. First of all, the solution is sustainable for all teams as it is available to all once it has been stored in the system. This is shown by using different participants from different business units. As consequents, all teams will obtain the same result and the same effects described in Figure 9 and 10. Therefore, the way of reuse planned in the creation phase is more sufficient. The second positive effect is the adaptation towards knowledge created in the “reuse” steps. If a team recognizes an alternative way to perform the reuse activities it is able to store this knowledge in the system. This requires training for the use of the Prometheus system, but other teams are now able to decide, which kind of transformation rule they want to use. (Reuser is Creator) Figure 11 shows both positive effects.

VII. CONCLUSION AND FUTURE WORK

The reuse of a software unit consists of different reuse activities. To perform such activities knowledge is required. Especially in an industrial environment this constitutes problem for a single team and in different teams of different business units. This paper shows the structure and result of an experiment aiming to demonstrate that it is possible to automate chosen reuse activities so that less experienced users are able to perform the activities. By comparing a group of software unit experts, a group of less experienced users within a normal development environment, and a group of less experienced users with the support of the focused automation approach following results are obtained: (1) It is possible to automate reuse activities. Expert users store their knowledge into a system, which is then able to perform the activity (knowledge extraction). (2) Less experienced users who are normally unable to perform such activities are now able to do this. (knowledge injection) (3) Analysing of the results demonstrated that this approach has positive effects for reuse of software units in industrial environments. (4) With automated support, a single team can decrease their reuse costs from the first time of reuse and thereby make it sustainable. Users utilizing the new approach are able to perform an activity faster than the software unit expert because the system provides the complete environment for the activity based on the expert users' knowledge. (5) By reusing the expert's knowledge, the variations are minimized. All teams use the same activity based on the same knowledge. (6) New automated activities are sustainable because the activity will be changed or a new one is stored in the system, therefore it can be used in each new reuse step of each team. Next to the positive effects, this paper's experiment is limited to two software reuse activities: Transformation and Integration. These activities were chosen because they require different amount of knowledge about tools, environment, and software units. But there also other reuse activities like test, validation, and deployment. Especially for deployment, for example on embedded devices, knowledge is required, but not all activities may be automated completely. The next step is the phase two of the experiment. The number of software units is raised to 10 and the number of inexperienced software engineers in the groups 2 and 3 is increased up.

Next to the fact that the results have to be confirmed by repeating the experiment with new software units and other software engineer the process has to be proofed by other companies. For that purpose the process of the experiment has to be formulated in a formal way. Additionally the following aspects are interesting for the future.

Horizontal extension of the research field: The concept presented in this work was demonstrated by using the example of integration and transformation. But, much more than the activities made use of in this experiment still exist in the area of software unit reuse. First, standard activities exist such as testing and validation of interfaces. These activities usually have a high degree of automation. However, these approaches are lacking in one approach, which is used to represent knowledge uniformly and then re-applied to the different existing automation systems. The scientific task is thus to consider whether the approach presented in this work can also be used for other horizontal activities. On the other hand, technological progress can ensure new activities in the area of reuse. The scientific problem in this case is to check whether the approach presented in this work is can also be used for new activities.

VIII. REFERENCES

[1] I. Sommerville, Software engineering, Pearson, 2011.

[2] F. Bobillo, M. Delgado, and J. Gómez-Romero, "Representation of context-dependant knowledge in ontologies: A model and an application," Expert Systems with Applications, vol. 35, no. 4, pp. 1899–1908, 2008.

[3] N. Juristo and A. M. Moreno, "Reliable knowledge for software development," IEEE Software, vol. 19, no. 5, pp. 98–99, 2002.

[4] R. Oliveto, G. Antoniol, A. Marcus, and J. Hayes, "Software Artefact Traceability: the Never-Ending Challenge,", pp. 485–488, 2007.

[5] M. Zinn, "Service based software construction process," in Proceedings of the Third Collaborative, Plymouth, UK, pp. 169–184, 2007.

[6] M. Zinn, G. Turetschek, and A. D. Phippen, "Definition of software construction artefacts for software construction," in In proceedings of the, pp. 79–91, 2008.

[7] J. Bosch and P. Bosch-Sijtsema, "From integration to composition: On the impact of software product lines, global development and ecosystems," Journal of Systems and Software, vol. 83, no. 1, pp. 67–76, 2010.

[8] V. C. Garcia, E. S. de Almeida, L. B. Lisboa, A. C. Martins, S. R. L. Meira, D. Lucredio, and R. P. de M. Fortes, "Toward a Code Search Engine Based on the State-of-Art and Practice,", 13th Asia Pacific Software Engineering Conference (APSEC'06), Bangalore, India, pp. 61–70, 2006

[9] T. Mens and P. Vangorp, "A Taxonomy of Model Transformation," Electronic Notes in Theoretical Computer Science, vol. 152, pp. 125–142, 2006.

[10] R. Stair and G. Raynolds, Principles of information systems, 10th ed. Boston Mass.: Course Technology Cengage Learning, 2011.

[11] T. Davenport, Working knowledge : how organizations manage what they know, Harvard Business School Press, 2000.

[12] A. Kleppe, MDA explained : the model driven architecture : practice and promise., Addison-Wesley, 2003.

[13] K. Czarnecki, Generative programming : methods, tools, and applications., Addison Wesley, 2000.

[14] M. Zinn, K. P. Fischer-Hellmann, and R. Schoop, "Reuseable Software Unit Knowledge for Device Deployment," presented at the Entwurf komplexer Automatisierungssysteme (EKA 2012), 2012.

[15] M. Zinn, K. P. Fischer-Hellmann. "Reusable Software Units Integration Knowledge in a Distributed Development Environment," International Workshop on Software Knowledge (SKY'11), pp. 24–35, 2011.

[16] J. Frijters, "IKVM," IKVM.NET Home Page, [Online], http://www.ikvm.net/. [retrieved: 09,2012].

[17] Microsoft, "ServiceModel Metadata Utility-Tool,"[Online], http://msdn.microsoft.com, [retrieved: 09,2012].

[18] Apache, "WebServices - Axis." [Online]. http://ws.apache.org /axis/java/user-guide.html, [retrieved: 09,2012].

[19] Socrades, "Socrades Website", [Online], http://www.socr ades.org [retrieved: 09,2012].

Learning Best K analogies from Data Distribution for Case-Based Software Effort Estimation

Mohammad Azzeh, Yousef Elsheikh
Faculty of Information Technology
Applied Science University
Amman, Jordan
e-mail: {m.y.azzeh, y_elsheikh}@asu.edu.jo

Abstract— **Case-Based Reasoning (CBR) has been widely used to generate good software effort estimates. The predictive performance of CBR is a dataset dependent and subject to extremely large space of configuration possibilities. Regardless of the type of adaptation technique, deciding on the optimal number of similar cases to be used before applying CBR is a key challenge. In this paper we propose a new technique based on Bisecting k-medoids clustering algorithm to better understanding the structure of a dataset and discovering the optimal cases for each individual project by excluding irrelevant cases. Results obtained showed that understanding of the data characteristic prior prediction stage can help in automatically finding the best number of cases for each test project. Performance figures of the proposed estimation method are better than those of other regular K-based CBR methods.**

Keywords- Software Effort Estimation; Case-Based Reasoning; Adjustment Techniques.

I. Introduction

Estimating the likely software project effort is a vital task for project planning, control and assigning resources [5, 23, 26, 28]. Although a variety of software effort estimation models have been proposed so far, the Case Based Reasoning (CBR) method is still the most widely investigated method. CBR is a knowledge management method based on premise that history almost repeats itself which leads to problem solving can be based upon retrieval by similarity [30]. It has been favored over regression techniques since software datasets often exhibit complex structure with a lot of discontinuities [2, 6, 20, 21].

The predictive performance of CBR suffers from common problems such as very large performance deviations as well as being highly dataset dependent. This is due to a large space of configuration possibilities and design decisions induced for each individual dataset [16]. Recent publications reported the importance of discovering the optimal K closest cases for generating better estimates in CBR [11, 27]. Conventional K-based CBR methods start with a single analogy and increase this number depending on the overall performance of the whole dataset then it uses the K value that produces the overall best performance. However, a fixed K value that produces overall best performance does not necessarily provide the best performance for individual projects. Our claim is that we can avoid sticking to a fixed best performing number of cases which changes from dataset to dataset or even from a single project to another within the same dataset. We propose an alternative technique to calibrate CBR by using Bisecting k-medoids (BK) clustering algorithm. The k-medoids is a clustering algorithm related to the centroid-based algorithms which groups similar individual instances within a dataset into N clusters known a priori [29, 30]. This enables us to discover the structure of dataset efficiently and automatically come up with the best number of K closest cases as well as excluding irrelevant cases for each individual test instance.

The rest of the article is organized as follows: Section 2 provides the Background of Case-Based Effort Estimation. Section 3 defines the Research question and introduces main problem. Section 4 presents the proposed technique. Section 5 presents experimental design. Section 6 presents the results we obtained. Section 7 presents threats to validity of this study. Lastly, Section 8 summarizes our conclusions and future work.

II. Background

Case-Based effort estimation is a variant of CBR which makes prediction for a new project by retrieving previously completed successful projects that have been encountered and remembered as historical projects [12, 13]. The data driven CBR method involves four major stages [25]: (1) retrieve the most similar training projects using Euclidean distance function as depicted in Eq. 1. Then (2) reuse the past solutions from the set of retrieved analogues to solve the new problem. (3) revise the proposed solution and to better adapt the target problem. Finally, (4) retain the solved problem for future problem solving.

$$d(p_i, p_j) = \frac{1}{m}\sqrt{\sum_{t=1}^{m} \Delta(p_{it}, p_{jt})} \tag{1}$$

where d is the similarity measure. m is the number of predictor features, t is the index of feature, p_i and p_j are projects under investigation and:

$$\Delta(p_{it}, p_{jt}) = \begin{cases} \frac{(p_{it} - p_{jt})^2}{\max_t - \min_t} & \text{if } t \text{ is continuous} \\ 0 & \text{if } t \text{ is categorical and } p_{it} = p_{jt} \\ 1 & \text{if } t \text{ is categorical and } p_{it} \neq p_{jt} \end{cases} \tag{2}$$

Although CBR generates successful performance figures in certain datasets, it still suffers from local tuning problems when they were to be applied in another setting [3]. Local tuning requires mainly learning appropriate K cases that fits procedure of adjustment and reflects dataset characteristics [16]. The classical approach uses a fixed number of cases (K=1, or 2 or…etc.) for all test projects, which is somewhat considered simpler but it relies heavily on the estimator intuitions [3]. In this direction, Kirsopp et al. [15] proposed making predictions from the K=2 nearest cases as it was found as the best value for their investigated datasets. In a further study Kirsopp et al. have increased their accuracy values with case and feature subset selection strategies. On the other hand, Idri et al. [9] proposed using all projects that fall within a certain similarity threshold. This approach could ignore some useful projects which might contribute better when similarity between selected and unselected cases is negligible. Li et al. [17] performed rigorous trials on actual and artificial datasets and they observed effect of various K values. However, we believe that reflection on dataset prior applying to different algorithms under multiple settings is of more significance. But, this is not enough because the selection of K cases is not only a dataset dependent but also adjustment method dependent. In this study we focus only on discovering the best number of cases to be used for each individual test project from the characteristics of the dataset.

III. Research Questions

Finding the appropriate number of cases to be used in CBR is a challenge on its own, and has a strong impact on the overall predictive performance. Conventional K-based CBR methods start with a single analogy and increase this number depending on the overall performance of the whole dataset then it uses the K value that produces the overall best performance. However, a fixed K value that produces overall best performance does not necessarily provide the best performance for individual projects. Furthermore, previous studies reported that the proper selection of K cases is a dataset dependent and subject to underlying distribution of the dataset [16]. For these reasons, we propose a new technique based on Bisecting k-medoids clustering algorithm to find the optimal number of cases to tune and configure CBR method. To the best of our knowledge, it has not been used previously in software effort estimation domain. Unlike regular K-based CBR methods, the proposed technique starts with all projects in the train dataset and gradually excludes irrelevant projects on the basis of compactness degree. The proposed work attempts to answer the following research questions:

1. How can we better understand the characteristics of a particular dataset and dynamically come up with optimum K number of analogies?
2. Does the performance of CBR improve with automatic dynamic selection of K cases for each individual project?

IV. The Proposed CBR based Bisecting k-medoids algorithm CBR(BK)

The k-medoids is a clustering algorithm related to the centroid-based algorithms which groups similar individual instances within a dataset into N clusters known a priori [29, 30]. It is more robust to noise and outliers as compared to k-means because it minimizes the sum of pairwise dissimilarities instead of a sum of squared Euclidean distances. A medoid can be defined as the instance of a cluster, whose average dissimilarity to all the instances in the cluster is minimal i.e. it is a most centrally located point in the cluster. The popularity of making use of k-medoids clustering is its ability to use arbitrary dissimilarity or distances functions, which also makes it an appealing choice of clustering method for software effort data as software effort datasets also exhibit very dissimilar characteristics.

Regardless of the k-medoids algorithm advantages it still has some challenges such as guessing the number of clusters that can be used to find the partitions that best fits the underlying data [30]. To avoid this challenge we employed bisecting procedure with k-medoids algorithm and propose Bisecting k-medoids algorithm (BK). BK is a variant of k-medoids algorithm that can produce hierarchical clustering by recursively applying the basic k-medoids. It starts by considering the whole dataset to be one cluster. At each step, one cluster is selected and bisected further into two sub clusters using the basic k-medoids. Note that by recursively using a bisecting k-medoids clustering procedure, the dataset can be partitioned into any given number of clusters in which the so-obtained clusters are structured as a hierarchical binary tree. The decision whether to continue clustering or stop it depends on the comparison of compactness degree between childes and their direct parent in the tree. If the maximum of compactness of child clusters is smaller than compactness of their direct parent then clustering is continued. Otherwise it is stopped and the parent cluster is considered as a leaf node. This criterion enables the BK to uniformly partition the dataset into homogenous clusters. In this paper the average cluster compactness as a measure of homogeneity of each cluster is used, it is defined as:

$$Compactness= \frac{1}{n}\sum_{i=1}^{k}\sum_{j=1,x_j\in C_i}^{n}\left\|x_j - v_i\right\|^2 \qquad (3)$$

where $\| \bullet \|$ is the usual Euclidean norm, x_j is the j^{th} data object, v_i is the center of i^{th} cluster (C_i) and k is the number of clusters. A smaller value of this measure indicates a high homogeneity (less scattering).

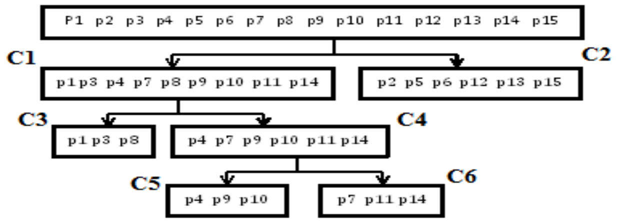

P1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14 p15
C1
p1 p3 p4 p7 p8 p9 p10 p11 p14
C2
p2 p5 p6 p12 p13 p15
C3
p1 p3 p8
C4
p4 p7 p9 p10 p11 p14
C5
p4 p9 p10
C6
p7 p11 p14

Figure 1. Illustration of Bisecting k-medoids algorithm

Figure 1 is a very simple BK tree that can be formed on a simple dataset of 15 projects. It shows how the main cluster is bisected recursively into four leaf clusters are: C2, C3, C5 and C6. Tricky point here is that unlike K-based CBR methods, BK does not need any expert interference to discover dataset characteristics so as to decide on the number of trees to be built or the number of cases to be used in estimation. To better understand the BK algorithm, see the pseudo code in Figure 2.

```
1: Input: The dataset X
2: Output: The set of k clusters S={C1, C2, C3, C4, ..Ck}
3: Initialization: Let V=X , S={}, NextLevl={}
4: Repeat while size(V)> 0
5:  foreach Cluster C in V
6:        Comp ← compactness (C)
7:        [C1,C2] ←  k-medoids(C,2)
8;        Comp1 ← compactness(C1)
9:        Comp2 ← compactness(C2)
10:       If(max(Comp1,Comp2)<Comp)
11:              NextLevel ← NextLevel ∪ {C1,C2}
12:       Else
13:              S ← S ∪ {C}
14:       End
15:       V ← NextLevel
16:       NextLevel ← {}
17: End
```

Figure 2. Bisecting k-medoids algorithm

Finally once BK tree is built, the estimation process starts. The un-weighted mean effort of the train projects of the leaf cluster whose medoid is closest to the test project becomes the estimated effort value for that test project as shown in Eq. 4. i.e. we choose to use K-many cases for estimation where K is the number of train instances that are in the selected cluster.

$$Effort(p_t) = \frac{1}{K}\sum_{i=1}^{K} Effort(p_i) \tag{4}$$

V. EXPERIMENTAL DESIGN

As it was reported [16] most of the methods in literature were tested on a single or a very limited number datasets, thereby reducing the credibility of the proposed method. To avoid this pitfall, we included nine datasets from two different sources namely PROMISE [4] and ISBSG [10]. PROMISE is an on-line publically available data repository and it consists of datasets donated by various researchers around the world. The datasets come from this source are: Desharnais [7], Kemerer [14], Albrecht [1], COCOMO [4], Maxwell [19], Telecom [4] and NASA93 [4] datasets. The other dataset comes from ISBSG data repository (release 10) which is a large data repository consists more than 4000 projects collected from different types of projects around the world. Since many projects have missing values only 500 projects with quality rating “A” are considered. 14 useful features were selected, 8 of which are numerical features and 6 of which are categorical features. The descriptive statistics of such datasets are summarized in Table 1.

TABLE 1 Statistical properties of the datasets

Dataset	**Cases #**	**Effort min**	**Effort max**	**Effort mean**
ISBSG	500	668	14938	2828.5
Desharnais	77	546	23940	5046.3
COCOMO	63	5.9	11400	683.5
Kemerer	15	23.2	1107.3	219.2
Albrecht	24	0.5	105.2	21.87
Maxwell	62	583	63694	8223.2
NASA93	18	8.4	824	624.4
China	499	26	54620	3921
Telecom	18	23.45	1115.5	284.3

For each dataset we follow the same testing strategy, we used Leave-one-out cross validation to identify test and train projects such that, in each run, we select one project as test set and the remaining projects as training set. This procedure is performed until all projects within dataset are used as test projects. In each run, The prediction accuracy of different techniques is assessed using MMRE, PRED(0.25) performance measure as shown in Eqs. 5 and 6. MMRE computes mean of the absolute percentage of error between actual and predicted project effort values. PRED(0.25) is used as a complementary criterion to count the percentage of estimates that fall within less than 0.25 of the actual values.

$$MMRE = \sum_{i=1}^{N} \frac{|Effort(p_i) - \overline{Effort(p_i)}|}{Effort(p_i)} \tag{5}$$

where $Effort(p_i)$ and $\overline{Effort(p_i)}$ are the actual value and predicted values of project p_i.

$$PRED(0.25) = \frac{\lambda}{N} \times 100 \tag{6}$$

where λ is the number of projects that have magnitude relative error less than 0.25, and N is the number of all observations. We also used Wilcoxon sum rank test to investigate the statistical significance of all the results, setting the confidence limit at 0.05. The Wilcoxon sum rank test is a nonparametric test that compares the medians of two samples. The reason behind using these tests is because all absolute residuals for all models used in this study were not normally distributed. In turn, the obtained results from the proposed approach have benchmarked to other regular *K*-based CBR methods that use a fixed number of *K* cases.

VI. RESULTS

A. Results for Research Question 1

This study explores the feasibility of learning best K analogy number from the dataset structure prior building

CBR method. The previous results and conclusions indicate that a single best performing K value that is producing the lowest MRE values for the whole dataset does not necessarily produce lowest MRE value for every single project. To illustrates our viewpoint and better understand this problem we carried out an extensive search to find the mean effort value of the best K number of analogies that produces lowest MRE value for every single test project as shown in Figure 3. For a dataset of size n, the best K value can range from 1 to n − 1. Since a few number of datasets were enough to illustrate our viewpoint, we selected 3 datasets that vary in the size (i.e. one small dataset (Albrecht), one medium (Maxwell) and one large (Desharnais)). Figure 3 shows the histogram of best selected K numbers for the three examined datasets, where x-axis represents K analogy number and y-axis represents frequency of K number (i.e. number of projects that chose that K value). It is clear that there is no global K number for all projects, for example in Albrecht dataset, two projects selected only the closest case (K=1), whilst four projects selected two closest cases (K=2) and so on. This indicates that every single project favors different number of closest analogies. The conclusion can be drawn here that using a fixed K number of cases for all test projects will far from optimum and there is provisional evidence that choosing of best K analogy for each individual project is relatively subject to data structure.

The conclusion can be drawn from previous empirical results that the optimum K number of analogies is not global and every project favors different K number. However, our first research question was how can we better understand the characteristics of a particular dataset and dynamically come up with optimum K number of analogies? In this paper we proposed Bisecting k-medoids algorithm to better understand the characteristics of software datasets and automatically come up with the optimum K number. To illustrate that, we executed CBR(BK) over all employed datasets and we recorded the best obtained K for every test project. Figure 4 shows the histogram of K number of analogies for every test project. This demonstrates the capability of BK technique to dynamically discovering the various K values for every test project that takes into account the characteristics of each dataset on the basis of compactness degree. The procedure of selecting has become easier than first (i.e. where the estimator intuition is heavily used to choose the optimum number of analogy) since the entire best K selection process has been left to the BK. The performance figures of the proposed technique are discussed in the next section.

B. *Results for Research Question 2*

The second research question was whether the predictive performance of CBR method can be improved when using BK algorithm? Apart from being able to choose the number of cases for each test instance on its own, BK outperforms all the other K-based CBR methods as can be seen in Table 2.

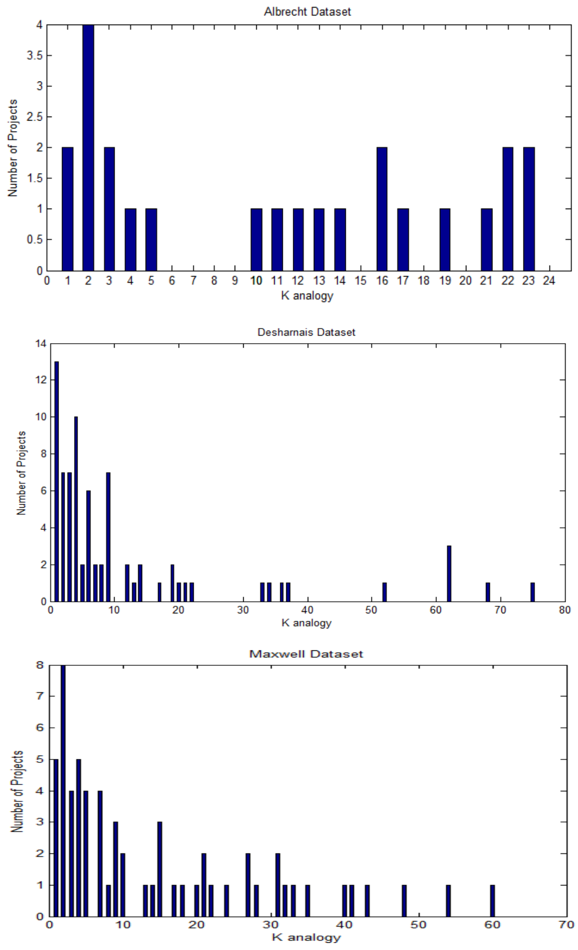


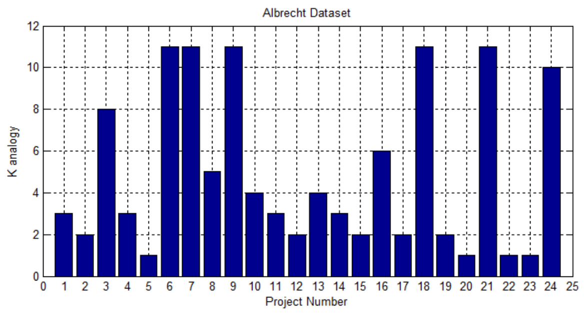
Figure 3. Distribution of K-cases

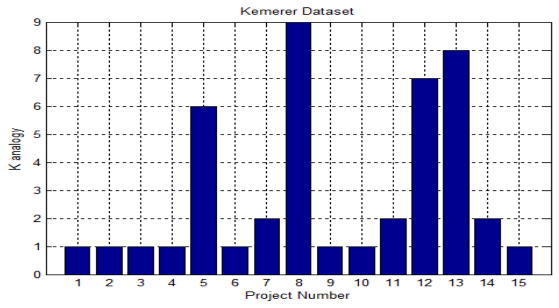
TABLE 2 MMRE results

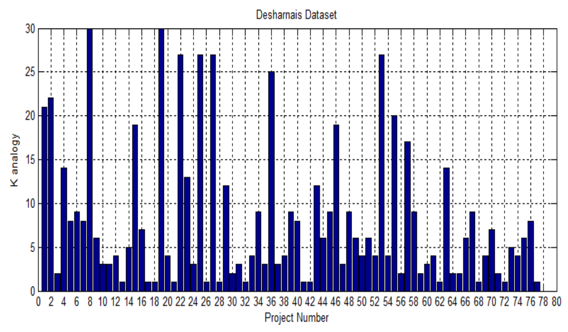
Dataset	CBR (BK)	CBR (K=1)	CBR (K=2)	CBR (K=4)	CBR (K=8)	CBR (K=16)
Albrecht	**45.4**	71.0	66.5	73.9	89.1	146.5
Kemerer	**41.9**	55.9	77.7	86.2	91.5	N/A
Desharnais	**29.4**	60.2	51.5	50.2	61.0	79.9
COCOMO	**60.43**	157.1	363.2	327.3	401.8	606.55
Maxwell	**41.3**	182.6	132.7	149.3	138.2	145.6
China	**27.7**	45.2	44.2	48.5	53.8	63.0
Telecom	**35.7**	60.0	45.2	77.4	115.3	175.3
ISBSG	**37.0**	72.6	73.2	74.7	71.7	71.7
NASA	**39.4**	81.2	97.5	77.6	77.1	227.6

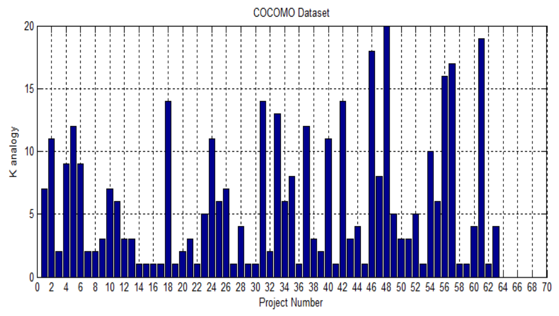
When we look closer at the MMRE values in Table 2, we can see that in all 9 datasets, BK has never been outperformed by other methods with the lowest MMRE values, which suggest that BK has attained better predictive performance values than all other regular K-based CBR methods. This also shows the capability of BK to support small-size datasets such as in Kemerer and Albrecht. However, although it proved inaccurate in this study, the strategy of using fixed K-analogy the effort values may be appropriate in situations where a potential analogues and target project are similar in size feature and other effort drivers. On the other hand, There may be little basis for believing that either increasing or decreasing the K-cases effort values of K-based CBR methods will not improve the accuracy of the estimation.

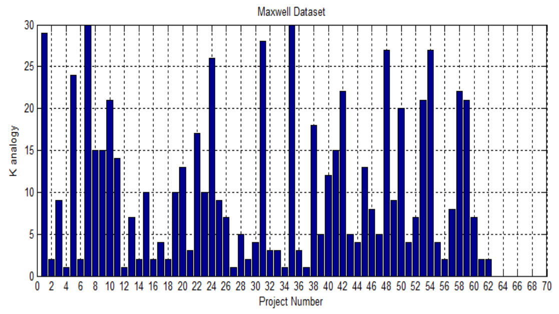
Table 3 shows that the proposed technique has achieved larger PRED values over eight datasets, which demonstrated that most of the predictions have very good accuracy with MRE vales are less than 0.25. However, overall results from Tables 2 and 3 revealed that there is reasonable believe that using dynamic K-cases for every test project has potential to improve prediction accuracy of CBR in terms of PRED. Concerning discontinuities in the dataset structure, there is clear evidence that the proposed method has capability to group similar projects together in the same cluster as appeared in the results of Maxwell, COCOMO, Kemerer and ISBSG.

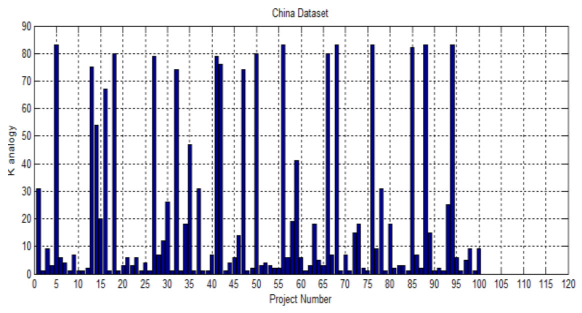


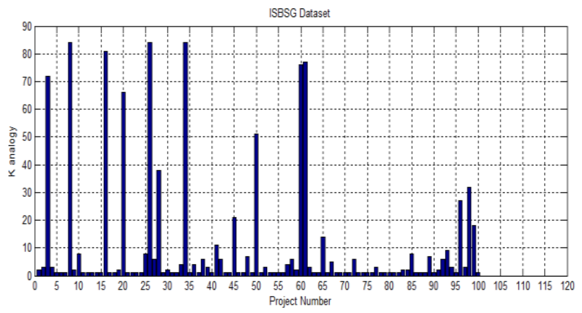












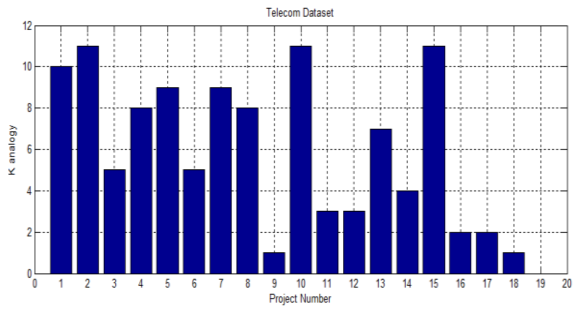


Figure 4. Histogram of K analogies obtained by CBR(BK) for all employed datasets

TABLE 3 PRED results

Dataset	CBR (BK)	CBR (K=1)	CBR (K=2)	CBR (K=4)	CBR (K=8)	CBR (K=16)
Albrecht	**40.8**	29.2	33.3	37.5	37.5	33.3
Kemerer	**43.3**	40.0	20.0	13.3	20	N/A
Desharnais	**40.3**	31.2	31.2	37.6	32.5	22.1
COCOMO	**19.3**	12.7	19.1	15.9	12.7	12.7
Maxwell	**22.6**	9.7	19.4	14.5	16.1	29
China	**52.7**	38.3	43.5	41.9	38.1	33.7
Telecom	**53.3**	33.3	50	44.4	38.9	22.2
ISBSG	38.4	**39.6**	30.7	29.7	25.7	22.2
NASA	**39.3**	33.3	38.9	22.2	11.1	0

The variants of CBR methods are taken and compared using Wilcoxon sum rank test. The results of Wilcoxon sum rank test of absolute residuals are presented in Table 4. Surprisingly, predictions based on CBR(BK) model presented statistically significant but necessarily accurate estimations than others, confirmed by the results of MMRE as shown in Table 2. Except for small datasets such as Albrecht, Kemerer, Telecom and NASA, the statistical test results demonstrate that there is no significant difference if the predictions generated by any CBR(BK) and other regular *K*-based CBR methods. So it seems that the small datasets are the most challenging ones. These datasets have relatively small number of instances and large degree of heterogeneity between projects so it is difficult to obtain a cluster of sufficient number of instances.

TABLE 4 Wilcoxon sum rank test results

Dataset	CBR (K=1)	CBR (K=2)	CBR (K=4)	CBR (K=8)	CBR (K=16)
Albrecht	0.8	0.64	0.39	0.45	0.69
Kemerer	0.7	0.17	0.07	0.07	0.69
Desharnais	**0.01***	**0.01***	**0.03***	**0.01***	**0.01***
COCOMO	**0.03***	**0.01***	**0.02***	**0.03***	**0.01***
Maxwell	**0.01***	**0.01***	**0.01***	**0.01***	**0.04***
China	**0.01***	**0.01***	**0.01***	**0.01***	**0.01***
Telecom	0.79	0.76	0.42	0.91	**0.03***
ISBSG	**0.04***	**0.01***	**0.01***	**0.01***	**0.01***
NASA	0.68	0.84	0.19	**0.04***	**0.01***

VII. THREAT TO VALIDITY

This section presents the comments on threats to validities of our study based on internal, external and construct validity. Internal validity is the degree to which conclusions can be drawn with regard to configuration setup of BK algorithm including: 1) the identification of initial medoids of BK for each dataset, 2) determining stopping criterion. Currently, there is no efficient method to choose initial medoids so we used random selection procedure. So we believe that this decision was reasonable even though it makes the k-medoids is computationally intensive. For stopping criterion we preferred to use the compactness performance measure to see when the BK should stop. Although there are plenty of compactness measures we believe that the used measure is sufficient to give us indication of how instances in the same clusters are strongly related.

Concerning construct validity which assures that we are measuring what we actually intended to measure. However, despite special emphasis was placed on the effectiveness of the performance measures, complete certainty with regard to this issue was challenged and we had to rely on common estimation-error based performance measures such as MMRE and PRED, which we no longer believe to be a completely trustworthy accuracy indicator [8, 24]. We do not consider that choice was a problem because (1) They are practical options for majority of researchers [2, 11, 13, 16, 22], and (2) using such measures enables our study to be benchmarked with previous effort estimation studies. On the other hand, in order to make apple-to-apple comparisons between different adaptation techniques we preferred to use Leave-one cross-validation strategy, though some authors favored n-Fold cross validation. The principal reason is that, the Leave-one cross-validation has been used in some previous studies and recommended to do comparison between different estimation models.

With regard to external validity, i.e. the ability to generalize the obtained findings of our comparative studies, we used 8 datasets from 2 different sources to ensure the generalizability of the obtained results. The employed datasets contain a wide diversity of projects in terms of their sources, their domains and the time period they were developed in. We also believe that reproducibility of results is an important factor for external validity. Therefore, we have purposely selected publicly available datasets. However, we consider that some datasets are very old to be used in software cost estimation because they represent different software development approaches and technologies. The reason for this is that these datasets are publically available, and still widely used for benchmarking purposes.

VIII. CONCLUSION AND FUTURE WORK

This paper proposed a new technique based on utilizing Bisecting k-medoids clustering algorithm and compactness degree to find the best K analogies number from the structure of dataset for each test project. Thus, rather than proposing a fixed best-K value a priori as the traditional CBR methods do, what CBR(BK) does is starting with all the training samples in the dataset, learning the dataset to form BK binary tree and excluding the irrelevant cases on the basis of compactness degree and then discovering the best-K value for each individual project. The proposed technique has the capability to support different-size datasets that have a lot of categorical features. Empirical results on various datasets indicate the performance of the proposed method over other regular K-based CBR methods. So the conclusion can be drawn that the choice of best K value is subject to the characteristics of a software dataset, and this value should be discovered from the structure of dataset. A future work is planned to study the impact of feature selection and weighting on discovering the optimal K value.

ACKNOWLEDGMENT

The authors are grateful to the Applied Science University, Amman, Jordan, for the financial support granted to cover the publication fee of this research article.

REFERENCES

[1] A. J. Albrecht and J. Gaffney, “Software function, source lines of code, and development effort prediction”, IEEE Trans on Software Engineering 9:639–648, 1983.

[2] M. Azzeh, “A replicated assessment and comparison of adaptation techniques for analogy-based effort estimation”, Empirical Software Engineering 17(1-2): 90-127, 2012.

[3] M. Azzeh, D. Neagu and P. Cowling, “Fuzzy grey relational analysis for software effort estimation”, Empirical Software Engineering, 15: 60-90, 2010.

[4] G. Boetticher, T. Menzies, T. Ostrand, PROMISE Repository of empirical software engineering data http://promisedata.org/ repository, West Virginia University, Department of Computer Science, 2010.

[5] L. C. Briand, K. El-Emam, D. Surmann and I. Wieczorek, K. D. Maxwell. An assessment and comparison of common cost estimation modeling techniques. Proceeding of the 1999 International Conference on Software Engineering, pp. 313–322, 1999.

[6] N. H. Chiu and S. J. Huang, “The adjusted analogy-based software effort estimation based on similarity distances”, Journal of Systems and Software 80:628–640. doi:10.1016/j.jss.2006.06.006, 2007.

[7] J. M. Desharnais, Analyse statistique de la productivitie des projets informatique a partie de la technique des point des foncti on. University of Montreal, 1989.

[8] T. Foss, E. Stensrud, B. Kitchenham and I. Myrtveit, “A simulation study of the model evaluation criterion MMRE”, IEEE Trans Softw Eng 29:985–995, 2003.

[9] A. Idri, A. Abran and T. Khoshgoftaar, Fuzzy Analogy: a New Approach for Software Effort Estimation, In: 11th International Workshop in Software Measurements, pp. 93-101, 2001.

[10] ISBSG, International software benchmark and standard group, Data CDRelease 10, www.isbsg.org, 2007

[11] M. Jorgensen, U. Indahl and D. Sjoberg Software effort estimation by analogy and “regression toward the mean”. Journal of Systems and Software 68:253–262, 2003.

[12] G. Kadoda, M. Cartwright, L. Chen and M. Shepperd, Experiences using case based reasoning to predict software project effort, in proceedings of EASE: Evaluation and Assessment in Software Engineering Conference, Keele, UK, 2000.

[13] J. Keung, B. Kitchenham and D. R. Jeffery, Analogy-X: Providing Statistical Inference to Analogy-Based Software Cost Estimation. IEEE Transaction on Software Engineering. 34(4): 471-484, 2008.

[14] C. F. Kemerer, “An empirical validation of software cost estimation models”, Comm. ACM 30: 416–429, 1987.

[15] C. Kirsopp, E. Mendes, R. Premraj, M. Shepperd, An empirical analysis of linear adaptation techniques for case-based prediction. International Conference on CBR. pp.231–245, 2003.

[16] E. Kocaguneli, T. Menzies, A. Bener and J. Keung, “Exploiting the Essential Assumptions of Analogy-based Effort Estimation”, IEEE transaction on Software Engineering. ISSN: 0098-5589, 2011.

[17] J. Z. Li, G. Ruhe, A. Al-Emran and M. Richter, “A flexible method for software effort estimation by analogy”, Empirical Software Engineering 12(1):65–106, 2007.

[18] Y. F. Li, M. Xie and T. N. Goh, “A study of the non-linear adjustment for analogy based software cost estimation”, Empirical Software Engineering 14:603–643, 2009.

[19] K. Maxwell, Applied statistics for software managers. Englewood Cliffs, NJ, Prentice-Hall, 2002.

[20] E. Mendes, N. Mosley and S. Counsell, Web metrics—Estimating design and authoring effort. IEEE Multimedia, Special Issue on Web Engineering, 50–57, 2001.

[21] E. Mendes, N. Mosley and S. Counsell, A replicated assessment of the use of adaptation rules to improve Web cost estimation, International Symposium on Empirical Software Engineering, pp. 100-109, 2003.

[22] E. Mendes, I. Watson, C. Triggs, N. Mosley and S Counsell, “A comparative study of cost estimation models for web hypermedia applications”, Empirical Software Engineering 8:163–196, 2003.

[23] T. Menzies, Z. Chen , J. Hihn and K. Lum, “Selecting Best Practices for Effort Estimation”, IEEE Transaction on Software Engineering. 32:883-895, 2006.

[24] I. Myrtveit, E. Stensrud and M. Shepperd, “Reliability and validity in comparative studies of software prediction models”, IEEE Trans on Software Engineering 31(5):380–391, 2005.

[25] M. Shepperd and C. Schofield, “Estimating software project effort using cases”, IEEE Transaction Software Engineering 23:736–743, 2006.

[26] M. Shepperd and G. Kadoda, “Comparing software prediction techniques using simulation”, IEEE Trans on Software Engineering 27(11):1014–1022, 2001.

[27] M. Shepperd and M. Cartwright, A Replication of the Use of Regression towards the Mean (R2M) as an Adjustment to Effort Estimation Models, 11th IEEE International Software Metrics Symposium (METRICS'05), pp.38, 2005.

[28] F. Walkerden and D. R. Jeffery, “An empirical study of analogy-based software effort Estimation”, Empirical Software Engineering 4(2):135–158, 1999.

[29] L. Kaufman and P. J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, New York, 1990.

[30] H-S. Park and C-H. Jun, “A simple and fast algorithm for K-medoids clustering”, J. Expert Systems with Applications, Volume 36, Issue 2, Part 2, Pages 3336-3341, 2009.

Predicting Risky Program Source Files

Syed Nadeem Ahsan, Syed Haider Abbas Naqvi and Kamran Raza
Faculty of Engineering Sciences and Technology
Iqra University, Karahi, Pakistan
sn_ahsan@yahoo.com, {haider0202, kraza}@iqra.edu.pk

***Abstract*—Change in source codes is an essential and routine activity of software development and maintenance. It has been observed that this activity might result in faults that might harm the use of the software. Therefore, it is always useful for software managers and programmers that before making any changes in source files, they should know the degree of risk associated with changing source files. In this paper, we present our approach to identify source files, which are risky or at least sensitive to new changes. We defined a set of metrics to compute the degree of risk associated to a source file. To validate our approach, an experiment has been performed by using Mozilla project's data. The experimental results show that the source files having higher risk values are more risky when applying the next change and thus should be tested more thoroughly.**

***Keywords*-Software evolution; code metrics; risk estimation;**

I. Introduction

In the field of software engineering, research studies reveal that over 90% of the software development cost spends on software maintenance and evolution. Moreover, software testing consume most of the development time and money [6]. Therefore, in today's software industry, software quality assurance personnel need those techniques which predict risky source code modules, so that more thorough testing can be performed for risky source code modules. Moreover, a developer can use such predictions to focus quality assurance activities.

In recent years, much research work has been performed to build models to predict risky or faulty source code modules. Mostly, researchers extract knowledge from the repository of software evolution [3][14]. This knowledge have been used to build models to predict faulty source code modules in the new release of a software product [2][7][9]. Therefore, the goal of our research work is to use the software evolution data, and define a new set of metrics to compute the degree of risk associated to the new changes in the source files. Such information is not only useful for developers but also for software managers in order to assign resources, e.g., for testing.

It has been found that the history of software change patterns might be used for the analysis of risky or faulty program files [1][10]. Therefore, in our research work, we first classify the changes in the source code into four well known types of software changes i.e., clean-changes, bug-introducing changes, bug-fix and introducing changes, and finally bug-fix changes [7]. Then, we define a set of metrics using the four different types of software changes. Finally, we derive an empirical relation for the risk model. To validate our research approach, we performed an experiment by extracting the change history data of source files from the software repository of Mozilla Project. After extracting the data, we applied an approach that allows us to identify the different types of code changes like, bug fixing, clean, bug introducing, and bug fix-introducing transactions [7]. We used the data of software changes and computed a set metrics. Finally, we used these metrics in our propose risk estimation model and obtained the degree of risk associated to source files.

The paper is organized as follow: In Section 2, we discuss related work. In Section 3, we describe the types of software changes. In Section 4, we describe our approach. In the next sections, i.e., Section 5 and 6, we discuss the obtained empirical results and conclude the paper.

II. Related Work

Software evolution data have been used to construct models for the classification of clean and faulty source code [3][14]. Asundi [8] highlight the importance and the major challenges of the risk estimation model of program source files. Whereas, most of the research works are based on data driven techniques and used machine learning algorithm to build model for the prediction of faulty source code modules [13]. It has been observed that very few attempts have been made to build empirical models for the computation of risk factor associated with the new changes in source codes [1][2].

Porter and Selby [4] used classification trees based on metrics from previous releases to identify components having high-risk properties. Mockus and Weiss [2] presented a model to predict the risk of new changes, based on previous information. The authors modeled the probability of causing failure of a change made to software, using properties of a change as model parameters. Sliwerski et al. [7] analyzed the CVS repository of Mozilla and Eclipse together with the information stored in the corresponding Bugzilla bug reporting system to identify fix inducing changes. Robert et al. [10] used code churn metrics (metrics which are based on addition, subtraction and modification of source code lines) for predicting faults in software. They found that change in the prior release is an essential component of fault prediction method. Similarly, our research work is focused to use the source code modification data, but instead of using direct code churn metrics, we used a new set of metrics, and derived a relation for the risk model of program source file. The research work present in this paper is the further enhancement in our previous risk model [12].

III. Types of Changes in Source Code

In this paper, we propose an approach to build model for the prediction of risky source code module. Our approach is based on software repository data, i.e., version controlling system (CVS) and bug tracking system (Bugzilla) [14]. First, we process the software repository data, and compute a set metrics using the following four types of source code changes: clean, bug introducing, bug fixing and bug fix-introducing changes. To obtain these types of source code changes, we used an approach which was given by Sliwerski et al. [7]. After obtaining the metrics data, we use these metrics in our risk model, and compute the risk factor associated to each source files. In the following paragraph we briefly describe the four different types of changes in source code (for details see [3][7]).

1) **Bug Fixing Change:** This type of changes are occurred when developers want to change the source code to fix a bug.
2) **Bug Introducing Change:** Once we identify that the change is a bug fixing change in the source file, then it is required to locate those changes in the source file revisions, which actually introduced the bug.
3) **Bug Fix-Introducing Change:** After identifying all the Bug-Introducing and the Bug-Fixing changes, then the next step is to list all those source file changes, which have both type of changes, i.e., Bug Fixing and Bug Introducing Changes.
4) **Clean Change:** Finally, the set of source file changes, which are not identified as bug fix or bug introducing or bug fix-introducing, are listed as clean changes.

IV. Our approach to Build a Risk Model

Our approach is based on a mathematical model, which can be derived using the definition of risk. In engineering risk corresponds to the costs in case of an accident [5], and according to IEEE, the standard definition of risk is: “An expression of the impact and possibility of a mishap in terms of potential mishap severity and probability of occurrence (MIL-STD-882-D, IEEE 1483).” Whereas, in our approach, “accident” or probability of “mishap” is basically the probability of faulty change in source code, which may introduce one or more bugs in a software, and “costs” is the time or effort needed to fix those bugs. Therefore, we process the data of source code’s change history, and compute the probability of faulty change of each source file. Finally, to obtain the risk value, we multiply the probability of faulty/buggy changes with the costs factor, and compute the risk value. Hence, in our case, risk can be defined as:

$$\text{Risk} = (Prob.\ of\ Bug) \times (Bug\ Impacts/Costs) \quad (1)$$

$$\text{Risk}_j = \text{p}_{BUG} \times \text{C}_j \quad (2)$$

where, Risk_j is the risk that a change in a source file could be a faulty change, and it could introduce j number of bugs. p_{BUG} is the probability that a new change could be a faulty change, and C_j represent the expected impact/costs of that faulty change. C_j represent the impact in term of an estimated costs that need in fixing the introducing number of faults. A faulty change in a program file may introduce *j* number of faults into a software system. C_j is the costs of fixing *j* faults. Therefore, the risk that a faulty change may introduce *j* faults is given by equation (2).

The probability of a change to lead to a bug (or accident), i.e., p_{BUG} can be obtained from the history of *bug introducing* and *bug fix-introducing* changes of a source file. For this purpose, the number of bugs introducing (n_I) and bug fix-introducing changes (n_{FI}) has to be divided by the number of all changes (n_C), e.g.,

$$\text{p}_{BUG} = \frac{\text{n}_I + \text{n}_{FI}}{\text{n}_C} \quad (3)$$

Now, to find the risk value of a source file, we have to know the expected costs of the bugs that could be generated. Let, p_j be the probability of introducing *j* bugs by a faulty change, and c_j is the costs of fixing *j* bugs.

$$\text{C}_j = \text{p}_j \times \text{c}_j = \frac{\text{n}_j}{\text{n}_C} \times \text{c}_j \quad (4)$$

$$\text{Risk}_j = \frac{\text{n}_I + \text{n}_{FI}}{\text{n}_C} \times \frac{\text{n}_j}{\text{n}_C} \times \text{c}_j \quad (5)$$

where, n_j is the number of changes that introduced *j* bugs per change. Its value can be obtained from the previous history of source file. If we consider the costs of each bug as a constant value, i.e., *c*, then Eq. (5) can be written as,

$$\text{Risk}_j = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{\text{n}_j}{\text{n}_C} \times \text{j} \times \text{c} \qquad (6)$$

In case of Open Source Software (OSS), there is no information regarding the costs for correcting a fault in terms of amount of work necessary, and moreover there is no information available about costs related to the bug when the program is executed. Hence, we might assume unit costs for each fault, i.e., *c*=1.

$$\text{Risk}_j = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{\text{n}_j}{\text{n}_C} \times \text{j} \qquad (7)$$

Equation (7) can be used to find the risk associated with source files, which could generate *j* faults. Every change might lead to several bugs. Hence, the expected costs can be estimated by averaging the costs of correcting "k" faults multiplied with the probability that a change causes "k" faults, i.e.,:

$$\text{Risk}_{avg(j)} = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{1}{\text{k}} \sum_{j=1}^{k} \frac{\text{n}_j}{\text{n}_C} \times j \qquad (8)$$

$$\text{Risk}_{avg(j)} = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{1}{\text{k}} \times \frac{1}{\text{n}_C} \times \sum_{j=1}^{k} (\text{n}_j \times j) \qquad (9)$$

where, $\sum_{j=1}^{k}(n_j \times j) = n_B$, and n_B is the total number of bugs occurred during the life of a source file. Its value can be obtained by adding the value of bug counts n_{BI}, which is generated by bug introducing changes and bug count value n_{BFI}, which is generated by bug fix-introducing changes (n_B=n_{BI}+n_{BFI}).

$$\text{Risk}_{avg(j)} = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{1}{\text{k}} \times \frac{\text{n}_B}{\text{n}_C} \qquad (10)$$

where, "k" is the maximum number of faults introduced by a single change in a source file. Furthermore, equation (10) can be used to compute the maximum risk instead of an average risk.

$$\text{Risk}_{max(j)} = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{\text{n}_B}{\text{n}_C} \qquad (11)$$

Equation (11) may be used to measure the maximum risk associated with the new changes in a source files. In equation (11), $\frac{n_B}{n_C}$ is the average number of faults per change. It may be written as $\text{C}_{max(j)}$. In our experiment, we used equation (11) to measure the risk value. In equation (11), we considered unit cost i.e., *c*=1, if we do not consider unit cost then equation (11) may be written as,

$$\text{Risk}_{max(j)} = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{\text{n}_B}{\text{n}_C} \times \text{c} \qquad (12)$$

Equation (12) is the main equation to compute the maximum risk factor associated with source file. The major challenge is to compute the cost value, i.e., "c" in equation (12). Since, in case of open source, the costs related to software change is not recoded, therefore, it is difficult to compute the costs value of software changes. However, we can make some assumption for costs value, like we did in equation (7). Moreover, it has been found in literatures that some attempts have been made to compute the effort or costs value of software changes [8] [11]. We are still working to find some better technique to compute the costs value of software changes. Now, In the next section, we discuss some experimental results, which are based on Eq. (1) to Eq. (11).

V. Results and Discussion

In order to validate our approach, we performed an experiment by downloading source files and bug reports data from the Mozilla project (evolution data is freely available on website). After downloading, we used an approach to process the downloaded data, and find the distribution of four different types of changes in source codes (for details see [7] [12]). Our experimental results are shown in Table I.

Table I depicts the obtained results of risk associated with 7 different source files of Mozilla project. For example, consider a source file, i.e., CBrowserShell.cpp from Table I. The number of fault-introducing changes and the number of fault-fix-introducing changes are 17 and 49 respectively. Hence, the probability of a change to introduce at least one bug in this example is: p_{BUG} = (n_I + n_{FI}) / n_C = (17+49)/93= 0.71. Moreover, the data of Table I, can easily be used to compute the expected cost of fault fixing in source file. Like in case of CBrowserShell.cpp source file, the cost is $\text{C}_{max(j)} = \frac{n_B}{n_C} = \frac{69}{93}$ = 0.74. From this, the risk factor associated with the new changes in the source file CBrowserShell.cpp can be computed as follow: $\text{R}_{max(j)}$ = p_{BUG} × C = 0.71 × 0.74 = 0.53.

Similarly, the risk can be computed for each source files. The results of this computation on sample data set of seven source files are given in Table I. Figure 1 shows the normalized risk factor associated with each source files. In our sample data set, the source file calDateTime.cpp is

Table I
COMPUTATION OF RISKY SOURCE FILES USING THE DISTRIBUTION OF FOUR DIFFERENT TYPES OF SOURCE FILE CHANGES

Mozilla Source Files (.cpp)	Four types of changes				Total Change	Total Faults	Total Faulty Changes	$p_{BUG} = (n_I+n_{FI}) / n_C$	$C_{max(j)} = n_B / n_C$	$Risk_{max(j)} = p_{BUG} \times C_{max(j)}$
(sample data)	Clean	Fix	n_I	n_{FI}	n_C	n_B	n_I+n_{FI}			
calDateTime	2	5	21	82	110	112	103	0.94	1.02	0.95
calICSService	1	6	9	60	76	73	69	0.91	0.96	0.87
CBrowserShell	11	16	17	49	93	69	66	0.71	0.74	0.53
CNavDTD	42	46	147	263	498	418	410	0.82	0.84	0.69
COtherDTD	8	20	57	65	150	128	122	0.81	0.85	0.69
dlldeps	32	44	35	92	203	132	127	0.63	0.65	0.41
EmbedPrivate	15	33	20	84	152	110	104	0.68	0.72	0.50

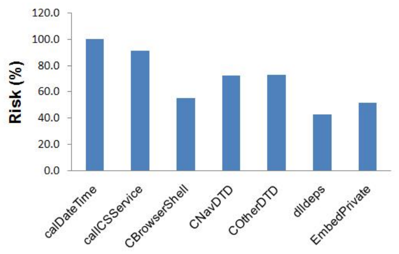


Figure 1. Risk factor associated with seven source files of Mozilla project

the most risky source file for further changes. Whereas, the source file dlldeps.cpp is the less risky file for further changes. A graph which is similar to Figure 1 can be used during software maintenance phase to identify those source files which have a high risk factor. Such information helpful to perform software maintenance task.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented our approach to build a mathematical model, which computes the risk factor associated with the new changes in source codes. Our risk model is based on a set of metrics. We obtained these set of metrics by using the evolution data of source codes. Furthermore, to validate our risk model, we performed an experiment by using the software evolution data of Mozilla project. We found that our model successfully measured the risk factor associated with source files. The source files which have higher risk factor values are more risky when applying the next change and thus should be tested more thoroughly. Currently, we are working to develop a method which can be used to compute the actual costs/impact value of a bug. Moreover, in future, we will further enhance our risk model by adding more metrics.

REFERENCES

[1] A. A. Phadke and E. B. Allen, "Predicting Risky Modules in Open-Source Software for High-Performance Computing," In proceeding of IWSEHPCS, 2005.

[2] A. Mockus and D. M. Weiss, "Predicting risk of software changes," Bell Labs Tech., vol. 5, Apr-June 2000, pp 169-180.

[3] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems." In Proceeding of the International Conference on Software Maintenance (ICSM), 2003, pp 13-23.

[4] A. A. Porter and R. W. Selby, "Empirically Guided Software Development Using Metric-Based Classification Trees," IEEE Software, volume 7(2), 1990, pp 46-54.

[5] E. Addison, "Managing Risk: Methods for Software Systems Development(Sei Series in Software Engineering)," Addison Wesley, February 28, 2005.

[6] L. Erlikh, "Leveraging legacy system dollars for e-business," IT professional, volume 2(3), 2000, pp 17-23.

[7] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?," Proceedings of the 2005 international workshop on Mining software repositories, 2005, pp 1-5.

[8] J. Asundi, "The need for effort estimation models for open source software projects." In Proceedings of the fifth workshop on Open source software engineering (5-WOSSE), ACM, New York, NY, USA, 2005, pp 1-3.

[9] K. A. Gunes and L. Hongfang, "Building Defect Prediction Models in Practice," IEEE Software, 22(6), 2005, pp 23-29.

[10] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "Does measuring code change improve fault prediction?," In Proceedings of the 7th International Conference on Predictive Models in Software Engineering (Promise), 2011.

[11] S. N. Ahsan, J. Ferzund and F. Wotawa, "Program File Bug Fix Effort Estimation Using Machine Learning Methods for OSS," 21st Int. Conference on Software Engg. and Knowledge Engg. (SEKE), Boston, USA, July 1-3, 2009, pp 129-134.

[12] S. N. Ahsan, J. Ferzund, and F. Wotawa, "A Database for the Analysis of Program Change Patterns," In proceeding of the 4th International conference on Networked Computing and Advanced Information Management, 2008.

[13] S. Kim, E. J. Whitehead, and Y. Zhang, "Classifying Software Changes: Clean or Buggy?," IEEE Transactions on Software Engineering, March/April, 2008, pp. 181-196.

[14] T. Zimmermann, P. Weisgerber, S. Diehl, A. Zeller, "Mining Version Histories to Guide Software Changes," In Proceedings of the 26th International Conference on Software Engineering, May 23-28, 2004, pp 563-572.

Supporting Time Planning Aligned with CMMI-DEV and PMBOK

Rafael Q. Gonçalves, André M. Pereira, Christiane Gresse von Wangenheim
Department of Informatics and Statistics
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil
rafael.q.g@hotmail.com, andremarquespereira@gmail.com, gresse@inf.ufsc.br

Abstract— **Software projects often fail, because they are not adequately managed. The establishment of effective and efficient project management practices especially with respect to time management still remains a key challenge to software organizations. Striving to address these needs, "best practice" models, such as, the Capability Maturity Model Integration (CMMI) or the Project Management Body of Knowledge (PMBOK), are being developed to assist organizations in improving project management. Yet, so far there does not exist a unified model focusing on the context of small and medium enterprises (SMEs). Therefore, this paper presents a generic model for time planning aligned with CMMI and PMBOK. In order to facilitate its application in practice an open-source tool (dotProject) has been enhanced and evaluated by project management specialists. The results of this research are expected to facilitate the adoption of time planning practices in SME contributing positively to their competitiveness.**

Keywords- time planning; PMBOK; CMMI; dotProject.

I. INTRODUCTION

Many software development projects still have problems to be delivered on time, within budget and with the complete scope defined [1]. In this context, one of the most important processes in the project management life cycle is time planning [2]. The aim of time planning processes is to deliver the project on time [2].

One of the reasons for these problems is a lack of project management, which indicates that establishing effective and efficient project management practices is still a challenge for many organizations [3].

As an attempt to improve this situation, "best practices" models have been developed to guide organizations interested in improving the project management process. This includes the CMMI-DEV (Capability Maturity Model Integration for Development) that guides the improvement and appraisal of a software organization's processes [4]. This model, although comprehensive, covers also "best practice" for project management. Another more specific "best practice" model for project management is the PMBOK (Guide to the Project Management Body of Knowledge) [2], which describes the life cycle of managing a project and the respective knowledge areas. Such maturity models, e.g., the CMMI, also indicate the importance of improving the project management process, as it is typically one of the first processes indicated to be improved associated to maturity level 2.

However, a large part of the IT market is composed of SMEs [5]. Many of these organizations typically struggle to implement these models [6]. Thus, in order to facilitate the adoption of best practices for time planning in SMEs, this work presents a generic model for a time planning process aligned with PMBOK and CMMI and customized to the characteristics of SMEs. In order to support the application of the model in practice we also enhanced one of the most popular open-source tools – dotProject – in conformance with the proposed model. The use of tools is important for SMEs as it can support and partially automate steps, increasing efficiency and improving the maturity of the process [7].

We first present the background to our research in Section 2. Analyzing and comparing both models, we map the respective best practices developing a unified model with respect to time planning as presented in Section 3. Section 4 describes the proposed generic model for time planning, demonstrating how processes/practices recommended by reference models could be applied to SME context. The tool enhancement and evaluation is presented in Section 5. Finalizing the paper, we discuss the results.

II. BACKGROUND

This section presents the key concepts with respect to project management and especially time planning and introduces PMBOK and CMMI.

Project management is the application of knowledge, skills, tools and techniques to project activities to meet their requirements [2]. A project is a temporary endeavor undertaken to create a unique product, service, or result. To achieve the goals defined in the project, knowledge, techniques and tools are applied that constitute project management. The project management life cycle is composed of five process groups [2] (Figure 1):

- Initiation: performed to initiate a new project or phase and obtain the authorization for its realization.
- Planning: performed to establish the project goals and scope and to define the actions necessary to ensure that the project meets its objectives.
- Execution: processes related to the execution of the project during which the work is carried out to complete the activities defined in the project plan.
- Monitoring and controlling: performed to monitor, review and adjust the project performance and progress, realizing corrective actions.
- Closing: performed to finalize all project activities in a formal way.

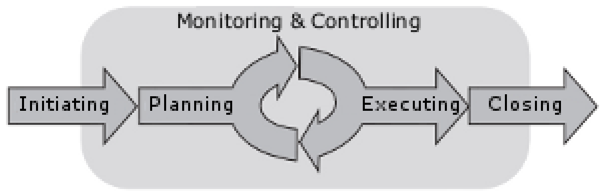


Figure 1 - Project management life cycle [2]

Orthogonal to this, project management processes are divided into 9 knowledge areas [2]: Integration, Scope, Time, Cost, Quality, Human Resource, Communication, Risk, and Procurement.

According to PMBOK, time planning requires processes to:

- Define Activities: a process to define the actions to be executed in order to produce project deliverables.
- Sequence activities: a process to define the logical dependencies between project activities, and also to define its execution order.
- Estimate activities resources: plan human and non human resources needed to execute project activities.
- Estimated activities duration: estimated the total time period needed to conclude the project activities. To do so it is necessary to know the activities scope, the resources available and others restrictions.
- Develop schedule: this process defines that each project activity has its own start and end dates, effort, duration, and resources estimated, as well the activity dependencies. Often it is represented as a Gantt chart.

A. *CMMI*

The Capability Maturity Model Integration (CMMI) provides a framework for improvement development process for software products and services. It describes the best practices associated to activities covering the life cycle of products from conception to delivery and maintenance [4]. Currently, there exist 3 different constellations: CMMI for Development (CMMI-DEV), CMMI for Acquisition (CMMI-ACQ) and CMMI for Services (CMMI-SVC). Here, due to our scope on software development, we focus on the CMMI-DEV constellation [4]. Its purpose is to help organizations improve their development and maintenance processes for both products and services. Within the CMMI Product Suite, a project is defined as a managed set of interrelated resources which delivers one or more products to a customer or end user. A project has a definite beginning and typically operates according to a plan. Such a plan is documented and specifies what is to be delivered or implemented, the sources and funds to be used, the work to be done, and a schedule for doing the work.

CMMI constellations are basically composed through two dimensions: process areas and capability/maturity levels. CMMI-DEV v1.3 defines 22 process areas grouped in four process categories. In this research, we focus mainly on project planning (PP) process area associated to maturity level 2, due to our specific focus on time planning practices that are supported by this process area. Specific practices related to time planning are:

- PP/SP 1.1 Estimate the Scope of the Project: practice related to project activates definition, derived from project Work Breakdown Structure (WBS).
- PP/SP 1.2 Establish Estimates of Work Product and Task Attributes: Accomplish estimations for work products size and task attributes, as estimated duration, start and end dates.
- PP/SP 2.1 Establish the Budget and Schedule: The budget is out of the scope of time planning, but the schedule is included, involving activities duration estimation and sequencing activities.
- PP/SP 2.4 Plan for Project Resources: Estimate for each project activity the human and non human resources and its quantities for accomplishing the activity.
- PP/SP 1.4 Determine Estimates of Effort and Cost: The cost estimation is out of scope of time planning, but the effort estimation is included. Estimation effort means the number of work periods that are needed to realize an activity.

III. PMBOK AND CMMI PRACTICES UNIFICATION

As a first step into the direction of a harmonized support aligned with both models, we analyzed and compared the best practices as required by both models for time planning and mapped them (TABLE I – column 3 and 4). The work is based on earlier research of the authors on unifying and harmonizing CMMI-DEV v1.2 (PP, PMC, SAM) and PMBOK processes [10], which has been revised and updated with respect to the new current version of CMMI-DEV v1.3.

Based on this we defined a set of Unified Best Practices (UBPs) covering completely both models (TABLE I – column 1 and 2).

TABLE I - UNIFIED BEST PRACTICES FOR TIME PLANNING

UBP	**Description**	**CMMI-DEV v1.3:2010**	**PMBOK 4ed:2008**
P1	Define Activities	PP/SP 1.1 Estimate the Scope of the Project	6.1 Define Activities
P2	Establish Estimates of Work Product and Activity Attributes	PP/SP 1.2 Establish Estimates of Work Product and Task	-
P3	Sequence Activities	PP/SP 2.1 Establish the Budget and Schedule	6.2 Sequence Activities
P4	Plan for Project Resources	PP/SP 2.4 Plan for Project Resources	6.3 Estimate Activity Resources
P5	Estimate Activity Durations	PP/SP 2.1 Establish the Budget and Schedule	6.4 Estimate Activity Durations

P6	Estimate Effort	PP/SP 1.4 Determine Estimates of Effort and Cost	
P7	Develop Schedule	PP/SP 2.1 Establish the Budget and Schedule	6.5 Develop Schedule

IV. GENERIC PROCESS MODEL

In accordance to the defined unification of both models and taking into consideration characteristics and needs of SMEs, we propose a generic process model for time planning. The proposed model (Figure 11) is defined using the formal notation SPEM which is maintained by OMG [8].

The model is composed of artifacts, processes, and tools (techniques and methods). The sequence of the processes is defined as shown in Figure 11. For each process inputs and outputs artifacts are defined. Each process is detailed in next sections.

A. *Define activities*

This process goal is to identify and document the work that has to be done to build the project deliveries. Project activities are identified based on the defined work packages in the WBS (*Work Breakdown Structure*). For each of the work packages one or more activities are identified – representing the work that has to be done to create the respective results. A technique to execute this process is named decomposition [2]. If existent, an organizational process model can also used as a basis.

B. *Sequence activities*

Sequence activities is the process related to identifying and documenting the logical relationships between project activities.

A technique used to define activities' dependencies is PDM (Precedence Diagram Method) [2]. It is based on a network diagram to represent all project activities and its dependencies (Figure 2).

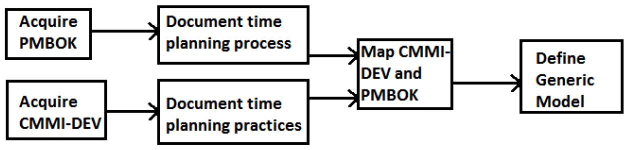


Figure 2 - Precedence diagram method

C. *Establish Estimates of Work Product and Activity Attributes*

This process aims at the estimation of the size and complexity of work packages. It is also related to the estimation of activity attributes as start and end dates, effort and duration.

The work package size is used to estimate the software dimension in a quantitative way. Size and complexity can be estimated using several units, for instance, size could be estimated using lines of code, function points or use case points [2]. It is considered an initial parameter, to perform other estimations as effort and cost.

D. *Plan for Project Resources*

This process aims at the estimation of the resources (people, equipments, etc.) and its quantities that will be needed to execute the project activities. Resources are typically estimated based on specialized opinion based on the roles defined in the organizational chart.

E. *Effort estimation*

The effort is the amount of work needed to execute an activity. Effort is typically estimated in terms of person-hours, person-months etc. Well accepted techniques for the estimation process include either the usage of historical data (often not available in organizations with an immature process such as many SMEs) or consensus-based techniques including wideband delphi [11] or planning poker [12].

F. *Estimate activities duration*

This process aims at estimating the duration of the activities. Typically the duration is estimated in periods of work (e.g.: hours or days) that are needed to conclude it. Several techniques typically adopted include:

- Expert opinion: Specialists provides their estimations for the duration of activities based on previous experiences.
- Analogue estimation: The duration of a similar activity, which was executed in a previous project to estimate the duration of this new activity, is used.
- Parametric estimation: The estimation is realized in a quantitative way. Making arithmetic calculus based on the amount of work and team productivity.

G. *Schedule development*

The schedule development is the process that determines the planned dates for start and ending each project activity. To develop the schedule in an optimized way, techniques typically adopted include:

- Schedule network analysis: It indicates the project duration. By applying this technique it is possible to know the project end date on worse and better cases.
- Critical path method (CPM): It identifies all activities which can't suffer any delay. A sequence of activities which can't suffer delay are called critical path. When some critical path activity suffers delay, the entire project will be delayed.

The main output for this process is the project schedule. The schedule typically is as a Gantt chart (Figure 3).

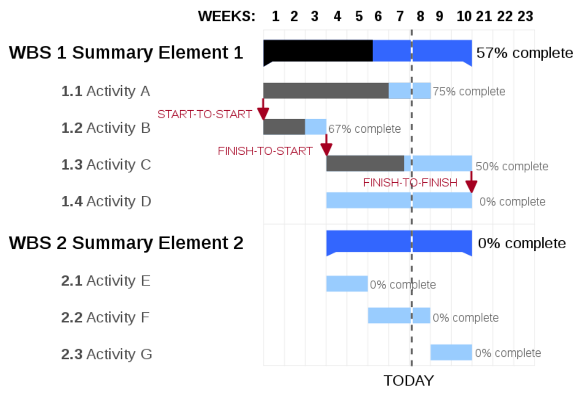


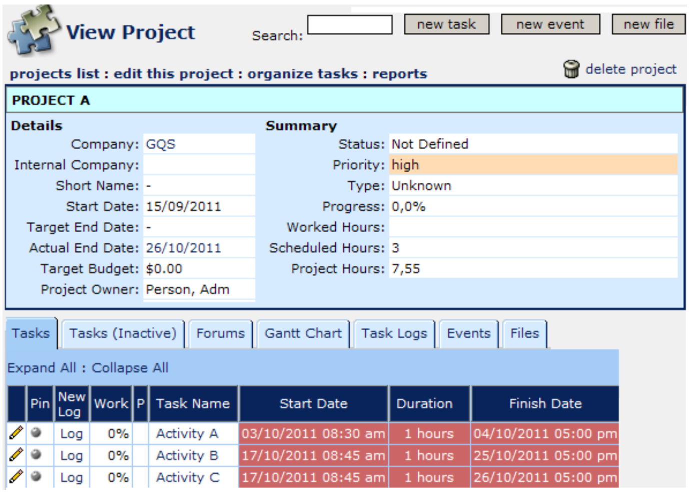
Figure 3 - Project schedule

V. TOOL SUPPORT

To facilitate the application of the proposed generic model in practice, tool support has been developed. Due to financial restrictions in the context of SMEs, we opted for enhancing one of the most popular free open-source tools - dotProject. One of the main reasons for choosing this tool, it is that new features can be built and installed as add-on modules using the tool's development framework. This possibility to adapt the tool usually is not available in commercial tools, e.g., ms project (microsoft.com/project) or primavera (oracle.com/primavera).

A. *DotProject*

DotProject [13] is a web-based tool for project management. It supports user management, projects listing, hierarchical task definition and schedule visualization (Gantt), client management, besides offering features such as contact list, file repository and calendar (Figure 4). The software supports MySQL or ADOdb databases and has been developed using PHP. Released in 2000, its current version is 2.1.6. It is an open-source system, published under General Public License (GPL), which means it can be customized and redistributed once the GPL is maintained.



PROJECT A

Details

Company:	GQS
Internal Company:	
Short Name:	-
Start Date:	15/09/2011
Target End Date:	-
Actual End Date:	26/10/2011
Target Budget:	$0.00
Project Owner:	Person, Adm

Summary

Status:	Not Defined
Priority:	high
Type:	Unknown
Progress:	0,0%
Worked Hours:	
Scheduled Hours:	3
Project Hours:	7,55

	Pin	New Log	Work	P	Task Name	Start Date	Duration	Finish Date
		Log	0%		Activity A	03/10/2011 08:30 am	1 hours	04/10/2011 05:00 pm
		Log	0%		Activity B	17/10/2011 08:45 am	1 hours	25/10/2011 05:00 pm
		Log	0%		Activity C	17/10/2011 08:45 am	1 hours	26/10/2011 05:00 pm

Figure 4 – dotProject

The current core version of dotProject does already partially support the time planning process. In a first step we analyzed the degree of support provided in conformance with the set of unified best practices (including PMBOK and CMMI). To assess the degree of support provided by dotProject in relation to each time planning UBP (TABLE I) we defined a 4-point ordinal rating scale as presented in TABLE II.

TABLE II - RATING SCALE [3]

Rating	Description
-	Does not provide any support.
*	Offers basic support, covering less than half of the UBP.
**	Covers more than half of the UBP.
***	Offers a complete set of elaborate functionalities for this UBP.

As a result, we identified that basically all required best practices are supported at least in a very simple way (TABLE III), yet with exception to the process "Develop Schedule" all processes need to be enhanced to provide full support.

TABLE III - DOTPROJECT TIME PLANNING SUPPORT [14]

UBP [3]	Description	dotProject version 2.1.6
P1	Define Activities	**
P2	Establish Estimates of Work Product and Task Attributes	*
P3	Sequence Activities	**
P4	Plan for Project Resources	**
P5	Estimate Activity Durations	**
P6	Estimate Effort	*
P7	Develop Schedule	***

VI. ENHANCEMENTS TO DOTPROJECT IN ORDER TO SUPPORT THE GENERIC MODEL

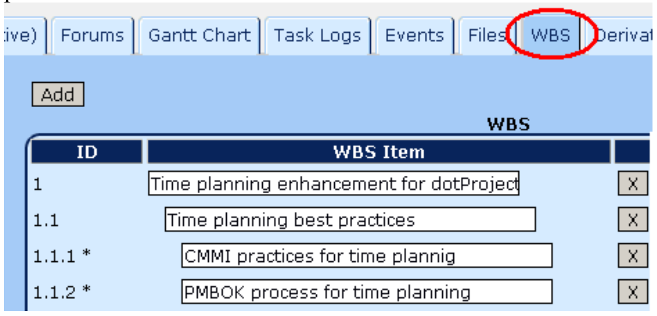
Based on the identified shortcomings as presented in TABLE III, we enhanced dotProject in alignment with PMBOK and CMMI [14]. This section presents the step by step modifications made.

Besides evolving functionality directly related to time planning processes, we also identified the need to develop features related to other processes, but required as inputs to the time planning processes, such as defining a WBS, so far not supported by dotProject core modules.

A. *Defining Work Breakdown Structure (WBS)*

In order to allow the systematic registration of the project scope in form of a WBS, we developed a new functionality, which supports the representation of project's

WBS (Figure 5). The created WBS is used as input to the process Define Activities.



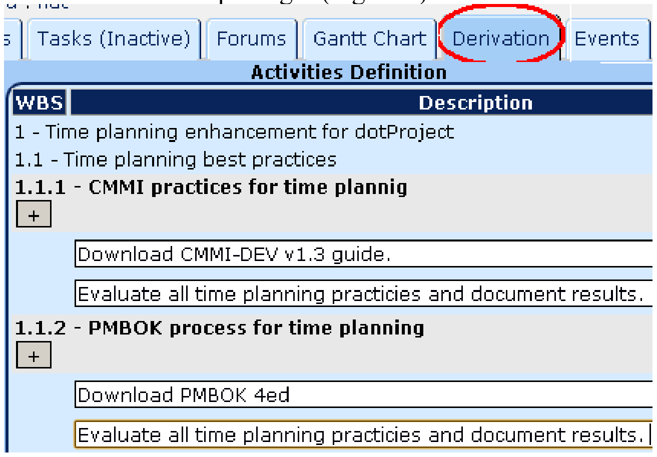
WBS

ID	WBS Item
1	Time planning enhancement for dotProject
1.1	Time planning best practices
1.1.1 *	CMMI practices for time plannig
1.1.2 *	PMBOK process for time planning

Figure 5 - Defining WBS

B. Defining activities based on work packages

Following PMBOK, activities are defined by decomposing work packages into activities. To support this process, we developed a functionality that visualizes the defined WBS and supports the addition of activities for each of the defined work packages (Figure 6).



Activities Definition

WBS	Description
1 - Time planning enhancement for dotProject	
1.1 - Time planning best practices	
1.1.1 - CMMI practices for time plannig	
	Download CMMI-DEV v1.3 guide.
	Evaluate all time planning practicies and document results.
1.1.2 - PMBOK process for time planning	
	Download PMBOK 4ed
	Evaluate all time planning practicies and document results.

Figure 6 - Defining project activities by work package

C. Sequence activities

The core implementation of dotProject allows the identification of relationships between activities, but not clearly.

Yet, in order to provide a better support we developed a feature that supports the definition of the dependencies between activities using the Precedence Diagram Method (PDM) in a graphical way visualizing project activities linked as a network.

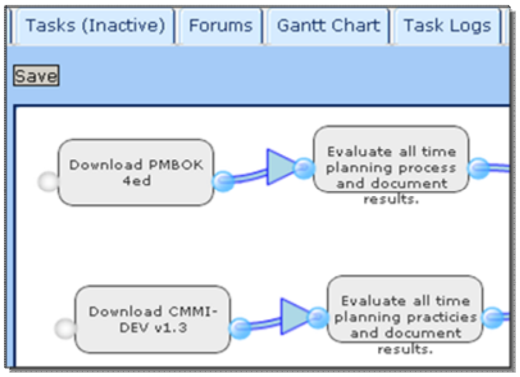
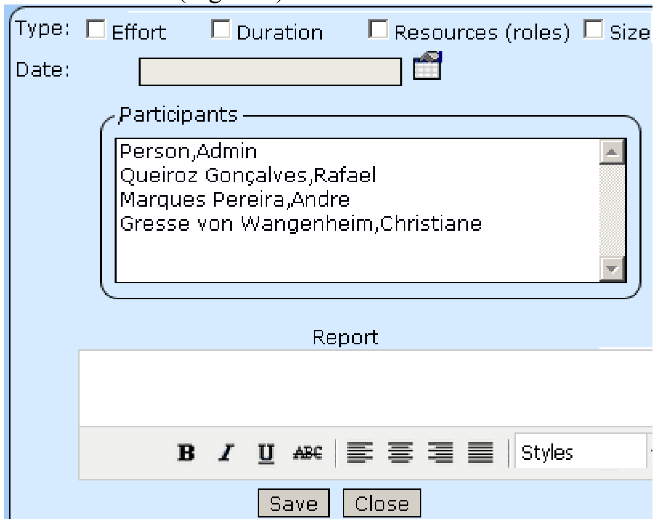

Tasks (Inactive)
Forums
Gantt Chart
Task Logs
Save
Download PMBOK 4ed
Evaluate all time planning process and document results.
Download CMMI-DEV v1.3
Evaluate all time planning practicies and document results.

Figure 7 - Sequence activities

D. Creating a meeting minute for estimation sessions

In accordance with the proposed process, estimations are made based on consensus, e.g., in planning poker sessions. In this context, tools support is basically required in terms of documenting how the estimates have been determined. Therefore, a new feature has been added to the tool, which supports the registering of a meeting minute for estimation sessions. As part of this meeting minute it is possible to document what has been estimated, when the session has taken place, who participated, as well as details with respect to the estimates (Figure 8).



Type: ☐ Effort ☐ Duration ☐ Resources (roles) ☐ Size

Date:

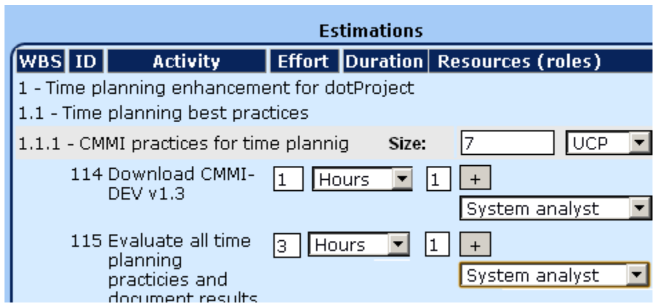
Participants

Person,Admin
Queiroz Gonçalves,Rafael
Marques Pereira,Andre
Gresse von Wangenheim,Christiane

Report

Figure 8 - Estimation minute form

The determined estimates (size/complexity) can be registered with respect to the specific work package. For each activity effort and/or duration estimates can be registered (Figure 9).


Estimations

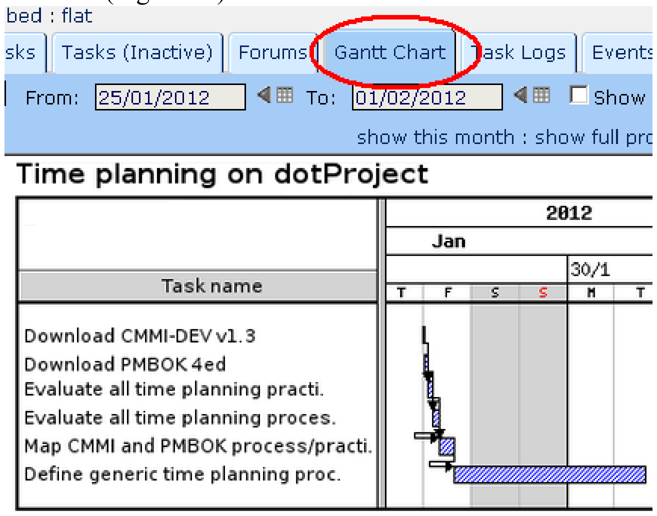
WBS	ID	Activity	Effort	Duration	Resources (roles)
1 - Time planning enhancement for dotProject					
1.1 - Time planning best practices					
1.1.1 - CMMI practices for time plannig		Size:	7	UCP	
	114	Download CMMI-DEV v1.3	1 Hours	1	System analyst
	115	Evaluate all time planning practicies and document results	3 Hours	1	System analyst

Figure 9 - Activities estimations details

For each activity, the tool also supports the estimation of the required resources by selecting required roles from the organizational chart.

E. Schedule development

To support schedule development a new functionality has been developed based on the Critical Path Method (CPM). It can be executed after all activities have been sequenced and their efforts have been estimated. This method calculates the start and end dates for activities based on project start date, sequence of activities, and estimated efforts. As a result, the tool also automatically creates a Gantt chart with this information in order to visualize the schedule (Figure 10).


bed : flat

sks | Tasks (Inactive) | Forums | Gantt Chart | Task Logs | Events

From: 25/01/2012 To: 01/02/2012 Show

show this month : show full pro

Time planning on dotProject

Task name

Download CMMI-DEV v1.3
Download PMBOK 4ed
Evaluate all time planning practi.
Evaluate all time planning proces.
Map CMMI and PMBOK process/practi.
Define generic time planning proc.

Figure 10 - Gantt chart

In this respect several enhancements have been developed in order to facilitate the application of the generic time planning process in practice, partially even automating steps as far as possible.

VII. EVALUATION

As part of our research we also evaluated the proposed generic model and the developed tool enhancements. Our evaluation goals are:

- Goal 1: Evaluate, if the enhancement of dotProject is helpful to support time planning in software projects in SMEs.
- Goal 2: Analyze if the generic process model is complete, consistent, and adequate for SMEs.
- Goal 3: Identify the strong points and the improvements points of proposed solution.
- Goal 4. Compare the degree of alignment of the enhancements of dotProject with dotProject v2.1.6.

With respect to the identified evaluation goals, we performed two types of studies. With respect to goals 1-3 we performed an expert panel and with respect to goal 4 we repeated the heuristic evaluation done in the beginning.

A. Expert panel

Adopting the GQM method [5], we decomposed these goals into questions and metrics. The required data has been collected through an expert panel capturing the opinion of software project management experts. Therefore, a questionnaire has been designed, transforming the metrics into affirmations and using a 5-point scale (1 – strongly disagree to 5 – strongly agree) (TABLE V).

TABLE IV – QUESTIONNAIRE ITEMS

Goals	Questions
Goal 1	1.1 I consider the evolution of dotProject useful for activity definition.
	1.2 I consider the evolution of dotProject useful for activity sequencing.
	1.3 I consider the evolution of dotProject useful for the documentation of size/effort/duration estimations.
	1.4 I consider the evolution of dotProject useful for the documentation of resources estimation documentation.
	1.5 I consider the evolution of dotProject useful for schedule development.
	1.6 The enhanced version of dotProject completely supports time planning.
	1.7 The enhancement of dotProject is consistent.
	1.8 The evolution of dotProject is adequate to support time planning on SME.
Goal 2	2.1 I consider the generic process model for time planning adequate for SMEs.
	2.2 I consider the generic process model for time planning consistent.
	2.3 The generic process model for time planning covers time planning completely to realize time planning.
Goal 3	3.1 What are the main strengths you observed?
	3.2 What are the main improvements suggestions?
	3.3 Do you have any other comment?

The evaluation was realized by project management specialists in SME context. The participants were chosen based on their availability to participate in a short period of time.

The specialists invited to join the experiment, did so in a voluntary capacity.

Beside the questionnaire the experts received an evaluation guide that explains the generic process model (Figure 11), and also demonstrates the enhanced tool functionality (Section V). The experts have been asked to follow the process model using dotProject on a time planning example and afterwards to respond the questionnaire.

B. *Results*

The evaluation was realized during the months of May and June 2012. We invited 34 experts, selection based on their software project management expertise and their short term availability. In total, we received 10 responses, representing a response rate of 29%. The invitation was sent by e-mail, containing the evaluation guide and the link to the online questionnaire.

Analyzing the experts' responses, TABLE V shows the median for each of the items.

TABLE V – MEDIANS PER ITEM/GOAL

Questionnaire item	**Median**
Goal 1	
1.1 I consider the evolution of dotProject useful for activity definition.	5
1.2 I consider the evolution of dotProject useful for activity sequencing.	5
1.3 I consider the evolution of dotProject useful for the documentation of size/effort/duration estimations.	4
1.4 I consider the evolution of dotProject useful for the documentation of resources estimation documentation.	4
1.5 I consider the evolution of dotProject useful for schedule development.	5
1.6 The enhanced version of dotProject completely supports time planning.	4
1.7 The enhancement of dotProject is consistent.	5
1.8 The evolution of dotProject is adequate to support time planning on SME.	4
Goal 2	
2.1 I consider the generic process model for time planning adequate for SMEs.	4
2.2 I consider the generic process model for time planning consistent.	4
2.3 The generic process model for time planning covers time planning completely to realize time planning.	4

B. *Heuristic evaluation*

As in the beginning of the enhancement of dotProject, we repeated the same heuristic evaluation with respect to the defined UBPs. The results are presented in TABLE VI.

TABLE VI - EVALUATION OF ENHANCEMENTS ON DOTPROJECT [14]

UBP [3]	**Description**	**dotProject version 2.1.6**	**Enhanced version dotProject**
P1	Define Activities	**	***
P2	Establish Estimates of Work Product and Task Attributes	*	***
P3	Sequence Activities	**	***
P4	Plan for Project Resources	**	***
P5	Estimate Activity Durations	**	***
P6	Estimate Effort	*	***
P7	Develop Schedule	***	***

Among the project management open-source tools, dotProject is considered the one that most provides support to the best practice models [3]. The enhancements made in this work, added several important features, such as, definition of WBS, and activities creation based on decomposition technique [2], clearly distinguishing the concepts of project work package and project activity. The sequence of the activities using the PDM [2], and also the estimations registration are important improvements to support the development of project schedule, which is the main output of time planning process.

C. *Discussion*

Analyzing the presented results, we can identify a very positive feedback in general.

In relation to Goal 1, we observed a strong tendency for total agreement of the experts regarding the questions whether the tool enhancement is helpful – all medians are between 4 and 5.

With respect to Goal 2, the experts also agreed that the proposed generic process model is complete, consistent, and adequate for SME. Experts agreed on all items with a median of 4. Yet, as no strong agreement has been obtained on average we can also identify that some further small improvements could be carried out.

With respect to Goal 3 the experts highlighted as strengths principally the harmonization between CMMI and PMBOK aligned to one single process model for time planning. One expert also stressed very positively the enhancement of an open-source tool to support the model. Several experts highlighted that the explicit separation, between work packages and project activities is a very useful feature, as most project management tools don't separate these concepts clearly.

In respected with Goal 4, we identified that the enhancement in fact has been done in correspondence with the PMBOK and CMMI and that with the enhancement made a full support in conformance with these models is provided.

D. *Threats to validity*

The evaluation we performed of course represents only a starting point. There can be identified several threats to validity of the results due to limitations of the evaluation for practical reasons. One threat is the small number of software project managers involved. As this number does not by any means provide statistical representativeness. At this point of time, we also involved only experts from Brazil. This of course limits strongly the generalization of the obtained results.

Another threat to validity can be the definition of metrics and data collection instruments. Yet, adopting GQM to systematically derive the metrics and questionnaire issues, such a threat may be small.

Another issue to be considered is the fact that dotProject in itself presents several usability problems and is therefore not very intuitive to be used.

Thus, evaluators may have found difficulties in the executing of the exemplar time planning tasks not related to the feature itself but due to general usability problems.

VIII. Conclusion and Future Work

This work intends to facilitate the adoption of systematic time planning in SMEs in conformance with the PMBOK and CMMI. Therefore, we unify best practices from both models, propose a generic process model for time planning customized to the context of SMEs and enhance a free open-source tool to support the process in practice.

A first evaluation provided a very positive feedback, stressing principally the tool enhancement as one of the strengths of the work.

Based on the obtained feedback, we are currently improving the process model and the dotProject evolution.

The evaluation we have performed focused on the application of generic process model in SMEs, and it was evaluated by project manager experts in SMEs context. There is a possibility that this model also could be applied in large organizations, but for that a new evaluation focused in this context should be applied.

As future work, we are amplifying the scope of our work aiming at the coverage of the complete project management life cycle, including e.g., monitoring & control, as well as all relevant knowledge areas (scope, cost, risks, etc.).

Acknowledgment

This work was supported by the CNPq (*Conselho Nacional de Desenvolvimento Científico e Tecnológico* – www.cnpq.br), an entity of the Brazilian government focused on scientific and technological development.

References

[1] The Standish Group International, “Chaos Summary for 2010”. Boston, MA, USA, 2010. <http://insyght.com.au/special/2010CHAOSSummary.pdf> 18.09.2012

[2] PMI – Project Management Institute, “A Guide to the Project Management Body of Knowledge”. 4th edition, Project Management Institute (PMI), Newtown Square, Pennsylvania, USA, 2008.

[3] C. Wangenheim, J. Hauck, and A. Wangenheim, “Enhancing Open Source Software in Alignment with CMMI-DEV”, IEEE Software, vol. 26, no. 2, 2009, pp.59-67.

[4] SEI – Software Engeenering Institute, “CMMI for Development (CMMI-DEV), Version 1.3”, Technical Report, CMU/SEI-2010-TR-033, 2010.

[5] ABES - Associação Brasileira das Empresas de Software, “Brazilian Software Market”. São Paulo, 2011. <http://www.abes.org.br/UserFiles/Image/PDFs/Mercado_BR 2011.pdf > 18.09.2012.

[6] SEI - SOFTWARE ENGINEERING INSTITUTE, “Process Maturity Profile CMMI/SCAMPI Class A Appraisal Results 2011 Mid-Year Update”. Pittsburgh: Carnegie Mellon® University, 2011. <http://www.sei.cmu.edu/cmmi/casestudies/profiles/pdfs/upload/2011SeptCMMI-2.pdf > 18.09.2012.

[7] H. Young, T. Fang, and C. Hu. “A Successful Practice of Applying Software Tools to CMMI Process Improvement”. Journal of Software Engineering Studies, vol. 1, no. 2, 2006, pp.78-95.

[8] OMG – Object Management Group, “Software & Systems Process Engineering Meta-Model Specification - 2ed version”, 2008. <http://www.omg.org/spec/SPEM/2.0/PDF> 18.09.2012 .

[9] V. Basili, G. Caldiera, and D. Rombach, “The experience factory”. Encyclopedia of Software Engineering, 1994, pp. 469-476.

[10] C. Wangenheim, D. Silva, L. Buglione, R. Scheidt, and R. Prikladnicki, “Best practice fusion of CMMI-DEV v1.2 (PP, PMC, SAM) and PMBOK 2008”. Journal on Information and Software Technology, vol. 52, no. 7, 2010, pp.749-757.

[11] B. Boehm, Software Engineering Economics, Prentice Hall PTR Upper Saddle River, NJ, USA, 1981.

[12] M. Cohn, Agile Estimating and Planning, Prentice Hall, USA, 2005.

[13] DotProject, “Project Management Software”. < www.dotproject.net > 26.10.2012

[14] R. Gonçalves, A. Pereira, C. Wangenheim, and J. Hauck, “Supporting time planning by enhancing an Open Source Software in Alignment with CMMI-DEV and PMBOK”. International. International Free Software Workshop. Porto Alegre, Brazil, 2012. <http://people.softwarelivre.org/wsl/2012/18.pdf> 18.09.2012

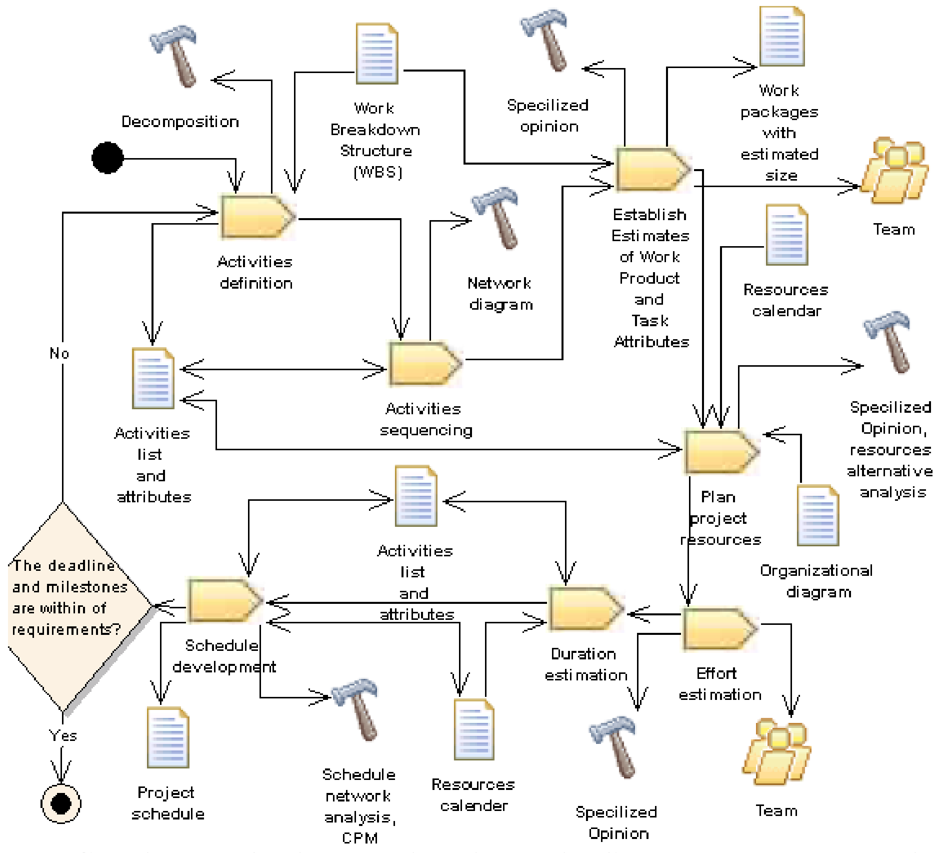

Decomposition
Work Breakdown Structure (WBS)
Specilized opinion
Work packages with estimated size
Team
Establish Estimates of Work Product and Task Attributes
Activities definition
Network diagram
Resources calendar
No
Activities list and attributes
Activities sequencing
Specilized Opinion, resources alternative analysis
Plan project resources
Organizational diagram
Activities list and attributes
The deadline and milestones are within of requirements?
Schedule development
Duration estimation
Effort estimation
Yes
Project schedule
Schedule network analysis, CPM
Resources calender
Specilized Opinion
Team

Figure 11 - Generic model for time planning aligned with CMMI-DEV and PMBOK for SME

Specification of UML Classes by Object Oriented Petri Nets

Radek Kočí and Vladimír Janoušek

Brno University of Technology, Faculty of Information Technology,
IT4Innovations Centre of Excellence
Bozetechova 2, 612 66 Brno, Czech Republic
{*koci,janousek*}*@fit.vutbr.cz*

***Abstract*—The UML class diagram defines a basic architectonic model of the system. Its behavior is then usually described by other UML diagrams, such as activity diagrams, sequence diagrams, etc. These models serve for the design purposes and are automatically or manually transformed in the next development stages, typically to the models with formal basis or to implementation (production) environment. There is no backward step allowing to investigate the system structure and its behavior with the designed models. On the other hand, there are approaches to system design combining design, testing, and implementing stages into one development technique. One of them uses Object Oriented Petri Nets (OOPN) as basic modeling formalism. Nevertheless, OOPN lacks for advisable architectonic view of modeled systems as it is offered by UML class diagram. The paper is aimed at using UML class diagrams for system architecture description and the OOPN formalism for description of classes behavior. Since UML classes and OOPN classes partially differs, we define formal transformation between UML classes and OOPN classes.**

***Keywords*-Class diagram; Object-Oriented Petri Nets; UML; transformation.**

I. INTRODUCTION

Design methodologies use models for system specification, i.e., for defining the structure and behavior of developed system. The most popular modeling language in software engineering is UML [1]. It serves as a standard for analytics, designers and programmers. But, own phraseology of UML does not have enough power allowing to realize some fundamental relationships and, in particular, rules, that are branch of every modeled system. To model dynamic aspects of the system, the designer usually describes them by static diagrams in a design phase and he cannot make certain of his partial ideas about the system behavior. Although the UML language can be completed by extensions, e.g., OCL (Object Constraint Language), making the system description more precise, it makes the checking of models correctness or validity by means of simulation complicated.

Therefore, new methodologies and approaches are investigated and developed for many years. They are commonly known as Model-Driven Software Development or Model-Based Design (MBD) [2], [3], [4]. An important feature of these methods is the fact that they use executable models, e.g., Model Driven Architecture (MDA) [5] and Executable UML [6], allowing to simulate models, i.e., to provide simulation testing. The created models can be (semi)automatically transformed to implementation language (the code generation). Nevertheless, the result has to be finalized manually, so it entails a possibility of semantic mistakes or imprecision between models and transformed code.

There are other similar methods that use the pure formal models (e.g., Petri Nets, calculus, etc.) allowing to use formal or simulation approaches to complete the design, testing, and implementation activities. In comparison with semi-formal models, formal models bring clear and understandable modeling and the possibility to test correctness with no need for model transformations. The design method, which is taken into account in this paper [7], [8], derives benefit from formalisms of Object Oriented Petri Nets (OOPN) [9], [10]. The paper is aimed at the class description using Object Oriented Petri Nets (OOPN). Since the UML classes and OOPN classes partially differ, we define formal transformation between UML classes and OOPN classes and formal constraints the classes and objects have to satisfy. The goal is to keep an eye to the system at the architectonic view with UML and at the behavioral view with the formalism of OOPN.

The paper is organized as follows. First, we briefly introduce used design methodology in Section III. Then the formalisms will be described in Section IV. Section V introduces relationships between UML classes and OOPN classes and a mechanism of class transformations. The proposed mechanism will be demonstrated with the example in Section VI.

II. RELATED WORK

The are works that are similar to the proposed one. First, the formalism of nets-within-nets (NwN) was introduced by Valk [11] and Moldt [12], [13]. The formalism of NwN is similar to OOPN, but OOPN fully support an integration of formal description of objects and objects from target environment, which facilitates, e.g., reality-in-the-loop simulation or usage of formal models into target application. Second, there are tools merging UML and Petri nets, for instance ArgoUML [14]. The difference is similar to the previous situation—these tools allow to *model* systems using combination of different formalisms, but do not allow to use formal models in system *implementation*.

III. Design Methodology

The design methodology [15] stems from the classic approach of class identification and definition and extends it to the new features. Primarily, there have to be found essential objects of a modeled system and their relationships. There we can successfully employ resources of UML such us *Use Case*, *Activity*, and *Class* diagrams. Thus, the design process comprises, among others, the identification of *use cases* of the system and the specification of *classes* and their behavior. To specify the behavior, the methodology distinguishes *roles* and *activity nets* as a special kinds of classes. These mentioned nets represent appropriate roles and use cases in the system and are layered hierarchically. Each role encapsulate activity nets and, moreover, each role can encapsulate another role. It allows to get a new view to the role based on the existing one.

Each role has its own set of allowed activities (activity nets) described by OOPN. If anybody wants to perform the activity, it has to ask the role for creating an instance of the activity and then it can use this activity as a use case of the system. The execution of nets are synchronized by means of synchronous ports. The nested nets define synchronous port for synchronization of executions and the net at higher layer is controlled by calling these ports. This principle will be demonstrated at the appropriate places in following parts.

IV. Formalisms

We will present a short introduction to formalisms and models used in this section.

A. Structural and Behavioral Views with UML

The UML modeling [1] uses a notion of *view*. A view of a system is a projection of the system on one of its relevant aspects. Such a projection focuses on certain aspects and ignores others. For our purposes, we mention only two views. The *structural view* describes layout between objects and classes, their associations and their possible communication channels. As an example, we can mention *Class diagram*. The *behavioral view* describes, how the system components interact, and characterizes the response to external system operations. For our purposes, we will not use UML diagrams, but OOPN for behavioral view.

B. Formalism of OOPN

An *Object Oriented Petri net* (OOPN) is a set of classes specified by high-level Petri nets. Formally, OOPN comprises constants $CONST$, variables VAR, net elements (such as places P and transitions T), class elements (such as object nets $ONET$, method nets $MNET$, synchronous ports $SYNC$, negative predicates $NPRED$ and message selectors MSG), classes $CLASS$, object identifiers OID, and method net instance identifiers MID. We denote $NET = ONET \cup MNET$ and $ID = OID \cup MID$.

A *class* is mainly specified by an object net (an element of $ONET$), a set of synchronous ports and negative predicates (a subset of $SYNC$ and $NPRED$), a set of method nets (a subset of $MNET$), and a set of message selectors (a subset of MSG) corresponding to its method nets, synchronous ports, and negative predicates. Object nets describe possible autonomous activities of objects, while method nets describe reactions of objects to messages sent to them from the outside.

An example illustrating the important elements of the OOPN formalism is shown in Figure 1. There are depicted two classes `C0` and `C1`. The object net of the class `C0` consists of places `p1` and `p2` and one transition `t1`. The object net of the class `C1` is empty. The class `C0` has a method `init:`, a synchronous port `get:`, and a negative predicate `empty`. The class `C1` has a method `doFor:`.

Synchronous ports are special (virtual) transitions, which cannot fire alone but only dynamically fused to some other transitions, which activate them from their guards via message sending. Every synchronous port embodies a set of conditions, preconditions, and postconditions over places of the appropriate object net, and further a guard, and a set of parameters. Parameters of an activated port s can be bound to constants or unified with variables defined on the level of the transition or port that activated the port s. An example is shown in Figure 1, the port named `get:` having one parameter `o`. This port is called from the transition `t2` (class `C1`) with unbound variable `n`—it means that the variable `n` will be unified with the content of the place `p2` (class `C0`).

Negative predicates are special variants of synchronous ports. Its semantics is inverted—the calling transition is fireable if the negative predicate is not fireable. The passed variable cannot be unbound (the unification is impossible) and the predicate cannot have a side effect. An example is shown in Figure 1, the predicate named `empty`. This predicate is called from the transition `t3` (class `C1`)—it means that the transition `t3` will be fireable if the place `p2` (class `C0`) will be empty.

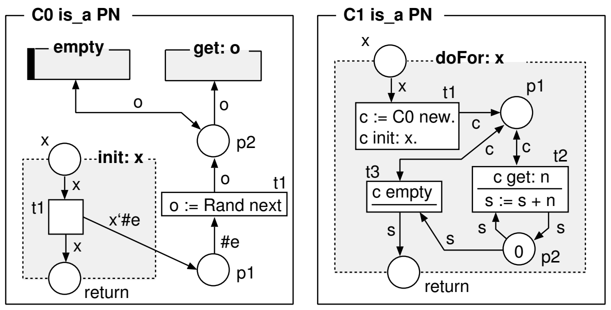


Figure 1. An OOPN example.

V. Relationship Between UML and OOPN Classes

We will present a relationship between classes of UML and OOPN and their reciprocal mapping.

A. Prerequisites

First, we define formal structures that will be used in next definitions. In pure object systems, everything is understood as an object, so that there is no requirement for defining special kind of types. Nevertheless, for our purpose we define $TYPE = CLASS \cup OCLASS \cup \{\varepsilon\}$, where $CLASS$ is a set of domain (OOPN) classes, $OCLASS$ is a set of other types (e.g., classes from the production environment or primitive types), and ε represents a special kind of type meaning *unspecified type*. Let the symbol $\triangleleft$ determines a relationship *is of a type* (*is an instance of*). For example, $o \triangleleft A$ means that the object (value) referred by the variable o is an instance of a class A (is of a type A).

The class can be defined as a tuple (n, V_C, I_C, B_C), where n is a class name, V_C is a set of instance variables, I_C is an interface (a set of operations), and B_C is a behavior, usually defined as a set of methods. The OOPN class be alternatively defined as a tuple $(n, P_{ON}, I_{PN}, B_{PN})$, where n is a class name, P_{ON} is a set of places from the object net representing instance variables, $I_{PN} \subseteq MSG$ is an interface, and B_{PN} is a behavior.

B. Interface

The interface of an OOPN class is defined as a subset of message selectors $I_{PN} \subseteq MSG$, where $MSG = MSG_M \cup MSG_S \cup MSG_P$. MSG_M corresponds with method nets, MSG_S corresponds with synchronous ports, and MSG_P corresponds with negative predicates.

There are several ways how I_C can be mapped to I_{PN}. Let f_I be a non-specific mapping $I_C \rightarrow MSG_M$. In this case, each operation is mapped into a message selector of a method net. This way is easy, but not sufficient for design methods that use Petri Nets [15]. Therefore, the operations from I_C are classified into three groups: *action group* $I_C^{Act} \subseteq I_C$ performing some actions on the object; *test group* $I_C^T \subseteq I_C$ performing some tests on the object, and *access group* $I_C^{Acc} \subseteq I_C$ which sets or gets a value of an instance variable. Analogically, let us define I_{PN}^{Act}, I_{PN}^{Acc}, and I_{PN}^T for the OOPN class. Then, the second way of mapping defines specific functions for appropriate group:

$$\begin{aligned} f_I^{Act} &: I_C^{Act} \rightarrow I_{PN}^{Act}, \text{where } I_{PN}^{Act} = MSG_M \cup MSG_S \\ f_I^{Acc} &: I_C^{Acc} \rightarrow I_{PN}^{Acc}, \text{where } I_{PN}^{Acc} = MSG_M \cup MSG_S \\ f_I^T &: I_C^T \rightarrow I_{PN}^T, \text{where } I_{PN}^T = MSG_S \cup MSG_P \end{aligned}$$

The action and access groups are mapped into the same subset of selectors of method nets and synchronous ports. The synchronous ports can influence on the object net during its firing (e.g., an object can be removed from or put into places in an object net), so that the calling a synchronous port from the interface has a direct effect in changing an object net state. Consequently, it can cause an activity of an object net. The negative predicate cannot have any side effects from the definition, so it cannot be a part of action and access groups. The testing group is mapped into a subset of synchronous ports or negative predicates—it depends on the positive or negative sense of the testing.

We can suppose, that the following statement holds for the UML class: $I_C^{Act} \cap I_C^T \cap I_C^{Acc} = \emptyset$. It means, that each operation is a member of only one group. For OOPN class, we can say $I_{PN}^{Act} \cap (I_{PN}^{Acc} \cup I_{PN}^T) = \emptyset$. It means, that operations from I_{PN}^{Act} cannot be members of other groups. Due to the definition of synchronous ports, the same synchronous port can serve for testing as well as for data accessing, so $I_{PN}^{Acc} \cap I_{PN}^T$ not have to be $\emptyset$.

C. Instance variables and types

A mapping of instance variables is defined as an injection $f_V : V_C \rightarrow P_{ON}$, where P_{ON} is a set of places of the object net. The consequence is that the variable is always a multiset of values. If the only one value has to be assigned to the place, as for an ordinary variable, it is possible to define a constraint, see Section V-D. The place in OOPN has assigned no type. But, for analysis and testing purpose, it is possible to 1) assign a set of types the objects can be of, 2) derive a set of types the objects are of from the model analysis or simulation.

Let T_P be a surjection $T_P : P \rightarrow \mathcal{P}(TYPE)$ assigning a set of types to a given place. The type of the place can be derived from the associations between classes, whereas there is no necessary to define only one type (and, thus, to allow all subtypes), but the set can be extended to next types. Implicitly, each place has assigned a type ε.

D. Constraints

Although the OOPN classes bring more intuitive modeling of behavior, they do not offer intrinsic definitions of invariants, a state of the place, or type checking. Nevertheless, there is very simple way how to define and test these conditions by means of OOPN. The advantage of this approach is that the designer has this feature under the control. We will call these definitions as *constraints*. Each such a constraint is defined formally and the definition is followed by its implementation in OOPN showed in Figure 2.

The test of *empty place* is defined as $\varphi(p) = \nexists x \in p$. It is implemented by the negative predicate `emptyPlace` in the OOPN formalism (see Figure 2). If there is no object in the place, the condition is not satisfied and it implies, that the negative predicate is evaluated as true.

The test of *nonempty place* is defined as $\psi(p) = \exists x \in p$. It is implemented by the synchronous port `nonEmptyPlace` in the OOPN formalism (see Figure 2). If there is at least

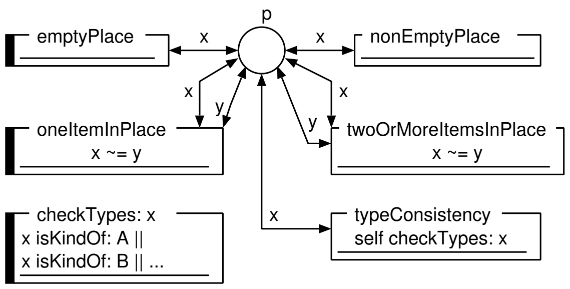


Figure 2. Invariants and testing conditions.

one object in the place, the condition is satisfied and the synchronous port is evaluated as true.

The test of *at most one item* (or *the capacity of the place is 1*) is defined as $\tau(p) = \nexists x, y \in p : x \neq y$. It is implemented by the negative predicate `oneItemInPlace` in the OOPN formalism (see Figure 2). If there is no object or only one object in the place, the conditions are not satisfied and the negative predicate is evaluated as true. In the other cases, it is evaluated as false.

The test of *two or more items* is defined as $\varsigma(p) = \exists x, y \in p : x \neq y$. It is implemented by the synchronous port `twoOrMoreItemsInPlace` in the OOPN formalism (see Figure 2). If there are at least two different objects in the place p, the synchronous port is evaluated as true.

The test of *type consistency* is defined as $\theta(p, ET) = \psi(p) \wedge \exists x \in p : \nexists t \in ET \wedge x \lhd t$. It is implemented by the synchronous port `typeConsistency` and the associated negative predicate `checkTypes:` in the OOPN formalism (see Figure 2). If there is an object x in the place p and there is no type t from the expected types set ET, the conditions of the negative predicate are not satisfied and it implies the negative predicate is evaluated as true. Then the synchronous port is evaluated as true for the object x—it means that this object x does not satisfy the expected types of the place p.

E. Behavior

The behavior B_{PN} is not simply a set of methods because the synchronous ports from interface can influence on the object net during its firing, as mentioned in Section V-B. The object net $n \in ONET$ is defined as a graph of Petri nets. The concrete behavior is usually provided by its part—a valid subnet of the Petri net graph. So we can define $\mathcal{S}(ONET)$ as a set of all valid subnets of the object nets. Then, the behavior B_{PN} can be defined as $B_{PN} \subseteq MNET \cup \mathcal{S}(ONET)$.

VI. Example

This section will present the relationship between UML and OOPN classes. To demonstrate this relationship, a very small part of the PNtalk system [16] was chosen. PNtalk is the tool intended to model and to simulate systems using OOPN. We depict a functionality of the method look-up.

A. UML Class Diagram

By following the design methodology [7], [15], we have to identify roles and use cases and classify them into classes. In the example, the only one role of *object* is identified and its use case *lookFor* (it does not strictly correspond with the real system, but for demonstration it is sufficient). These elements are classified into two classes, the class `Object` for the role and `LookFor` for the activity of method searching (the use case).

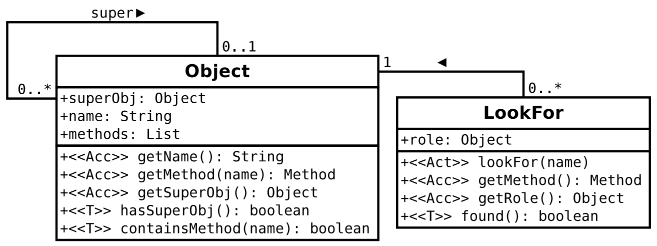

super▶
0..1
0..*
Object
+superObj: Object
+name: String
+methods: List
+<<Acc>> getName(): String
+<<Acc>> getMethod(name): Method
+<<Acc>> getSuperObj(): Object
+<<T>> hasSuperObj(): boolean
+<<T>> containsMethod(name): boolean
1
◀
0..*
LookFor
+role: Object
+<<Act>> lookFor(name)
+<<Acc>> getMethod(): Method
+<<Acc>> getRole(): Object
+<<T>> found(): boolean

Figure 3. The class diagram of the method look-up.

The `Object` has attributes of the object name, the object's superobject (in the terms of inheritance hierarchy), and the list of object's methods. It offers methods for getting values of attributes (see the stereotype `<<Acc>>` in Figure 3) and methods for testing the object's state (see the stereotype `<<T>>` in Figure 3).

The `LookFor` has an attribute of the role the activity is intended for. It offers a method for the look-up (see the stereotype `<<Act>>` in Figure 3), a method for testing the result of searching (see the stereotype `<<T>>` in Figure 3), and methods for getting values (see the stereotype `<<Acc>>` in Figure 3).

B. The class Object

Let us analyze the class `Object`. It contains three instance variables, so that there will be three places in the OOPN class, according to the function $f_V(Object) = \{name \rightarrow name, methods \rightarrow methods, superObj \rightarrow superObject\}$.

We can identify the following operations from the interface: $I_C^{Act}(Object) = \emptyset$, $I_C^{Acc}(Object) = \{getName, getMethod, getSuperObj\}$, $I_C^{T}(Object) = \{hasSuperObj, containsMethod\}$. The class `Object` offers no operations in I_C^{Acc}, so that there is nothing to transform. There are three operations in $I_C^{Acc}(Object)$, that are transformed into synchronous ports: $f_I^{Acc}(Object) = \{getName \rightarrow name{:}, getMethod \rightarrow method{:}named{:}, getSuperObj \rightarrow superObject{:}\}$.

The test group $I_C^{T}(Object)$ offers two operations, that are transformed into synchronous ports and negative predicates: $f_I^{T}(Object) = \{hasSuperObj \rightarrow \{superObject{:}, notSuperObject\}, containsMethod \rightarrow$

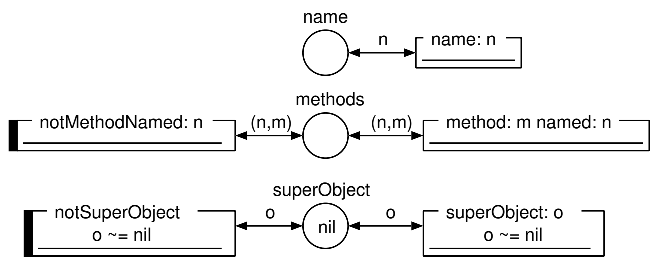


Figure 4. The OOPN class `Object`.

$\{method{:}named{:}, notMethodNamed{:}\}\}$. The synchronous ports allow to get a value of instance variables (using the unification principle) and, at the same time, to test if the variable contains a given value. So, the test operation is transformed usually into a pair of a synchronous port (it allows also for accessing, so that it is a part is the access group) and a negative predicate.

Finally, the interface of the OOPN class `Object` is defined as follows: $I^{Act}_{PN}(Object) = \emptyset$, $I^{Acc}_{PN}(Object) = \{name{:}, method{:}named{:}, superObject{:}\}$, $I^{T}_{PN}(Object) = I^{Acc}_{PN}(Object) \cup \{notMethodNamed{:}, notSuperObject\}$. The graphic notation is shown in Figure 4.

C. The class LookFor

Let us analyze the class `LookFor`. It contains one instance variable, so that there will be one place in the OOPN class, according to the function $f_V(LookFor) = \{role \rightarrow role\}$.

We can identify the following operations from the interface: $I^{Act}_{C}(LookFor) = \{lookFor\}$, $I^{Acc}_{C}(LookFor) = \{getMethod, getRole\}$, $I^{T}_{C}(LookFor) = \{found\}$. There is one operation in $I^{Act}_{C}(LookFor)$, which is transformed into the synchronous port $f^{Act}_{I}(LookFor) = \{lookFor \rightarrow lookFor{:}\}$. There are two operations in $I^{Acc}_{C}(LookFor)$, that are transformed into the synchronous ports $f^{Acc}_{I}(LookFor) = \{getRole \rightarrow role{:}, getMethod \rightarrow found{:}\}$. There is one operation in $I^{T}_{C}(LookFor)$, which is transformed into the synchronous port and the negative predicate $f^{T}_{I}(LookFor) = \{found \rightarrow \{found{:}, failed\}\}$ testing the positive or negative state of the search result.

Finally, the interface of the OOPN class `LookFor` is defined as follows: $I^{Act}_{PN}(LookFor) = \{lookFor{:}\}$, $I^{Acc}_{PN}(LookFor) = \{role{:}, found{:}\}$, $I^{T}_{PN}(LookFor) = I^{Acc}_{PN}(LookFor) \cup \{failed\}$. The graphic notation is shown in Figure 5.

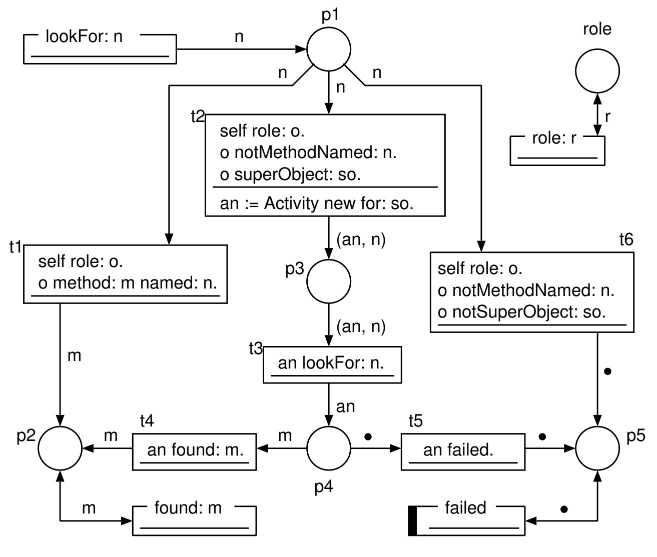


Figure 5. The OOPN class `LookFor`.

D. Behavior

The behavior of the activity net `LookFor` can be divided into three basic subnets (the subnet is described as a set of vertexes, i.e., places and transitions): $\delta_1 = \{p1, t1, p2\}$, $\delta_2 = \{p1, t2, p3, t3, p4, t4, p2, t5, p5\}$, and $\delta_3 = \{p1, t6, p5\}$. The δ_1 is a behavior for a situation if the method is found directly in the object (see the transition $t1$). The δ_3 is a behavior for a situation if the method is not found directly in the object and the object does not have an superobject (see the transition $t6$). The δ_2 is a behavior for a situation if the method is not found directly in the object and the object has an superobject. Then the new activity net is created for the superobject (see the transition $t2$). Then the operation $lookFor{:}$ is called (the transition $t3$) and the result is tested (the transitions $t4$ and $t5$). The places $p2$ and $p5$ store the state of the operation, which can be tested by $found{:}$ and $failed$. The synchronous port $found{:}$ serves even as an access operation for getting the found method.

E. Constraints

Now, we demonstrate an usage of constraints in the class definition. We chosen the place $superObject$ from the class `Object`. First, the place is initialized by a special value nil representing an information that the object does not have a superobject. If the object has an superobject, the value nil is replaced. So there is one invariant: *the place* $superObject$ *contains just one value*. This constraint is tested by $\varsigma(superObject)$. Second, the place can contain only objects of a type `Object`. This constraint is tested by $\theta(superObject, \{Object\})$. Declaration of both constraints in the OOPN class is shown in Figure 6a.

The constraints are realized by synchronous ports or negative predicates. Their definition does not evocate any activity or testing without its calling. Hence, it is possible to define many constraints on the classes with no influence on the system performance. In order to activate the tests, they have to be called, as shown in Figure 6b. The tested object is stored in the place p and the associated transitions provide the appropriate tests. These transitions can be a part

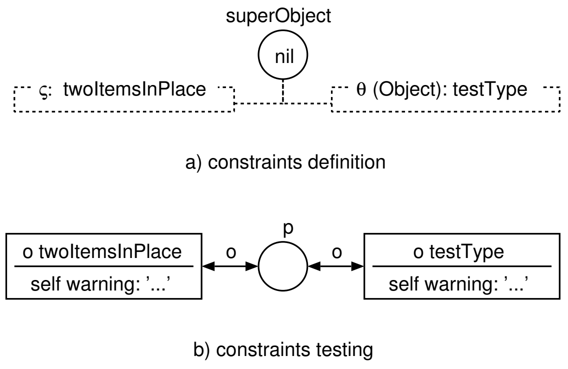


Figure 6. The class `Object`: a) constraints definition and b) testing.

of any object nets (then the transition is fired immediately the condition occurs) or any method net (then the tests are provided on demand).

VII. Conclusion and Future Work

The paper dealt with a formal approach to describe system structure and behavior. Proposed approach extends system modeling using formalism of Object Oriented Petri Nets (OOPN) with selected UML diagrams. First, the class diagram was taken into account. The approach stems from UML classes for system structure specification, where classes behavior is modeled by OOPN. Since the UML classes and OOPN classes differ, the transformation technique has been introduced. The presented approach is a part of the development methodology, which allows to use formal models in all phases of system development. Formal models should be used as basic design, analysis and also programming means with a vision to allow for combining of simulated and real components and to deploy models as the target system with no code generation. Using UML classes together with formalism of OOPN satisfies the development methodology, because one-to-one assignability enables to keep an eye to the system with UML and OOPN formalisms and, together, to use OOPN models as a programming means. In the future, we plan to complete transformation mechanisms with class associations, extend modeling with use case diagrams, and investigate simulation techniques for an assistance in the system modeling.

Acknowledgment: This work has been supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), by BUT FIT grant FIT-S-11-1, and by the Ministry of Education, Youth and Sports under the contract MSM 0021630528.

References

[1] J. Arlow and I. Neustadt, *UML and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison-Wesley Professional, 2001.

[2] S. Beydeda, M. Book, and V. Gruhn, *Model-Driven Software Development*. Springer-Verlag, 2005.

[3] J. Greenfield, K. Short, S. Cook, S. Kent, and J. Crupi, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, 2004.

[4] M. Broy, J. Gruenbauer, D. Harel, and T. Hoare, Eds., *Engineering Theories of Software Intensive Systems: Proceedings of the NATO Advanced Study Institute*. Kluwer Academic Publishers, 2005.

[5] D. S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*, ser. 17 (MS-17). John Wiley & Sons, 2003.

[6] C. Raistrick, P. Francis, J. Wright, C. Carter, and I. Wilkie, *Model Driven Architecture with Executable UML*. Cambridge University Press, 2004.

[7] R. Kočí and V. Janoušek, "System Design with Object Oriented Petri Nets Formalism," in *The Third International Conference on Software Engineering Advances Proceedings ICSEA 2008*. IEEE Computer Society, 2008, pp. 421–426.

[8] R. Kočí and V. Janoušek, "OOPN and DEVS Formalisms for System Specification and Analysis," in *The Fifth International Conference on Software Engineering Advances*. IEEE Computer Society, 2010, pp. 305–310.

[9] M. Češka, V. Janoušek, and T. Vojnar, *PNtalk — a Computerized Tool for Object Oriented Petri Nets Modelling*, ser. Lecture Notes in Computer Science. Springer Verlag, 1997, vol. 1333, pp. 591–610.

[10] R. Kočí and V. Janoušek, *Simulation Based Design of Control Systems Using DEVS and Petri Nets*, ser. Lecture Notes in Computer Science. Springer Verlag, 2009, vol. 5717, pp. 849–856.

[11] R. Valk, "Petri Nets as Token Objects: An Introduction to Elementary Object Nets." in *Jorg Desel, Manuel Silva (eds.): Application and Theory of Petri Nets; Lecture Notes in Computer Science*, vol. 120. Springer-Verlag, 1998.

[12] D. Moldt, "OOA and Petri Nets for System Specification," in *Object-Oriented Programming and Models of Concurrency*. Italy, 1995.

[13] L. Cabac, M. Duvigneau, D. Moldt, and H. Rölke, "Modeling dynamic architectures using nets-within-nets," in *Applications and Theory of Petri Nets 2005. 26th International Conference, ICATPN 2005, Miami, USA*, 2005, pp. 148–167.

[14] Tigris.org, "ArgoUML: open source UML modeling tool," http://argouml.tigris.org/, 2012.

[15] R. Kočí and V. Janoušek, "Modeling and Simulation-Based Design Using Object-Oriented Petri Nets: A Case Study," in *Proceeding of the International Workshop on Petri Nets and Software Engineering 2012*, vol. 851. CEUR, 2012, pp. 253–266.

[16] V. Janoušek and R. Kočí, "Embedding Object-Oriented Petri Nets into a DEVS-based Simulation Framework," in *Proceedings of the 16th International Conference on System Science*, ser. volume 1, 2007, pp. 386–395.

A Report on Using Simplified Function Point Measurement Processes

Luigi Lavazza Geng Liu
Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
luigi.lavazza@uninsubria.it, giulio.liu@gmail.com

Abstract—**Background: Function Point Analysis is widely used, especially to quantify the size of applications in the early stages of development, when effort estimates are needed. However, the measurement process is often too long or too expensive or requires more knowledge than available when development effort estimates are due. To overcome these problems, simplified methods have been proposed to measure Function Points.**
Objectives: The work reported here concerns the experimentation of simplified functional size measurement methods in the sizing of both "traditional" and real-time applications. The goal is to evaluate the accuracy of the sizing with respect to full-fledged Function Point Analysis.
Method: A set of projects, which had already been measured by means of Function Point Analysis, have been measured using the NESMA and Early&Quick Function Points simplified processes: the resulting size measures were compared.
Results: while NESMA indicative method appears to quite overestimate the size of the considered applications, the other methods provide much more accurate estimates of functional size. EQFP methods proved more accurate in estimating the size of non Real-Time applications, while the NESMA estimated method proved fairly good in estimating both Real-Time and non Real-Time applications.
Conclusions: The results of the experiment reported here show that in general it is possible to size software via simplified measurement processes with an acceptable accuracy. In particular, the simplification of the measurement process allows the measurer to skip the function weighting phases, which are usually expensive, since they require a thorough analysis of the internals of both data and operations.

Keywords-Functional Size Measures; Function Points; Simplified measurement processes; Early&Quick Function Points (EQFP); NESMA estimated; NESMA indicative.

I. INTRODUCTION

Function Point Analysis (FPA) [1][4][2][3] is widely used. Among the reasons for the success of FPA is the fact that it can provide measures of size in the early stages of software development, when they are most needed for cost estimation.

However, FPA performed by a certified function point consultant proceeds at a rather slow pace: between 400 and 600 function points (FP) per day, according to Capers Jones [13], between 200 and 300 function points per day according to experts from Total Metrics [14]. As a consequence, measuring the size of a moderately large application can take too long if cost estimation is needed urgently. Also the cost of measurement can be considered excessive by software developers. In addition, cost estimates may be needed when requirements have not yet been specified in detail and completely. To overcome these problems, simplified FP measurement processes have been proposed. Among these are the NESMA (Netherland Software Metrics Association) indicative and estimated methods, and the Early & Quick Function Points method. The proposers of these methods claim that they allow measurers to compute good approximations of functional size measures with little effort and in a fairly short time.

The goal of the work reported here is to test the mentioned simplified functional size measurement processes on real projects in both the "traditional" and real-time domains. Function Points are often reported as not suited for measuring the functional size of embedded applications. The motivation is that FP –conceived by Albrecht when the programs to be sized were mostly Electronic Data Processing applications– capture well the functional size of data storage and movement operations, but are ill-suited for representing the complexity of control and elaboration that are typical of embedded and real-time software. However, a careful interpretation of FP counting rules makes it possible to apply FPA to embedded software as well [10].

In this paper we apply the International Function Points User Group (IFPUG) measurement rules [2] to size a set of programs for non-real time playing on the internet, and we apply the guidelines given in [8] (which are as IFPUG-compliant as possible) to measure a set of embedded real-time avionic applications. All these measures are used to test the accuracy of simplified functional size measurement processes.

The goal of the paper is therefore to evaluate if simplified functional size measurement processes can be used to size real-time and embedded applications, as well as "traditional" business applications.

The paper is organized as follows: Section II briefly introduces the simplified functional size measurement processes used in the paper. Section III describes the projects being measured and gives their sizes measured according to the full-fledged, canonical FPA process. Section IV illustrates the sizes obtained via simplified functional size measurement processes. Section V discusses the accuracy of the measures obtained via the simplified methods used and outlines the lessons that can be learned from the reported experiment.

Section VI accounts for related work. Finally, Section VII draws some conclusions and outlines future work.

II. A BRIEF INTRODUCTION TO SIMPLIFIED SIZE MEASUREMENT PROCESSES

The FP measurement process involves (among others) the following activities:

– Identifying logic data;
– Identifying elementary processes;
– Classifying logic data as internal logic files (ILF) or external interface files (EIF);
– Classifying elementary processes as external inputs (EI), outputs (EO), or queries (EQ);
– Weighting data functions;
– Weighting transaction functions.

Simplified measurement processes allow measurers to skip –possibly in part– one or more of the aforementioned activities, thus making the measurement process faster and cheaper.

The most well-known approach for simplifying the process of FP counting is probably the Early & Quick Function Points (EQFP) method [5][7]. EQFP descends from the consideration that estimates are sometimes needed before the analysis of requirements is completed, when the information on the software to be measured is incomplete or not sufficiently detailed.

Since several details for performing a correct measurement following the rules of the FP manual [2] are not used in EQFP, the result is a less precise measure. The trade-off between reduced measurement time and costs is also a reason for adopting the EQFP method even when full specifications are available, but there is the need for completing the measurement in a short time, or at a lower cost. An advantage of the method is that different parts of the system can be measured at different detail levels: for instance, a part of the system can be measured following the IFPUG manual rules [2][3], while other parts can be measured on the basis of coarser-grained information. In fact, the EQFP method is based on the classification of the processes and data of an application according to a hierarchy (see Figure 1. [7]).

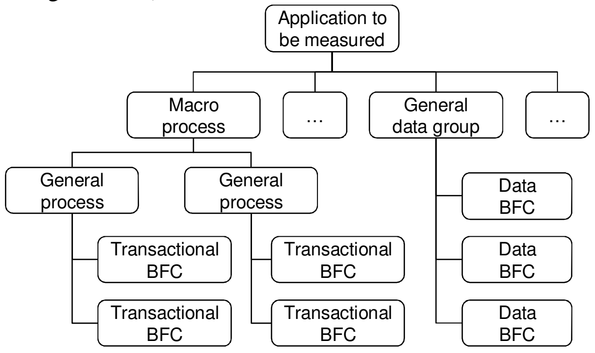


Figure 1. Functional hierarchy in the Early & Quick FP technique

Transactional BFC (Base Functional Components) and Data BFC correspond to IFPUG's elementary processes and LogicData, while the other elements are aggregations of processes or data groups. The idea is that if you have enough information at the most detailed level you count FP according to IFPUG rules; otherwise, you can estimate the size of larger elements (e.g., General or Macro processes) either on the basis of analogy (e.g., a given General process is "similar" to a known one) or according to the structured aggregation (e.g., a General process is composed of 3 Transactional BFC). Therefore, by considering elements that are coarser-grained than the FPA BFC, the EQFP measurement process leads to an approximate measure of size in IFPUG FP.

Tables taking into account the previous experiences with the usage of EQFP are provided to facilitate the task of assigning a minimum, maximum and most likely quantitative size to each component. For instance, TABLE I. provides minimum, maximum and most likely values for generic (i.e., not weighted) functions as given in [7]. Using this method involves the activities indicated in TABLE III. The time and effort required by the weighting phases are saved. Such saving can be relevant, since weighting a data or transaction function requires analyzing it in detail.

TABLE I. EQFP: FUNCTION TYPE WEIGHTS FOR GENERIC FUNCTIONS

Function type	**Complexity**		
	Low	***Likely***	***High***
Generic ILF	7.4	7.7	8.1
Generic EIF	5.2	5.4	5.7
Generic EI	4	4.2	4.4
Generic EO	4.9	5.2	5.4
Generic EQ	3.7	3.9	4.1

The size of unspecified generic processes (i.e., transactions that have not been yet classified as inputs, outputs or queries) and unspecified generic data groups (i.e., logical files that have not been yet classified as ILF or EIF) as given in [7] are illustrated in TABLE II. When using this method, only the identification of logical data and elementary processes needs to be done, as shown in TABLE III. Both the classification of data and transaction functions and their weighting are skipped. Consequently, sizing based on unspecified generic processes and data groups is even more convenient –in terms of time and effort spent– than sizing based on generic (i.e., non weighted) functions.

TABLE II. EQFP: FUNCTION TYPE WEIGHTS FOR UNSPECIFIED GENERIC PROCESSES AND DATA GROUPS

Function type	**Complexity**		
	Low	***Likely***	***High***
Unspefied Generic Processes	4.3	4.6	4.8
Unspefied Generic Data Group	6.4	7.0	7.8

Methods for simplifying the counting of FP, the Indicative NESMA method [6] simplifies the process by only requiring the identification of LogicData from a data model. The Function Point size is then computed by applying predefined weights, whose value depends on whether the data model is normalized in 3rd normal form:

Non normalized model: Function Points = Number of ILF × 35 + Number of EIF × 15

Normalized model: Function Points = Number of ILF × 25 + Number of EIF × 10

The process of applying the NESMA indicative method involves only identifying logic data and classifying logic data as ILF or EIF. Accordingly, it requires less time and effort than the EQFP methods described above, in general. However, it is quite clear that the Indicative NESMA method is quite rough in its computation. The official NESMA counting manual specifies that errors in functional size with this approach can be up to 50%.

The Estimated NESMA method requires the identification and classification of all data and transaction functions, but does not require the assessment of the complexity of each function: Data Functions (ILF and EIF) are assumed to be of low complexity, while Transactions Functions (EI, EQ and EO) are assumed to be of average complexity. Accordingly, the Estimated NESMA method is expected to be more approximated than the EQFP method based on generic functions, as the latter uses *likely* values for transactions of unknown complexity, derived from statistic analysis.

TABLE III. ACTIVITIES REQUIRED BY DIFFERENT SIMPLIFIED MEASUREMENT PROCESSES

Measurement activities	**IFPUG**	**NESMA indic.**	**NESMA estim.**	**EQFP Generic func.**	**EQFP Unspec. generic func.**
Identifying logic data	✓	✓	✓	✓	✓
Identifying elementary processes	✓		✓	✓	✓
Classifying logic data as ILF or EIF	✓	✓	✓	✓	
Classifying elementary processes as EI, EO, or EQ	✓		✓	✓	
Weighting data functions	✓				
Weighting transaction functions	✓				

The activities required by the simplified functional size measurement methods considered in the paper are reported in TABLE III. Of course, the IFPUG method requires all the activities listed in TABLE III.

III. THE CASE STUDY

A. Real-time projects

The real-time projects measured are from a European organization that develops software for avionic applications, and for other types of embedded and real-time applications.

The projects' FUR were modeled using UML as described in [9], and were then measured according to IFPUG measurement rules [2]. When the real-time nature of the software made IFPUG guidelines inapplicable, we adopted ad-hoc counting criteria, using common sense and striving to preserve the principles of Function Point Analysis, as described in [10].

The same projects were then sized using the NESMA and EQFP simplified functional size measurement processes, using the data that were already available as a result of the IFPUG measurement.

All the measured projects concerned typical real-time applications for avionics or electro-optical projects, and involved algorithms, interface management, process control and graphical visualization.

For each project the measurement of the functional size was carried out in two steps. First, a model of the product was built. The models were written in UML and represented the requirements, including all the information needed for the measurement of FPs and excluding the unnecessary details [9]. Then, the function points were counted, on the basis of the model, according to IFPUG rules.

TABLE IV. reports the size in FP of the measured projects, together with the BFC and –in parentheses– the number of unweighted BFC. For instance, project 1 involved 18 Internal Logic Files, having a size of 164 FP.

TABLE IV. REAL-TIME PROJECTS' SIZES (IFPUG METHOD)

Project ID.	**ILF**	**EIF**	**EI**	**EO**	**EQ**	**FP**
1	164 (18)	5 (1)	90 (21)	8 (2)	22 (5)	289
2	56 (8)	0 (0)	21 (6)	18 (3)	6 (1)	101
3	73 (7)	0 (0)	12 (2)	47 (8)	4 (1)	136
4	130 (15)	15 (3)	44 (11)	0 (0)	6 (1)	195
5	39 (4)	0 (0)	28 (8)	39 (8)	0 (0)	106
6	71 (9)	5 (1)	8 (2)	139 (28)	0 (0)	223
7	7 (1)	0 (0)	3 (1)	5 (1)	0 (0)	15

B. Non Real-time projects

The non real-time project considered are programs that allow users to play board or card games vs. remote players via the internet.

The projects were measured –as the real-time ones– in two steps: the UML model of each product was built along the guidelines described in [9]; then, the function points were counted, on the basis of the model, according to IFPUG rules.

TABLE V. reports the size in FP of the measured projects, together with the BFC and –in parentheses– the number of unweighted BFC.

TABLE V. NON REAL-TIME PROJECTS' SIZES (IFPUG METHOD)

Project ID.	**ILF**	**EIF**	**EI**	**EO**	**EQ**	**FP**
1	45 (6)	7 (1)	34 (10)	6 (1)	0 (0)	92
2	28 (4)	20 (4)	37 (9)	5 (1)	4 (1)	94
3	21 (3)	5 (1)	27 (7)	8 (2)	18 (6)	79
4	31 (4)	0 (0)	49 (16)	13 (3)	3 (1)	96
5	24 (3)	0 (0)	45 (14)	21 (5)	0 (0)	90
6	49 (7)	0 (0)	36 (9)	0 (0)	6 (2)	91

IV. Results of Simplified Measurement

Simplified measurement processes were applied following their definitions, which require data that can be easily derived from the tables above. So, for instance, the data required for Real-Time project 1 are the following:

– The NESMA indicative method requires the numbers of ILF and EIF. TABLE I. shows that the number of ILF is 18, and the number of EIF is 1.
– The NESMA estimated method and the EQFP generic functions methods require the numbers of ILF, EIF, EI, EO and EQ. TABLE I. shows that the numbers of ILF, EIF, EI, EO and EQ are, respectively, 18, 1, 21, 2, and 5.
– The EQFP unspecified generic functions method requires the numbers of data groups (that is, the number of ILF plus the number of EIF) and the number of transactions (that is, the sum of the numbers of EI, EO and EQ). TABLE I. shows that the number of data groups is 18+1 = 19, and the number of transactions is 21+2+5 = 28.

A. Applying NESMA indicative

The applications to be measured were modeled according to the guidelines described in [9]. The logic data files –modeled as UML classes– provide a data model that cannot be easily recognized as normalized or not normalized. Therefore, we applied both the formulae for the normalized and not normalized models.

TABLE VI. MEASURES OF REAL-TIME PROJECTS OBTAINED VIA THE NESMA METHODS

Project ID	IFPUG	NESMA indicative non normalized	NESMA indicative normalized	NESMA estimated
1	289	645	460	245
2	101	280	200	99
3	136	245	175	101
4	195	570	405	168
5	106	140	100	100
6	223	330	235	216
7	15	35	25	16

TABLE VII. MEASURES OF NON REAL-TIME PROJECTS OBTAINED VIA THE NESMA METHODS

Project ID	IFPUG	NESMA indicative non normalized	NESMA indicative normalized	NESMA estimated
1	92	225	160	81
2	94	200	140	82
3	79	120	85	73
4	96	140	100	91
5	90	105	75	83
6	91	245	175	82

The formulae of the NESMA indicative method were applied to the number of ILF and EIF that had been identified during the IFPUG function point counting process. The results are given in TABLE VI. for Real-Time projects and in TABLE VII. for non Real-Time projects.

B. Applying NESMA estimated

The formulae of the NESMA indicative method were applied to the number of ILF, EIF, EI, EO, and EQ that had been identified during the IFPUG function point counting process. The results are given in TABLE VI. for Real-Time projects and in TABLE VII. for non Real-Time projects.

C. Applying EQFP

As described in Figure 1. , the EQFP method can be applied at different levels. Since we had the necessary data, we used the BFC aggregation level.

TABLE VIII. MEASURES OF REAL-TIME PROJECTS OBTAINED VIA THE EQFP METHOD

Project ID	IFPUG	EQFP – unspecified generic processes and data groups	EQFP –generic transactions and data files
1	289	262	262
2	101	102	106
3	136	100	108
4	195	181	182
5	106	102	106
6	223	208	229
7	15	16	17

At this level it is possible to use the data functions and transaction functions without weighting them or even without classifying transactions into EI, EO and EQ and logic data into ILF and EIF. In the former case (generic functions) the weights given in TABLE I. are used, while in the latter case (unspecified generic functions) the weights given in 0are used.

The results of the application of EQFP are given in TABLE VIII. for Real-Time projects, and in TABLE IX. for non Real-Time projects.

TABLE IX. MEASURES OF NON REAL-TIME PROJECTS OBTAINED VIA THE EQFP METHOD

Project ID	IFPUG	EQFP – unspecified generic processes and data groups	EQFP –generic transactions and data files
1	92	100	99
2	94	107	99
3	79	97	92
4	96	120	118
5	90	108	108
6	91	100	100

V. Summary and Lessons Learned

To ease comparisons, all the size measures of RT projects are reported in TABLE X. and those of non RT projects are reported in TABLE XI.

TABLE X. MEASURES OF REAL-TIME PROJECTS OBTAINED VIA THE VARIOUS METHODS

Proj ID	IFPUG	NESMA ind. non norm.	NESMA ind. norm.	NESMA estim.	EQFP unspec.	EQFP generic
1	289	645	460	245	262	262
2	101	280	200	99	102	106
3	136	245	175	101	100	108
4	195	570	405	168	181	182
5	106	140	100	100	102	106
6	223	330	235	216	208	229
7	15	35	25	16	16	17

TABLE XI. MEASURES OF NON REAL-TIME PROJECTS OBTAINED VIA THE VARIOUS METHODS

Proj ID	IFPUG	NESMA ind. non norm.	NESMA ind. norm.	NESMA estim.	EQFP unspec.	EQFP generic
1	92	225	160	81	100	99
2	94	200	140	82	107	99
3	79	120	85	73	97	92
4	96	140	100	91	120	118
5	90	105	75	83	108	108
6	91	245	175	82	100	100

It is easy to see that the NESMA indicative method yields the greatest errors. On the contrary, the NESMA estimated and EQFP methods yield size estimates that are close to the actual size.

The relative measurement errors are given in TABLE XII. and TABLE XIII. , where the least error for each project is in bold. It is easy to see that the NESMA indicative methods are generally outperformed by the other methods. For Real-Time projects EQFP (either in the unspecified or generic flavor) tend to provide the most accurate results, while the NESMA estimated method provides quite reasonable estimates. For non Real-Time projects the NESMA estimated method appears even better than the EQFP methods. Quite noticeably, NESMA estimated underestimates all non Real-Time projects except one.

TABLE XII. RELATIVE MEASUREMENT ERRORS (REAL-TIME PROJECTS)

Proj ID	NESMA ind. non norm.	NESMA ind. norm.	NESMA estim.	EQFP unspec.	EQFP generic
1	123%	59%	-15%	**-9%**	**-9%**
2	177%	98%	-2%	**1%**	5%
3	80%	29%	-26%	-26%	**-21%**
4	192%	108%	-14%	**-7%**	**-7%**
5	32%	-6%	-6%	-4%	**0%**
6	48%	5%	**-3%**	-7%	**3%**
7	133%	67%	**7%**	**7%**	13%

TABLE XIII. RELATIVE MEASUREMENT ERRORS (NON REAL-TIME PROJECTS)

Proj ID	NESMA ind. non norm.	NESMA ind. norm.	NESMA estim.	EQFP unspec.	EQFP generic
1	145%	74%	-12%	9%	**8%**
2	113%	49%	-13%	14%	**5%**
3	52%	8%	**-8%**	23%	16%
4	46%	4%	**-5%**	25%	23%
5	17%	-17%	**-8%**	20%	20%
6	169%	92%	**-10%**	**10%**	**10%**

The accuracy of the used methods is summarized in TABLE XIV. , where the mean and standard deviation of the *absolute* relative errors are given for Real-Time projects, for non Real-Time projects and for the entire set of projects.

TABLE XIV. MEAN AND STDEV OF ABSOLUTE RELATIVE ERRORS

	NESMA ind. non norm.	NESMA ind. norm.	NESMA estim.	EQFP unspec.	EQFP generic
Mean (RT only)	112%	53%	10%	9%	8%
Stdev (RT only)	62%	42%	8%	8%	8%
Mean (non RT)	90%	41%	9%	17%	14%
Stdev (non RT)	61%	37%	3%	7%	7%
Mean (all)	102%	47%	10%	13%	11%
Stdev (all)	60%	38%	6%	9%	8%

VI. RELATED WORK

Meli and Santillo were among the first to recognize the need of comparing the various functional size methods proposed in the literature [18]. To this end, they also provided a benchmarking model.

In [12], van Heeringen et al. report the results of measuring 42 projects with the full-fledged, indicative and estimated NESMA methods. They found a 1.5% mean error of NESMA estimated method and a 16.5% mean error of NESMA indicative method.

Using a database of about 100 applications, NESMA did some research on the accuracy of the estimated and indicative function point counts. They got very good results (http://www.nesma.nl/section/fpa/earlyfpa.htm), although no statistics (e.g., mean relative error) are given.

In [16] Frank Vogelezang summarized the two techniques to simplified measuring given in the COSMIC measurement manual: the approximate technique (comparable to NESMA's indicative technique) and the refined approximate technique (comparable to NESMA's rough technique). In the approximate technique the average size of a functional process is multiplied with the number of functional processes the software should provide. In the refined approximate technique the functional processes to be provided can already be classified as small, medium, large or very large, each with its own average size. The precision of the COSMIC-FFP approximate technique is good enough with less than 10% deviation on a portfolio and less than 15% on a project within a specified environment [16].

Popović and Bojić compared different functional size measures –including NESMA indicative and estimated– by evaluating their accuracy in effort estimation in various phases of the development lifecycle [15]. Not surprisingly, they found that the NESMA indicative method provided the best accuracy at the beginning of the project. With respect to Popović and Bojić, we made two quite different choices: the accuracy of the method is evaluated against the actual size of the software product, not the required development effort, and –consistently– all the information needed to perform measurement is available to all processes.

There is no indication that real-time projects were among those measured by van Heeringen et al. or by NESMA.

VII. Conclusion

Sometimes, FPA is too slow or too expensive for practical usage. Moreover, FPA requires a knowledge of requirements that may not be available when the measures of size are required, i.e., at the very first stages of development, when development costs have to be estimated. To overcome these problems, simplified measurement processes have been proposed.

In this paper we applied simplified functional size measurement processes to both traditional software applications and Real-Time applications. The obtained results are fairly good, as a few of the tested methods provided average errors not greater than 13% (with fairly small standard deviations).

EQFP methods proved more accurate in estimating the size of non Real-Time applications, while the NESMA estimated method proved fairly good in estimating both Real-Time and non Real-Time applications. However, the relatively small number of projects involved in the analysis does not allow generalizing these results.

Even considering the relatively small dataset, it is however probably not casual that the NESMA estimated method happened to underestimate *all* projects. Probably NESMA should consider reviewing the weights used in the estimated method, in the sense of increasing them.

It is noticeable that all the used methods underestimated one of the Real-Time projects by over 20%. This can be quite dangerous, as underestimating size usually leads to dramatically underestimating the development effort, which can very easily cause the failure of the project. Our observations seem to suggest that the projects that are most likely to be underestimated by simplified methods are those characterized by the need to store or communicate many data at a time. The frequent occurrence of this condition should be checked before adopting a simplified measurement process.

Finally, we must point out that the results presented here are based on datasets in which the largest project has size of 289 FP. Further work for verifying the precision of simplified measurement methods when dealing with larger project is needed.

Among the future work is also the experimentation of simplified measurement processes in conjunction with measurement-oriented UML modeling, as described in [11].

Acknowledgment

The research presented in this paper has been partially supported by the project "Metodi, tecniche e strumenti per l'analisi, l'implementazione e la valutazione di sistemi software" funded by the Università degli Studi dell'Insubria

References

[1] A.J. Albrecht, Measuring Application Development Productivity, *Joint SHARE/ GUIDE/IBM Application Development Symposium*, 1979.

[2] International Function Point Users Group. Function Point Counting Practices Manual - Release 4.3.1, 2010.

[3] ISO/IEC 20926: 2003, Software engineering – IFPUG 4.1 Unadjusted functional size measurement method – Counting Practices Manual, ISO, Geneva, 2003.

[4] A.J. Albrecht and J.E. Gaffney, "Software function, lines of code and development effort prediction: a software science validation" IEEE Transactions on Software Engineering 9(6), 1983.

[5] L. Santillo, M. Conte and R. Meli, "Early & Quick function point: sizing more with less", *Software Metrics, 2005. 11th IEEE International Symposium*, Como, 19-22 Sept. 2005.

[6] ISO, Iec 24570: 2004, Software Engineering-NESMA Functional Size Measurement Method version 2.1 - Definitions and Counting Guidelines for the Application of Function Point Analysis. International Organization for Standardization, Geneva, 2004.

[7] Early & Quick Function Points for IFPUG methods v. 3.1 Reference Manual 1.1, April 2012.

[8] L. Lavazza and C. Garavaglia, "Using Function Point in the Estimation of Real-Time Software: an Experience", Software Measurement European Forum – SMEF 2008, Milano, 28-30 May 2008.

[9] L. Lavazza, V. del Bianco, C. Garavaglia, "Model-based Functional Size Measurement", ESEM 2008, 2nd International Symposium on Empirical Software Engineering and Measurement, Incorporating ISESE and Metrics, Kaiserslautern, Germany. October 9-10, 2008.

[10] L. Lavazza and C. Garavaglia, "Using Function Points to Measure and Estimate Real-Time and Embedded Software: Experiences and Guidelines", ESEM 2009, 3rd Int. Symp. on Empirical SW Engineering and Measurement, October 15-16, 2009, Lake Buena Vista, Florida.

[11] V. del Bianco, L. Lavazza, and S. Morasca, "A Proposal for Simplified Model-Based Cost Estimation Models", 13th International Conference on Product-Focused Software Development and Process Improvement – PROFES 2012, Madrid, June 13-15, 2012.

[12] H. van Heeringen, E. van Gorp, and T. Prins, "Functional size measurement - Accuracy versus costs - Is it really worth it?", Software Measurement European Forum – SMEF, Rome, 28 - 29 May 2009

[13] C. Jones, "A new business model for function point metrics", http://www.itmpi.org/assets/base/images/itmpi/privaterooms/capersjones /FunctPtBusModel2008.pdf, 2008

[14] "Methods for Software Sizing – How to Decide which Method to Use", Total Metrics, www.totalmetrics.com/function-point-resources/downloads/R185_Why-use-Function-Points.pdf, August 2007.

[15] J. Popović and D. Bojić, "A Comparative Evaluation of Effort Estimation Methods in the Software Life Cycle", Computer Science and Information Systems, Vol. 9, n. 1, January 2012)

[16] F.W. Vogelezang, "COSMIC Full Function Points, the Next Generation", in Measure! Knowledge! Action! – The NESMA anniversary book, NESMA, 2004.

[17] F.W. Vogelezang, A.J.E. Dekkers, "One year experience with COSMIC-FFP", Software Measurement European Forum – SMEF 2004, January 28-30, Rome, 2004.

[18] R. Meli and L. Santillo, "Function point estimation methods: a comparative overview", FESMA '99, Amsterdam October, 4-8, 1999.

An Empirical Evaluation of Effort Prediction Models Based on Functional Size Measures

Luigi Lavazza Sandro Morasca
Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
e-mail: {luigi.lavazza, sandro.morasca}@uninsubria.it

Gabriela Robiolo
Facultad de Ingeniería
Universidad Austral
Buenos.Aires, Argentina
e-mail: grobiolo@austral.edu.ar

***Abstract*— Software development effort estimation is among the most interesting issues for project managers, since reliable estimates are at the base of good planning and project control. Several different techniques have been proposed for effort estimation, and practitioners need evidence, based on which they can choose accurate estimation methods.**

The work reported here aims at evaluating the accuracy of software development effort estimates that can be obtained via popular techniques, such as those using regression models and those based on analogy.

The functional size and the development effort of twenty software development projects were measured, and the resulting dataset was used to derive effort estimation models and evaluate their accuracy.

Our data analysis shows that estimation based on the closest analogues provides better results for most models, but very bad estimates in a few cases. To mitigate this behavior, the correction of regression toward the mean proved effective.

According to the results of our analysis, it is advisable that regression to the mean correction is used when the estimates are based on closest analogues. Once corrected, the accuracy of analogy-based estimation is not substantially different from the accuracy of regression based models.

Keywords- Functional size measurement; function points; effort estimation; Regression Toward the Mean; Least Median of Squares.

I. INTRODUCTION

Several different types of models have been proposed for estimating the effort required to develop a software system whose functional size is known.

In this paper, we use a dataset of 20 projects to evaluate the accuracy of different estimation models. For each project in the dataset, we assume that the other 19 projects' sizes and effort data are known, and that the considered project development effort is estimated based on this data.

The model types considered are:

$$\text{Estimated\_Effort} = a + b \times \text{Size} \quad (1)$$

$$\text{Estimated\_Effort} = a\ \text{Size}^b \quad (2)$$

$$\text{Estimated\_Effort} = \text{Size} / \text{Productivity} \quad (3)$$

where Productivity is defined as the ratio Size/Effort; Effort is measured in person hours and Size is measured in Function Points [1][2].

Models of type (1) are obtained via Ordinary Least Square (OLS) and Least Median of Squares (LMS) linear regressions, while models of type (2) are obtained via OLS regression after log-log transformation. Models of type (3) are obtained using two different values of productivity:

a) Productivity_{CA}: the productivity of the projects that are Closest Analogues (CA) to the project to be estimated.
b) $\text{Productivity}_{RTM}$: the productivity obtained from Productivity_{CA} by correcting Regression Toward the Mean (RTM).

The goal of the paper is to evaluate the accuracy of software development effort estimates obtained via popular techniques, such as regression and analogy. These estimation techniques are among the most used by practitioners. Maybe LMS is not so widely used as the other ones, however, since its introduction [3], LMS has been used in several Empirical Software Engineering studies (a list appears in [4]). The great advantage of LMS for practitioners is that it takes the burden of dealing with outlier identification and exclusion away from the user. A disadvantage for practitioners is that LMS excludes half of the datapoints from the model, so that relatively large datasets are needed to apply it.

More sophisticated techniques were not considered because they have not yet achieved great popularity among practitioners. In fact, our paper is mainly directed to practitioners that have collected –often with some difficulty and effort– a set of historical data, and wonder what is the best way to use this data. Accordingly, we show how some popular estimation methods can be applied to historical datasets of average size, and what the accuracy of the resulting estimates is. This may help practitioners choosing among the many available types of effort estimation models.

The paper is organized as follows: Section 2 describes the dataset and illustrates the derivation of models for every project in the dataset, and the application of such models to get effort estimates. Section 3 evaluates the accuracy of the obtained estimates. Section 4 discusses the threats to the validity of this study. Section 5 accounts for related work. Finally, Section 6 discusses the results found, draws some conclusions and outlines future work.

II. MODEL BUILDING

The analysed projects are a superset of those described in [5]. Also, the data is available from the authors on request.

They were selected because they have the following characteristics:

a) Requirements specifications were documented in a homogeneous way, namely via use cases.
b) Use cases were completely implemented, therefore the effort employed in every project concerns the same overall activity, consisting in complete implementation.
c) The hours worked in each project were homogeneously and accurately registered.

Some projects were developed at Universidad Austral, in a software engineering undergraduate context, as an assignment, consisting in the development of a business application. The hours worked on programming were verified and measured by the team leader and by a professor, as this was one of the academic requirements. The other projects were developed in two different contexts: the System and Technology (S&T) Department at Universidad Austral and a CMM level 4 Company. The S&T Department develops software for the university and other parties, with a contractual relationship with the students, similar to that one they would have in a company. The hours worked on programming were obtained from the company registration files. The information about such hours was used in each company to do quality control or for future project estimation. The involved human resources in both environments shared a similar profile: advanced undergraduate students –who had been similarly trained– worked in academy, at the S&T Department and at the CMM level 4 Company.

Table I reports the values of Productivity$_M$ (the mean productivity), Productivity$_{CA}$, and Productivity$_{RTM}$. These productivity values were computed for each project by taking into account the rest of the project data. So, for instance, ProductivityM for project 1 is given by the mean of the productivity values of projects 2 to 20.

Productivity$_{CA}$ was computed as the mean productivity of the two projects having minimum size distance with respect to the considered project. In case three or more projects had the same distance, they were all considered. Let us consider project 5, which has size of 110 UFP: the closest analogue is project 14 (distant just 1 UFP), while projects 4 and 13 are at a distance of 3 UFP. In this case, ProductivityCA is given by the average of the productivities of projects 4, 13 and 14, i.e., Productivity values (113/285 + 107/348 + 111/242.5)/3 = 0.386 UFP/PersonHour.

Having computed Productivity$_{CA}$, it was then possible to check for the conditions under which the regression to the mean phenomenon is bound to occur. In our case, the RTM is expected to occur in Productivity$_{CA}$ with respect to the actual productivity, which is given by the ratio size/effort.

The conditions for RTM are: a) the distributions of the actual productivity and Productivity$_{CA}$ are normal, b) they have similar variance, and c) they are not perfectly correlated. In our case, all these conditions are satisfied since:

TABLE I. PRODUCTIVITY VALUES

ProjID	Actual	Mean	CA	RTM corrected
1	0.451	0.397	0.543	0.515
2	0.568	0.391	0.450	0.438
3	0.447	0.397	0.240	0.270
4	0.396	0.400	0.515	0.493
5	0.335	0.403	0.386	0.389
6	0.434	0.398	0.269	0.294
7	0.170	0.412	0.266	0.294
8	0.296	0.405	0.072	0.136
9	0.867	0.375	0.786	0.707
10	0.658	0.386	0.525	0.498
11	0.878	0.375	0.553	0.519
12	0.161	0.412	0.236	0.270
13	0.307	0.405	0.317	0.333
14	0.458	0.397	0.389	0.391
15	0.133	0.414	0.179	0.224
16	0.401	0.400	0.373	0.378
17	0.361	0.402	0.386	0.389
18	0.595	0.389	0.256	0.281
19	0.037	0.419	0.072	0.139
20	0.039	0.419	0.068	0.135

a) the Shapiro-Wilk test applied to Productivity$_{CA}$ and the actual productivity rejects the hypothesis of non-normality (p-value > 0.4 for both distributions);
b) the standard deviations of the two distributions are similar (being 0.24 and 0.19);
c) the Pearson correlation factor r is 0.808 (p-value < 10^{-3}). Accordingly, the percent of regression effects –computed via the formula $(1 - r) \times 100$– is 19.2%.

Even though the effect of regression toward the mean is not extremely relevant, we wanted to check whether RTM could still provide good results (as in [6]) or even better ones, when compared to other techniques. Moreover, some values of Productivity$_{CA}$ are quite far from the actual productivity: it is thus worthwhile trying RTM correction to see if such deviations can be eliminated. To this end, we applied a correction formula suggested by Campbell and Kenny [7]:

$$\text{Productivity}_{RTM} = \text{Productivity}_{CA} + (1\text{-}r) \times (\text{Productivity}_M - \text{Productivity}_{CA}) \quad (4)$$

The resulting values of Productivity$_{RTM}$ are given in the rightmost column of Table I: it is easy to see that the RTM correction decreases high values of productivity and increases the small ones. This is exactly what RTM correction is expected to do. We shall evaluate if such correction actually improves the accuracy of estimates.

We also computed Effort vs. Size models using OLS regression, both with and without log-log transformation, and using LMS regression. As for productivity, each project's model was derived by excluding the project's data from the dataset.

When deriving the models via OLS regression, we excluded outliers according to Cook's distance [8]. Cook's distance is commonly used to identify projects that jointly exhibit a large influence and large residual. Projects with Cook's distance greater than 4/n, where n represents the total number of projects, are considered to have a high influence on the results [9].

This explains why some projects are associated with the same model. Consider for instance project 7: during the computation, Cook's distance of project 8 suggests that project 8 be excluded from the dataset. Similarly, project 7 is an outlier for project 8, according to Cook's distance. Therefore, the models for projects 7 and 8 are computed over the same dataset, which excludes the data from projects 7 and 8.

LMS linear regressions compute the model using only half the available data, thus it is quite expected that several projects share the same model.

We applied the models found to get effort estimates. The resulting estimates are given in Table II.

TABLE II. EFFORT ESTIMATES

ID	Actual Effort	Estimates				
		Lin. OLS	LogLog OLS	Linear LMS	CA	RTM corr.
1	–	322	351	341	341	360
2	–	433	494	405	598	614
3	–	321	332	331	712	633
4	285	279	285	287	220	229
5	328	237	232	216	285	283
6	198	208	196	270	320	293
7	442	190	175	235	282	255
8	723	408	405	451	2972	1574
9	392	605	586	647	433	481
10	–	369	369	397	341	359
11	131	263	271	297	208	222
12	1042	336	334	380	712	623
13	348	230	225	211	338	321
14	243	279	283	285	285	284
15	300	107	106	181	223	178
16	147	169	144	250	158	156
17	169	168	136	251	158	157
18	121	196	182	230	282	256
19	16809	1049	951	1087	8600	4484
20	5221	387	385	431	2972	1488

A 0.05 statistical significance threshold was used throughout the paper, as is customary in Empirical Software Engineering studies. All the results reported in the paper are characterized by p-value < 0.05. All validity requirements for the proposed models (e.g., the normal distribution of residuals of OLS regressions) were rigorously verified.

III. EVALUATION OF MODELS

After having obtained the estimates for all projects using the different models (Table II), we computed the estimation errors as the differences between the actual and estimated efforts.

With effort estimation, the size of an error is possibly not as relevant as its relative size. For instance, a 10 PersonMonth error is generally more easily accepted for a 200 PersonMonth project than a 4 PersonMonth error is considered acceptable for a 12 PersonMonth project. In fact, even though the former error is two and a half times the latter, it is just 5% of the entire effort, while in the second case it is 33%.

Accordingly, we computed the relative errors of the estimates, and reported them in Table III. Table III shows that the biggest errors occur with the estimation based on analogy. It is noticeable that the four biggest (in absolute terms) errors with Productivity$_{CA}$ (concerning projects 8, 18, 3 and 6) are effectively reduced by the RTM correction. However, RTM correction has also the effect of increasing the estimation error for some projects (see, for instance, projects 11 and 15).

TABLE III. ESTIMATION RELATIVE ERRORS

ID	Linear OLS	Loglog OLS	Linear LMS	CA	RTM corr.
1	–	-14%	-17%	-17%	-12%
2	–	4%	-14%	26%	30%
3	–	-13%	-13%	86%	65%
4	-2%	0%	1%	-23%	-20%
5	-28%	-29%	-34%	-13%	-14%
6	5%	-1%	36%	62%	48%
7	-57%	-60%	-47%	-36%	-42%
8	-44%	-44%	-38%	311%	118%
9	54%	50%	65%	10%	23%
10	–	35%	46%	25%	32%
11	101%	107%	127%	59%	69%
12	-68%	-68%	-64%	-32%	-40%
13	-34%	-35%	-39%	-3%	-8%
14	15%	17%	18%	18%	17%
15	-64%	-65%	-40%	-25%	-40%
16	15%	-2%	70%	7%	6%
17	-1%	-20%	49%	-7%	-7%
18	62%	50%	90%	133%	111%
19	-94%	-94%	-94%	-49%	-73%
20	-93%	-93%	-92%	-43%	-71%

The distribution of errors is represented via boxplots in Figure 1, where the errors concerning projects 19 and 20 are omitted. As shown in Table II, all models estimate these projects with large errors. Including them in the boxplot would have resulted in squeezing the plots, thus making them hardly readable.

The distribution of relative errors is represented via boxplots in Figure 2. The mean value of errors is represented as a diamond on each boxplot. It is easy to see that the estimation based on the closest analogues provides good accuracy with respect to other models, except for a single project (project 8). It is also quite evident that the RTM correction eliminates such anomaly, though negative errors worsen slightly.

To fully appreciate the differences in accuracy, it is useful to look also at absolute relative errors, which are illustrated by the boxplots in Figure 3. It is also quite apparent in Figure 3 that the RTM correction eliminates the problem of huge relative estimation errors, at the expense of a higher median absolute relative error.

However, the effects of RTM correction should be evaluated by taking into consideration the effects on the projects (19 and 20) that required the biggest effort. By looking at Table II, it is easy to see that the effort of such projects is estimated much better by analogy without correction than with RTM correction. The fact that the corrections concerning these projects are relatively smaller than others (e.g. the one concerning project 8) does not imply that they are acceptable. Actually, all the models based on regression consider projects 19 and 20 outliers, thus excluding them from the models. Estimation based on closest analogues is the only way of taking into account these projects, but ,unfortunately, RTM corrections in these cases operate in the wrong direction, decreasing estimates that are already underestimated. In conclusion, we must be aware that projects (like 19 and 20) that feature quite unusual productivity values, can reduce the effectiveness of RTM correction.

Concerning RTM correction, our results are similar to those reported in the literature. In particular, the MMRE and MdMRE [10] for our set of projects (see Table IV), are close to those reported in [6] and [11].

Table V summarizes the results of some representative papers. It can be seen that our results (given in Table IV), are in line with these studies.

TABLE IV. MMRE AND MDMRE OF ANALOGY BASED ESTIMATES

	CA	RTM
MMRE	49.3%	42.3%
MdMRE	25.5%	36.0%

TABLE V. ACCURACY OF ESTIMATES REPORTED IN THE LITERATURE

Ref.	Results	
	MMRE	MdMRE
[12]	[23.2 -51.1]	[14.8 – 48.0]
[13]	[36.15 – 73.85]	[14.23 – 44.95]
[14]	[11.3 – 32.8]	[7. 2– 24.3]
[15]	[32.82 – 82.20]	[20.44 – 50.54]
[16]	N.A.	[26 – 85]

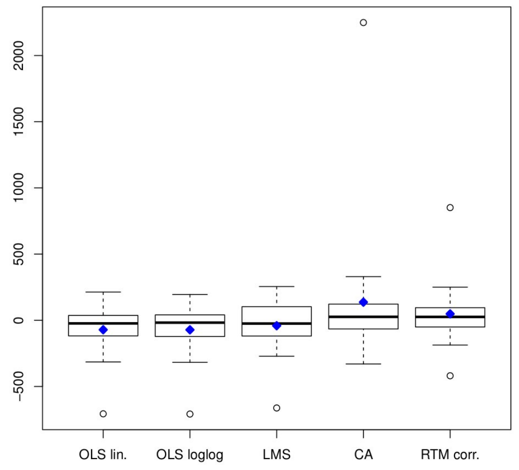


Figure 1. Boxplot of errors.

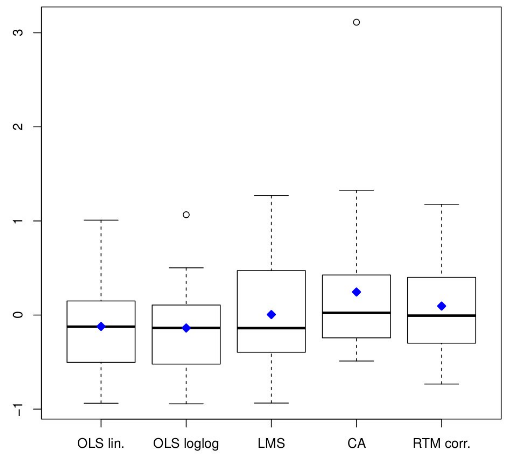


Figure 2. Boxplot of relative errors.

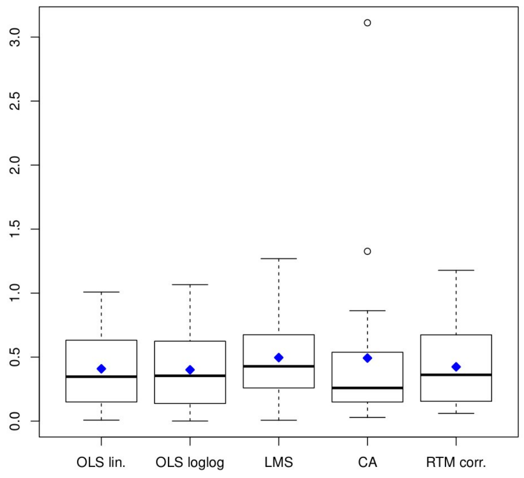


Figure 3. Boxplot of absolute relative errors.

IV. Threats to Validity

Some of the projects that originated the dataset were carried out in industry, while some others were carried out in academia. So, treating all these projects as a single class of projects could be inaccurate, in principle. To make data as homogeneous as possible, academic developments were organized and conducted as industrial ones.

During the construction of models, we tested alternate ways of searching for analogue projects; one included the usage of projects carried out in the same environment. In such case, we also used different Productivity$_M$ to correct RTM. So, for instance, the productivity of academic projects was estimated on the basis of the academic projects of similar size. Then RTM was corrected using in equation (4) the mean productivity of academic projects only. However, taking into account the development environment did not change much the results presented in Section III.

Another issue that deserves attention is the size of the considered projects: only three projects are substantially bigger than 200 FP. Accordingly, practitioners have to be cautious when applying the results reported in this paper to larger projects.

V. Related Work

The phenomenon of "regression towards the mean" (RTM) is thoroughly described in [7]. RTM occurs where the estimation model is inaccurate and extreme observations appear, i.e., the values of the attribute of interest are much higher or lower than the expected value. The presence of these "extreme" values calls for the correction of regression models. Several adjustment approaches were proposed by Campebell and Kenny [7]. Jørgensen et al. were the first authors who described the occurrence of RTM in the context of software effort estimation and used one of the adjustment approaches by Campebell and Kenny [7] to evaluate five data sets [6]. They showed that analogy based effort estimates can be significantly improved through RTM-adjustments. Jørgensen et al. also hypothesized that, in cases with less extreme analogues and more accurate estimation models, there would be an expected improvement, if the underlying assumptions of the RTM-adjustment are met. However, they did not prove this hypothesis in [6].

Shepperd and Cartwright [11] performed an independent replication of the study by Jorgensen et al.: they used two further industrial data sets in which they compared accuracy levels with and without the RTM adjustment. Their results were consistent with those reported in [6], as using the RTM resulted in a small increase in predictive accuracy. However, for one data set it was necessary to first partition it into more homogeneous subsets. Their results added further support for the RTM approach, in that there is a small, but positive, effect upon prediction accuracy.

The RTM adjustment was improved by using the Model Tree adaptation strategy. Model Tree based attribute distance was proposed by Azzeh to adjust estimation by analogy and derive new estimates [17]. This is advantageous because it deals with categorical attributes, minimizes user interaction and improves the efficiency of model learning through classification. The experimental results showed that the proposed approach produced better results when compared to those obtained by using analogy based linear size adaptation, linear similarity adaptation, 'regression towards the mean' and null adaptation. However, this approach may only be applied to complex data sets with large number of categorical attributes.

The interest of finding analogies arises when historical data sets are available, thus making it possible to look for projects that are "similar" to the one for which an estimate is required. Similarity is defined as Euclidean distance in n-dimensional space where n is the number of considered project features. Each dimension is usually standardized, so that all dimensions have equal weight. The known effort values of the nearest neighbors to the new project are then used as the basis for the prediction. Shepperd and Schofield argued that estimation by analogy is a viable technique that can be used by project managers to complement current estimation techniques [18].

Several papers propose improvements of estimation based on closest analogues method. Chiu and Huan [19] investigated the effects on estimates obtained when a genetic algorithm method is adopted to adjust historical effort based on the similarity of distances between pairs of projects. The empirical results obtained using two data sets of 23 and 21 projects each showed that applying a suitable linear model to adjust the analogy-based estimations is a feasible approach to improve the accuracy of software effort estimates. A project selection technique for analogy-based estimations (PSABE), was then added to reduce the whole project base into a small subset that consists only of representative projects. The experimental results showed that applying the genetic algorithm to determine suitable weighted similarity measures of software effort drivers is a feasible approach to improve the accuracy of software effort estimates analogy-based. They also demonstrated that the nonlinearly weighted analogy method has better estimate accuracy than those obtained by using other methods [20].

Li and Ruhe [21] pointed out that a careful selection and weighting of attributes may improve the performance of the estimation methods. They considered the impact of weighting (and selecting) attributes as extensions of their estimation by analogy method AQUA+. With AQUA+ a qualitative analysis pre-step using rough set analysis –a machine learning technique for classification of objects– is performed to weight attribute. They reported that AQUA+ can improve the estimation accuracy, according to the empirical studies performed with six data sets.

Mittas, Athanasiades and Angelis [22] exploited the relationship between the estimation by analogy method and the nearest neighbor non-parametric regression technique in order to suggest a resampling procedure, known as iterated bagging, to reduce the prediction error. The positive effect of iterated bagging on estimation by analogy was validated using both artificial and real datasets from the literature.

Azzeh, Neagu, and Cowling [13] proposed a new formal estimation by analogy model based on the integration of the Fuzzy set theory with the Grey Relational Analysis (GRA) to overcome the inherent uncertainty in software attribute measurement. The Fuzzy set theory provides a representation scheme and mathematical operations to deal with uncertain, imprecise and vague concepts. GRA is a problem solving method that is used to assess the similarity between two tuples with M features, which is mainly used to reduce the uncertainty in the distance measurement between two software projects, for both continuous and categorical features. Both techniques are suitable when the relationship between effort and other effort drivers is complex. Experimental results showed that using integration of GRA with Fuzzy logic produced credible estimates, when compared to the results obtained using Case-Based Reasoning, Multiple Linear Regression and Artificial Neural Networks methods. In another paper [12], the same authors proposed a new approach to deal with each attribute, which has different influence on the project retrieval, based upon the idea of Kendall's coefficient of concordance between the similarity matrix of project attributes and the similarity matrix of known effort values of the dataset. The results showed improved prediction accuracy when multiple project attributes are used with certain weights. Moreover, they integrated analogy-based estimation with Fuzzy numbers in order to improve the performance of software project effort estimation during the early stages of a software development lifecycle, using all the available early data [14]. Particularly, this paper proposes a new software project similarity measurement and a new adaptation technique based on Fuzzy numbers. The results have also shown that the proposed method outperforms some well known estimation techniques, such as case-based reasoning and stepwise regression.

To overcome the inherent uncertainties of the estimation process, Li, Xie, and Goh focused on the generation of interval based estimates with a certain probability [15]. They proposed a novel method named Analogy Based Sampling (ABS) and compared it to the well established Bootstrapped Analogy Based Estimation method (BABE), which is the only existing variant of the analogy based method which has the capability to generate interval predictions. The results and comparisons showed that ABS could improve the performance of BABE with much higher efficiency and more accurate interval predictions. In another paper [23] they proposed a genetic algorithm to simultaneously optimize the K parameter and the feature weights for. Experiment results showed that their methods could significantly improve the prediction accuracy of conventional ABE.

Walkerden and Jeffery [24] stated that Analogy-based estimation is potentially easier to understand and apply than algorithmic methods. They compared several methods of analogy-based software effort estimation with each other and also with a simple linear regression model. The results showed that people are better than tools at selecting analogues. In particular, estimates based on selections made by people, with a linear size adjustment to the analogue's effort value, proved more accurate than estimates based on analogues selected by tools, and also more accurate than estimates based on the simple regression model.

However, controversial results were reported on this subject. Briand, Langley and Wieczorek [16] agree with Stensrud and Myrtveit [25] that estimation by analogy does not outperform regression models. Myrtveit and Stensrud [26] suggested that the results are sensitive to the experimental design, including dataset characteristics, criteria for removing outliers, test metrics, significance levels, the involvement of people, etc. Also, they pointed out that neither their results nor previous ones were robust enough to claim general validity.

Similarly, Mair and Shepperd found that there is approximately equal evidence in favour of and against analogy-based methods [27].

VI. Conclusions and Future Work

In this paper, we have tested a few estimation models, using an experimental dataset. To the best of our knowledge, this is the first time that an effort prediction accuracy comparison is performed for a set of methods including OLS models, LMS models, and analogy-based methods both with and without RTM correction.

By looking at the results given in the tables and boxplot in Section III, it is possible to conclude that all the considered models yield similar performances, as far as estimation accuracy is concerned. Actually, the model based exclusively on analogy is slightly less precise then the others, but RTM correction makes the precision of analogy based estimation very close to that of regression models. It is also possible to consider RTM corrected models preferable over those based on regression because the median is closer to zero (Figure 2).

In conclusion, we can say that our results are of interest for practitioners, especially considering that a small dataset – i.e., a dataset very similar to the datasets that can be collected in most development environment– and popular techniques were used.

In the future, we aim at gaining a deeper theoretical understanding of, and generalizing the results presented here by studying larger projects, possibly involving additional effort-related factors, like product complexity and factors depending on the development environment.

Acknowledgement

The research presented here was partially funded by the project "Metodi, tecniche e strumenti per l'analisi, l'implementazione e la valutazione di sistemi software" funded by Università degli Studi dell'Insubria, and by the Research Fund of Austral University School of Engineering.

References

[1] A. Albrecht, Measuring application development productivity, IBM Application Development Symp. I.B.M. Press, 1979.

[2] ISO. 2003. ISO/IEC 20926: 2003, Software engineering – IFPUG 4.1 Unadjusted functional size measurement method – Counting Practices Manual.

[3] P. J. Rousseeuw and A. M Leroy, Robust regression and outlier detection, John Wiley & Sons, Inc., New York, 1987.

[4] L. Lavazza and S. Morasca, "Using a Generalized Robust Linear Regression Technique to Predict Effort in Software Engineering Projects", 16th Int. Conf. on Evaluation and Assessment in Software Engineering (May 2012).

[5] L. Lavazza and G.Robiolo, “The Role of the Measure of Functional Complexity in Effort Estimation”, 6th Int. Conf. on Predictive Models in Software Engineering, ACM, New York, NY, USA, Article 6, 10 pages, 2010.

[6] M. Jørgensen, D. I. K. Sjøberg and U. Indahl, “Software effort estimation by analogy and regression toward the mean”, Journal of Systems and Software 68, 3 (Dec. 2003), pp. 37-46

[7] D. Campbell and D. Kenny, A primer on regression artifacts. The Guilford Press, 2002.

[8] R. D. Cook and S. Weisberg, Residuals and Influence in Regression. Chapman and Hall, London, 1982.

[9] E. Mendes, Cost Estimation Techniques for Web Projects. IGI Publishing, Hershey, New York, 2008.

[10] S. Conte, H. Dunsmore, and V. Shen, Software Engineering, Metrics and Models. Benjamin/Cummings, 1986.

[11] M. Shepperd, and M. Cartwright, “A replication of the use of regression towards the mean (R2M) as an adjustment to effort estimation models”, 11th IEEE Int. Symp. on Software Metrics (Como, Italy, Sept. 19-22 2005), METRICS 2005, pp. 38-47.

[12] M. Azzeh, D. Neagu, and P. Cowling, “Software effort estimation based on weighted fuzzy grey relational analysis”, 5th Int. Conf. on Predictor Models in Software Engineering (PROMISE '09). ACM, New York, NY, USA, 2009.

[13] M. Azzeh, D. Neagu, and P. Cowling, “Fuzzy grey relational analysis for software effort estimation”, Empirical Software Engineering, 15, 1. Springer, 2010.

[14] M. Azzeh, D. Neagu and P. Cowling,“Analogy-based software effort estimation using Fuzzy numbers”, Journal of Systems and Software, 84, 2, Elsevier, 2011, pp. 270-284.

[15] Y.F. Li, M. Xie, and T.N. Goh, “A Study of Analogy Based Sampling for interval based cost estimation for software project management”, 4th IEEE Int.Con. on Management of Innovation and Technology (Bangkok, 2008). ICMIT 2008, pp. 281 – 286.

[16] L.C. Briand, T. Langley and I. Wieczorek., “A replicated assessment and comparison of common software cost modeling techniques”, in Proceedings of the 22nd international conference on Software engineering (ICSE '00), ACM, New York, NY, USA, pp. 377-386, 2000, doi: 10.1145/337180.337223.

[17] M. Azzeh, “Model Tree Based Adaption Strategy for Software Effort Estimation by Analogy”, 2011 IEEE 11th International Conference on Computer and Information Technology (CIT), Pafos, Sept. 2 2011, pp. 328 – 335.

[18] M. Shepperd, C. Schofield, "Estimating Software Project Effort Using Analogies," IEEE Trans. on Software Eng., vol. 23, no. 11, pp. 736-743, Nov. 1997.

[19] N.H Chiu and S.J. Huang, “The adjusted analogy-based software effort estimation based on similarity distances”, Journal of Systems and Software, 80, 4 (April 2007), pp. 628-640.

[20] Y.F. Li, and M. Xie, and TN. Goh, “A study of project selection and feature weighting for analogy based software cost estimation”, Journal of Systems and Software, 82, 2, Elsevier, pp. 241-252.

[21] J. Li, and G. Ruhe, “Analysis of attribute weighting heuristics for analogy-based software effort estimation method AQUA+”, Empirical Software Engineering, 13, 1, Springer, 2008.

[22] N.Mittas, M. Athanasiades, and L. Angelis. “Improving analogy-based software cost estimation by a resampling method”, Information and Software Technology, 50, 3, Elsevier, 2008.

[23] Y.F Li., M. Xie, and T.N. Goh, “Optimization of feature weights and number of neighbors for Analogy based cost Estimation in software project management”, IEEE Int. Con. on Industrial Engineering and Engineering Management (Singapore, 2008) IEEM 2008, pp. 1542 – 1546.

[24] F. Walkerden and R. Jeffery, “An Empirical Study of Analogy-based Software Effort Estimation”, Empirical Softw. Engg. 4, 2 (June 1999), pp. 135-158, doi:10.1023/A:1009872202035.

[25] E. Stensrud and I. Myrtveit, “Human Performance Estimating with Analogy and Regression Models: An Empirical Validation”, in Proceedings of the 5th International Symposium on Software Metrics (METRICS '98), IEEE Computer Society, Washington, DC, USA, 1998, pp. 205-.

[26] I. Myrtveit, E. Stensrud, "A Controlled Experiment to Assess the Benefits of Estimating with Analogy and Regression Models", IEEE Trans. on Software Eng., vol. 25, no. 4, pp. 510-525, 1999.

[27] C. Mair, M. J. Shepperd, “ The consistency of empirical comparisons of regression and analogy-based software project cost prediction”, in 2005 International Symposium on Empirical Software Engineering (ISESE 2005), 17-18 November 2005, Noosa Heads, Australia, pp. 509-518, IEEE, 2005.

Mapping ASM Specifications to Spec Explorer: Guidelines, Benefits and Challenges

Jameleddine Hassine
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals, Dhahran, Kingdom of Saudi Arabia
jhassine@kfupm.edu.sa

***Abstract*—Model-based testing (MBT) focuses on the generation of test suites, using models of system requirements and behavior. In order to get full benefits from model-based testing and to drive its adoption by practitioners, automation support is required. Spec Explorer 2010 is an MBT tool that offers a rich set of features allowing for modeling and analyzing software behavior using graphical visualization and automatic generation of test code from models. In this paper, we propose and discuss a set of guidelines to map the core Abstract State Machines (ASM) concepts and constructs into Spec Explorer 2010. We illustrate our mapping approach using examples and features from ASM-based formal specification languages *CoreASM* and *AsmL*. Furthermore, we discuss the benefits, the challenges and the limitations of our proposed mapping guidelines. Finally, we illustrate our approach using a case study of the well known dining philosophers problem.**

***Keywords*-Model-based testing; Spec Explorer 2010; Abstract State Machines (ASM); CoreASM; AsmL**

I. Introduction

Model-based testing (MBT) is a variant of testing that relies on explicit behavior models that describe the intended behaviors of a System Under Test (SUT) and/or the behavior of its environment. Test cases are then generated from one of these models or their combination, and then executed on the SUT [1]. MBT offers significant promise in reducing the cost of test suite generation, increasing the effectiveness of the produced test cases, and shortening the testing cycle.

Driven by technological advances and by the ever-growing need for producing high quality software, model-based testing (MBT) has emerged as a potential research domain. Given the popularization of models in software design and development, MBT has moved from a research topic [2], [3] to an emerging practice in the industry [4], [5], with increasing commercial tool support [6]. Dias-Neto et al. [2] have performed a systematic review of MBT research literature and have proposed a classification of the MBT approaches. A recent systematic review of state-based MBT tools by Shafique and Labiche [6] provides a detailed classification of nine commercial and research MBT tools. In a more recent work, Utting et al. [1] have developed a taxonomy of six dimensions that covers the key aspects of MBT approaches.

Model-based testing relies on three key aspects - the modeling notation, the algorithms used to guide test generation and the tools supporting on-the-fly generation or off-line realization of executable tests.

Many different notations and languages, with different expressive power, have been used for modeling the behavior of systems for test generation purposes. Many of these languages are discussed and classified by Hierons et al. [3] and, more recently, by Utting et al. [1]. Examples of such notations classification include *state-based Notations* (e.g., *Z*, *B*, etc.), *transition-based Notations* (e.g., FSMs, statecharts, etc.), *history-based notations* (e.g., temporal logics, *MSC*, etc.), *operational notations* (e.g., process algebra such as *CSP* and *CCS*).

Many test generation algorithms have been proposed [1], [2], [3]. Examples of such test generation approaches include *Random generation* (achieved by sampling the input space of a system), *Search-based algorithms* (e.g., graph search algorithms), *Model Checking* (for checking system properties), *Symbolic execution* (generating symbolic traces that should be instantiated to obtain test cases), *Deductive theorem proving* (used to check the satisfiability of formulas), *Constraint solving* (useful for selecting data values from complex data domains), etc.

Utting et al. [1] have proposed a classification of MBT tools [6] according to which kinds of test selection criteria they support (e.g., *structural model coverage*, *data coverage*, *requirements-based coverage*, *fault-based criteria*, etc.). Microsoft's *Spec Explorer 2010* [7], is one of the commercial tools that have led the MBT scene. The new release of *Spec Explorer 2010* [7], version 3.5 and integrated within Visual Studio 2010, offers a rich and powerful set of features for modeling and analyzing system functional behavior, as well as automatically generating test code.

The widespread interest in model-based testing techniques and tools provides the major motivation of this research. In particular, this paper serves the following purposes:

- Bridge the gap between ASM-based languages (e.g., *CoreASM* [8], *AsmL* [9], etc.) and the new release of *Microsoft Spec Explorer 2010* [7], where the SUT model is specified using .NET $C\#$. The previous version, *Spec Explorer 2006* [10], accepts models written in *AsmL* (Abstract State Machines Language) [9], which is no longer supported in *Spec Explorer 2010* [7]. To bridge this gap, we propose a set of guidelines to map ASM core concepts and constructs into *Spec*

Explorer 2010 modelling notation (i.e., $C\#$ program models and *Cord* scripts [11]).

- Allow for ASM model exploration and analysis. Indeed, *Microsoft Spec Explorer 2010* [7] provides several strategies for managing the exploration of a model, including data coverage of parameter values and structural model criteria [1]. Furthermore, the resulting *Spec Explorer 2010* models can be used to automatically generate test cases.
- Discuss the benefits, the challenges, and the limitations of the *ASM* to *Spec Explorer 2010* mapping approach.

The remainder of this paper is organized as follows. The next section presents some related work. A brief introduction to *Spec Explorer 2010* and an overview of its specification language are presented in Section III. Section IV represents the core of our paper, where we describe and discuss the *ASM* to *Spec Explorer 2010* mapping guidelines. In order to demonstrate the applicability of our proposed mapping approach, a case study of the well known *Dining Philosophers* problem is presented in Section V. A discussion of the challenges and the limitations of our proposed approach is presented in Section VI. Finally, conclusions are drawn in Section VII.

II. Related Work

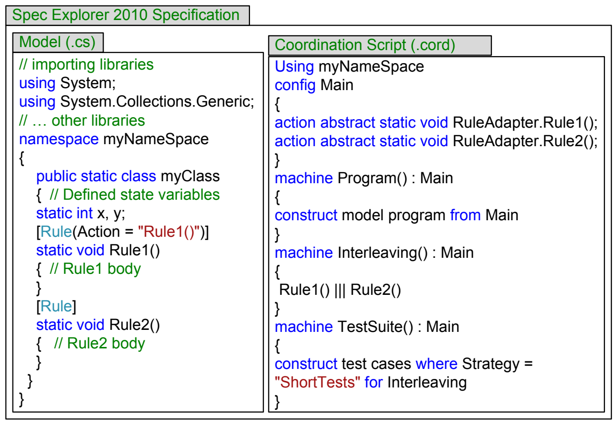
Several approaches have been proposed to translate ASM specifications to languages and notations in order to enable testing and verification. Winter [12] has proposed a schema for transforming ASM models into the language of SMV. A similar approach has been implemented in *AsmetaSMV* by Arcaini et al. [13] for mapping and verifying ASM models, written in *AsmetaL*, into NuSMV notation. Gargantini et al. [14] have introduced an algorithm to automatically encode an ASM specification in *Promela*, the language of the model checker *SPIN*.

Grieskamp et al. [15] have proposed an algorithm that derives a finite state machine (FSM) from a given abstract state machine (ASM) specification. The generated ASM states, often infinite, are grouped into hyperstates which are the nodes of the FSM. The links of the FSM are induced by the ASM state transitions. This transformation allows for the integration of ASM specifications with the existing tools for test case generation from FSMs. In a related work, Barnett et al. [16] have presented a tool environment for model-based testing with the Abstract State Machine Language (*AsmL*) [9]. The environment supports semi-automatic parameter generation, FSM generation from ASMs [15], call sequence generation and conformance testing.

The most closely related work to ours is the one by Veanes et al. [17]. The authors have provided a symbolic analysis of model programs written in AsmL [9], in terms of a background $\mathcal{T}$ of linear arithmetic, sets, tuples and maps.

III. Overview of Spec Explorer 2010

Spec Explorer 2010 [7], which we simply refer to as *Spec Explorer*, provides a model editing, composition, exploration, and visualization environment within Visual Studio, and can generate on-line and off-line test suites. A Spec Explorer model consists of a set of rules, written in $C\#$, expressed in a model program (i.e., the *.cs* file) combined with behavioral descriptions coded in a scripting language called *Cord* [11] (i.e., the *.cord* file). The model program and the *Cord* script work together to make a testable model of the SUT.



```
// importing libraries
using System;
using System.Collections.Generic;
// ... other libraries
namespace myNameSpace
{
   public static class myClass
   {  // Defined state variables
   static int x, y;
   [Rule(Action = "Rule1()")]
   static void Rule1()
   {  // Rule1 body
   }
   [Rule]
   static void Rule2()
   {   // Rule2 body
   }
 }
}
```

```
Using myNameSpace
config Main
{
action abstract static void RuleAdapter.Rule1();
action abstract static void RuleAdapter.Rule2();
}
machine Program() : Main
{
construct model program from Main
}
machine Interleaving() : Main
{
 Rule1() ||| Rule2()
}
machine TestSuite() : Main
{
construct test cases where Strategy =
"ShortTests" for Interleaving
}
```

Figure 1. Structure of a typical *Spec Explorer* specification

Figure 1 illustrates a typical structure of a *Spec Explorer* specification. The model program (.cs) defines the classes of the system (e.g., *myClass*), the fields used to hold state data (e.g., x, y), and the rule methods for the actions (e.g., *Rule1* and *Rule2*) already declared in the *Cord* configuration. The model program is joined by a *Cord* script that defines the context signature (e.g., *Main*) and action machines (e.g., *Program*) expressing behaviors that further constrain the model for purposes of exploring submodels and generating practical test cases. A machine is based on one or more configurations and represents the unit of exploration. Many operators can be used to compose machine behavior from configurations and other behaviors. For example, the parallel behavior operator $|||$ is used in machine *Interleaving* to construct the interleaved parallel composition of *Rule1* and *Rule2*. *Cord* scripts also provide the means to generate test cases from the model. For a detailed description of *Spec Explorer*, we refer the reader to [7].

IV. ASM to Spec Explorer Mapping Guidelines

In this section, we present a brief overview of the basic ASM concepts and we provide a set of guidelines for translating them into *Spec Explorer*. For a detailed description of Abstract State Machines concepts and features, an interested reader is referred to [18].

Table I
EXAMPLES OF MAPPINGS OF ASM BUILT-IN TYPES

CoreASM [8]	**AsmL** [9]	**Spec Explorer** [7]
Boolean	Boolean	bool
enum/universe	enum	enum
String	String	string
Number	Integer/Byte/short	int/byte/short
	Long/Float/Double	long/float/double
Set	Set of T	SetContainer<T>
List	Seq of T	SequenceContainer<T>
Map	Map of T to S	MapContainer<T,S>

Although we illustrate the mapping guidelines using features and examples from *CoreASM* [8] and *AsmL* [9] languages, our proposed mapping guidelines can be applied to any ASM-based language, thus maintaining the generality of the discussion.

A. Mapping of States

Abstract State Machines (ASM) [18] define a state-based computational model, where computations (runs) are finite or infinite sequences of states $\{S_i\}$ obtained from a given initial state S_0 by repeatedly executing transitions δ_i:

$$S_0 \xrightarrow{\delta_1} S_1 \xrightarrow{\delta_2} S_2 \qquad \cdots \qquad \xrightarrow{\delta_n} S_n$$

The states of an ASM are multi-sorted first-order structures, i.e., domains of objects with functions and predicates defined on them.

1) Domains:: A domain consists of a set of declarations that establish the ASM vocabulary. Each declaration establishes the meaning of an identifier within its scope. *Spec Explorer* is based on *C#* language and it offers a rich set of data types covering almost all domains of ASM-based languages. Hence, the mapping of domains into their corresponding *Spec Explorer* data types is straightforward. Table I shows some examples of supported ASM-based built-in types and their corresponding *Spec Explorer* data types.

In order to match the mutable nature of ASM sets, sequences, and maps, we use the *Spec Explorer* mutable containers types: *SetContainer*, *SequenceContainer* and *MapContainer*.

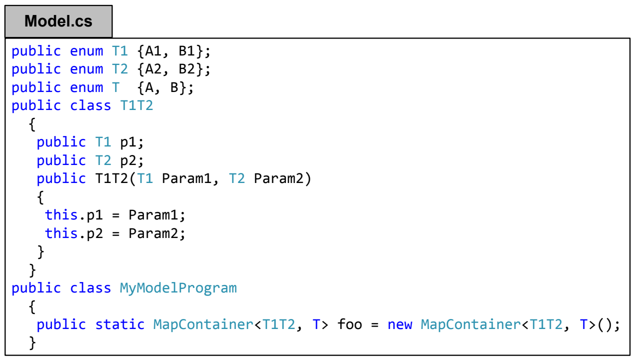
2) Function Names:: Each function name *f* has an arity (number of arguments that the function takes). Function names can be static (i.e., fixed interpretation in each computation state) or dynamic (i.e., can be altered by transitions fired in a computation step). Static nullary (i.e., 0-ary) function names (i.e., called constants) and dynamic nullary functions (i.e., called variables) are mapped respectively into *Spec Explorer* constants and variables of the types presented in Table I.

ASM *n-ary* functions (i.e., f: T_1 x T_2 x …... $T_n \rightarrow$ T), may be described using the *Spec Explorer MapContainer* construct. The same mapping applies to the ASM monitored, controlled and shared functions. For example, the following *CoreASM* code defines three enumeration domains *T1*, *T1* and *T*, each having 2 elements, and a 2-ary function *foo* defined over *T1* and *T2* and returning a value of type *T*:

```
enum T1 = {A1,B1}
enum T2 = {A2,B2}
enum T  = {A,B}
foo: T1 * T2 -> T
```

Figure 2 illustrates its corresponding *Spec Explorer* model implementation. A new class *T1T2* is created to map function *foo* input types *T1* and *T2*. The arity of the function defines the number of the new class attributes. The function name *foo* is mapped to a *MapContainer* object having keys of type *T1T2* and values of type *T*. It is worth noting that the new class *T1T2* could have been created as CompoundValue (i.e., public class T1T2 : CompoundValue), which is mutable but it cannot be updated outside of the constructor.

Another possible solution, proposed by Arcaini et al. [13], is to create $\prod_{i=1}^{n} |D_i|$ variables to cover all combinations of the domains elements. Considering the example above, four variables of type *T* can be created to cover the product of enumerations *T1* and *T2*: *public static T foo_A1_A2, foo_A1_B2, foo_B1_A2, foo_B1_B2;* This solution does not scale well in presence of domains with large sets of elements.


Model.cs

```
public enum T1 {A1, B1};
public enum T2 {A2, B2};
public enum T  {A, B};
public class T1T2
  {
   public T1 p1;
   public T2 p2;
   public T1T2(T1 Param1, T2 Param2)
   {
    this.p1 = Param1;
    this.p2 = Param2;
   }
  }
public class MyModelProgram
  {
   public static MapContainer<T1T2, T> foo = new MapContainer<T1T2, T>();
  }
```

Figure 2. Implementing n-ary functions in Spec Explorer

B. Mapping of Transition Rules

The transition relation is specified by guarded function updates, called *rules*, describing the modification of the functions from one state to the next. An ASM state transition is performed by firing a set of rules in one step.

1) Conditional Rule:: Typically, an ASM transition system appears as a set of guarded rules:

if *cond* **then** *RuleThen* **else** *RuleElse* **endif**

RuleThen and *RuleElse* are transition rules and *cond*, the guard, is a term representing a boolean condition. The ASM conditional rule can be implemented using *Spec Explorer*

Condition class, which provides conditions that can control the exploration of a rule. *Spec Explorer* does not generate a transition for a rule if it encounters in the execution path of the rule any condition that is not satisfied. The *Condition* class contains a rich set of methods for controlling exploration (e.g., IsTrue(Boolean), IsFalse(Boolean), IsNull(Object), etc.

Figure 3 illustrates the mapping of the ASM conditional rule. Condition *Condition.IsTrue(cond)* enables the exploration of *RuleThen()* if *cond* evaluates to true, while condition *Condition.IsFalse(cond)* enables the exploration of *RuleElse()* if *cond* evaluates to false.

```
static class ModelProgram
{
  static bool cond=true;
  static int stateVar;
  [Rule(Action = "RuleThen()")]
  static void RuleThen()
  {
   Condition.IsTrue(cond);
   // update state Variables
  }
```

```
[Rule(Action = "RuleElse()")]
  static void RuleElse()
  {
  Condition.IsFalse(cond);
  // update state Variables
  }
}
```

Figure 3. Implementation of the conditional rule in Spec Explorer

2) Update Rule:: The basic form of a transition rule is a function update:

$$f(t_1,t_2,...,t_n) := \text{value}$$

(f, $(t_1,t_2,...,t_n)$) represents the location and *value* is its content. An update can be implemented in *Spec Explorer* as a rule containing simple assignment statements in the case of dynamic nullary functions (e.g., var = value) or as an update of a *MapContainer* in case of n-ary functions, where an update results in adding a new element to the *MapContainer* (if the key does not exist), or as an update of a *MapContainer* element (if the key does exist). Figure 4 illustrates the mapping of the ASM update rule to *Spec Explorer* using the data structures introduced in Figure 2. The *MapContainer* class method *ContainsKey(D)* returns whether the key exists in the *MapContainer* (e.g., foo.ContainsKey(key)), while the method *Add(D, R)* adds a key-value pair to the *MapContainer* (e.g., foo.Add(key, value)).

Model.cs

```
public static MapContainer<T1T2, T> foo = new MapContainer<T1T2, T>();
[Rule]
public static void UpdateRule(T1T2 key, T value)
 {
  if (foo.ContainsKey(key))
     foo[key] = value;
  else
     foo.Add(key, value);
 }
```

Figure 4. Spec Explorer implementation of the update rule

3) Sequence Rule:: The sequence rule aims at executing rules/function updates in sequence:

$$\textbf{seq}\ Rule1,\ \ldots,\ RuleN\ \text{or}\ \textbf{seq}\ Update_1,\ \ldots,\ Update_n$$

The resulting update set is the sequential composition of the updates, generated by $Rule1 \ldots RuleN$ in case of rules and generated by $Update_1, \ldots, Update_n$ in case of function updates. ASM sequential function updates can be mapped into one single rule in *Spec Explorer* as described in Figure 5(a), while ASM sequential rules can be implemented using the *Cord* composition operator ";" as described in Figure 5(b).

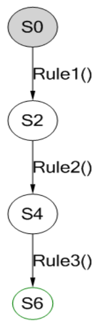
Model.cs

```
[Rule]
static void SequentialUpdates()
   { // Update1
     var1 = v1;
     // UpdateN
     varN = vN;
   }
```

(a) Model program for sequential function updates

Config.cord

```
config Main
{
action abstract static void RuleAdapter.Rule1();
action abstract static void RuleAdapter.Rule2();
action abstract static void RuleAdapter.Rule3();
}
machine Program() : Main
{
construct model program from Main
}
machine Sequence() : Main
{
Rule1() ; Rule2() ; Rule3()
}
```



(b) Cord file and generated FSM for the sequential rules

Figure 5. Spec Explorer implementation of the sequence rule

4) Choose Rule:: The choose rule consists in selecting elements (non deterministically), from specified domains, satisfying the guard φ, and then evaluates *Rule1*. If no such elements exist (i.e., ifnone), *Rule2* is evaluated.

$$\textbf{choose}\ x\ \textbf{in}\ D\ \textbf{with}\ \varphi(\text{x})\ \textbf{do}\ Rule1\ \textbf{ifnone}\ Rule2$$

Model.cs

```
[Rule(Action = "ChoiceRule()")]
static void ChoiceRule()
 {
  Predicate<int> P = (y => (y >= 1 && y <= 4));
  if (x == Choice.Some<int>(P))
    Rule1(x);
  else
    Rule2();
 }
```

Figure 6. Spec Explorer implementation of the choose rule

The *choose* rule may be implemented in *Spec Explorer* using the non-deterministic method *Some* of the *Choice* class (i.e., Choice.Some<T>(Predicate<T> predicate)), which selects a value from a range of a specific type *T*, that satisfies a certain condition: *predicate*. If such an element exists, it will be passed to Rule1, otherwise a call to Rule2 is performed. Figure 6 illustrates an example of a mapping of the ASM choose rule to *Spec Explorer*. The predicate P corresponds to the guard φ, while the selected value, if any, is stored in variable *x*. An integer value is selected randomly from the range 1 . . . 4, then passed to Rule1.

5) Extend Rule:: The extend rule is used to construct new elements and add them to a specific domain. The resulting update set is the updates generated by *Rule1*.

$$\textbf{extend } D \textbf{ with } x_1, \ldots, x_n \textbf{ do } \mathit{Rule1}$$

In *Spec Explorer*, the extend rule may be implemented by adding new elements to a mutable container such as SetContainer, SequenceContainer, MapContainer, etc., then call *Rule1*. As shown in Figure 7, a new integer *x1* is added (using the *Add* method) to the existing SetContainer *D*, then *Rule1* is called on the new added element.

Model.cs

```
static SetContainer<int> D= new SetContainer<int>(1,2);
[Rule]
static void ExtendRule(int x1)
  {
   D.Add(x1);
   Rule1(x1);
  }
```

Figure 7. Spec Explorer implementation of the extend rule

It is worth noting that the *CoreASM* [8] extend rule has different semantics depending on whether *D* is a background (e.g., Collection, List, Set, etc.) or a *universe*. Such distinction is not made at the *Spec Explorer* level.

6) Block Rule:: If a set of ASM transition rules have to be executed simultaneously, a block rule is used:

$$\textbf{par } Rule_1 \ \ldots \ Rule_n \textbf{ endpar}$$

CoreASM uses the **par**. . . **endpar** syntax, while *AsmL* uses **step** $Rule_1$ . . . $Rule_n$ syntax to describe parallel updates. The update generated by this rule is the union of all the updates generated by $Rule_1$ . . . $Rule_n$. A set of ASM updates is called inconsistent, if it contains updates with the same locations, i.e., two elements (loc,v) and (loc,v') with v≠v'. In the case of inconsistency, the computation does not yield a next state. For example, Figures 8(a) and 8(b) show respectively *CoreASM* code for (1) a rule (*rule1*) having a conflicting parallel function update of variable x, and (2) one block rule (*InitRule*) firing conflicting parallel updates from *rule1* and *rule2*. Both examples do not yield a next state.

Since *Spec Explorer* is based on .NET, a rule method code is not different from an ordinary sequential $C\#$ code. Hence, statements within a rule are executed in sequence,

```
// Two conflicting
// parallel updates
rule rule1 =
par
x := 1
x := 5
endpar
```

(a) Inconsistent set of function updates

```
rule rule1 = x := 1
rule rule2 = x :=5
// Conflicting call to rule1 and rule2
rule InitRule =
par
rule1
rule2
endpar
```

(b) Inconsistent set of rule updates

Figure 8. Inconsistent ASM updates

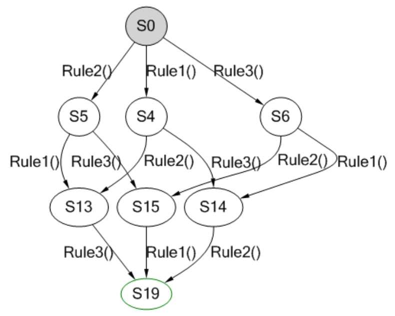
not concurrently. At every step, *Spec Explorer* enables all the rules satisfying their respective preconditions, leading to a non-deterministic interleaving of rule executions. Therefore the parallel update semantics, previously supported in *Spec Explorer 2006* [10]), is abandoned because it conflicts with the default sequential semantics of $C\#$.

If we consider rules updating different locations (i.e., consistent updates), *Spec Explorer* parallel execution of the selected rules can be reduced to a non-deterministic interleaving (with extra created transitions and states). It can be achieved using the *Cord* composition operator ||| as shown in Figure 9.

Config.cord

```
config Main
{
action abstract static void RuleAdapter.Rule1();
action abstract static void RuleAdapter.Rule2();
action abstract static void RuleAdapter.Rule3();
}
machine Interleaving(): Main
{
Rule1() ||| Rule2() ||| Rule3();
}
```

(a) SpecExplorer implementation of the block Rule



(b) Block rule associated FSM

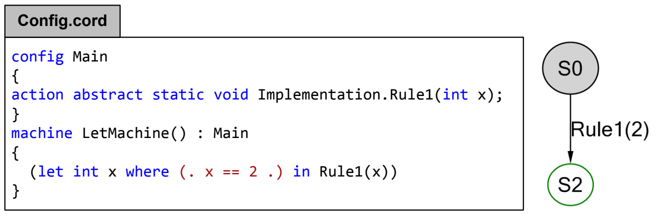
Figure 9. Spec Explorer implementation of the block rule and its corresponding FSM

A possible work-around for detecting inconsistent updates is to design a wrapper method to check and analyze whether there are any inconsistencies between the different update statements.

7) Let Rule:: The *let* rule assigns a value of a term *t* to the variable *x* and then executes the rule *Rule*. The syntax of a Let rule is:

let (x = t) **in** $Rule$ **endlet**

The *let* rule may be implemented in *Spec Explorer* using the *LetBehavior* construct. This construct introduces a set of local variables with an optional associated constraint. The scope of the declared variables is the given behavior. Figure 10 shows an example of a *Cord* configuration relative to a machine that produces a transition of *Rule1* on *x* equal to 2.



Config.cord

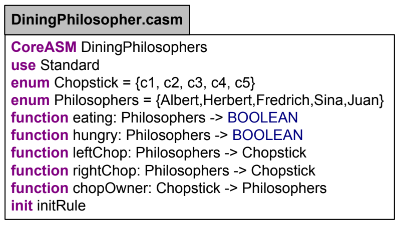
```
config Main
{
action abstract static void Implementation.Rule1(int x);
}
machine LetMachine() : Main
{
  (let int x where (. x == 2 .) in Rule1(x))
}
```

Figure 10. Spec Explorer implementation of the let rule

Other ASM rules such as *forall* rule, *iterate* rule, etc., are not covered in this work due to the lack of space.

V. Case Study: The Dining Philosophers Problem

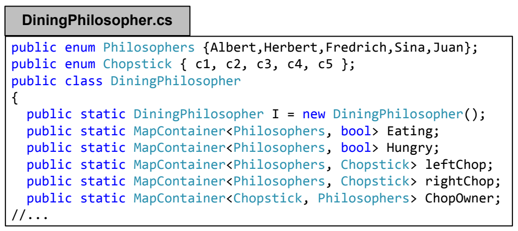
The dining philosophers problem is a classic problem in concurrent programming invented by E. W. Dijkstra. Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table there is a bowl of rice, and the table is laid with five single chopsticks. In what follows, we map the *CoreASM* specification of the Dining Philosophers [19] into Spec Explorer.



DiningPhilosopher.casm

```
CoreASM DiningPhilosophers
use Standard
enum Chopstick = {c1, c2, c3, c4, c5}
enum Philosophers = {Albert,Herbert,Fredrich,Sina,Juan}
function eating: Philosophers -> BOOLEAN
function hungry: Philosophers -> BOOLEAN
function leftChop: Philosophers -> Chopstick
function rightChop: Philosophers -> Chopstick
function chopOwner: Chopstick -> Philosophers
init initRule
```

(a) Dining Philosophers CoreASM Declaration



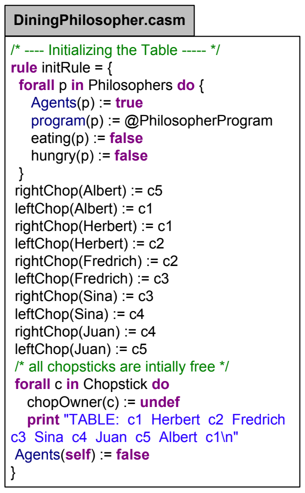
DiningPhilosopher.cs

```
public enum Philosophers {Albert,Herbert,Fredrich,Sina,Juan};
public enum Chopstick { c1, c2, c3, c4, c5 };
public class DiningPhilosopher
{
  public static DiningPhilosopher I = new DiningPhilosopher();
  public static MapContainer<Philosophers, bool> Eating;
  public static MapContainer<Philosophers, bool> Hungry;
  public static MapContainer<Philosophers, Chopstick> leftChop;
  public static MapContainer<Philosophers, Chopstick> rightChop;
  public static MapContainer<Chopstick, Philosophers> ChopOwner;
//...
```

(b) Dining Philosophers Spec Explorer Declaration

Figure 11. Dining philosophers CoreASM declarations and their mappings in Spec Explorer

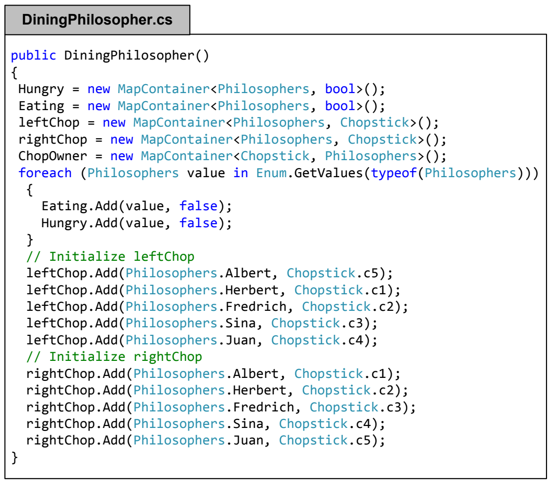
Figure 11 illustrates the *CoreASM* declarations and their mappings in *Spec Explorer*. Enumerations are mapped to Spec Explorer enumerations and functions are mapped to MapContainers. The creation of the set of philosophers (through agents in *CoreASM*) is done implicitly through the class constructor in SpecExplorer. In addition, there is no need to initialize *ChopOwner* as *undef* in *Spec Explorer* since its *MapContainer* is initially empty. Figure 12 illustrates the *CoreASM* init rule and its mapping as a constructor of the class DiningPhilosopher.



DiningPhilosopher.casm

```
/* ---- Initializing the Table ----- */
rule initRule = {
 forall p in Philosophers do {
   Agents(p) := true
   program(p) := @PhilosopherProgram
   eating(p) := false
   hungry(p) := false
 }
 rightChop(Albert) := c5
 leftChop(Albert) := c1
 rightChop(Herbert) := c1
 leftChop(Herbert) := c2
 rightChop(Fredrich) := c2
 leftChop(Fredrich) := c3
 rightChop(Sina) := c3
 leftChop(Sina) := c4
 rightChop(Juan) := c4
 leftChop(Juan) := c5
 /* all chopsticks are intially free */
 forall c in Chopstick do
   chopOwner(c) := undef
   print "TABLE:  c1  Herbert  c2  Fredrich
c3  Sina  c4  Juan  c5  Albert  c1\n"
 Agents(self) := false
}
```

(a) CoreASM init rule



DiningPhilosopher.cs

```
public DiningPhilosopher()
{
 Hungry = new MapContainer<Philosophers, bool>();
 Eating = new MapContainer<Philosophers, bool>();
 leftChop = new MapContainer<Philosophers, Chopstick>();
 rightChop = new MapContainer<Philosophers, Chopstick>();
 ChopOwner = new MapContainer<Chopstick, Philosophers>();
 foreach (Philosophers value in Enum.GetValues(typeof(Philosophers)))
  {
    Eating.Add(value, false);
    Hungry.Add(value, false);
  }
  // Initialize leftChop
  leftChop.Add(Philosophers.Albert, Chopstick.c5);
  leftChop.Add(Philosophers.Herbert, Chopstick.c1);
  leftChop.Add(Philosophers.Fredrich, Chopstick.c2);
  leftChop.Add(Philosophers.Sina, Chopstick.c3);
  leftChop.Add(Philosophers.Juan, Chopstick.c4);
  // Initialize rightChop
  rightChop.Add(Philosophers.Albert, Chopstick.c1);
  rightChop.Add(Philosophers.Herbert, Chopstick.c2);
  rightChop.Add(Philosophers.Fredrich, Chopstick.c3);
  rightChop.Add(Philosophers.Sina, Chopstick.c4);
  rightChop.Add(Philosophers.Juan, Chopstick.c5);
}
```

(b) Spec Explorer class constructor

Figure 12. CoreASM init rule and its mapping in Spec Explorer

Figure 13 describes the CoreASM derived functions and their mappings in *Spec Explorer*.

Figure 14 shows the three rules of the system and their mappings in Spec Explorer. *CoreASM* rules *StartEating* and *StopEating* are called only from the *PhilosopherProgram* rule. In *Spec Explorer*, these two rules are enabled at every transition (see Figure 14(b)). To mimic the *CoreASM*

DiningPhilosopher.casm

```
/* ---- Derived Functions ----- */
derived canPickBothChopsticks =
  (chopOwner(leftChop(self)) = undef)
  and (chopOwner(rightChop(self)) = undef)
// flip of a coin
derived flip = pick b in BOOLEAN
```

(a) CoreASM Derived Functions

DiningPhilosopher.cs

```
public static bool canPickBothChopsticks(Philosophers P)
{
 if (!ChopOwner.ContainsKey(leftChop[P]) &&
!ChopOwner.ContainsKey(rightChop[P]))
   return true;
 else
   return false;
}
public static bool flip()
{
 return Choice.Some<bool>();
}
```

(b) Spec Explorer Functions

Figure 13. CoreASM derived functions and their mappings in Spec Explorer

behavior, two guard conditions *StartEatingEnabled* and *StopEatingEnabled* are added to protect the execution of rules *StartEating* and *StopEating* respectively (see Figure 14(c)).

VI. ASM to Spec Explorer Mapping Challenges and Limitations

At first glance, the mapping seemed to be simple because *Spec Explorer* supports the concept of rules with actions. Later, it appeared that it was a challenging task from many perspectives:

- The most important challenge was dealing with parallel updates/rules generating a single next step when there are no inconsistencies. Indeed, *Spec Explorer 2010* is based on .NET, so a rule method is not different from an ordinary *C#* sequential code. Spec Explorer 2010 [7] differs from the original *Spec Explorer 2006* [10], where ASMs were supported directly through *AsmL*. As a core difference, ASM parallel updates semantics are not directly supported in *Spec Explorer 2010* but have to be encoded, if desired.
- A related challenge to encoding parallel updates, is the detection of inconsistencies in updates. One possible work-around for this limitation is to design a wrapper method to check and analyze whether there are any inconsistencies between the different update statements.
- An ASM-based language, such as *CoreASM*, may define a program rule from which all other rules are called (see Figure 14(a)). *Spec Explorer 2010* will fire all rules whenever their preconditions are satisfied. To implement such behavior, the called rules can be guarded with entry and exit boolean conditions (as described in Figure 14(c) using variables *StartEatingEnabled* and *StopEatingEnabled*).
- The proposed mapping guidelines represent a good and direct translation of ASM constructs into *Spec Explorer*. However, different mapping may also be valid. One example is to map an ASM initialization rule as a regular rule, as an event or as a class constructor.

VII. Conclusion and Future Work

The general goal of this work is to apply advanced model-based testing techniques to Abstract State Machines. More precisely, this paper aimed to bridge the gap between ASM-based languages and the new version of *Spec Explorer*. It discussed the core aspects of ASMs and how one can map them to *Spec Explorer 2010*, where modeling is done through a special extension of *C#* and coordination language *Cord*. The major limitation of the proposed mappings is the implementation of ASM parallel updates semantics, which are not directly supported in *Spec Explorer 2010*. We believe, this was mainly due to the conceived ease of adoption by testers, and therefore the parallel update semantics was abandoned because it conflicts with the default sequential semantics of *C#*.

As a future work, we are planning to extend this work to cover more ASM constructs and to investigate the automation of the proposed guidelines in a target ASM language such as *CoreASM*.

References

[1] M. Utting, A. Pretschner, and B. Legeard, “A taxonomy of model-based testing approaches,” *Software Testing, Verification and Reliability*, 2011, published online. Paper version to appear.

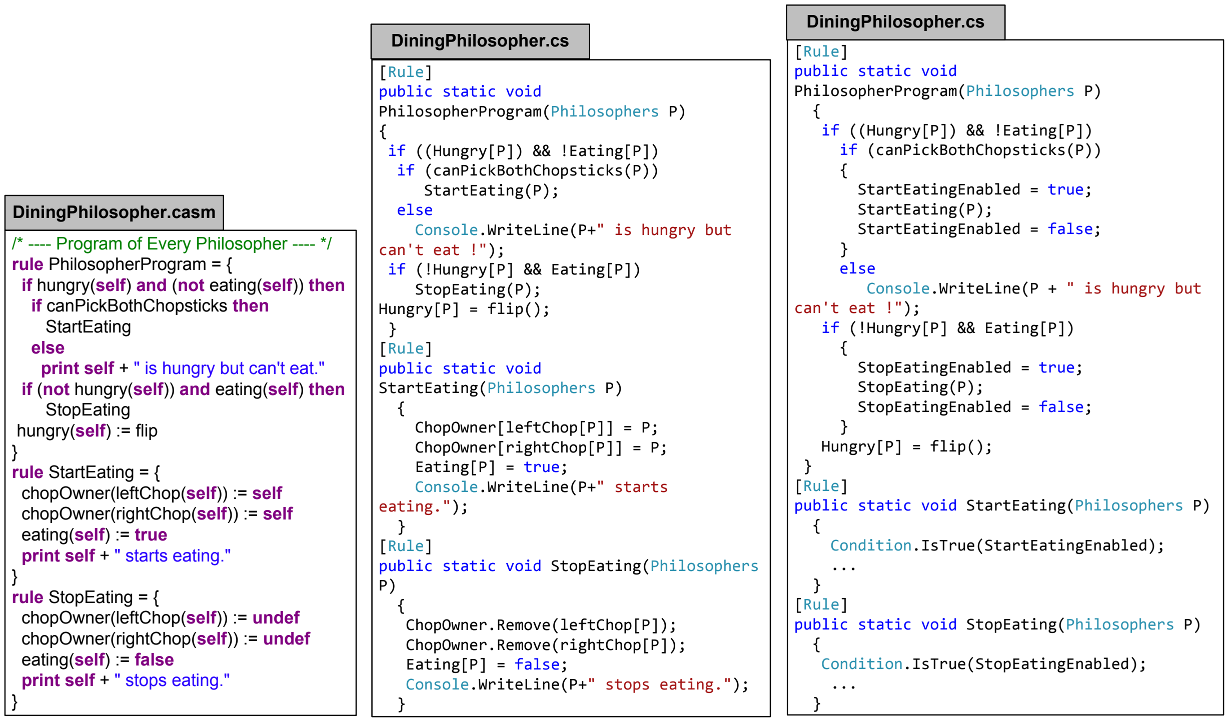
[2] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, “A survey on model-based testing approaches: a systematic review,” in *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies*, ser. WEASELTech ’07. New York, NY, USA: ACM, 2007, pp. 31–36.

[3] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan, “Using formal specifications to support testing,” *ACM Comput. Surv.*, vol. 41, pp. 9:1–9:76, February 2009.

[4] M. Sarma, P. V. R. Murthy, S. Jell, and A. Ulrich, “Model-based testing in industry: a case study with two mbt tools,” in *Proceedings of the 5th Workshop on Automation of Software Test*, ser. AST ’10. New York, NY, USA: ACM, 2010, pp. 87–90.

[5] K. Stobie, “Model based testing in practice at Microsoft,” *Electron. Notes Theor. Comput. Sci.*, vol. 111, pp. 5–12, January 2005.

[6] M. Shafique and Y. Labiche, “A systematic review of model based testing tool support,” Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, Tech. Rep. SCE-10-04, May 2010.



```
DiningPhilosopher.casm
/* ---- Program of Every Philosopher ---- */
rule PhilosopherProgram = {
  if hungry(self) and (not eating(self)) then
    if canPickBothChopsticks then
       StartEating
    else
      print self + " is hungry but can't eat."
  if (not hungry(self)) and eating(self) then
       StopEating
 hungry(self) := flip
}
rule StartEating = {
  chopOwner(leftChop(self)) := self
  chopOwner(rightChop(self)) := self
  eating(self) := true
  print self + " starts eating."
}
rule StopEating = {
  chopOwner(leftChop(self)) := undef
  chopOwner(rightChop(self)) := undef
  eating(self) := false
  print self + " stops eating."
}
```

(a) CoreASM Rules

```
DiningPhilosopher.cs
[Rule]
public static void
PhilosopherProgram(Philosophers P)
{
 if ((Hungry[P]) && !Eating[P])
  if (canPickBothChopsticks(P))
     StartEating(P);
  else
    Console.WriteLine(P+" is hungry but
can't eat !");
 if (!Hungry[P] && Eating[P])
    StopEating(P);
Hungry[P] = flip();
 }
[Rule]
public static void
StartEating(Philosophers P)
  {
    ChopOwner[leftChop[P]] = P;
    ChopOwner[rightChop[P]] = P;
    Eating[P] = true;
    Console.WriteLine(P+" starts
eating.");
  }
[Rule]
public static void StopEating(Philosophers
P)
  {
   ChopOwner.Remove(leftChop[P]);
   ChopOwner.Remove(rightChop[P]);
   Eating[P] = false;
   Console.WriteLine(P+" stops eating.");
  }
```

(b) Spec Explorer Rules

```
DiningPhilosopher.cs
[Rule]
public static void
PhilosopherProgram(Philosophers P)
  {
   if ((Hungry[P]) && !Eating[P])
     if (canPickBothChopsticks(P))
     {
       StartEatingEnabled = true;
       StartEating(P);
       StartEatingEnabled = false;
     }
     else
        Console.WriteLine(P + " is hungry but
can't eat !");
   if (!Hungry[P] && Eating[P])
     {
       StopEatingEnabled = true;
       StopEating(P);
       StopEatingEnabled = false;
     }
   Hungry[P] = flip();
 }
[Rule]
public static void StartEating(Philosophers P)
  {
    Condition.IsTrue(StartEatingEnabled);
    ...
  }
[Rule]
public static void StopEating(Philosophers P)
  {
   Condition.IsTrue(StopEatingEnabled);
    ...
  }
```

(c) Spec Explorer Rules Adjusted

Figure 14. CoreASM rules and their mappings in Spec Explorer

[7] Microsoft, "Spec Explorer 2010 Visual Studio Power Tool, Version 3.5," http://visualstudiogallery.msdn.microsoft.com/271d0904-f178-4ce9-956b-d9bfa4902745, 2010.

[8] R. Farahbod, V. Gervasi, and U. Glässer, "CoreASM: An Extensible ASM Execution Engine," *Fundamenta Informaticae*, vol. 77, pp. 71–103, January 2007.

[9] AsmL, "Microsoft Research: The Abstract State Machine Language," http://research.microsoft.com/en-us/projects/asml/, 2006.

[10] Microsoft, "Microsoft Research: Spec Explorer 2006 tool," http://research.microsoft.com/en-us/downloads/b33add8c-6172-444d-b1b1-6a91323ad7cc/default.aspx, 2006.

[11] W. Grieskamp and N. Kicillof, "A schema language for coordinating construction and composition of partial behavior descriptions," in *Proceedings of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools*, ser. SCESM '06. New York, NY, USA: ACM, 2006, pp. 59–66.

[12] K. Winter, "Model checking for abstract state machines," *Journal of Universal Computer Science*, vol. 3, no. 5, pp. 689–701, 1997.

[13] P. Arcaini, A. Gargantini, and E. Riccobene, "AsmetaSMV: A Way to Link High-Level ASM Models to Low-Level NuSMV Specifications," in *Abstract State Machines, Alloy, B and Z, Second International Conference, ABZ 2010, Orford, QC, Canada*, 2010, pp. 61–74.

[14] A. Gargantini, E. Riccobene, and S. Rinzivillo, "Using Spin to generate tests from ASM specifications," in *Proceedings of the abstract state machines 10th international conference on Advances in theory and practice*, ser. ASM'03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 263–277.

[15] W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes, "Generating finite state machines from abstract state machines," in *Proceedings of the International Symposium on Software Testing and Analysis, July 22-24, Roma, Italy. ISSTA'02*, 2002, pp. 112–122.

[16] M. Barnett, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes, "Towards a tool environment for model-based testing with AsmL," in *Formal Approaches to Software Testing, Third International Workshop on Formal Approaches to Testing of Software, FATES 2003, Montreal, Quebec, Canada*, 2003, pp. 252–266.

[17] M. Veanes, N. Bjørner, Y. Gurevich, and W. Schulte, "Symbolic bounded model checking of abstract state machines," *Int. J. Software and Informatics*, vol. 3, no. 2-3, pp. 149–170, 2009.

[18] E. Börger and R. F. Stark, *Abstract State Machines: A Method for High-Level System Design and Analysis*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[19] G. Ma and R. Farahbod, "Dining Philosphers: A Sample Specification in CoreASM," 2006. [Online]. Available: http://coreasm.svn.sourceforge.net/viewvc/coreasm/engine-carma/trunk/sampleSpecs/DiningPhilosophers.coreasm

Object Segmentation by Edges Features of Graph Cuts

Weiwei Du, Yuki Masumoto, Nobuyuki Nakamori
Dept. Information Science
Hashigamicho, Matugasaki, Kyoto, 606-8585 Japan
e-mail: {duweiwei, m2622037, nakamori}@kit.ac.jp

***Abstract*—This paper proposes a simple graph cuts algorithm based edges features to object segmentation problems. Graph cuts are used to find the global optimum of a cost function based on boundary and region of an image. Gaussian Mixture Models (GMMs) are built based on the seeds which are given by user to the object and background in an image. The contribution of this paper is to add edges features to GMMs. The proposal can segment an object region having noisy edges and colors similarity between the object and background. Experimental results illustrate the validity of the proposal.**

Keywords-object segmentations; edges features; graph cuts; Gaussian Mixture Models

I. INTRODUCTION

Object Segmentation in static images is one of the most fundamental tasks in image content analysis, object recognition, image matting and so on. Many algorithms of object segmentation have been proposed. For example, an object can be segmented by obtaining a α which is expressed at $I= F\alpha +B (1- \alpha)$. F, B and α are foreground colors, background colors, opacity respectively. However, F, B and α are unknown, object segmentation is the highly ill-posed problem. In order to solve the above problem, some object segmentation algorithms have proposed such as the trimap algorithm [1][2], membership propagation algorithm [3]. However, they are time-consuming.

Alternatively, from minimize certain energy functions view, graph cuts algorithms are widely explored such as early organ segmentation [4], interactive graph cuts [5], and recently efficient N-D image segmentation [6], image segmentation using multi-scale smoothing [8] and graph cuts segmentation using local texture features [9]. The goal of these algorithms is to obtain a minimize certain energy function which is defined in terms of boundary and region in an image. Their basic idea of the algorithms is Min-Cut/Max-Flow in graph theory. The minimize energy can be obtained based on adding or removing any constraints by user.

Early graph cuts algorithms developed with an image are difficult to segment an image with complex noisy edges, because these noisy edges interfere with the values among neighboring pixels. Thus, [8] proposed a coarse-to-fine approach to detect the true boundaries using graph cuts. However, [8] cannot segment the kind of images which have some analogue colors between the object and background. Graph cuts segmentation using local texture features of multiresolution [9] can solve the above problem, to some extent, but it is not easy to get the texture features from an image.

This paper proposes a simple graph cuts algorithm based on edges features to object segmentation problems. Graph cuts are used to find the global optimum of a cost function based on boundary and region of an image. Gaussian Mixture Models (GMMs) are built based on the seeds which are given by user to the background and foreground in an image. The contribution of this paper is to add edges features to GMMs. The proposal can segment an object region having noisy edges and colors similarity between the background and foreground, and improve precision. Experimental results illustrate the validity of the proposal.

The paper is organized as following: Section two introduces theory of graph cuts in an image. Section three interprets the proposal of this paper. Section four expounds the procedures of the approach. Section five draws a conclusion and gives a future work.

II. GRAPH CUTS FOR AN IMAGE SEGMENTATION

Boykov et al. [4][5][6] apply the theory of graph cuts algorithm to image segmentation. An undirected graph $G=(V,E)$ is defined as a set of nodes V and a set of undirected edges E that connect these nodes. A simple graph corresponding to an image is shown in Figure 1. In the graph, there are two terminals which are called as "a source" and "a sink". In the image, the source is considered as the object and a sink is considered as the background. The image can be segmented when every pixel corresponds to the source or the sink. However, all pixels of an image cannot be labeled, we must judge the unlabeled pixels which belong to the object or the background based on labeled pixels. We call the edges of the labels corresponding to a source or a sink *t-link*. The edges of neighboring pixels are called *n-link*. We obtain the segmentation boundary between the object and the background by computing *n-link* and *t-link*. Obtaining the segmentation boundary means finding the minimum cost cut on the graph. It is note that locations with high intensity gradient correspond to cheap *n-link*. Thus, they are attractive choices for optimal segmentation boundary. The minimum cut can be computed exactly in polynomial time using well known algorithms such as max-flow [10] or push-relabeled [11].

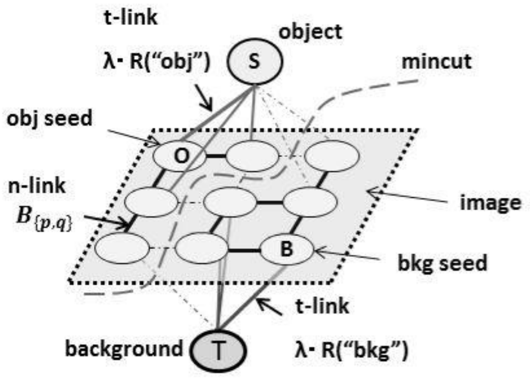


Figure 1. Graph representing a 3×3 image

Actually, image segmentation is considered as a binary labeling problem. The nodes are pixels *p* on the image *P* and the edges have adjacency relationships with four or eight connections between neighboring pixels $q \in N$. *N* is a set of neighboring pixels. The labeling problem is assigned a unique label *A* to each node. $A = (A_1, A_2, ..., A_p, ...A_{|p|})$ can be obtained by minimizing the energy *E(A)* in Eq.(1). A is a binary vector i.e. $A_p \in \{"obj", "bkg"\}$. "*obj*" and *O* are represented object while "*bkg*" and *B* are represented background in an image. *P* is the number of pixels on the image.

$$E(A) = \lambda \cdot R(A) + B(A) \tag{1}$$

Where

$$R(A) = \sum_{p \in P} R_p(A_p) \tag{2}$$

$$B(A) = \sum_{\{p,q\} \in N} B_{\{p,q\}} \cdot \delta(A_p, A_q) \tag{3}$$

$$\delta(A_p, A_q) = \begin{cases} 1 & A_p \neq A_q \\ 0 & otherwise \end{cases}$$

The coefficient $\lambda \geq 0$ in Eq. (1) specifies the relative importance of the region properties term *R(A)* shown at Eq. (2), to the boundary properties term *B(A)* shown at Eq. (3). The term *R(.)* reflects how the intensity of pixel *p* fits into a known intensity model of object and background. The term *B(A)* comprises the boundary properties of segmentation. *B(A)* is interpreted as a penalty for discontinuity between pixels *p* and *q*. $B_{\{p,q\}}$ is normally large when *p* and *q* are similar.

Table I

EDGE COST

edge		cost	for
n-link	{*p*, *q*}	$B\{p,q\}$	$\{p,q\} \in N$
t-link	{*p*, *S*}	$\lambda \cdot R_p("bkg")$	$p \in P, p \notin O \cup B$
		K	$p \in O$
		0	$p \in B$
	{*p*, *T*}	$\lambda \cdot R_p("obj")$	$p \in P, p \notin O \cup B$
		0	$p \in O$
		K	$p \in B$

Table I lists the edge costs of the graph. The region term and boundary term in Table I are calculated by:

$$\begin{cases} R_p("obj") = -\ln \Pr(O \mid C_p) \\ R_p("bkg") = -\ln \Pr(B \mid C_p) \end{cases} \tag{5}$$

$$B_{\{p,q\}} \propto \exp\left(\frac{(I_p - I_q)^2}{2\sigma^2}\right) \cdot \frac{1}{dist(p,q)} \tag{6}$$

$$K = 1 + \max_{p \in P} \sum_{q:\{p,q\} \in N} B_{\{p,q\}} \tag{7}$$

I_p is the brightness values and C_p is *RGB* color features at pixel *p*. In Eq. (5), the likelihood is computed based on Gaussian Mixture Models. The boundary between the object and the background is found by searching for the minimum cost [7] on graph *G*.

III. OUR APPROACH

The paper proposes a simple graph cuts algorithm based edges features to object segmentation problems. Graph cuts are used to find the global optimum of a cost function based on boundary and region of an image. Smoothing the image using downsampling and upsampling is to obtain the minimize energy while we do not need add or reduce seeds by user. The *n-link* can be computed after downsampling and upsampling of the image. Gaussian Mixture Models (GMMs) are built based on the seeds which are given by user to the background and foreground in an image. The *t-link* is computed by GMMs. The contribution of this paper is to add edges features to GMMs. The proposal can segment an object region having noisy edges and colors similarity between the object and background.

A. *Smoothing image by downsampling and upsampling(n-link)*

We use a max flow algorithm [10] to determine the minimum cut corresponding to the optimal segmentation. The max flow algorithm gradually increases the flow sent from the source *S* to the sink *T* along edges in *G* given their capacities (weights). Upon termination the maximum flow saturates the graph. The saturated edges correspond to the minimum cost cut on *G* giving us an optimal segmentation. In original image segmentation by graph cuts [4], User added or reduced seeds for changing the capacities of graph. In this paper, we change the capacities of graph by the coarse-to-fine level which is shown at the Figure 2. It is the same as Nagahashi [8].

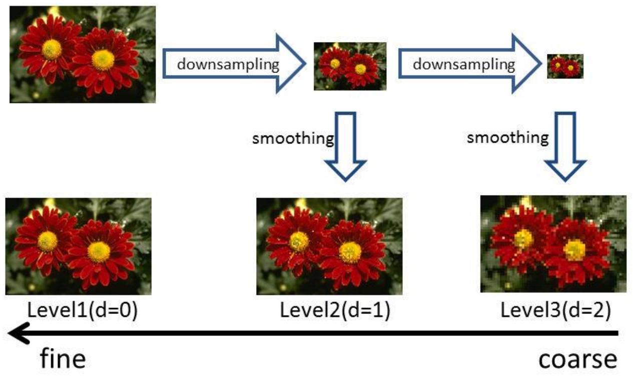


Figure 2. Smoothing the image by downsampling

B. *Edges features by the monochrome images*

As we all know, the traditional graph cuts algorithm is difficult to handle the kind of images which have some noise or analogue colors between the object and background. That is because GMMs just used the colors information. Thus, we add the edges features to GMMs. As a sobel filter has the characteristic to control the noise, we adopt it for obtaining edges features of the image. The monochrome images are used with formula Y=0.299R+0.587G+0.114B in order to compute the sobel filter conveniently. Y stands for luminance. We need obtain edges features of every image which is shown in Figure3.

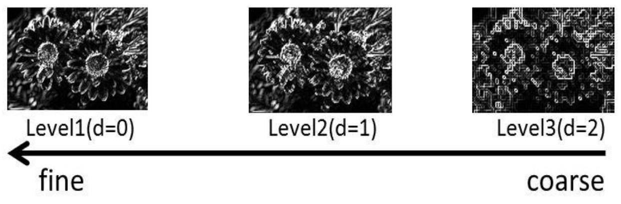


Figure 3. Edges features in the monochrome image

C. *Object segmentation by graph cuts*

4 dimensional features $X_p = \{C_p, E_p\}$ are derived from *R, G, B* color features C_p and edges features E_p. In Eq. (5), *t-link* edge costs are transformed to the posterior probability to achieve greater further accuracy as follows:

$$\begin{cases} R_p{}'("obj") = -\ln \Pr(O \mid X_p) \\ R_p{}'("bkg") = -\ln \Pr(B \mid X_p) \end{cases} \quad (8)$$

The posterior probability is proportional to the product of the prior probability (*Pr(O)*, *Pr(B)*) and the features likelihood according to Bayes' theorem as follows:

$$\begin{cases} \Pr(O \mid X_p) = \dfrac{\Pr(X_p \mid O)\Pr(O)}{\Pr(X_p)} \\ \Pr(B \mid X_p) = \dfrac{\Pr(X_p \mid B)\Pr(B)}{\Pr(X_p)} \end{cases} \quad (9)$$

$$\begin{cases} \Pr(O) = \begin{cases} d_{obj} & d_{obj} \le d_{bkg} \\ 1 - d_{obj} & otherwise \end{cases} \\ \Pr(B) = 1 - \Pr(B) \end{cases} \quad (10)$$

The feature likelihoods $\Pr(X_p \mid O)$, $\Pr(X_p \mid B)$ are derived using Gaussian Mixture Models. The GMMs is obtained by:

$$\Pr(X_p \mid \cdot) = \sum_{i=1}^{K} \alpha_i p_i (X_p \mid \mu_i, \Sigma_i) \quad (11)$$

$$p(X_p \mid \mu, \Sigma) = \frac{1}{(2\pi)^{4/2} |\Sigma|^{1/2}} \cdot e^{(\frac{1}{2}(X_p - \mu)^T \Sigma^{-1} (X_p - \mu))} \quad (12)$$

We use the EM algorithm to fit the GMMs.

We initialize *Pr(O)*=*Pr(B)*=0.5 in *d* level, As we consider a pixel of the image just has two classes which are the object or background. The likelihood $\Pr(X_p \mid O)$, $\Pr(X_p \mid B)$ are derived using Gaussian Mixture Models with seeds of the object and background by user. *t-link*s of the object and background are computed as the (*d*-1)th level image in Eq. (8). The (*d*-1)th level image can be segmented based in Eq. (13). The prior probabilities *Pr(O)* and *Pr(B)* of (*d*-2)th level are computed using the distance transform of the segmentation result of (d-1)th level based on Eq. (10). The distance from the boundary is normalized from 0.5 to 1.0. d_{obj} is defined as the normalized distance to the object, and d_{bkg} is defined as the normalized distance to the background. $\Pr(X_p|O)$, $\Pr(X_p|B)$ of (d-2)th level are computed based on (d-1)th level image segmentation. It is shown in Figure 4. This processing is repeated until d=0.

$$\begin{cases} p \in O & \left|\lambda \cdot R_p'("obj") - \sum_{q:\{p,q\}\in N} B_{\{p,q\}}\right| \\ & < \left|\lambda \cdot R_p'("bkg") - \sum_{q:\{p,q\}} B_{\{p,q\}}\right| \\ p \in B & \left|\lambda \cdot R_p'("obj") - \sum_{q:\{p,q\}\in N} B_{\{p,q\}}\right| \\ & > \left|\lambda \cdot R_p'("bkg") - \sum_{q:\{p,q\}} B_{\{p,q\}}\right| \end{cases} \tag{13}$$

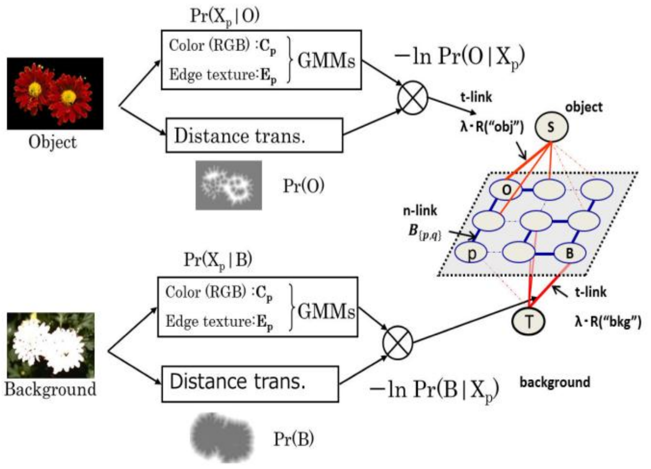


Figure 4. The posterior probability for {p, S} and {p, T}

IV. STEPS OF OUR ALGORITHM

Figure 5 shows the process of the algorithm. It is carried out according to the following procedure.

1) *Degrade a color image to the low resolution image with a downsampling method.*
2) *Give some seeds the dth level color image.*
3) *Obtain Y value of the low resolution monochrome image with the formula Y=0.299R+0.587G+0.114B.*
4) *Extract the edges features of the monochrome image.*
5) *Obtain t-link using the prior probability GMMs.*
6) *Obtain n-link by the dth level color image.*
7) *Obtain a segmentation image by graph cuts.*

Repeat from step 3 to step 6 until the original image is obtained. After that, go ahead to step 7 in order to get a segmentation image.

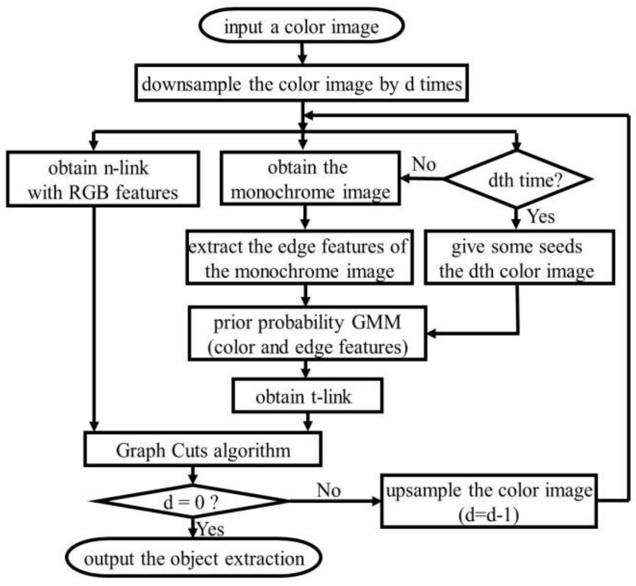

input a color image
downsample the color image by d times
obtain n-link with RGB features
obtain the monochrome image
No
dth time?
Yes
extract the edge features of the monochrome image
give some seeds the dth color image
prior probability GMM (color and edge features)
obtain t-link
Graph Cuts algorithm
d = 0 ?
No
upsample the color image (d=d-1)
Yes
output the object extraction

Figure 5. Flowchart on the process of our proposal

V. EXPERIMENT

The images of segmentation experiments come from the Grab Cuts Database [12]. The image database has the original images and mask images of humans, animals, landscapes and so on. User gives seeds to the original images and the differences between mask images given in the database and output images are computed as error rate as shown in Figure 11. We compare interactive graph [5] (method1), image segmentation using multi-scale smoothing [8] (method2), graph cuts segmentation by texture features [9] (method3) and our proposal. The segmentation error rate is defines as:

$$Error[\%] = \left(\frac{undetected \quad pixels \quad in \quad background}{image \quad size} + \frac{undertected \quad pixels \quad in \quad object}{image \quad size}\right) \times 100\% \tag{14}$$

A. An image with noise and analogue colors between the foreground and background

We give two images for describing our approach. Figure 6 is a difference colors image between the object and background by their histograms which are shown at Figure 8. Figure 7 is an analogue colors image between the object and background by their histograms which are shown at Figure 9. The horizontal axis of figure 8 and 9 is bins from 0 to 255 and the vertical one is the average values of R, G and B. Figure 6 and Figure 7 give the accurate objects and backgrounds based on [12]. We consider that figure 7 has an analogue colors image between the object and background, because they have analogue histograms between the object and background like a Gaussian distribution based on Figure 9. Conversely, we obtain a failure result referring to figure 12, because they have a difference colors image between the object and background based on Figure 8.

Finally, we add Gaussian noises to Figure 7. The experimental result is shown at Figure 10. The error rate of mehtod1 without edges features is 6.78% while the error rate of our approach with edges features is 2.06%. Our approach is effective to the kind of images which have noise and an analogue colors image between the foreground and background.

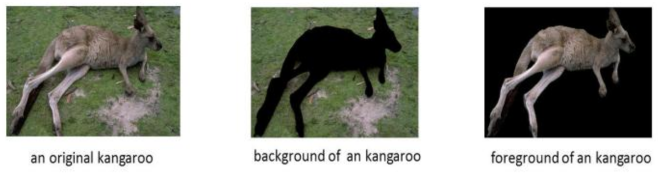


Figure 6. A different colors image between the foreground and background

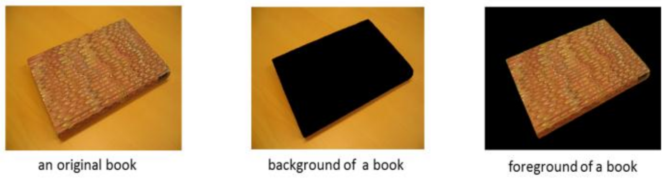


Figure 7. An analogue colors image between the foreground and background

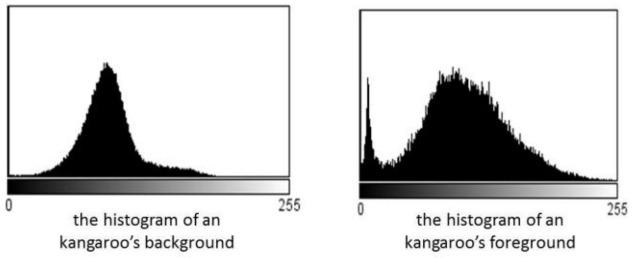


Figure 8. Histogram of the kangaroo image between the foreground and background

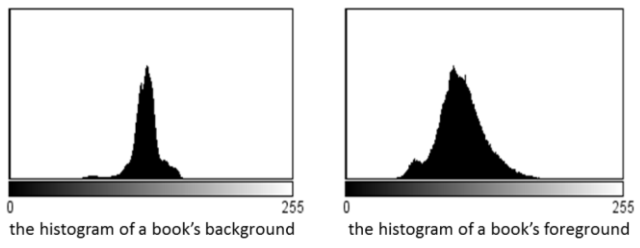


Figure 9. Histogram of the book between the foreground and background

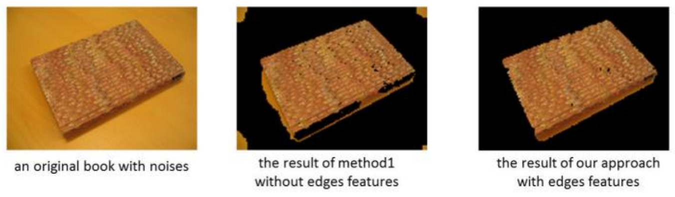

an original book with noises
the result of method1 without edges features
the result of our approach with edges features

Figure 10. The results of the book image with noises

B. Comparison with other methods

Our approach obtains the smallest error rate comparison with other methods. The results are shown in Figure 11. However, our proposal does not always obtain the smallest error rate to all images such as Figure 12. Generally speaking, our approach is always prior to method1 and method2. When the image has heavy texture change, our approach is inferior to method3.

VI. CONCLUSION AND FUTURE WORK

We propose a simple graph cuts algorithm for object segmentation based on edge features of an image. We just add the edges features in GMMs in order to compute *t-link*. The proposal can improve the segmentation error rate compared to the conventional methods to the kind of the images which have an object region having noisy edges and colors similarity between the object and background. Experiments' results also certified the effectiveness of the approach. However, we must find the appropriate the values of σ and λ in order to apply our proposal. Therefore, future work is required adding texture feature for segmentation and the appropriate parameters σ and λ can be obtained automatically.

REFERENCES

[1] Ruzon, M. and Tomasi, C.: "Alpha estimation in natural images," Proc. CVPR, pp. 18-25, 2000.

[2] Chunang, Y. Y., Curless, D., Salesin, D., and Szeliski, R. : "A bayesian approach to digital matting," Proc. CVPR, pp. 264-271, 2000.

[3] W. Du and K. Urahama : "Natural image matting with membership propagation," IPSJ Trans. CVA, vol. 1, no. 1, pp. 3-11, 2009.

[4] Y.Boykov and M.-P. Jollu.:"Interactive organ segmentation using graph cuts," In Medical Image Computing and Computer-Assisted Intervention, pp. 276-286, 2000.

[5] Y.Boykov, Y., Jolly, and M.-P.: "Interactive graph cuts for optimal boundary and region segmentation of object in N-D image segmentation," Internation Conference on Computer Vision, vol. I, pp. 105-112, 2001.

[6] Y.Boykov and G. Funka-Lea: "Graph cuts and efficient N-D image segmentation," Int. J. Comput. Vis. vol. 70, no. 2, pp. 109-131, 2006.

[7] Y.Boykov and V. Kolmorov: "An experimental comparison of mincut/max-flow algorithms for energy minimization in vision," IEEE Trands. Pattern Anal. Mach. Intell. vol. 26, no. 9, pp. 1124-1137, Sep. 2004.

[8] T. Nagahashi, H.Fujiyoshi, and T.Kanade: "Image segmentation using iterated graph cuts based on multi-scale smoothing," Asian Conference on Computer Vision, Part II, LNCS 4844, pp. 806-816, 2007.

[9] K. Fukuda, T. Takiguchi, and Y. Araki: "Graph cuts segmentation by using local texture features of multiresolution analysis," IEICE TRANs. INF. and SYST. vol. E92-D, no. 7, pp. 1453-1460, 2009.

[10] L. Ford and D. Fulkerson.: "Flows on networks," Princeton University Press, 1962.

[11] A. Goldberg and R. Tarjan. : "A new approach to the maximum flow problem," Journal of the Association for Computing Machinery, 35(4): 921-940, October 1988.

[12] http://research.microsoft.com/en-us/um/cambridge/projects/visionimagevideoediting/segmentation/grabcut.htm/. Retrieved: September, 2012

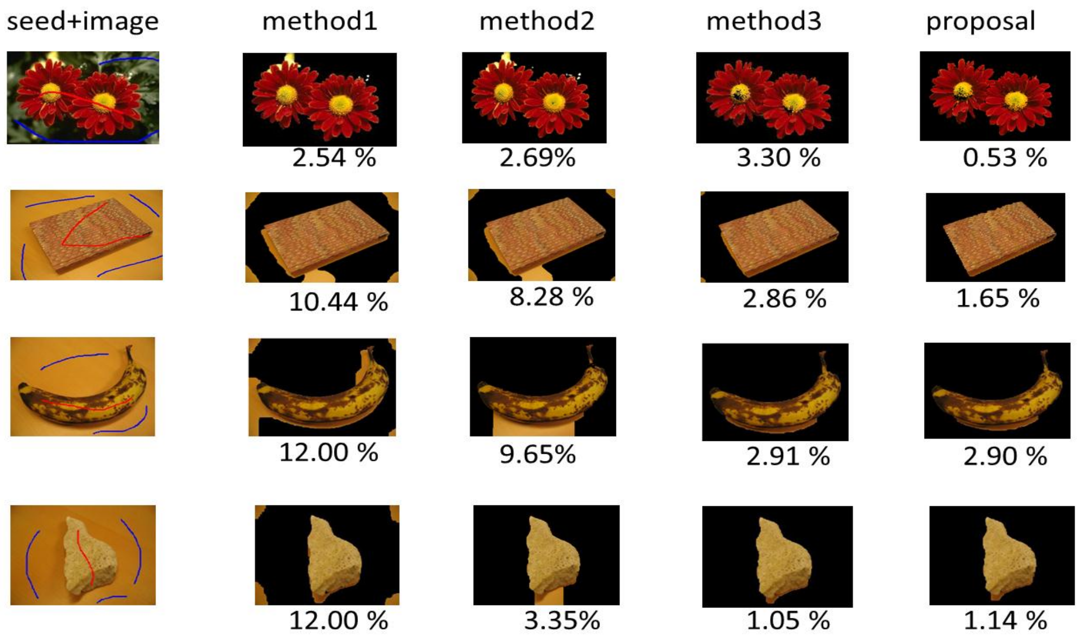


Figure 11. Examples of segmentation results

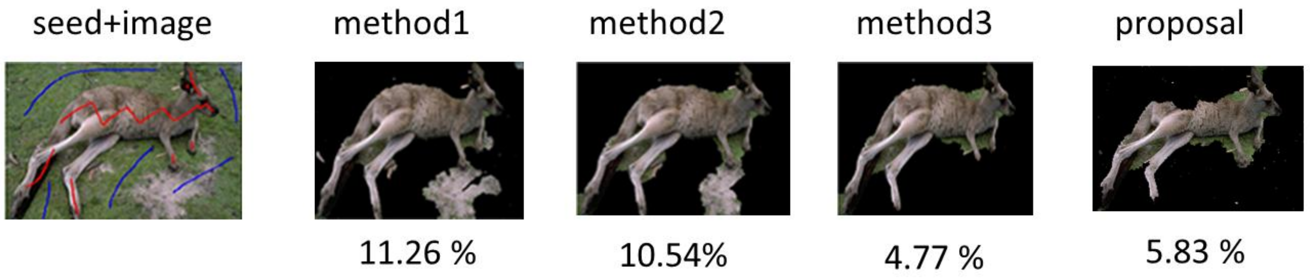


Figure 12. A failure example of segmentation result

Product Development Time Improvement with Requirements Reuse

Semra Yilmaz Tastekin
Informatics Institute
Middle East Technical University
Ankara, Turkey
e-mail: syilmaz@aselsan.com.tr

Yusuf Murat Erten
Innova Inc.
Ankara, Turkey
e-mail: merten@innova.com.tr

Semih Bilgen
Electrical and Electronics Engineering
Middle East Technical University
Ankara, Turkey
e-mail: bilgen@eee.metu.edu.tr

Abstract—**Product development time estimation is important for project management tasks. This study investigates the impact of requirements reuse on product development duration for different products in a similar domain. We propose an analytical tool to estimate the minimum time to be saved given the percentage of requirements reused from earlier projects. Empirical results of industrial case studies are used as inputs to this study. Three cases from different organizations have been studied for software and system development projects, which consist of hardware and software components. The results of the case studies are compared with a study in the literature on product development time. According to the industrial empirical results, a modified product development time estimation method is proposed for software projects.**

Keywords - requirements reuse; product development time.

I. Introduction

In most organizations, project deliverables are generally systems that combine hardware and software components. For system projects, which include hardware and software components, software is becoming an increasingly significant constituent [1][2] and project management is accordingly becoming more complex.

Engineers working in these projects discover most of the problems at the integration phase. Isolation of the source of these problems at this stage can take time and this may affect the project duration. According to [3], 50% of total time and cost of a project is spent for testing. So, minimizing possible system faults at earlier stages will minimize the test efforts. Since most defects are found in the integration and test phases, these phases are generally stressful for developers and testers since they are responsible for correcting the defects. Therefore, reuse of the components created during each phase of different projects has an important role in eliminating the defects in the product, correspondingly reducing the engineering effort in the project.

This study focuses on reducing some of the sources of defects hence the product development time via requirements reuse. The motivation behind the reuse of requirements items from previous projects is that these are already validated and accepted by end users in previous projects. In this study, three case studies are presented to demonstrate the benefits of requirements reuse for a system project (containing both hardware and software) and a software product. In the literature, there are some studies on the reuse of some components and they have generally been performed for software code reuse. One of the contributions of the present study is to provide empirical results, which demonstrate the reduction of project duration due to requirements reuse for industrial products. Another contribution is to show that the product development time estimations proposed earlier in the literature for manufacturing industries [19] can be applied to system and software projects. We also propose a modification to the current formulation to represent software-based projects more accurately [19].

This paper is organized as follows. Section II briefly reviews the software product line literature, focusing specifically on product development time. Section III discusses the impact of requirements reuse in the context of industrial case studies. Section IV presents a modified product development time proposal. Section V concludes the paper and suggests future directions for reuse in project management.

II. Related Work

This section reviews the literature on Software Product Line (SPL) and product development time.

A. Software Product Line

When reuse is applied in all stages of the product development cycle, this corresponds to the product line concept. The aim of the product line concept is to enhance the quality of the product and decrease the engineering cost. In the literature, there are many studies, which evaluate the establishment of software and system product lines in organizations [5][6][9][10].

The accepted SPL approach is based on two-life cycles, which are Domain Engineering (DE) and Application Engineering (AE). SPL mostly focuses on the DE activities [5], which create a common infrastructure for the related domain. On the other hand, AE is active when there is a new project. To produce a product, the required assets are selected from the common assets created by DE. For the remaining part of the product, new assets are created from scratch. Although SPL applies to all phases of projects life cycle, this study covers only the requirements definition phase. Thus, in the following sections, requirements definition phase, which is a part of SPL, is studied.

B. Product Development Time

While preparing project proposals, little information is available concerning the development details. Before starting the development, it is important to have information about development time to estimate the cost of the project used for proposals.

In the literature, there is some information about the methods for estimating the duration for product development. Griffin [18] classifies the metrics, which affect the development time in four groups as;

- Changes during the product generation,
- The complexity of product,
- Whether a formal process is used in the organization, and
- Whether a cross functional team is used.

Changes in requirements have a considerable effect on the development workload ([11][20][21]). Therefore, it is important to define the requirements accurately to ensure minimum change during the development stage. The number of main functions the product performs gives the complexity of the product [19]. If organizations do not have formal development processes, the development time is higher compared to those with formal development processes. A study by Olson et al. [12] emphasizes that the use of a cross functional team is also an important parameter in increasing project performance.

Griffin [18][19] defines Development Time (DT) and Concept To Customer (CTC) as two separate parameters. DT begins from design up to the introduction to the customer and CTC begins with concept development and continues to specification definition until the introduction to the customer. Requirements engineering activities are covered within CTC. If DT is subtracted from CTC this will give the time spent for requirements engineering activities. DT and CTC formulations proposed for the manufacturing industries are given below [19].

$$DT = \alpha + \beta_{1DT} * PC + \beta_{2DT} * NN + \beta_{3DT} * (PC * FP) + \beta_{4DT} * (NN * XFT) + \epsilon_{DT} \quad (1)$$

$$CTC = \alpha + \beta_{1CTC} * PC + \beta_{2CTC} * NN + \beta_{3CTC} *(PC * FP) + \beta_{4CTC} * (NN * XFT) + \epsilon_{CTC} \quad (2)$$

where α is the cycle time constant, PC and NN are product complexity and product newness/uncertainty, respectively. If NN value increases, the change probability on the product during the development also increases. FP and XFT show whether formal processes and cross functional teams are used, respectively. If formal development processes are not used, then FP=0. ϵ is the error term. The unit of β_1 and β_3 are the months/function designed in the product. The unit of β_2 and β_4 are the months/percentage of change in the product. The estimation of the coefficients α and β, based on data collected from many companies, is given in Table I [19].

TABLE I. COEFFICIENTS USED IN THE DT AND CTC FORMULATIONS

	α	β_1	β_2	β_3	β_4
DT	8.4	4.2	0.09	-1.9	-0.09
CTC	10.4	3.7	0.16	0.1	-0.16

III. PROBLEM DEFINITION AND QUANTITATIVE DATA

The requirements phase will have the greatest impact on the subsequent steps in the project life cycle. If there are defects in the requirements phase, its negative effect is reflected on the later phases. Considering that 7-15% of the total project resources are used for requirements engineering [14][15], requirements-related phases of the development lifecycle should be realized as effectively as possible. Besides, the customer generally needs the product within the shortest possible time. This puts pressure on the project manager and engineers to find ways to shorten the project duration. This can be achieved by shortening the phases in the life cycle.

The requirements engineering stage can be shortened by effective reuse of requirements, which are generated at the initial step of the project. In the final step, test activities are performed to determine whether the system meets defined requirements. Once the system is accepted by the customer, all the requirements defined at the beginning of the project are tested and approved. So, reusing these requirements in other projects will enable easy acceptance of the same features in similar systems. This will increase the quality of the requirements and will reduce the project duration [8][13][22].

In the literature, there are many studies discussing guidelines for reuse but few compare project development time with reuse and without reuse. For example, Johnson [23] proposes a model, which integrates SPL with a software development environment; however, this study does not have measured results. Similarly, Catal and Diri [4] proposes a framework for software fault prediction in SPL but does not present an analysis due to the lack of fault data. There are several theoretical studies concerning software components reuse [6][7][8][16][17], but little research about hardware components. In the present study, the requirements engineering phase in software intensive systems projects is analyzed for two different organizations using a reuse approach. Following paragraphs present the quantitative data from these organizations. The results of the case studies are compared with the results of Griffin's study [19]. Of the three case studies the first involved the reuse of the requirements for a system with hardware and software functionalities. The second and third cases involved the reuse of the requirements for software products. Cross functional teams are not used for the projects in these case studies.

A. Characteristics of the Case Studies

To analyze the product development time using the requirements reuse approach, it was intended to gather data from different companies and the requirements engineering phase would be analyzed for three different organizations. Data is collected with a joint effort of the project technical managers and the authors. Unfortunately, there were some difficulties in gathering data from different companies. First, organizations generally do not keep the project related data in a systematic way. To overcome this difficulty, some interviews with the project technical managers were undertaken and the related data were collected using related documents and organizational database. Second, even when metrics were kept systematically, the organizations would prefer not to release this data for external use.

The data collected in this study is quantitative ([24]). The necessary data to perform the case studies include the total number of requirements, the number of requirements reused from a previous project and the duration of the requirements definition phase. These data are presented in tabular format below (Table II, Table III, Table V, Table VI, Table VII, Table VIII, Table X, Table XI).

The project related data are derived from the responses to the questions given below, following the methods as described:

Question 1: Are there any similar products which can be in the same domain or are derivative products in the company?

Method Used for Answering Question 1: Discussions with different project technical managers from different companies are performed and the details of the projects are evaluated.

Question 2: Is there recorded data for the number of requirements for each project in the same domain?

Method Used for Answering Question 2: The System Requirements Documents for each project are used to obtain the necessary data.

Question 3: Is there recorded data for reused requirements?

Method Used for Answering Question 3: If the metrics are kept systematically, data is retrieved from organization database. If they are not kept in an organizational database, reused requirements are derived from the system requirements documents by technical personnel involved in the projects.

Question 4: Is there duration data for requirement definition phases?

Method Used for Answering Question 4: The enterprise resource planning systems of the companies are used to extract this data.

Question 5: What is the complexity level of the product to be studied?

Method Used for Answering Question 5: As defined in [19], the main functions of the products are defined to obtain the complexity level of the product with the help of technical managers of the projects. If there is critical technology to be developed within the scope of the product, this factor is also added to the complexity. For example, if the product has 4 main functions and includes 2 technological infrastructure developments, the complexity level is defined for this product as 6.

The data from the case studies are used for the following purposes;

- Deriving the realized duration for requirement definition phases of system products and software products,
- The comparison of durations for different projects in the same domain,
- Comparison of realized duration with the theoretically predicted product development time ([19]),
- Separately analyzing the software products and system products,
- Definition of any necessary modification to product development time proposed in [19] if required.

Using Griffin's study [19] and case studies data from current research, the following issues will be resolved at the end of this study;

- The product development time proposed in [19] was appropriate for system products which involve hardware and software components,
- But product development time proposed in [19] was not appropriate for software products,
- There was a need to modify the product development time prediction for more accurate estimations for software products.

B. Case Study-1

Project A1 and Project A2 in the same domain from Company A were analyzed. Some of the requirements of Project A1 have been reused in Project A2. Table II shows the number of requirements in Project A2. Project A2 had 183 requirements in total. 104 requirements of those were reused from Project A1. Remaining 79 requirements were created from scratch for Project A2. The realized duration for requirements engineering (RE) activities for both projects is given in Table III.

Change probability of 104 reused requirements was very low in Project A2, because they were tested, and approved by the customer previously. This implies that;

- 57% of total requirements (104 requirements) for Project A2 were almost fixed.
- 43% of total requirements (79 new requirements) could still be changed in Project A2.

By reusing the requirements, change probability in the product (NN) is minimized. While NN varies between 0% and 100%, by reusing the requirements it can be decreased in the range of 0% to 43% for Project A2. By using Griffin's CTC formulation (2), this situation indicates that for all possible changes in the requirements, the organizations would require an additional 16 months ($\beta_{2CTC} * NN = 0.16 * 100\%$) if requirements were not reused. On the other hand, the organization would only require an additional 6.88 months (0.16*43%) maximum when requirements were reused. So, change effect is reduced by 9.12 months for Case Study-1.

TABLE II. REQUIREMENTS USED IN PROJECT A2

	Total # of Req. in Project A2	**Req. of Project A1 Reused in Project A2**
Total	183	104 (57% of Project A2)

TABLE III. DURATION EXPENDED IN PROJECT A1 AND PROJECT A2

	Project A1	**Project A2**	**Possible Impact of Reuse**
Total Duration (months)	8	5	37 % decrease in duration

To estimate the time spent for RE activities, the calculations of DT and CTC given in Appendix-A for 100% and 43% cases are performed using (1) and (2). 100% indicates that product requirements/features were totally new. 43% indicates the amount of new requirements, and is taken as the change probability of the requirements. The complexity level of the product developed in the scope of Project A2 was taken as 6 based on the number of main functions the product has. Eventually, calculated time spent for RE activities is summarized in Table IV.

Even if the maximum change (43%) occurs in the requirements, there would be at least 22% decrease (from 18 months to 14.01 months) in the duration of RE activities. If the change in the requirements were less than 43% change, the improvement would be expected to be greater than 22%.

When this result is compared with the actual results of Case Study-1 in Table III, the decrease in the Project A2 shows agreement with these calculations. Griffin's formulation predicts at least 22% reduction in duration, likewise a reduction of 37% (more than 22%) was obtained. Thus, formulations proposed by Griffin for the estimation of project duration applies for the system, which involves both hardware and software.

C. Case Study-2

For this case, three software projects from Company B were analyzed; Projects B2 and B3 used Project B1 as a baseline. Table V and Table VI show the number of requirements in Project B2 and Project B3. Project B2 had 376 requirements in total. 314 of those were reused from Project B1. Remaining 62 requirements were created from scratch for Project B2. Similarly, Project B3 had totally 323 requirements, 230 of those were reused from Project B1 and 93 new requirements were created. For the requirements engineering phase of both projects, the duration data are given in Table VII and Table VIII.

As shown in Table V, for Project B2;

- 84% of total requirements (314 requirements) were almost fixed and change probability of those was very low.
- 16% of total requirements (62 new requirements) could still be changed during the product cycle time.

Similarly for Project B3, as shown in Table VI;

- 71% of total requirements (230 requirements) were almost fixed and change probability of those was very low.
- 29% of total requirements (93 new requirements) could still be changed during the product cycle time.

When the requirements are reused, changes in product could be decreased in the range of 0% to 16% for Project B2 and 0% to 29% for Project B3. Using (2), if the requirements were not reused, the organizations would require an additional 16 months (0.16*100%). On the other hand organization would only require an additional 2.56 months (0.16*16%) maximum for Project B2 and 4.64 months (0.16*29%) maximum for Project B3 if requirements were reused.

TABLE IV. CASE STUDY-1: ESTIMATED TIME SPENT FOR RE WORKS

	RE works for 100% change	RE works for 43% change	% of Decrease in RE works
CTC-DT	18 months	14.01 months	$\geq$ 22%

TABLE V. REQUIREMENTS USED IN PROJECT B2

	Total # of Req. in Project B2	Req. of Project B1 Reused in Project B2
Total	376	314 (84% of Project B2)

TABLE VI. REQUIREMENTS USED IN PROJECT B3

	Total # of Req. in Project B3	Req. of Project B1 Reused in Project B3
Total	323	230 (71% of Project B3)

Detailed calculations of DT and CTC are given for possible changes of 100%, 16% and 29% in Appendix-B using (1) and (2). 16% and 29% indicate those requirements, which were new and can be changed for Project B2 and Project B3, respectively. The complexity level of the products in Project B2 and B3 was taken as 3 again based on the number functions in the software. By using the results of calculations, the time spent for RE activities is summarized in Table IX.

However, calculated results are not similar to the actual results of Case Study-2 in Table VII and Table VIII. The decreases in both Project B2 and Project B3 were 34% in real life. But Griffin's formulation predicts at least 44% and 37% decreases in Project B3 and Project B4, respectively. This is explained and tackled in Section IV.

D. Case Study-3

Two software projects from Company A were studied as Project A3 and Project A4. In Project A4, some of the requirements from Project A3 were reused as well as some additional requirements included by the customer. Table X shows the number of requirements in Project A4. Project A4 had 342 requirements in total. 255 of those were reused from Project A3. For the requirements engineering phase of both projects, the data related to the duration are given in Table XI.

As shown in Table X, for Project A4;

- 75% of total requirements (255 requirements) were almost fixed and change probability of those was very low.
- 25% of total requirements (87 new requirements) could still be changed during the product cycle time.

By reusing the requirements, changes in product could be decreased in the range of 0% to 25% for Project A4.

TABLE VII. DURATION EXPENDED IN PROJECT B1 AND PROJECT B2

	Project B1	Project B2	Possible Impact of Reuse
Total Duration (months)	7,5	5	34 % decrease in duration

TABLE VIII. DURATION EXPENDED IN PROJECT B1 AND PROJECT B3

	Project B1	Project B3	Possible Impact of Reuse
Total Duration (months)	7,5	5	34 % decrease in duration

TABLE IX. CASE STUDY-2: ESTIMATED TIME SPENT FOR RE WORKS

	RE works for 100% change	RE works for 16% and 29% change	% of Decrease in RE works
CTC-DT	13.5 months	7.62 months 8.53 months	$\geq$ 44% (16% change) $\geq$ 37% (29% change)

Using (2), if requirements were not reused, the organizations would require an additional 16 months (0.16*100%), while the organization would only require an additional 4 months (0.16*25%) maximum for Project A4 if requirements were reused.

Detailed calculations of DT and CTC are given for 100% and 25% changes in Appendix-C using (1) and (2). The durations for 100% and 25% changes is summarized in Table XII. The complexity level of the product was defined as 5.

Again, these results are not similar to the actual results of Case Study-3 in Table XI. The decrease in the Project A4 was 25% in real life. But Griffin's formulation predicts at least 32% decrease in Project A4.

The results of all case studies are tabulated in Table XIII.

IV. DISCUSSION

According to the empirical results of Case Study-1, Griffin's formulation for product development time is validated for system projects. But, empirical results of Case Study-2 and Case Study-3 show that it is necessary to modify Griffin's formulation for software projects. Software requirements may change more easily when compared to hardware requirements. The nature of software allows the customer to feel more comfortable while requesting changes. Thus, changes in software projects were more than expected. There are some decreases in project durations for Case Study-2 and Case Study-3, but these are less than what would be expected according to Griffin's formulation.

Changes in product features, i.e., in requirements are denoted by NN (newness/uncertainty) variable in (1) and (2). NN variable in [19] should be re-evaluated for software projects as this parameter should have a more significant effect on the software product development time according to the case studies performed in the scope of this study. By referring to these case studies, if the effect of NN variable is multiplied by at least 2.1 but not more than 3.4 for the cases where the product is not totally new, the results of Griffin's formulation show similarities with the real life results. Using a multiplier out of this range does not produce the realized results for the case studies. When the multiplication coefficient (referred as δ) is chosen to be close to 3.4, the decrease in the duration is going to be smaller. For values larger than 3.4, the results are similar to the case where requirements are not reused at all. So, δ is selected as 2.1 to see the best effect of requirements reuse. The projects for each case study selected from Company A and Company B are in the same domain and share the common requirements. Therefore, evaluations regarding the results of the case studies do not cover totally new products.

TABLE X. REQUIREMENTS USED IN PROJECT A4

	Total # of Req. in Project A4	Req. of Project A3 Reused in Project A4
Total	342	255 (75% of Project A4)

TABLE XI. DURATION EXPENDED IN PROJECT A3 AND PROJECT A4

	Project A3	Project A4	Possible Impact of Reuse
Total Duration (months)	6	4.5	25 % decrease in duration

TABLE XII. CASE STUDY-3: TIME SPENT FOR RE WORKS

	RE works for 100% change	RE works for 25% change	% of Decrease in RE works
CTC-DT	16.5 months	11.25 months	$\geq$ 32%

TABLE XIII. EXPECTED AND ACTUAL CHANGES IN DURATION OF RE ACTIVITIES FOR PROJECTS A2, B2, B3, A4 USIG GRIFFIN'S FORMULATION

Project	Max. Expected % of Change in Req.	Expected % of Duration Decrease in RE Works	Actual % of Duration Decrease in RE Works
A2	43%	$\geq$ 22%	37%
B2	16%	$\geq$ 44%	34%
B3	29%	$\geq$ 37%	34%
A4	25%	$\geq$ 32%	25%

The modified versions of (1) and (2) are proposed as below. The duration estimations include the engineering efforts during the requirements engineering phases. Other departments such as marketing, finance etc. are not included in the scope of the case studies. Therefore, this modification assumes that a cross functional team is not used in the organizations.

$$DT = \alpha + \beta_{1DT} * PC + \beta_{2DT} * \delta * NN + \beta_{3DT} * (PC * FP) + \epsilon_{DT} \quad (3)$$

$$CTC = \alpha + \beta_{1CTC} * PC + \beta_{2CTC} * \delta * NN + \beta_{3CTC} *(PC * FP) + \epsilon_{CTC} \quad (4)$$

where 2.1 $\leq \delta \leq$3.4.

Calculations for Case Study-2 of Company B are repeated in Appendix-D using (3) and (4). Equations (1) and (2) are used without any modification for the case when the product is totally new (NN=100%). Even if the maximum change (16%) occurred in the requirements for Project B2, the calculation indicates that there would be at least a 34% decrease in the duration (from 13.5 months to 8.85 months) of the RE activities. Real-life duration reduction, which was 34% as given in Table VII is in agreement with this result.

Similarly, if the maximum change (29%) occurred in the requirements for Project B3, the calculation indicates that there would be at least a 20% decrease in the duration (from 13.5 months to 10.76 months) of the RE activities. Consequently, real-life reduction of duration, which was 34% as given in Table VIII is in agreement with this result.

Similar calculations for Case Study-3 of Company A are performed in Appendix-E using (3) and (4). Again (1) and (2) are used for NN=100%. Even if the maximum change (25%) occurred in the requirements for Project A4, the calculation indicates that there would be at least a 20% decrease in the duration (from 16.5 months to 13.17 months) of the RE activities. The actual reduction in the duration, which was 25% as given in Table XI is in agreement with this result.

Summarized results of Case Study-2 and Case Study-3 using proposed formulations are given in Table XIV.

V. Conclusion and Future Work

The focus of this study is the reuse of requirements for different products in similar domains. The effects of requirement reuse for two different product types, one consisting of hardware and software, and the other purely software, have been investigated. For this investigation three case studies have been performed and their results have been compared with a theoretical study ([19]). It is very likely that different projects in the same domain have many common requirements and if these requirements were maintained and shared in a common database that employees could access, systems engineers would choose to use these requirements in different projects. In the context of such an opportunity, the product would be developed within a common understanding of the requirements. Besides, availability of applicable product development time estimations has an importance on the project management tasks. By using an applicable estimation model for industrial products, it should be possible to make project budget and resource allocation with minimum error. It might be difficult to work with minimum error at the beginning of the project with less information about the development details. From the case studies it is concluded that the proposed method [19] can be applied to system projects with hardware and software. However, it does not yield the same results with real-life software projects. This is because software requirements may change more easily when compared to hardware requirements. So, Griffin's formulation is revised for the software products and proposed formulation has more effect for the change probability of the product.

A. Future Work

This study showed the effects of requirements reuse on project duration based on empirical results of the case studies in which the data are collected from industry. This study covers only the requirements analysis phase. In terms of the future work, the duration can be further reduced by investigating reuse in the other phases of the project life cycle.

Table XIV. Expected and Actual Changes in Duration of RE Activities for Projects B2, B3, A4 Using the Proposed Formulation

Project	Max. Expected % of Change in Req.	Expected % of Duration Decrease in RE Works	Actual % of Duration Decrease in RE Works
B2	16%	$\geq$ 34%	34%
B3	29%	$\geq$ 20%	34%
A4	25%	$\geq$ 20%	25%

To enhance the validity of the δ value in proposed method, additional case studies can be performed. The effect of reuse can also be studied for the organizations in which cross functional teams are used.

Appendix

A. Calculations for Case Study-1

$$CTC_{100} = 10.4+3.7*6+0.16*100\%+0.1*6 = 49.2 \text{ months}$$

$$CTC_{43} = 10.4+3.7*6+0.16*43\%+0.1*6 = 40.08 \text{ months}$$

$$DT_{100} = 8.4+4.2*6+0.09*100\% - 1.9*6 = 31.2 \text{ months}$$

$$DT_{43} = 8.4+4.2*6+0.09*43\% - 1.9*6 = 26.07 \text{ months}$$

The time spent for requirements engineering is:

$$CTC_{100} - DT_{100} = 49.2 - 31.2 = 18 \text{ months}$$

$$CTC_{43} - DT_{43} = 40.08 - 26.07 = 14.01 \text{ months}$$

B. Calculations for Case Study-2

$$CTC_{100} = 10.4+3.7*3+0.16*100\%+0.1*3 = 37.8 \text{ months}$$

$$CTC_{16} = 10.4+3.7*3+0.16*16\%+0.1*3 = 24.36 \text{ months}$$

$$CTC_{29} = 10.4+3.7*3+0.16*29\%+0.1*3 = 26.44 \text{ months}$$

$$DT_{100} = 8.4+4.2*3+0.09*100\% - 1.9*3 = 24.3 \text{ months}$$

$$DT_{16} = 8.4+4.2*3+0.09*16\% - 1.9*3 = 16.74 \text{ months}$$

$$DT_{29} = 8.4+4.2*3+0.09*29\% - 1.9*3 = 17.91 \text{ months}$$

The time spent for requirements engineering is:

$$CTC_{100} - DT_{100} = 37.8 - 24.3 = 13.5 \text{ months}$$

$$CTC_{16} - DT_{16} = 24.36 - 16.74 = 7.62 \text{ months}$$

$$CTC_{29} - DT_{29} = 26.44 - 17.91 = 8.53 \text{ months}$$

C. Calculations for Case Study-3

$$CTC_{100} = 10.4+3.7*5+0.16*100\%+0.1*5 = 45.4 \text{ months}$$

$$CTC_{25} = 10.4+3.7*5+0.16*25\%+0.1*5 = 33.4 \text{ months}$$

$$DT_{100} = 8.4+4.2*5+0.09*100\% - 1.9*5 = 28.9 \text{ months}$$

$$DT_{25} = 8.4+4.2*5+0.09*25\% - 1.9*5 = 22.15 \text{ months}$$

The time spent for requirements engineering is:

$$CTC_{100} - DT_{100} = 45.4 - 28.9 = 16.5 \text{ months}$$

$$CTC_{25} - DT_{25} = 33.4 - 22.15 = 11.25 \text{ months}$$

D. Calculations for Case Study-2 according to the proposed formulation (δ = 2.1)

$$CTC_{100} = 10.4+3.7*3+0.16*100\%+0.1*3 = 37.8 \text{ months}$$

$$CTC_{16} = 10.4+3.7*3+\mathbf{2.1}*0.16*16\%+0.1*3 = 27.18 \text{ months}$$

$$CTC_{29} = 10.4+3.7*3+\mathbf{2.1}*0.16*29\%+0.1*3 = 31.54 \text{ months}$$

$$DT_{100} = 8.4+4.2*3+0.09*100\%-1.9*3 = 24.3 \text{ months}$$

$$DT_{16} = 8.4+4.2*3+\mathbf{2.1}*0.09*16\%-1.9*3 = 18.33 \text{ months}$$

$$DT_{29} = 8.4+4.2*3+\mathbf{2.1}*0.09*29\%-1.9*3 = 20.78 \text{ months}$$

Time spent for requirements engineering is calculated as;

$$CTC_{100} - DT_{100} = 37.8 - 24.3 = 13.5 \text{ months}$$

$$CTC_{16} - DT_{16} = 27.18 - 18.33 = 8.85 \text{ months}$$

$$CTC_{29} - DT_{29} = 31.54 - 20.78 = 10.76 \text{ months}$$

E. Calculations for Case Study-3 according to the proposed formulation (δ = 2.1)

$$CTC_{100} = 10.4+3.7*5+0.16*100\%+0.1*5 = 45.4 \text{ months}$$

$$CTC_{25} = 10.4+3.7*5+\mathbf{2.1}*0.16*25\%+0.1*5 = 37.8 \text{ months}$$

$$DT_{100} = 8.4+4.2*5+0.09*100\%-1.9*5 = 28.9 \text{ months}$$

$$DT_{25} = 8.4+4.2*5+\mathbf{2.1}*0.09*25\%-1.9*5 = 24.63 \text{ months}$$

Time spent for requirements engineering is calculated as;

$$CTC_{100} - DT_{100} = 45.4 - 28.9 = 16.5 \text{ months}$$

$$CTC_{25} - DT_{25} = 37.8 - 24.63 = 13.17 \text{ months}$$

ACKNOWLEDGMENTS

The authors would like to thank the companies who have contributed the necessary data for the case studies to complete this work.

REFERENCES

[1] Matthew R. Kennedy and David A. Umphress, "An Agile Systems Engineering Process - The Missing Link", *CrossTalk, The Journal of Defense Software Engineering"*, May-June 2011.
[2] Richard Turner, "Toward Agile Systems Engineering Processes", *Cross Talk, The Journal of Defense Software Engineering*, April 2007.
[3] Shaojie Guo, Weiqin Tong, Juan Zhang, and Zongheng Li, "An Application of Ontology to Test Case Reuse", *International Conference on Mechatronic Science, Electric Engineering and Computer*, pp. 775-778, 2011.
[4] Cagatay Catal and Banu Diri, "A Conceptual Framework to Integrate Fault Prediction Sub-process for Software Product Lines", *2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*, pp. 99-106, 2008.
[5] Timo Kkl, "Standards Initiatives for Software Product Line Engineering and Management within the International Organization for Standardization", *Proceedings of the 43rd Hawaii International Conference on System Sciences*, pp. 1-10, 2010.
[6] Sholom Cohen, "Guidelines for Developing a Product Line Concept of Operations", Technical Report. CMU/SEI-99-TR-008, August 1999.
[7] John K. Bergey, Gary Chastek, Sholom Cohen, Patrick Donohoe, Lawrence G. Jones, and Linda Northrop, "Software Product Lines: Report of the 2010 U.S. Army Software Product Line Workshop", Technical Report, CMU/SEI-2010-TR-014, June 2010.
[8] Parastoo Mohagheghi and Reidar Conradi, "An Empirical Investigation of Software Reuse Benefits in a Large Telecom Product", *ACM Transactions on Software Engineering and Methodology*, Vol. 17, Issue 3, Article No 13, June 2008.
[9] Frank Dordowsky, Richard Bridges, and Holger Tschpe, "Implementing a Software Product Line for a complex Avionics System", *15th International Software Product Line Conference*, pp. 241-250, 2011.
[10] Kentaro Yoshimura, Jun Shimabukuro, Takatoshi Ohara, Chikashi Okamoto, Yoshitaka Atarashi, Shinobu Koizumi, Shigeru Watanabe, and Kazumi Funakoshi, "Key Activities for Introducing Software Product Lines into Multiple Divisions: Experience at Hitachi", *15th International Software Product Line Conference*, pp. 261-266, 2011.
[11] Andy J. Nolan, Silvia Abraho, Paul C. Clements, and Andy Pickard, "Requirements Uncertainty in a Software Product Line", *15th International Software Product Line Conference*, pp. 223-231, 2011.
[12] Eric M. Olson, Orville C. Walker, Robert W. Ruekert, and Joseph M. Bonner, "Patterns of Cooperation During New Product Development Among Marketing, Operations and R&D: Implications for Project Performance", *Journal of Product Innovation Management*, Vol 18, Issue 4. pp. 258-271, July 2001.
[13] Wayne C. Lim, "Effects of Reuse on Quality, Productivity, and Economics", *IEEE Software*, Vol. 11, Issue 5. pp. 23-30, 1994
[14] Gerald Kotonya and Ian Sommerville, *Requirements Engineering Processes and Techniques*, John Wiley & Sons Press, 1998.
[15] Tony Gorschek and Alan M. Davis, "Requirements Engineering: In Search of the Dependent Variables", *Information and Software Technology*, Vol. 50, No. 1-2. pp. 67-75, January 2008.
[16] Oscar Lopez Villegas and Miguel Angel Laguna, "Requirements Reuse for Software Development", *5th IEEE International Symposium on Requirements Engineering*, 2001.
[17] Donald Firesmith, "Achieving Quality Requirements with Reused Software Components: Challenges to Successful Reuse", *2nd International Workshop on Models and Processes for the Evaluation of off-the-shelf Components*, 2005.
[18] Abbie Griffin, "Metrics for Measuring Product Development Cycle Time", *Journal of Product Innovation Management*, Vol. 10, No. 2, pp. 112-125, March 1993.
[19] Abbie Griffin, "The Effect of Project and Process Characteristics on Product Development Cycle Time", *Journal of Marketing Research*, Vol. 34, No. 1, pp. 24-35, February 1997.
[20] He-Biao Yang, Zhi-Hong Liu, and Zheng-Hua Ma, "An Algorithm for Evaluating Impact of Requirement Change", *Journal of Information and Computing Science*, Vol. 2, No. 1, 2007.
[21] Susan Ferreira, James Collofello, Dan Shunk, and Gerald Mackulak, "Understanding the Effects of Requirements Volatility in Software Engineering by Using Analytical Modeling and Software Process Simulation", *Journal of Systems and Software*, Vol. 82, Issue 10, pp. 1568-1577, October 2009.
[22] William W. Thomas, Alex Delis, and Victor R. Basili, "An Analysis of Errors in a Reuse-Oriented Development Environment", *Journal of Systems and Software*, pp. 211-224, 1997.
[23] Mark Johnson, "From Engineering to System Engineering to System of Systems Engineering", *IEEE SMC Third International Conference on System of Systems Engineering*, pp. 1-6, 2008.
[24] Per Runeson and Martin Host, "Guidelines for Conducting and Reporting Case Study Research in Software Engineering", *Journal of Empirical Software Engineering*, Vol. 14. pp. 131-164, 2009.

Specifying Class Hierarchies and MOOSE Metrics in Z

Younès El Amrani
LCS laboratory, faculty of Sciences
University Mohamed V-Agdal
Rabat, Morocco
e-mail: elamrani@fsr.ac.ma

***Abstract*—Metrics put into numbers the quality of software's design and contribute to reinforce an organization's software development competitive advantage. Ultimately, an organization would gain impressive benefits in terms of quality, costs, cycle time and productivity in using metrics to quantify software artifacts. Metrics should be formally defined to ensure every stakeholder understands what is measurable in design, and what is actually measured. The formal specification should be easily formulated. A short and concise formal model is introduced in this article and is used to specify the MOOSE metrics suite. The formal specification proposed provides, for the first time, an unambiguous specification of the LCOM metric.**

Keywords-Metric; Design; Quality; Measure; MOOSE; LCOM; Z; Formal Specification.

I. INTRODUCTION

Even though the main concern of this article is to formally specify the MOOSE metrics suite [1], one should put one's mind to understand object-oriented terminology at first place. This article reviews the basic concepts in object-oriented terminology in the formal specification language Z [2]. Z provides not only a means for formulating concise specifications, but also an integrated framework for conducting proofs. One of the advantages of using pure Z is that one is unencumbered by many of the complications evolved in syntax-extensions introduced to reflect object-oriented concepts.

Section 2 is devoted to related works. Section 3 settles notation for expressing object-oriented concepts and reviews those features of object-orientation that will emerge again in Section 4 later on. Many aspects of object-orientation, of which there is abundance, are not covered. Only those that are needed to specify the MOOSE metrics suite [1] are covered. For fuller coverage, the reader is referred to the standard reviews published on the subject, some of which are mentioned in the references, such as [3]. Our main purpose is to set landmarks that will help readers to navigate through the concepts behind all object-oriented design metrics. Section 4 is intended to specify formally the MOOSE metrics suite [1] using Z. The remaining Section 5 is used to conclude and to explore future works that extend this research.

II. RELATED WORKS

It is vitally important to precisely specify the metrics used in software engineering to gain confidence in obtained measurements. Such precision is particularly important since the object-oriented paradigm abounds in terms and concepts. Introducing formalisms into the paradigm is important to the establishment of a sound theoretical foundation for the measurements in software engineering. The reader is referred to existing surveys, such as [3], in the combination of the object-oriented paradigm and formal specification. Three combinations are possible [3]: the first incarnates in a full transformation into an object-oriented language, the second proposes extension to the syntax of the formal language to cope with object-oriented concepts and necessitates to set up a transformational semantic, and finally, the third proposes to specify the system in an object-oriented fashion to keep available the proof system. The model proposed in the Section 3 belongs to the third approach.

Most of the third approach's specifications fall into two basic styles, depending on whether the properties are modeled as functions from identities to property values or modeled by a value in the object state. Hall's style [4-5] falls in the latter approach, whereas France's style [6-7] falls in the former approach. Both styles specify functional properties, called methods or function-members in object-oriented jargon, using schema operation. This common feature in both styles has tremendous consequences on the expressiveness of the specifications. The first limitation is that "there is no way of stating that a subclass must have all the operations of its superclass" [4] and a second limitation, in both styles, is that: "if the methods of different subclasses are in fact different in any way at all, it is not possible to give them the same name in Z" [4]. In the next section, the model presented circumvents these two limitations and empowers the object-oriented paradigm with an expressive formal model. The MOOSE metrics suite [1] is formally specified to illustrate the usefulness of the model. The MOOSE metrics suite [1] is also formally specified using the formal language Z in [8]; the main difference between our specification and [8] is that they used a formal specification of the UML [9] metamodel to express the same set of metrics, whereas in this article a smaller and more concise model is achieving more by adding several object-oriented consistency rules in the inheritance tree specification. However, object-oriented consistency rules are not the main target of this article; the reader is referred to [10] for a devoted article to UML [9] consistency rules using Z and Hall's style. The scientific contribution of this work is to formally define the object-oriented concepts that could not be defined in other models [14], notably the concept of virtual methods. The formal definition of the MOOSE metrics is provided to illustrate how the problem of a formal definition of some metrics, like Lack of Cohesion Metric (LCOM), encountered when using other models [14] is circumvented then overcome in the model presented in this article.

III. Formal Specification of Class Hierarchies in Z

At the heart of formal specification in Z is the ability to introduce new datatypes and to define functions and operations that manipulate their values. Datatypes can be introduced as given sets. The model proposed, uses five given sets: ID is the set of all identifiers, SIGNATURE is used for method's signature. NAME is the set of all names, including methods' names and variables' names. EXPRESSION defines the set of all expressions found in methods' bodies. Finally, the given set TYPE is the set of all variable types in the specification.
[ID, TYPE, SIGNATURE, EXPRESSION, NAME]

Sets can also be defined using Z enumerated sets. Only one enumerated set is used in this model. It is used to specify the concept of visibility for properties (methods and attributes)

Visibility ::= public | private | protected | package

Identifiers' sets for the main set are specified as subsets of ID: the set of all identifiers introduces earlier.

```
│ ClassID, ObjectID, AttributeID, MethodID, PropertyID: ℙ ID
├──────────────────────────
│ ⟨AttributeID, MethodID⟩ partition PropertyID
```

The method and attributes are modeled as Cartesian products. This specification allows different methods to share the same name (this is commonly called operator's overloading)

Method == MethodID × Visibility × NAME × SIGNATURE
Attribute == AttributeID × Visibility × NAME × TYPE

Variable and attribute are the same in the model: they define two syntaxic equivalences.

Variable == Attribute
VariableID == AttributeID

The method's body is itself modeled, the state variables is the set of all the variables used in the method's body. Whereas methods is the set of all the methods called in a given implementation. A set of expressions is defined. To specify a sequential execution, the power set can be replaced by seq EXPRESSION. Finally, the complexity of the method is provided as an instance variable.

```
┌─MethodBody ────────────────────────────
│ variables: ℙ AttributeID
│ calls: ℙ MethodID
│ expressions: ℙ EXPRESSION
│ complexity: ℕ
└────────────────────────────────────────
```

A small set is defined as a return value for get functions. It can easily be replaced by a Boolean set.

YesNo ::= Yes | No

We use a forward declaration for a method that checks whether a method is abstract in a class ancestry. The method's complete definition will be defined later on.

```
│ isMethodAbstractInParentClass: MethodID × ClassID ⇸ YesNo
```

The method getMethodID is used to obtain the method's ID.

```
│ getMethodID: Method → MethodID
├──────────────────────────
│ ∀ method: Method; methodid: MethodID; visibility: Visibility; name:
NAME;
│   signature: SIGNATURE
│     • method = (methodid, visibility, name, signature)
│       ∧ getMethodID method = methodid
```

Now we can define a class. The defined attributes are separated from the inherited attributes (iattributes) as well as the defined methods are separated from the inherited methods (imethods).

```
┌─Class──────────────────────────────────
│ self: ClassID
│ parents: ℙ ClassID
│ children: ℙ ClassID
│ attributes: ℙ Attribute
│ methods: ℙ Method
│ iattributes: ℙ Attribute
│ imethods: ℙ Method
│ isAbstract: YesNo
│ implementation: ℙ (MethodID × MethodBody)
├──────────────────────────
│ ∃ m: methods | getMethodID m ∉ dom implementation • isAbstract =
Yes
│ ∃ m: imethods
│    | isMethodAbstractInParentClass ((getMethodID m), self) = Yes
│      ∧ getMethodID m ∉ dom implementation • isAbstract = Yes
└────────────────────────────────────────
```

The first predicate states that if a defined method (not inherited) has no implementation then the class is abstract: the condition is sufficient. The second predicate states that if a method is abstract in class' ancestry and has not been attributed an implementation, then the class is abstract.

Inheritance is specified with a relation named *inheritFrom*. When a class C_1 inherits from a class C_2, then the pair (C_1,C_2) belongs to *inheritsFrom*. This relation is sematically equivalent to the relation *subSuper* used in Hall's style.

$inheritsFrom: Class \leftrightarrow Class$

$inheritsFrom$
$\quad = \{\ C_1: Class; C_2: Class \mid C_1 \in getClassFromID\ (\ C_2.parents\ ) \bullet (C_1, C_2)\ \}$

Now, the inheritance tree can be formally specified.

InheritanceTree

$children: Class \rightarrowtail \mathbb{P}\ Class$
$parents: Class \rightarrowtail \mathbb{P}\ Class$
$offspring: Class \rightarrowtail \mathbb{P}\ Class$
$ancestry: Class \rightarrowtail \mathbb{P}\ Class$

$\forall\ C: Class$
$\quad \bullet\ children\ C = inheritsFrom\ (\ \{C\}\ )$
$\quad\quad \wedge\ C \notin children\ C$
$\quad\quad \wedge\ children\ C = getClassFromID\ (\ C.children\ )$
$\forall\ C: Class$
$\quad \bullet\ parents\ C = inheritsFrom^{\sim}\ (\ \{C\}\ )$
$\quad\quad \wedge\ C \notin parents\ C$
$\quad\quad \wedge\ parents\ C = getClassFromID\ (\ C.parents\ )$
$\forall\ C: Class \bullet\ offspring\ C = inheritsFrom^{+}\ (\ \{C\}\ ) \wedge C \notin offspring\ C$
$\forall\ C: Class \bullet\ ancestry\ C = (inheritsFrom^{\sim})^{+}\ (\ \{C\}\ ) \wedge C \notin ancestry\ C$

The first and the second predicate use the *inheritsFrom* relation to specify the children and the parents of a class. The third and the fourth predicate define respectively the offspring as the transitive closure of the relation *inheritsFrom* whereas ancestry is the transitive closure of the inverse relation.

Two utility functions named *getAncestryOf* and *getOffspringOf* are formally introduced now. Both functions use relation *inheritsFrom*. This two functions are introduced now because both are used in the next section introducing the formal definition of the MOOSE metrics suite [1].

The first utility function *getAncestryOf* returns the ancestry of the class provided as input.

The method *isMethodAbstractInParentClass*, previously declared, can now be defined (Z does not allow using a function before declaring it) The definition is provided in the second predicate following the first predicate which defines the function *getAncestryOf*.

$\mathrm{getAncestryOf: Class} \rightarrow \mathbb{P}\ \mathrm{Class}$

$\forall\ \mathrm{C: Class} \bullet\ \mathrm{getAncestryOf\ C = inheritsFrom}^{+}\ (\ \{\mathrm{C}\}\ )$
$\mathrm{isMethodAbstractInParentClass}$
$\quad = \{\ \mathrm{mid: MethodID; Cid: ClassID; C: Class; ancestry:}\ \mathbb{P}\ \mathrm{Class}$
$\quad\quad \bullet\ \mathbf{if}\ \mathrm{C = getClassFromID\ Cid}$
$\quad\quad\quad \wedge\ \mathrm{ancestry = getAncestryOf\ C}$
$\quad\quad\quad \wedge\ \mathrm{mid} \notin \bigcup\{\ \mathrm{C_1: ancestry} \bullet\ (\mathrm{dom\ C_1.implementation})\ \}$
$\quad\quad \mathbf{then}\ (\mathrm{mid, Cid}) \mapsto \mathrm{Yes}$
$\quad\quad \mathbf{else}\ (\mathrm{mid, Cid}) \mapsto \mathrm{No}\ \}$

The second utility function *getOffspringOf* returns the offspring of the class provided as input.

$\mathrm{getOffspringOf: Class} \rightarrow \mathbb{P}\ \mathrm{Class}$

$\forall\ \mathrm{C: Class} \bullet\ \mathrm{getOffspringOf\ C = (inheritsFrom}^{\sim})^{+}\ (\ \{\mathrm{C}\}\ )$

The formal specification of the MOOSE metrics suite [1] is now illustrated in Section 4.

IV. Formal Specification of the MOOSE Metrics Suite

The MOOSE Metrics Suite defines a set of six metrics: NOC is the total number of children in a class, DIT measures the depth of the inheritance tree, LCOM measures the lack of cohesion in the set of methods in a class, RFC measures the response for a class, WMC is the weighted methods per class it measures the complexity of the set of methods of the class, finally CBO measures the coupling between object. In the subsequent subsection, we define formally and precisely this set of metrics.

A. The NOC metric

The NOC metric is defined informally as the number of children of a given class. Its formal definition is straightforward in the model introduced in section 3.

$NOC: Class \rightarrow \mathbb{N}$

$\forall\ C: Class \bullet\ NOC\ C = \#\ C.children$

B. The DIT metric

The DIT metric is defined informally as the longest path from the input class to the inheritance tree root. Firstly the formal specification of the set of all paths leading to the root is provided, secondly the maximum length is specified and that is DIT.

A function *isRoot* is used to check if a class is a root. A class is a root when is has no parent.

$isRoot: Class \rightarrowtail YesNo$

$\forall\ C: Class \bullet\ \mathbf{if}\ C.parents = \varnothing\ \mathbf{then}\ isRoot\ C = Yes\ \mathbf{else}\ isRoot\ C = No$

The function *getClassFromID* returns the class associated to the input *ClassID*

$$getClassFromID: ClassID \rightarrowtail Class$$

The function *allPathLengthToRoot* returns the set of all paths to root.

$$\begin{array}{l}
allPathLengthToRoot: Class \rightarrow \mathbb{P}\,\mathbb{N} \\
\hline
\forall\, C: Class \\
\quad \bullet\ allPathLengthToRoot\ C \\
\qquad = \{\ path: \mathrm{seq}\ Class;\ i: \mathbb{N} \\
\qquad\quad \mid\ i \in 1 \mathrel{..} \#\,path \\
\qquad\quad \wedge\ path\ 1 = C \\
\qquad\quad \wedge\ path\ i \in getClassFromID\ (\,(path\ (i - 1)).parents\,) \\
\qquad\quad \wedge\ isRoot\ (last\ path) = Yes \bullet\ \#\,path\ \}
\end{array}$$

The function *maxi* returns the maximum of a set of Integers.

$$\begin{array}{l}
maxi: \mathbb{P}\,\mathbb{N} \rightarrow \mathbb{N} \\
\hline
\forall\, I: \mathbb{P}\,\mathbb{N} \bullet\ \forall\, n: I \bullet\ \exists_1\, m: I \mid\ m > n \bullet\ maxi\ I = m
\end{array}$$

The DIT metric is now straightforward to define formally: it is the longest path to root.

$$\begin{array}{l}
DIT: Class \rightarrow \mathbb{N} \\
\hline
\forall\, C: Class \bullet\ DIT\ C = maxi\ (allPathLengthToRoot\ C)
\end{array}$$

C. The LCOM metric

The LCOM metric comes also easily, the two sets P and Q defined in [1] are formally specified as follow:

$$\begin{array}{l}
P: Class \nrightarrow \mathbb{P}\,(Method \times Method) \\
\hline
\forall\, C: Class \\
\quad \bullet\ P\ C \\
\qquad = \{\ m_1: C.methods;\ m_2: C.methods \\
\qquad\quad \mid\ (C.implementation\ (getMethodID\ m_1)).variables \\
\qquad\qquad \cap\ (C.implementation\ (getMethodID\ m_2)).variables = \emptyset \\
\qquad\quad \bullet\ (m_1, m_2)\ \}
\end{array}$$

P is the set of all methods couples that do not use any variable (attribute) in common.

$$\begin{array}{l}
Q: Class \nrightarrow \mathbb{P}\,(Method \times Method) \\
\hline
\forall\, C: Class \\
\quad \bullet\ Q\ C \\
\qquad = \{\ m_1: C.methods;\ m_2: C.methods \\
\qquad\quad \mid\ (C.implementation\ (getMethodID\ m_1)).variables \\
\qquad\qquad \cap\ (C.implementation\ (getMethodID\ m_2)).variables \neq \emptyset \\
\qquad\quad \bullet\ (m_1, m_2)\ \}
\end{array}$$

Q is the set of methods couples which implementations have some attributes in common.

LCOM is equal to zero if there a more couples in Q than in C, otherwise it is equal to the difference between the two.

$$\begin{array}{l}
LCOM: Class \nrightarrow \mathbb{N} \\
\hline
\forall\, C: Class \\
\quad \bullet\ \mathbf{if}\ \#\,(Q\ C) > \#\,(P\ C)\ \mathbf{then}\ LCOM\ C = 0\ \mathbf{else}\ LCOM\ C = \#\,(P\ C) - \#\,(Q\ C)
\end{array}$$

D. The RFC metric

The RFC metric computes how many different calls can occur as a response to a message received by a class. Of course defined methods and inherited methods are counted and added to the number of different calls that occur in implementations.

$$\begin{array}{l}
RFC: Class \rightarrow \mathbb{N} \\
\hline
\forall\, C: Class \\
\quad \bullet\ RFC\ C \\
\qquad = \#\,C.methods + \#\,C.imethods \\
\qquad\quad + \#\,(\bigcup\,\{\ mob: MethodBody \\
\qquad\qquad \mid\ mob \in C.implementation\ (\,getMethodID\ (\,C.methods\,)\,) \\
\qquad\qquad \bullet\ mob.methods\ \}) \\
\qquad\quad + \#\,(\bigcup\,\{\ imob: MethodBody \\
\qquad\qquad \mid\ imob \in C.implementation\ (\,getMethodID\ (\,C.imethods\,)\,) \\
\qquad\qquad \bullet\ imob.methods\ \})
\end{array}$$

E. The WMC metric

The WMC metric computes the sum of methods complexities. The method complexity is a state variable of the method's body. A function that sums all the complexities of a set of method's bodies is defined and is used to compute the complexity of a class.

$$\begin{array}{l}
sumComplexity: \mathbb{P}\,MethodBody \rightarrow \mathbb{R} \\
\hline
\forall\, B: \mathbb{P}\,MethodBody \\
\quad \bullet\ \mathbf{if}\ B = \emptyset \\
\qquad \mathbf{then}\ sumComplexity\ B = 0 \\
\qquad \mathbf{else}\ \exists\, mb: B \\
\qquad\quad \bullet\ sumComplexity\ B = mb.complexity + sumComplexity\ (B \setminus \{mb\})
\end{array}$$

$$\begin{array}{l}
WMC: Class \rightarrow \mathbb{R} \\
\hline
\forall\, C: Class \\
\quad \bullet\ \exists\, B: \mathbb{P}\,MethodBody \mid\ B = C.implementation\ (\,getMethodID\ (\,C.methods\,)\,) \\
\qquad \bullet\ \mathbf{if}\ B = \emptyset \\
\qquad\quad \mathbf{then}\ WMC\ C = 0 \\
\qquad\quad \mathbf{else}\ \exists\, mob: B \bullet\ WMC\ C = mob.complexity + sumComplexity\ (B \setminus \{mob\})
\end{array}$$

F. *The CBO metric*

The remaining metric from the MOOSE metrics suite [1] is the CBO metric. This metric computes the coupling of a class with all the other classes of a provided design.

First, the function useMethods is defined. It has the value Yes if at least a method of one class uses one methods of the other class:

```
$useMethods: Class \times Class \rightarrow YesNo$
────────────────────
$\forall C_1: Class; C_2: Class$
  $\bullet$ if $\mathrm{dom}\ C_1.implementation \cap (getMethodID\ (\ C_2.methods\ )) \neq \emptyset$
    then $useMethods\ (C_1, C_2) = Yes$
    else $useMethods\ (C_1, C_2) = No$
```

Second, the function useVariables is defined. It has the value Yes if at least a method of one class uses some attributes of the other class:

```
$getAttributeID: Attribute \rightarrow AttributeID$
────────────────────
$\forall attribute: Attribute; attributeid: AttributeID; visibility: Visibility;$
  $name: NAME; type: TYPE$
  $\bullet\ attribute = (attributeid, visibility, name, type)$
    $\wedge\ getAttributeID\ attribute = attributeid$
```

```
$useVariables: Class \times Class \rightarrow YesNo$
────────────────────
$\forall C_1: Class; C_2: Class$
  $\bullet$ if $\bigcup \{\ mob: MethodBody$
              $\mid mob \in C_1.implementation\ (\ getMethodID\ (\ C_1.methods\ )\ )$
              $\bullet\ mob.variables\ \}$
         $\cap\ (getAttributeID\ (\ C_2.attributes\ )) \neq \emptyset$
    then $useVariables\ (C_1, C_2) = Yes$
    else $useVariables\ (C_1, C_2) = No$
```

Then, the coupling between two classes is defined:

```
$CBO_1: Class \times Class \rightarrow \{0, 1\}$
────────────────────
$\forall C_1: Class; C_2: Class$
  $\bullet$ if $useVariables\ (C_1, C_2) = Yes$
      $\vee\ useVariables\ (C_2, C_1) = Yes$
      $\vee\ useMethods\ (C_1, C_2) = Yes$
      $\vee\ useMethods\ (C_2, C_1) = Yes$
    then $CBO_1\ (C_1, C_2) = 1$
    else $CBO_1\ (C_1, C_2) = 0$
```

An object oriented design is formally specified as a set of classes.

$Design == \mathbb{P}\ Class$

And the coupling metric for a class is the sum of all coupling with other classes except the class itself.

```
$CBO: Class \times Design \rightarrow \mathbb{N}$
────────────────────
$\forall C: Class; design: Design$
  $\bullet$ if $design \setminus \{C\} = \emptyset \vee C \notin design$
    then $CBO\ (C, design) = 0$
    else $\exists C_1: design \setminus \{C\}$
         $\bullet\ CBO\ (C, design) = CBO_1\ (C, C_1) + CBO\ (C, (design \setminus \{C_1\}))$
```

All presented specifications have been thoroughly checked using the Z/EVES [12] system.

V. Conclusion and Future Work

This article provided a formal specification for object-oriented concepts and illustrated the power of the proposed specification by providing a complete and formal definition of the MOOSE metrics suite [7]. A formal definition of the MOOD metrics suite [11] and others metrics can be specified with the model presented in Section 3. Additional object-oriented consistency rules can be specified by adding predicates in the inheritance tree. Concepts like the overriding in object-oriented paradigm can easily be specified with this framework. There are many object-oriented concepts that could be clarified and put in a clear mathematical predicate along the road. All the specifications presented in this article have been thoroughly tested using the Z/EVES [12] system. Because of its importance to the subsequent development of software engineering, the proposed formal specification of MOOSE metrics should be extended, in future work, to the set of metrics reviewed in [13].

[1] Chidamber S.R. and Kemerer, C.F.: A metric suite for Object Oriented Design. J. Trans. on Soft. Eng. vol. 20. IEEE Press, New York (1994)

[2] Spivey, J.M.: The Z Notation: A Reference Manual. Prentice Hall International, Oxford (1998)

[3] Ruiz-Delgado, A., Pitt, D., Smythe, C.: A Review of Object-oriented Approaches in Formal Methods. J. Comp. vol. 38, pp. 777-784 (1995)

[4] Hall, J.A.: Specifying and Interpreting Class Hierarchies in Z. In: Bowen J.P., Hall J.A. (eds.) Cambridge 1994. Z User Workshop, pp. 120-138. Springer, New York (1994)

[5] Hall, J.A.: Using Z as a Specification Calculus for Object-Oriented Systems. In: Bjorner, D., Hoare, C.A.R., Langmaack, H. (eds.) VDM and Z, Third International Symposium on VDM Europe Kiel, 1990. LNCS, vol. 428, pp. 290-318. Springer, Heidelberg (1990)

[6] France, R.B., Bruel, J.M., Larrondo-Petrie, M.M., Shroff, M.: Exploring the Semantics of UML Type Structures with Z. In: Proceedings of the Formal Methods for Open Object-based Distributed Systems. FMOODS, pp. 247-257. Springer, New York (1997)

[7] Shroff, M., France, R.B.: Towards a Formalization of UML Class Structures in Z. In: 21th Computer Software and Application. COMPSAC, pp. 646-651. IEEE Press, New York (1997)

[8] Lamrani, M., El Amrani, Y., Ettouhami, A.: Formal Specification of Software Design Metrics. In: Sixth International Conference on Software Engineering Advances. Barcelona (2011)

[9] The Object Management Group: UML 2.3 superstructure specification. http://www.omg.org/spec/uml/ (09/11/2012)

[10] El Miloudi, K., El Amrani, Y., Ettouhami, A.: An Automated Translation of UML Class Diagrams into a Formal Specification to

Detect UML Inconsistencies. In: Sixth International Conference on Software Engineering Advances. Barcelona (2011)

[11] Abreu, F.B.: The MOOD Metrics Set. In: Workshop on Metrics, ECOOP. Aarhus (1995)

[12] 12. Saaltink, M.: The Z/EVES System. In: Bowen, J.P., Hinchey, M.G., Hill, D. (eds.) Ten International Conference of Z Users Reading 1997. LNCS, vol. 1212, pp. 72-85. Springer, Heidelberg (1990)

[13] Xenos, M., Stavrinoudis, D., Zikouli, K., Christodoulakis, D.: Object Oriented Metrics: A Survey. In: Proceedings of the Federation of European Software Measurement Association. FESMA 2000. Madrid (2000)

[14] Wieringa, R.: A Survey of Structured and Object-Oriented Software Specification Methods and Techniques. In: ACM Computing Surveys. Vol. 30, No. 4, pp. 459-527, New-York (1998) doi=10.1145/299917.299919

Applying Algebraic Specification To Cloud Computing

-- A Case Study of Infrastructure-as-a-Service GoGrid

Dongmei Liu
School of Computer Science and Technology
Nanjing University of Science and Technology
Nanjing, 210094, P.R. China
Email:dmliukz@njust.edu.cn

Hong Zhu and Ian Bayley
Dept of Computing and Communication Technologies
Oxford Brookes University
Oxford, OX33 1HX, UK
Email:hzhu@brookes.ac.uk, ibayley@brookes.ac.uk

Abstract— **Cloud Computing has attracted attention from both the research community and the industry. It is highly desirable to specify the syntax and semantics of the services precisely and accurately without giving away any design and implementation details. This challenge is even greater for cloud services based on RESTful web services techniques, where the invocation is through HTTP queries and there is no agreed standard exists for their specifications. In this paper, we propose an algebraic approach and apply it, as a case study, to the GoGrid Cloud Computing API. Not only does this give a formal unambiguous specification that is easy to write and understand, but it also identifies and eliminates errors in the existing documentation.**

Keywords-cloud computing; formal specification; algebraic specification; RESTful Web services, Infrastructure-as-a-Service

I. INTRODUCTION

Precise and accurate documentation of software systems has long been a challenge to the software engineering communities. The advent of cloud computing, and other forms of service-oriented computing, has raised the demands further, since software engineers, when developing their applications, have to depend solely on documents of the services provided by the cloud. Moreover, when services are dynamically discovered and composed at runtime, the specifications of the services must be machine readable, in the senses of both syntax and semantics.

To meet the challenges of software specifications, formal methods have been developed in the past forty years and have advanced significantly [1]. However, their application to services thus far has been limited, restricted to ontology definition languages and business process description languages, such as the Business Process Execution Language BPEL [2].

A formal specification technique for services must satisfy two requirements. First, it must be uniformly applicable to each of the various levels of services: Infrastructure, Platform and Software as a service. Secondly, it must be flexible enough to support dynamic discovery and composition of services without revealing vendor-specific design and implementation details.

This paper explores the applicability of algebraic specification to RESTful web services [3], which is widely employed by cloud service providers.

A. *Related works*

Many cloud services provide an application programming interface (API) with which their customers can dynamically configure, manage and use their resources through a programmatic interface. For Infrastructure-as-a-Service (IaaS), the resources are hardware entities, such as servers and load balancers, etc. For Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS), the resources are platform and software entities respectively. Examples include virtual machines, network middleware, databases, software components, etc.

The current practice is to define the API informally with an open specification. The API is accessed through the RESTful web service protocol. In contrast to traditional SOAP-based web services [4], no agreed standards exist for describing RESTful services, either at the semantic level or the syntactic level. Documentation is often in natural language, leaving space for ambiguity and for errors in the definition of the services. The formal specification of RESTful web services is still an open problem. It is highly desirable that the API specification is formalized to reduce ambiguity, redundancy and inconsistency to the minimum, whilst still being easy to understand and requiring minimal training.

Current research on the description of RESTful web services is mostly focused on formats for annotating the syntax and semantics of RESTful web services. The most well-known such efforts include WADL [5], hRESTS/ MicroWSMO [6], and SA-REST [7]. They describe the syntax and data types of the input and output as well as the operations of the WS using a machine readable format in XML or HTML. The main problem of these approaches is that they rely heavily on the RPC-based operation model, which does not align well with the principles of RESTful web service. Moreover, the semantics are described at the level of ontology, rather than the effect on the states of the resources that the services operate on. In [8], Liskin et al. proposed an extension to UML state machine diagram to describe in graphic notation how RESTful web services changes the states of the resources. However, it is open to dispute whether graphic notations can capture the full complexity of the state changes involved in resources managed and manipulated by these services.

Algebraic specification was first proposed in the 1970s as an implementation-independent specification technique for abstract data types [9][10]. Since then, it has been extended to concurrent systems, state-based systems and software components, all by applying theories of behavioural algebras [11] and co-algebras [12][13][14][15]. The specifications produced are at a high level of abstraction, completely independent of any implementation detail. Properties can be proven for these specifications, they can be refined to implementations, and implementations can be proven correct with respect to the specifications [16], etc.

A particularly attractive feature of algebraic specifications is that they can be used directly by automated software testing tools [17][18]. This is particularly important when services bind dynamically since testing must be done on-the-fly.

B. *Main Contributions of the paper*

This paper reports a case study of an algebraic specification of *GoGrid* [19], a real industrial-strength system that provides infrastructure-as-a-service. GoGrid provides an API, defined by an open specification [21] and accessed through a RESTful interface. The specification language used in the case study is CASOCC-WS, which we proposed in [20] for the specification of SOAP-based web services. By successfully specifying every operation of the GoGrid API, we demonstrate that CASOCC-WS can be used for RESTful Web services too.

During the formalization, we detected non-trivial errors in GoGrid's documentation. These errors included ambiguity, inconsistency and also incompleteness. This case study shows, therefore, that formal specification can improve the precision and accuracy of service documentation.

It shows too that algebraic specifications can be abstract and implementation independent, since the GoGrid API, like many other RESTful web services, supports multiple programming languages, such as Java, Ruby, Python, C#, as well as shell script languages such as Bash.

Finally, our case study also demonstrates that algebraic specifications can be easy to understand with minimal training, confirming the findings in [22].

To our knowledge, there is no similar work in the literature on the algebraic specification of web services, nor on RESTful web services in particular.

C. *Organisation of the paper*

The remainder of this paper is organized as follows. Section II briefly outlines the algebraic specification language CASOCC-WS. In Section III, we use GoGrid API as a case study to demonstrate how the algebraic approach can be used to specify cloud computing interface. Section IV discusses some of the benefits of applying algebraic approach on cloud computing interface. Section V concludes the paper and discusses possible future work.

II. ALGEBRAIC SPECIFICATION LANGUAGE CASOCC-WS

The CASOCC-WS language is an extension of the CASOCC language [17][18]. The specifications in CASOCC-WS are modular. A specification is built from a number of units, one for each software entity in the system. A software entity can be an abstract data type, a class, a component, a web service, and so on. A specification has the following syntactic form, in a variant of BNF:

```
<Specification> ::= {<Spec Unit>}
<Spec Unit> ::=
    Spec <Sort Name> [<Observability>];  <Signature> [<Axioms>]
    End
<Sort Name> ::= <Identifier>
<Observability> ::=
     is observable by <Operator ID> | is unobservable
<Operator ID> ::= <Identifier>
```

Each specification unit contains two main parts: a *signature* and a set of *axioms*. The <Sort Name> is an identifier that names the main sort of the unit. Observability is an important property of software entities. A software entity is directly observable if its state or value can be tested for equality; otherwise, its state or value has to be checked by other means, e.g. through observers. The operator for an observable entity must be a Boolean function.

A. *Signature*

The signature specifies the syntactic aspect of the software entity. A signature has the following syntactic form:

```
<Signature> ::=  [<Imported Sorts>;] <Operations>
<Imported Sorts> ::= Sort <Imported Sort List>
<Imported Sort List> ::=  <Sort Name>[, <Imported Sort List>]
<Operations> ::=
    Operators:  [<Creators>;][<Transformers>;][<Observers>;]
<Creators> ::= Creator: <OpList>
<Transformers> ::= Transformer: <OpList>
<Observers> ::= Observer: <OpList>
<OpList> ::= <Operation> [; <OpList>]
<Operation> ::= <Operator ID> :['['<Context Sort>']']
        [<Domain Type>] -> <Co-domain Type>
<Context Sort> ::= <Sort Name>
<Domain Type> ::= <Type> | VOID
<Co-domain Type> ::= <Type> | VOID
<Type> ::= <Sort Name> [, <Type>]
```

The *Imported Sorts* clause is a comma-separated list of the sorts upon which the sort currently being specified depends. The operators defined for a sort are classified into creator, transformer and observer. Take STACK for example, its signature is as follows.

```
Spec STACK;
    Sort BOOL, NAT;
        Operators:
            Creator:        newStack:  -> STACK;
            Transformer:    push: STACK, NAT -> STACK;
                            pop: STACK -> STACK;
            Observer:       isNewStack: STACK -> BOOL;
                            top: STACK -> NAT;
End
```

This means that STACK depends on BOOL for Boolean values and NAT for natural numbers. newStack is a creator, push and pop are transformers, isNewStack and top are observers.

Note that, in a traditional algebraic specification language, the co-domain of an operator must be a singleton. Such a signature is called *algebraic*; STACK has such a signature. More recent languages, based on co-algebras, require instead the domain to be singleton; such signatures are called *co-algebraic*, and can be used.

CASOCC-WS extends the algebraic and co-algebraic approaches by allowing both the domain and the co-domain of an operator to be non-singleton at the same time. This makes it possible to specify stateful services naturally. For example, infinite streams of natural numbers are specified as follows. Each application of the operator next to a stream will give a natural number and change the state of the stream.

```
Spec STREAM is unobservable;
    Sort NAT;
        Operators:
            Transformer:    next: STREAM -> STREAM, NAT;
End
```

In general, when the main sort of the unit occurs in both the domain and the co-domain of an operator, we call it the *context sort* of the operator. In such a case, CASOCC-WS use the following format to indicate the context sort, while omitting it from the domain and co-domain.

$$Op : [s]\ s_1, \dots , s_n \rightarrow s'_1, \dots , s'_k,$$

where s is the context sort.

If the main sort is the only sort in an operator's domain or co-domain, we write VOID for the type of the latter. For example, the signature of the operator push of STACK and the operator next of STREAM can now be specified respectively as follows.

```
    push: [STACK] NAT -> VOID;
    next: [STREAM] VOID -> NAT;
```

B. Axiom

Each specification unit consists of logical axioms describing the properties that functions are required to satisfy. An axiom consists of a variable declarations block and a list of conditional equations.

```
<Axioms> ::= Axiom: <Axiom List>
<Axiom List> ::= <Axiom> [<Axiom List>]
<Axiom> ::= <Var Declarations> <Equations> End
<Equations> ::= <Equation> [<Equations>]
```

1) Variable declarations

Variable declarations declare a list of variables and their types. Variables are declared "globally" to all equations in the axiom using "For all" keywords.

```
<Var Declarations> ::= For all <Var-Sort Pairs> that
<Var-Sort Pairs> ::=  <Var IDs> : <Sort Name> [, <Var-Sort Pairs>]
<Var IDs> ::= <Var ID> [, <Var IDs>]
<Var ID> ::= <Identifier>
```

where the sort name can only be the main sort or a sort listed in the imported sorts clause. The variable identifiers must be unique: they must not clash with sort names, operator names nor with any such names in any sorts imported and other variables in this axiom.

2) Equation

Equations declare a list of conditional equations. The syntax rule of an equation is as follows.

```
<Equation> ::=  [<Label ID>:] <Condition> [, if <Conditions>];
                 | Let <Var Definitions> in <Equations> End
<Label ID> ::= <Identifier>
<Conditions> ::= <Condition> [(, | or) <Conditions>]
<Condition> ::=  <Bool Term> | <Term> <Relation OP> <Term>
<Bool Term> ::= True | False
<Relation OP> ::=  "==" | "<>" | ">" | "<" | ">=" | "<="
<Var Definitions> ::=  <Var Assignment> [, <Var Definitions>]
<Var Assignment> ::= <Var ID> = <Term>
```

The most basic form of an equation is thus $t_1 == t_2$. Here is an example of sort STACK, assuming that sort BOOL is predefined.

```
For all s: STACK, n: NAT that
     isNewStack(push(s,n)) == False;
    pop(push(s, n)) == s;
    top(push(s, n)) == n;
End
```

The second syntax rule for equations is designed to allow *local variable* definitions in the form

Let $x_1 = \tau_1, \dots, x_n = \tau_n$ ***in*** *equs* ***End***

where $x_1, \dots, x_n$ are local variables, limited in scope to *equs*, and $\tau_1, \dots, \tau_n$ are terms denoting the values that are assigned to the variables. Local variables must have unique names, not clashing with other variables in this equation and any other names, just as with global variables. The above example can be specified as follows.

```
For all s: STACK, n: NAT that
    Let s1 = push(s,n)  in
        isNewStack(s1) == False;
        pop(s1) == s;
        top(s1) == n;
    End
End
```

3) Term

A term is constructed from constants and variables by the application of operators. All names used in terms may be qualified with the intended type and the intended sort of the term may be specified. In particular, a term is called *ground* term if it contains no variable. The syntax rules for term are as follows.

```
<Term> ::=   <Var ID> | "(" <Term> ")" | "<" <Term List> ">"
    | <Operator ID> ["(" [<Parameters>] ")"]
    | "[" <Term>  "]" | <Term> "." <Term>  | NULL
    | <Term> "#" <Term> | <numeric_expression>
    | <string_expression> | <literal_expression>
<Parameters> ::= <Term List>
<Term List> ::= <Term> [, <Term List>]
<numeric_expression> ::= <Term> <Algorithm OP> <Term>
<string_expression> ::= <Term> ("+"|"+=") <Term>
<literal_expression> ::= <integer_literal>  | <float_literal>
    | <string_literal> | <character_literal>
<Algorithm OP> ::= "+" | "-" | "*" |  "/"
```

Any operator in a term must either be declared in the signature part of the sort being specified or in the signature of an imported sort. For example, if *s* is a variable of the STACK sort, and *m* and *n* are variables of the NAT sort, then the following are STACK-terms of the STACK sort.

```
push(s, n)
push(push(s,n),m)
pop(push(push(s,n),m))
pop(push(pop(push(s,n)),m))
```

Note that when an operator φ is declared in the form φ: $[s]s_1 \rightarrow s_2$ using a sort *s* as the context, the type of a term like φ(τ) is s_2, rather than (s, s_2). The new context state in the sort *s* after applying the operator φ to τ is given by the expression [φ(τ)]. For example, let NatSt: STREAM be an infinite stream of natural numbers. Then NatSt.next is the natural number at the front of the stream and [NatSt.next] is state of the stream after the next operation; i.e., the stream after taken the front number away.

III. Case Study

In this section, we specify GoGrid API in CASOCC-WS as a case study.

GoGrid is the world's largest pure-play Infrastructure-as-a-Service (IaaS) provider specializing in Cloud infrastructure solutions. It provides an API, defined by an open specification, with which its customers can deploy and manage their applications and workloads through a programmatic interface.

A. GoGrid API

The GoGrid API is a REST-like query interface. RESTful web services, unlike SOAP/WSDL, are based on the HTTP protocol, so each GoGrid API call is an individual HTTP query. For HTTP GET calls, the input data are passed via the query string. For HTTP POST calls, the input data are passed in the request body, which is URL-encoded. Only GET and POST are used in GoGrid API. The server responds to each request by changing the internal state of the service if need be and by returning a message to the service requester.

The latest version of GoGrid API (version 1.8) has 11 different types of objects and 5 types of common operators. Some of the operators are not applicable to some types of objects. There are 3 types of objects that are only used as parameters of the operators, so no operators are applicable on them, while some objects have special operators. **Table 1** gives the applicable operators for each type of object.

Table 1. Applicable Operators on Objects

Object	List	Get	Add	Delete	Edit	Other Ops
Server	Yes	Yes	Yes	Yes	Yes	Power
Server image	Yes	Yes		Yes	Yes	Save, Restore
Load Balancer	Yes	Yes	Yes	Yes	Yes	
Job	Yes	Yes				
IP	Yes					
Password	Yes	Yes				
Billing		Yes				
Option	Yes					

It is worth noting that some operators have different meanings for different types of object, so in our specification of GoGrid, the definitions were grouped by object rather than by operator. For each object, we start by specifying the requests and responses of the operations, defining their structures and the constraints on the values of the elements.

The requests (or responses) for one operator are specified in one specification unit. But, there may be a number of other specification units that specify the elements in the structure of the requests (or responses). Then, we specify the semantics of the operators on the type of objects by defining the relationships between the requests and the responses. Note that, the internal states of an object that the operator changes cannot be observed directly. They can only be observed by applying observers, which are API requests, too. The set of operators for one type of object is specified in one specification unit, but there may be auxiliary specification units, such as for lists of objects, as found in the responses to some operators.

Here, we only give the details of the specification of the operators applicable to the object type Server. This is the most important object of the system and also the most complicated to specify. Other operators are similar but less complex.

B. Requests and Responses

1) Common query parameters in requests

There are four query parameters common to all GoGrid API calls, and they are specified as follows:

```
Spec CommonQueryParameter ;
    Operators:
        Observer:
            api_key, sig, v, format:
                CommonQueryParameter -> string;
     Axiom:
        For all CQP: CommonQueryParameter that
            CQP.api_key <> NULL;
            CQP.sig <> NULL;
            CQP.v <> NULL;
        End
End
```

where api_key is a key generated by GoGrid for security in the access of resources, sig is an MD5 [23] signature of the API request data, v is the version id of the API, and format is an optional field to indicate the response format required. NULL is a value that represents no information. The signature can be generated by an MD5 hash from the api_key, which is obtained before API calls can be made, the user's shared secret, which is a string of characters set by the user and known only by the GoGrid server, and a Unix timestamp, which is the number of seconds since the Unix Epoch of the time when the request was made. The api_key and shared secret act as an authentication mechanism.

However, because the signature is time-dependent, and therefore, also dependent on the context, the relationship between these query parameters cannot be specified without the context of the request. So, the axiom part of the specification states only that these parts cannot be omitted. We specify the authentication mechanism later in the systematic specification.

2) Request of the List operator

In addition to the parameters common to all requests, each type of request also contains variable parts. Below, we only give the specification of the requests of the List operation as an example. A *server list* call returns a list of server objects of a certain type in the cloud.

```
Spec ServerListRequest;
    Sort CommonQueryParameter, ListofString;
    Operators:
        Observer:
            para: ServerListRequest -> CommonQueryParameter;
            num_items, page, timestamp: ServerListRequest -> int;
            server_type, datacenter: ServerListRequest -> string;
            isSandbox: ServerListRequest -> boolean;
    Axiom:
        For all SLR: ServerListRequest that
            SLR.num_items >=0;
            SLR.page >=0, if SLR.num_items > 0;
        End
End
```

where para is the common query parameters defined above. num_items is the number of items to return. Its value is used to paginate the results into a number of pages so that each page contains num_items number of items. page is the index of the page to be returned when the results are paginated. The index starts from 0. This parameter is ignored if num_items is not specified. server_type, isSandbox, and datacenter are used to filter server objects. timestamp is used in authentication.

3) Responses to the List Operation

The GoGrid API responses can be in three different formats: *JSON* (JavaScript Object Notation), *XML*, and *CSV* (Comma Separated Values). The default format, used when the optional *format* parameter is omitted, is JSON. However, one benefit of using algebraic specification is that we need only one formal specification for all output formats.

The response to a list call contains the response status, request method, summary of the list and a list of returned objects. The summary part of the responses can be specified as follows:

```
Spec ListResSummary;
    Operators:
        Observer:
            Total, start,
returned, numpages:
                ListResSummary -> int;
    Axiom:
        For all LRS: ListResSummary that
            LRS.total >= 0;
            LRS.start >= 0;
            LRS.returned >= 0;
        LRS.numpages >= 0;
    End
End
```

where total is the total number of objects in the list; start is the current start index for this list of objects; returned is the number of objects returned in this list; and numpages is the total number of pages available given the num-items value in the request.

The structure of the responses of the list operator when applied to server object can be specified as follows.

```
Spec ServerListResponse;
    Sort ListofServer, ListResSummary, ListofString;
    Operators:
        Observer:
            Status, request_method: ServerListResponse -> string;
            summary: ServerListResponse -> ListResSummary;
            objects: ServerListResponse -> ListofServer;
            statusCode: ServerListResponse -> int;
    Axiom:
        For all SLR: ServerListResponse that
            SLR.request_method == "/grid/server/list";
        End
        For all SLR: ServerListResponse, i, j: int that
            SLR.objects.items(i).id <> SLR.objects.items(j).id,
                if   status == "success", i <> j,
                     0<= i, i <= SLR.summary.returned,
                     0<= j, j <= SLR.summary.returned;
        End
        For all SLR: ServerListResponse, i: int,
                X: ServerListRequest that
    search(X.datacenter,SLR.objects.items(i).datacenter.name)
        == True,
                if   status == "success",
                     0<= i, i <= SLR.summary.returned,
                     X.datacenter.length > 0;
            SLR.objects.items(i).type.name == X.server_type,
                if   status == "success",
                     0<= i, i <= SLR.summary.returned,
                     X.server_type <> NULL;
            SLR.objects.items(i).isSandbox == X.isSandbox,
                if   status == "success",
                     0<= i, i <= SLR.summary.returned,
                     X.isSandbox <> NULL;
        End
End
```

where search is an auxiliary function of the type ListofString, string -> Boolean.

In addition to status, request method, summary of the list and a list of returned objects, each response will contain a status code: 200 means that the call is successful, and 4xx means there is an error in the client's request, of which 400 means the argument is illegal, 401 means unauthorised, 403 means authentication failed, and 404 means not found. A status code of 5xx means that a server error occurred.

C. Semantics of the operations

For each type of request, we define an operator that takes common query parameters and various typed parts as input and produces a response as the output. All such operators have GoGrid as the context. Some are transformers, such as Add, Delete and Edit; some are observers, such as List and Get. We also need to know the clock time on the grid and also the shared secret chosen by each user for checking the authentication of access. Also we define some auxiliary functions. Thus, we have the following signature for the sort ServerGoGrid, which represents the Server web service of GoGrid cloud computing system.

```
Spec ServerGoGrid;
    Sort      Server, ListofServer,
              ServerListRequest, ServerListResponse,
              ServerAddRequest, ServerAddResponse, …
    Operators:
        Observer:
            clockTime: -> int;
            sharedSecret: string -> string;
            List: [ServerGoGrid]
                ServerListRequest -> ServerListResponse;
            Get: [ServerGoGrid]
                ServerGetRequest -> ServerGetResponse;
        Transformer:
            Add: [ServerGoGrid]
                ServerAddRequest -> ServerAddResponse;
            Delete: [ServerGoGrid]
                ServerDeleteRequest -> ServerDeleteResponse;
            Edit: [ServerGoGrid]
                ServerEditRequest -> ServerEditResponse;
            Power: [ServerGrid]
                ServerPowerRequest -> ServerPowerResponse;

    Axiom
    …
End
```

where the following auxiliary functions are used.

```
        MD5: string, string, int -> string ;
        abs: int -> int;
        insert: [ListofServer]ListofServer -> VOID;
        remove: [ListofServer]ListofServer -> VOID;
        update: [ListofServer]ServerPowerRequest -> VOID;
```

For each operator, its semantics can be characterised by a set of axioms. For the sake of space, here we only give the axioms that define the semantics of the list operator.

First of all, GoGrid checks the authentication of each API call using the MD5 function to reconstruct the signature from the api-key, the user's shared secret and the time stamp. It then compares this to the signature contained in the request parameter. It also checks the time stamp with its server clock time, allowing a discrepancy of up to 10 minutes. This authentication rule can be specified as follows.

```
Axiom <Authentication>:
    For all G: ServerGoGrid, X: ServerListRequest that
        Let key = X.para.api_key,
            sig_Re = MD5(key, G.sharedSecret(key), X.timeStamp)
        in  G.List(X).statusCode == 403,
                If   X.para.sig <> sig_Re
                     or abs(X.timeStamp - G.clockTime) > 600;
        End
    End
```

The second axiom is about the semantics of the List operation. It states that if the number of items per page required by the request is greater than 0 (i.e., $N_{per\,page} > 0$) and the call is successful, then, in the response summary, the number N_{pages} of pages, total number N_{items} of items and the number $N_{per\ page}$ of items on each page has the following relationship:

$$N_{pages} = \lceil N_{items} / N_{per\,page} \rceil,$$

```
Axiom <list.pagination1>:
    For all G: ServerGoGrid, X: ServerListRequest that
        Let res = G.List(X),
            sCode = G.List(X).statusCode,
            itemsPerPage = X.num_items
        in
    res.summary.numpages == res.summary.total/itemsPerPage,
        if    sCode == 200, itemsPerPage > 0;
        End
    End
```

The third axiom of the List operator states that, when each page contains N items, the i'th item on page k must be the same as the j'th item when there is no pagination, where

$$j = k \times N + i$$

```
Axiom <list.pagination2>:
For all   G: ServerGoGrid, i, j: int,
          X, X1: ServerListRequest that
    Let
        res = G.List(X),        sCode   = G.List(X).statusCode,
        res1 =   G.List(X1),    sCode1 = G.List(X1).statusCode,
        n = X.num_items,     n1= X1.num_items
        k = X.page,
    in
        res.objects.items(i) == res1.objects.items(j),
            if    sCode == 200, sCode1 == 200,
                  n > 0,    n1== 0,
                  j == k*n + i,
                  0 <= i, i < res.summary.returned;
    End
End
```

The fourth axiom states that when the number of items per page is specified, the list of objects in a page either contains exactly the number of objects if the page is not the last, or at most that number if the page is the last. Moreover, the number of items in the result must equal the value of parameter returned in the result summary.

```
Axiom <list.pagination3>:
    For all  G: ServerGoGrid, i, j: int,
             X: ServerListRequest that
        Let
             result = G.List(X).objects,
             sCode = G.List(X).statusCode,
             n = X.num_items,
             nr = G.List(X).summary.returned,
             numPages = G.List(X).summary.numpages,
             lpage = X.page
        in
             result.length == nr,  if  sCode == 200, n > 0;
             nr == n,  if  sCode == 200, n > 0, lpage < numPages;
             nr <= n,  if sCode == 200, n > 0, lpage == numPages;
    End
End
```

An important property of the List operator is that, being an observer, it will not change the state of the system to which it is applied. This can be stated in the following axiom, though this need not be included in the specification because we have already declared the operator as an observer.

```
Axiom <List-Op>:
    For all  G: ServerGoGrid, X: ServerListRequest,
             X1: ServerXOpRequest that
        [G.List(X)].XOp(X1) == G.XOp(X1);
End
```

where XOp can be any of the operators List, Get, Add, Edit, Delete, etc.

Finally, when an operation does changes the state of the system, the List operator should be able to observe the difference accordingly. For example, the following axioms state the effect of Add when observed by the List operator.

```
Axiom <Add-List>:
    For all  G: ServerGoGrid, X1: ServerAddRequest,
             X2: ServerListRequest that
         [G.Add(X1)].List(X2).objects ==
              insert(G.List(X2).objects, G.Add(X1).objects),
                  If   X2.num_items == 0, X2.server_type == NULL,
                       X2.isSandbox == NULL, X2.datacenter == NULL,
                       G.Add(X1).statusCode == 200,
                       G.List(X2).statusCode == 200;
    End
```

The corresponding axioms for other operators are similar and are omitted to save space.

IV. Discussion

In this section, we report the main findings of the case study.

A. Improving Document Preciseness

As one may expect, the ambiguity in the original documentation of the GoGrid API [21] was detected in the process of formalization. This documentation [21] specifies the data types of the API request parameters and their corresponding responses, and describes the meaning of each in normative text. Sample requests and responses are also given to explain the semantics and usage of the API.

In most cases, the meanings of the operations are left to the reader to interpret, according to his understanding. For example, in the description of the results of the List operator, the meaning of pagination according to num-items is not formally defined. We, however, specified it exactly to ensure that there is only one interpretation.

B. Detecting Incompleteness

Ambiguity in natural language documents is often caused when the specification is incomplete, meaning that some information is missing. The GoGrid documentation has this problem too. For example, in some cases, it is unclear about the range of values for a parameter and what will happen when the value is out of the range. An example of this is the num_items parameter, for the number of items in a page in a List request, which must be greater than 0, with no alternative behaviour specified for when it is not. Error codes are another example. It is unclear when each code will be returned. Both this and the num_items issue have been left unresolved because we do not have the relevant information.

There are two more serious cases of incompleteness, however. The List operation can list all the jobs in the system for a specified range of dates, but it can also list all the jobs for a specified object type, of a certain state or belonging to a certain owner. There is no documentation of these additional features. Another example concerns the relationship between requests and responses. The *id* parameter occurs in both the request and the response for the Get operator on Job objects. There is no statement explaining what the *id* parameter is for and the two occurrences are different in the samples given in the document, again with no explanation. Writing the formal specification has forced us to be precise and complete, making this incompleteness immediately apparent.

C. Checking Consistency

Cloud Computing is a relatively young field; so, some evolution in cloud software is inevitable. New API versions may emerge frequently. This often causes a mismatch between the software and its documentation. We detected many such cases for GoGrid. Here are three examples:

1. from version 1.5, GoGrid added a new general attribute called *datacenter* as a request query parameter but in the documentation of the Job object, there is no mention of this attribute.
2. similarly, there is no description of the attribute *numpages* in the documentation of the Get, Edit, and Delete operators even though it appears in the sample responses for these operators.
3. moreover, the parameter *port* in the Edit operation on

LoadBalancer objects must satisfy *port >=0* but the condition is *port >0* for the other operations.

D. Reducing Redundancy

In an unstructured document, redundancy is also a common problem. The same information may arise in several different places with different descriptions although the meanings are the same. This often causes confusion. Take error code for example. There is a detailed description of error code in the chapter *Anatomy of a GoGrid API Call* but this is duplicated in the documentation for every API call. Another example is the three different response formats associated with each operation. These descriptions are duplications and occupy most of the space. The algebraic specification that we presented in the paper uses specification units to organize the document structure. Consequently, the redundancy is reduced.

E. Understandability of Document

An advantage of natural language documentation is its understandability. It is widely perceived by industry that formal methods are difficult to learn and expensive to apply. However, our case study demonstrates that without much training ordinary software developers can write algebraic specification, even for real industrial strength software systems like GoGrid. This confirms the discovery reported in [20]; that algebraic specification is easy to write and easy to understand. It can be part of the job of any ordinary software developer.

V. CONCLUSION AND FUTURE WORKS

In this paper, we applied the CASOCC-WS specification language to cloud computing interface with a case study on the GoGrid system. This demonstrated the value of algebraic specification for RESTful web services.

We are currently extending the algebraic specification language and studying its theoretical foundation. We are also developing a tool that uses the language as input to support automated testing of a cloud computing interface. The case study reported in this paper specifies only the functions of resource management that the GoGrid API original document specifies. However, the specification of the properties and dynamic behaviours of the resources are left as an open problem. Further case studies of the formal specification of PaaS and SaaS will also be conducted.

ACKNOWLEDGEMENT

The work reported in this paper is partially supported by EU FP7 project MONICA on Mobile Cloud Computing (Grant No.: PIRSES-GA-2011-295222) and National Natural Science Foundation of Jiangsu Province, China (Grant No. BK2011022).

REFERENCES

[1] A. van Lamsweerde, Formal specification: a roadmap, in Proc. of ICSE 2000 - Future of SE Track, 2000, pp. 147--159.

[2] M. Juric, B. Mathew and P. Sarang, Business Process Execution Language for Web Services, 2nd ed., Packt Publishing, Jan 2006.

[3] L. Richardson and S. Ruby, RESTful web services, O'Reilly, 2007.

[4] M. Papazoglou, Web services & SOA: principles and technology, Prentice Hall, 2012

[5] M. J. Hadley, Web Application Description Language (WADL), Sun Microsystems Inc., CA, USA, SMLI TR-2006-153, March 2006.

[6] J. Kopecky, K. Gomadam, and T. Vitvar, hRESTS: An HTML microformat for describing RESTful web services. In: Proc. of WI-IAT'08, Dec 2008, Sydney, Australia, IEEE/WIC/ACM.

[7] J. Lathem, K. Gomadam and A. P. Sheth, SA-REST and (S)mashups: Adding Semantics to RESTful Services, in Proc. of ICSC'07, 2007, pp469-476.

[8] O. Liskin, L. Singer, K. Schneider, Welcome to the Real World: A Notation for Modeling REST Services, IEEE Internet Computing, pp. 36-44, July-Aug., 2012.

[9] J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright, Initial algebra semantics and continuous algebras, Journal of ACM, vol. 24, no. 1, pp. 68-95, 1977.

[10] H.-D. Ehrich, On the theory of specification, implementation, and parametrization of abstract data types, Journal of ACM, vol. 29, no. 1, pp. 206-227, 1982.

[11] J. A. Goguen and G. Malcolm, A hidden agenda, Theoretical Computer Science, vol. 245, no. 1, pp. 55-101, 2000.

[12] C. Cirstea, Coalgebra semantics for hidden algebra: Parameterised objects an inheritance, in Proc. of WADT'97, 1997, pp. 174-189.

[13] J. M. Rutten, Universal coalgebra: a theory of systems, Theoretical Computer Science, vol. 249, no. 1, pp. 3-80, 2000.

[14] C. Cirstea, A coalgebraic equational approach to specifying observational structures, Theoretical Computer Science, vol. 280, no. 1-2, pp. 35–68, 2002.

[15] F. Bonchi and U. Montanari, A coalgebraic theory of reactive systems, Electr. Notes Theor. Comput. Sci., vol.209, pp.201-215, 2008.

[16] D. Sannella and A. Tarlecki, Algebraic methods for specification and formal development of programs, ACM Computing Surveys, vol. 31, no. 3es, p. 10, 1999.

[17] L. Kong, H. Zhu, and B. Zhou, Automated testing EJB components based on algebraic specifications, in Proc. of COMPSAC'07, vol.2, 2007, pp. 717-722.

[18] B. Yu, L. Kong, Y. Zhang, and H. Zhu, Testing Java components based on algebraic specifications, in Proc. of ICST'08, 2008, pp. 190-199.

[19] GoGrid.com, http://www.gogrid.com, last access: July 10, 2012.

[20] H. Zhu and B. Yu, Algebraic specification of web services, in Proc. of QSIC'10, 2010, pp. 457-464.

[21] GoGrid.com, GoGrid wiki,https://wiki.gogrid.com/wiki/index.php, last access: July 10, 2012.

[22] H. Zhu and B. Yu, An experiment with algebraic specifications of software components, in Proc. of QSIC'10, 2010, pp. 190-199.

[23] T. A., Berson, Differential Cryptanalysis Mod 2^{32} with Applications to MD5. Proc. of *EUROCRYPT'92*. pp. 71–80, 1992.

A Holistic Approach to Energy Efficiency Management Systems

Ignacio González
University of Oviedo
Oviedo, Spain
gonzalezaloignacio@uniovi.es

María Rodríguez Fernández
University of Oviedo
Oviedo, Spain
rodriguezfmaria@uniovi.es

Juan Jacobo Peralta
Andalusian Institute of Technology
Málaga, Spain
jjperalta@iat.es

Adolfo Cortés
Ingenia
Málaga, Spain
adolfo@ingenia.es

***Abstract*—Improvement in energy efficiency is one of the most effective ways in economic terms to increase supply security and reduce emissions of greenhouse gases. Furthermore, the increased cost of energy resources has encouraged the development of new technologies that allows their efficient use. Through the identification and control of end users consumptions - businesses, residential, public, etc. – these technologies allow us to have more efficient consumption without lowering the threshold of comfort users are used to. The Smart Home Energy project makes a profitable use of these technologies to provide a complete solution that, through user interaction with electrical devices present in the home and integrated into the network, allows on one hand to manage, control, plan and in most cases reduce the electric bill, being aware about the cost of it, and on the other hand, help to improve the environment.**

Keywords-Digital Home; Energy Efficiency; Bill reduction; Cloud.

I. INTRODUCTION

A. *Interoperability between systems*

A Digital Home [1] offers users an intelligent environment that learns and adapts to the references and needs of its occupants. However, there are many restrictions: the high cost of certain systems, capacity problems, lack of standardization, etc. The most important restriction to solve is the lack of real interoperability between different systems.

Both domotic systems and service robots allow a major modernization in Spanish homes and improve energy efficiency thereof. These two technologies are the ones that provide the greatest technological advances in homes, public buildings or workplaces.

The lack of interoperability makes both systems work and interact independently. This means that if we make them work together, there would be a duplication of performed tasks. So if both systems could coordinate, resources will be optimized and it would improve energy efficiency.

Therefore, it is essential that those devices communicate with each other in a complementary way, i.e. sharing the services to know what is happening around them and take decisions accordingly.

Technically, it is possible to create a common communication protocol at the application level on the protocol stack TCP / IP, and the corresponding adapters for each device. This protocol, as standard, allows the intercommunication between devices that comply with the label that ensures the adequacy of their adapters. It is named DH Compliant [2] and it is a universal and open standard. Because of this, the Smart Home Energy project (hereinafter SHE) is based on technology standards, giving them some advantages that other protocols lack:

- There is neither technological nor brand dependence.
- Savings in solution investment.
- Intellectual and economic richness both in companies and national research institutions.
- Control and simultaneous management of service robots, smart home applications and smart appliances.

B. *Device management*

In recent years, computers and electrical appliances that are being incorporated at homes are reaching more and more energy efficiency levels. However, its contribution is still not enough in the current energy scenario, where external energy dependence and indefinite rising prices cast doubt on the profitability of these devices compared to the useful life and the necessary investment.

This means that energy saving measures and common recommendations (energy-efficient light bulbs, A + + electrical appliances, awareness campaigns , etc.) are increasingly ineffective in achieving a significant reduction of consumption, prompting the need for more advanced strategies.

From a technical point of view, a home is an extremely complex system with many uncertainty sources depending on environmental conditions and human behavior.

Therefore, the inclusion of monitoring and control systems in real time in our homes is becoming increasingly necessary.

The incorporation of these management and control devices allows the quantification of the energy performance of a home, recording the consumption values in order to characterize the profiles and habitual patterns. This provides the user - using appropriate predictive models- with the necessary information to anticipate and minimize energy losses.

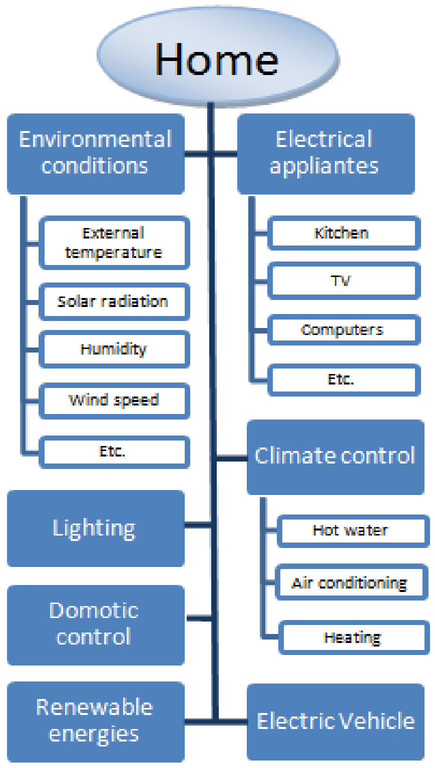


Figure 1. Elements involved in home energy management

C. *Holistic approach*

From a physical standpoint, all elements on a system may interact between each other that may indicate a dependence level that could be characterized by statistical analysis based on observation of variables and parameters.

This fact also takes place in the field of HVAC in homes, since all the elements of the domestic system (people, electrical appliances, lighting, outdoor conditions, etc.) exchange heat with each other by changing comfort conditions (temperature and humidity). For that reason, quantification and analysis of all variables involved in heat exchange must be made from a holistic approach, measuring the contribution of the element of each system and modeling its interaction to propose real energy-efficient alternatives within the conditions of comfort that the user sets.

Applying this integration approach to a energy management system using a suitable learning system, optimal decisions would be made in real time to reduce energy losses (based on "experience" of the system from the beginning of its operation); decisions that could be ineffective or even counterproductive if the mentioned approach is not considered.

This will result in most cases in a reduction on the electric bill, taking into account that the electrical company has an important role in the system.

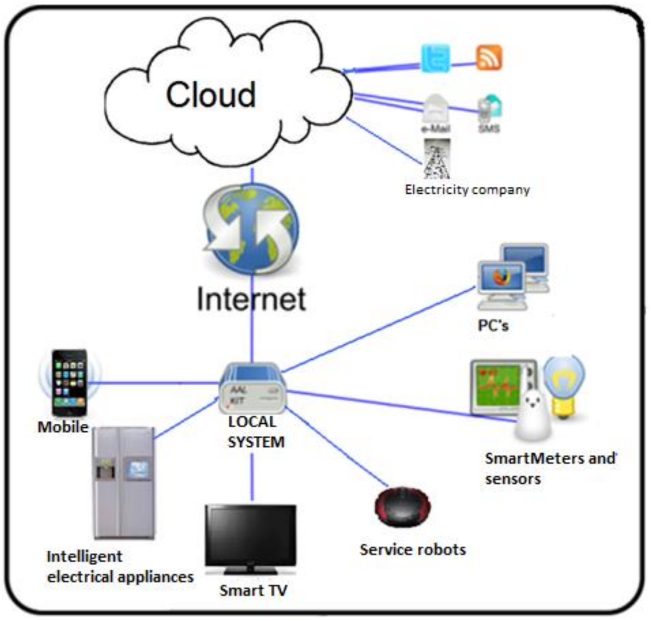


Figure 2. Multidevice architecture scheme

As seen in Figure 2, the great advantage of the architecture stands on the fact of implementing a solution without having to think about which device is addressed.

The rest of the paper is organized as follows. Section II describes the state of the art. Section III explains the proposed solution. In the last section, we draw the conclusions and indicate the future work.

II. STATE OF THE ART AND DISCUSSION ON HOW TO IMPLEMENT THE SOLUTION

Integration of various automation and control technologies in the domestic environment is called “Digital Home”, an idea not only applying to domestic tasks through smart home appliances, but also aiming to cover needs of personal assistance, education and entertainment as well as security and surveillance.

The following sections will discuss the different points to be solved and decisions on the most appropriate technologies to be used in the proposed solution:

A. *Efficient energy management*

The Digital Home proposes an efficient energy management by integrating some features in common with BMS (Building Management System) [3]. such as issues related to the HVAC control, a correct monitoring of lighting, allowing a control of consumption and, therefore, its associated costs.

Control of electrical appliances and robots is not covered by these systems, so BMS are a good starting point to define the features and functionality of SHE, even though their ultimate goal does not match the prototype to be developed. In terms of architecture, BMS has a similar structure and characteristics to those presented by the prototype: hardware components (sensors, controllers and actuators), computer processing power and control of all sensors and actuators, user interface (smart-phones, PCs, tablets, etc.) as well as means of transmission.

The meters that will be referenced in Section III generate XML reports, allowing to be processed quickly and giving the necessary information for operation to the SHE system. In contrast to other research, all business logic will be in the cloud and it is going to be described in the next section. As will be seen, the core protocol requires only basic input parameters in order to generate configurable and customizable reports and recommendations, among other functions, independent of the measuring device.

B. *Cloud*

It is necessary to store the information from the digital home in a centralized way in order to have a database that allows comparative analysis and heuristics. Cloud technology [4] allows us to centralize the information without requiring high-capacity storage in homes as well as to manage and maintain the integrity, security and availability of data, storing replicated data to ensure recovery in case of a loss of information.

In turn, the Cloud server-side solution provides an elastic system that is able to be sized according to demand using an infrastructure that does not require a high initial investment. It also allows an adaptive storage capacity and information processing to specific needs. This adaptive capacity also improves the overall energy efficiency of the system.

Therefore, this solution provides facilities for software updating, software improvements and developments. This is immediate and transparent for the user because most of the software is centralized and not distributed on each node.

From the user’s perspective, Cloud technology provides access to the stored information from any device, anywhere [5].

C. *Communication*

After the study of various technologies, we chose to use of DPWS (Devices Profile for Web Services) [7] for the communication within the housing as it maintains the philosophy of SOA combined with the convenience of Web Services.

It is necessary also to take into account that the system requires the input in an agile way and an easy integration of different devices. Furthermore, the access to and from the outside will be made over different types of networks – wired and wireless- so communications between digital homes nodes and the Cloud are made using REST API [8].

The implementation of the REST API allows defining a communication interface between software components (API) where an URL represents an object or resource whose content is accessed via HTTP. This solution means that a digital home can notify captured information to the Cloud. These are the advantages of the approach:

- Portability between different languages. This is highly important for integrating different manufacturers and technologies in digital home nodes.
- Performance improvement comparing to other APIs (XML, SOAP, etc.), which is particularly critical due to the large number of potential nodes of the digital homes network.
- Easiness of scalability to consider a probable exponential in the adoption of digital homes.
- REST communication is less "heavyweight" compared to SOAP, so it is faster and consumes less bandwidth.
- Lack of strong typing and changeable data structure at any request.

On the other hand, applications such as consumers’ information transferred to and processed in Cloud also will make use of REST API and will serve of these facilities. Furthermore, the REST API functionality can be extended in client applications using widgets or the own applications or if the API is opened by applications developed by the users or communities of these manufacturers, utilities, etc. Next, the qualities of the REST API to exploit Cloud information are highlighted:

- Exploitation and use of information and services in an agile way, as well as easy integration into several applications and devices.
- Resources access is more accessible than SOAP and other heavier protocols that require more processing of the response to its interpretation.
- Customer accessing a REST communication does not require large resources compared to SOAP.

D. *DHC Adapters*

The SHE (Smart Home Energy) kernel must be based on an open, extensible and modular protocol, so DHCompliant [2] including its open energy service DHC Energy is considered a very suitable option as it allows integration of consumption measuring devices, such as CurrentCost, with the other technologies in the home. The services of DHC communication protocol are set out below:

- DHC- Security & Privacy service provides the security and privacy to prevent fraudulent use of devices and access to data by people or devices outside the system.
- DHC- Groups service coordinates the implementation of collaborative tasks between different devices connected to the system.
- DHC-Localization Service provides the information to locate devices within the home and help them in navigation.

- DHC-Intelligence service provides intelligence to the system by managing rules that control the tasks and predicting sensor data from a previous training.
- DHC-Energy Service [9] improves energy efficiency being probably one of the most effective ways in economic terms increasing supply security and reducing emissions of greenhouse gases and other pollutants.
  DHC-Energy arises from that need by establishing the concept of energy and smart grids [10] in the DHCompliant communication protocol. It defines a set of concepts and energy management savings that allow the user to know in detail the energy consumption data in the Digital Home environment.

The prototype, using an expert system with a rule editor, allows the recommendation of actions on a digital home environment depending on environmental conditions, creating a knowledge database with the consumption.

For this purpose, the expert system, based on its experience, uses the rules to model the system. In addition, this editor enables you to test and simulate the rules. The rule execution is done with an inference engine based on an execution of rules and tree forward.

III. PROPOSED SOLUTION

Therefore, after studying the state of the art, the SHE architecture is proposed. It is specified in SysML, the standard and open language for systems engineering.

A. *Requirements*

The need to have an intelligent measuring device, Smart Meter: will report in real time about the consumption of gas and electricity, showing the actual rates. The Smart meter [6] shown in Figure 3 (a) and its adapter for connection in Figure 3 (b) are examples of the device that are been used:

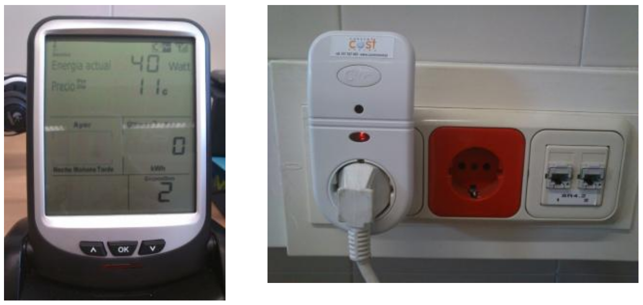


Figure 3. Smart meter (Current Cost) and his wireless connection

For tasks that require cooperation among multiple robots, first a leader is chosen among them, depending on the energy consumption of each robot.

B. *Establishment of the charging information*

We selected the most economic energy configuration. The user can see the types of charging (maximum consumption, cost per kWh, etc.), being able to select the best rate, so that the user can choose what time will a device be enabled or disabled according to tasks being performed, time of day and the tariffs to be applied.

Figure 4 shows a sequence diagram describing the process.

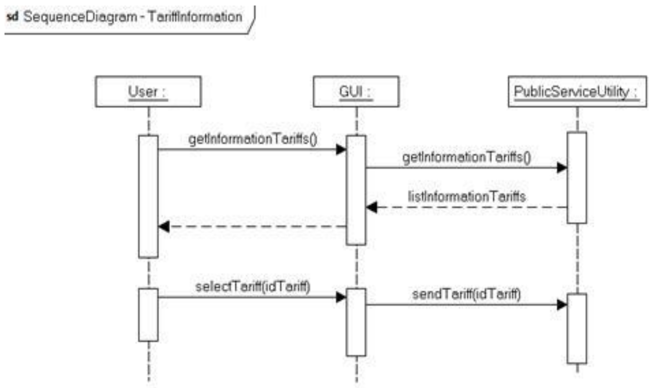


Figure 4. Sequence diagram: Obtain pricing information

C. *Device states*

It is also important to know the user's preferences to be compared with the data that shows the state of the device and the power source - either renewable or normal electrical supply. The sequence diagram that represents the different states of the device is shown on Figure 5.

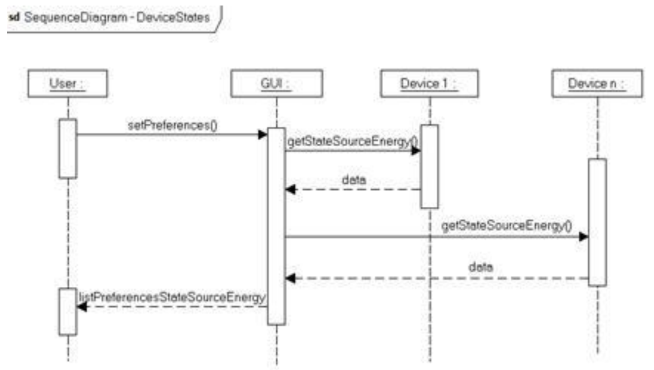


Figure 5. Sequence diagram: Device state

D. *Temperature regulation*

Using metadata is one of the main ways of managing energy savings. Through that, the internal and external temperatures are known, as well as other certain characteristics of the environment (humidity, number of sunlight hours, rain, wind, etc.). It is described in the following sequence diagram.

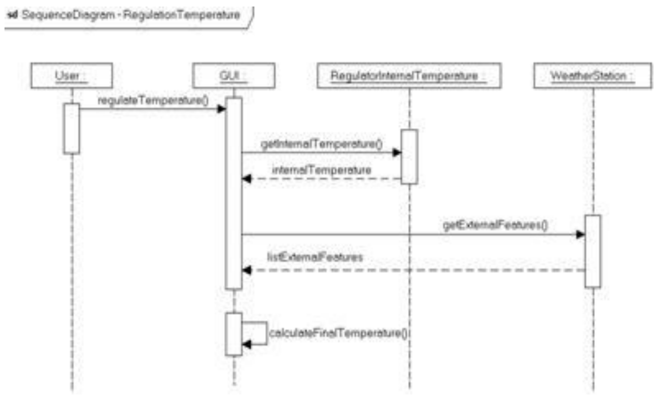


Figure 6. Sequence diagram: Temperature regulation

E. Energy profiles

There is a large number of devices that are held to regulate their energy consumption (TV, mobile phones, computers, etc.). Therefore, the DHC-Energy defined a set of patterns of energy consumption:

- Off – This profile is used to point out that the user does not need the device and the power consumption is zero.
- Stand by – The device is awaiting orders. Its power consumption is minimal.
- Low - The device is operating at low capacity. In this profile, the energy saving rate will be the highest.
- High - The device is operating at full capacity. In this profile, the energy saving rate is the lowest.
- Emergency - Full capacity of use. There will be no energy savings.

Assignment of a profile for a device: The user will be able to choose one of the define profiles for each device or each home.

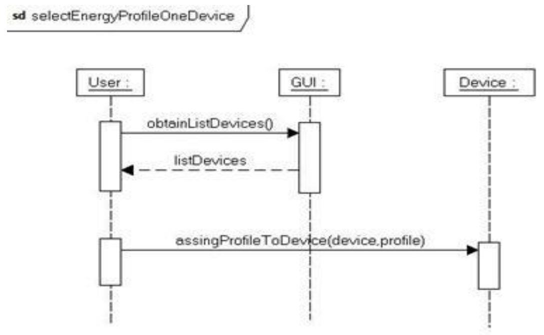

sd selectEnergyProfileOneDevice
User :
GUI :
Device :
obtainListDevices()
listDevices
assingProfileToDevice(device.profile)

Figure 7. Sequence diagram: Election of the energy profile

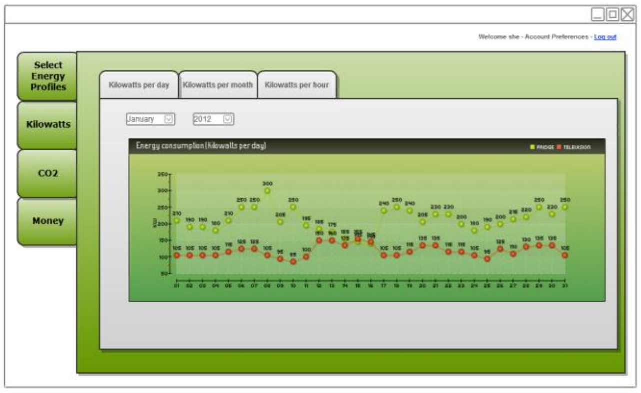
F. Change Profile

If a device remains in an inactive state, the device must change its energy profile for a lower profile.

G. Service and devices description

The DHCompliant device description should contain a list of DHC-Energy services. This XML description must contain the name of the DHC-Energy service and SCPDURL description with the URL to the extended service description.

This architecture is summarized in a main block, which is implemented using Cloud Servers, Smart Home Energy Management System, and blocks for adapters that will have the common interface ConnectedDevice (device connected). On the other hand, the recommendation system will have a distributed architecture between the cloud and the devices in the home. The information sharing took place by exchanging an XML file. After being obtained from the SHE adapter (DHC-Energy), this information is sent to the cloud.



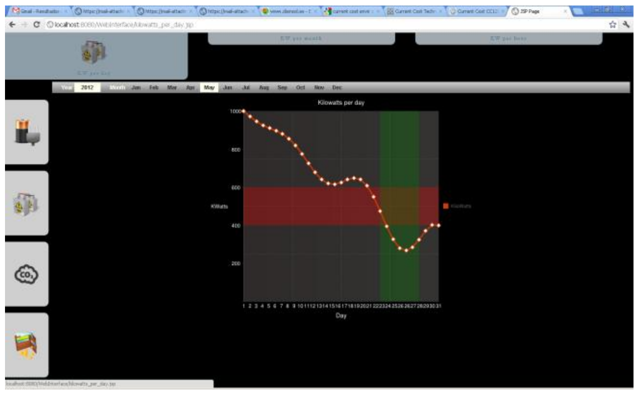

Year 2012 Month Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Kilowatts per day
Kilowatts
Day

Figure 8. Prototype interfaces

As it is shown in Figure 8, the information will be displayed on screens, which take into account different aspects of usability, accessibility and of course, the functional aspects of providing the user of the information that allows to manage the environment devices and Smart Grid that are in the user digital home. As there are different manufacturers working within the SHE, implementations depend on each particular company or manufacturer. Prototypes that are independent of a particular solution have been proposed and they are being used by various technologically neutral members of the consortium.

IV. Conclusion and Future Work

This architecture work allows to determine that an open stage of interaction between devices and the Smart Grid can be set by providing more capabilities than pure traditional energy efficiency (such as accounting and reductions in consumption). It also allows the establishment of a consumption profile of the different heterogeneous devices a user has at home, as well as a referral system in the cloud associated with business intelligence that allows reducing even more the energy expenditure. All this is done in a distributed way but through a single point where the user interacts.

It can be also concluded that this technology has advantages over other approaches because it is open, distributed, scalable and requires little or no configuration by the end user. In addition to the technical advantages, other advantages include the open environments and standards that will produce more open market scenarios. The results of these market changes are beyond the scope of this communication.

This first interaction has been outlined in architecture, prototyping the various adapters and software to deploy in the cloud. It remains to be completed, and tested for selected use cases in the project Smart Home Environment.

One open issue is the analysis of the scalability of the technology. Different scenarios could be simulated (for example using queuing theory) to understand how a large-scale deployment would consume more or less computational resources on the Cloud side and of the recommenders.

Acknowledgment

This project is funded by the Ministry of Economy and Competitiveness of Spain (IPT-2011-1237-920000).
We are also very grateful to all the members of the consortium (Ingenia, Satec, Ingho, Tecopysa, Cotesa, IAT, University of Oviedo) and to the programmers of the entities participating in the project (Víctor García, Alejandro Álvarez, José María Ocón).

References

[1] I.G. Alonso, O.A. Fres, A.A. Fernandez, P.G. del Torno, J.M. Maestre, and M.D.A.G. Fuente. Towards a new open communication standard between homes and service robots, the DH Compliant case. Robotics and Autonomous Systems. Volume 60, Issue 6, June 2012, Pages 889–900.

[2] DH Compliant Project, 2009. Web Site. http://www.dhcompliant.com/ [retrieved: october, 2012]

[3] R. Spinar, P. Muthukumaran, R. Paz, D. Pesch, W. Song, S. Chaudhry, C.J. Sreenan, E. Jafer, and B. O'Flynn. Demo Abstract: Efficient Building Management with IP-based Wireless Sensor Network, 6th European Conference on Wireless Sensor Networks, Cork, 2009. Pp 1-2.

[4] J. Rhoton and R. Haukioja. Cloud Computing Architected: Solution Design Handbook. Recursive Press; 2011.

[5] L. Spaanenburg and H. Spaanenburg. Cloud Connectivity and Embedded Sensory Systems. Available at: http://www.springer.com/engineering/circuits+%26+systems/book/978-1-4419-7544-7 [retrieved: july, 2012]

[6] F.J. Casellas, G. Velasco, F. Guinjoan, and R. Piqué. El concepto de Smart Metering en el nuevo escenario de distribución eléctrica. Seminario Anual de Automática, Electrónica Industrial e Instrumentación. Bilbao: 2010, p. 752-757 978-84-95809-75-9

[7] OASIS. Devices Profile for Web Services. 2009.http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01 [retrieved: july, 2012]

[8] M. Masse. REST API Design Rulebook. O'Reilly Media; 2011.

[9] DH Compliant Project. Energy Management. 2009. Web Site. http://156.35.46.38/index.php/blog/show/DHCompliant-Protocol-v.0.2-%28Energy-Subsystem%29.html [retrieved: october, 2012]

[10] S. Chen, S. Song, L. Li, and J. Shen. Survey on Smart Grid Technology. Power System Technology; 2009-08

Structuring Software Reusability Metrics for Component-Based Software Development

Danail Hristov, Oliver Hummel, Mahmudul Huq, Werner Janjic
Software Engineering Group
University of Mannheim
Mannheim, Germany
e-mail: {dhristov, hummel, shuq, janjic}@mail.uni-mannheim.de

Abstract — **The idea of reusing software components has been present in software engineering for several decades. Although the software industry developed massively in recent decades, component reuse is still facing numerous challenges and lacking adoption by practitioners. One of the impediments preventing efficient and effective reuse is the difficulty to determine which artifacts are best suited to solve a particular problem in a given context and how easy it will be to reuse them there. So far, no clear framework is describing the reusability of software and structuring appropriate metrics that can be found in literature. Nevertheless, a good understanding of reusability as well as adequate and easy to use metrics for quantification of reusability are necessary to simplify and accelerate the adoption of component reuse in software development. Thus, we propose an initial version of such a framework intended to structure existing reusability metrics for component-based software development that we have collected for this paper.**

Keywords- Software Reusability; Software Reusability Metrics; Component-Based Software Development.

I. INTRODUCTION

The idea of software reuse [1] is not new: its roots date back to 1968 when McIlroy has presented his seminal work on reusable components [2] at the NATO Software Engineering Conference in Garmisch, Germany. However, there has only been limited practical experience with reuse until the late 1980s, when large-scale reuse programs were adopted by companies, mainly in the US (e.g., by IBM and Hewlett Packard [3]) and Japan (e.g., by Toshiba and Fujitsu); these efforts have also pushed forward the research in the 1990s, and in turn created a growing interest in systematic software reuse and reuse programs for organizations at that time [4]. After the turn of the millenium, widely available broadband internet connections and the raise of the open source movement have clearly created opportunities for broader inter-organizational reuse [5] and resulted in a huge amount of source code and components that is freely available [6]. Even the recently rising popularity of agile methodologies and practices has created numerous interesting ideas on how to facilitate reuse in that context [7,8]. However, as stated by Frakes and Kang, there are still numerous open issues to be solved [9] – including a better understanding of reusability.

Before being able to go into more details on the challenges tackled in this paper, we have to clarify what is not in its scope since reuse is a broad concept that cannot only be applied on components, but also on numerous other artifacts necessary for software development. Such artifacts are, for example, design structures [3,11,12], architectures [1,11,12], or even documentation [12]. However, the results presented in this paper will focus on software components, in both source code and binary form or in other words on the white- and black-box reuse of these software building blocks. In white-box reuse, the source code is available to the developer and can be changed before it is integrated into a new context, while in black-box reuse this is not the case and therefore only a component's interface (containing the public methods and attributes) and the documentation are visible [13]. Services in Service Oriented Architectures (SOA) [14] are conceptually certainly similar, but research on service reusability could not be considered in this publication due to space limitations. Black-box reuse probably tended to be the more facilitated approach in the past [15], however, the wide availability of search engines for open source software [7] has certainly brought the possibility of white-box reuse back into the center of interest .

It has been observed that software reuse research is rather scattered than focused and consecutive [9]. It is also evident that – though the software industry has developed massively in last decades – the paradigm of component reuse is still facing numerous issues and hence lacking adoption from practitioners. One of the central impediments that prevent efficient and effective reuse today is the difficulty to determine which artifacts are best suitable to solve a particular problem in a certain context and how easy it will be to reuse them. In other words, no comprehensive framework describing the reusability of software and structuring appropriate metrics exists in literature so far. Nevertheless, a good understanding of software reusability as well as adequate and easy to use metrics for its quantification are crucial in order to facilitate the adoption of reuse in software development. The focus of the research presented in this paper is on the reusability of software components in an ad-hoc reuse scenario. By "ad-hoc reuse scenario" we mean the spontaneous decision of a developer to use a component repository or search engine [7], indexing e.g. open source software not specifically built for being reused, in order to search for a component that might match the given requirements. This type of software reuse is probably one of the most promising that does not require larger upfront investments as for example the creation of a software product line [17] or a planned reuse program [4]. Nevertheless, it is likely that the insights gained from this work can be transferred to the latter areas where a measure for judging the reusability of software artifacts is also important.

Hence, in this paper we are distilling an initial framework for structuring software reusability metrics in component-based software development based on a comprehensive survey of metrics proposed in the literature. Its remainder is organized in the following order: in Section II we discuss the general concept of software reusability before we turn to the current state of the art in reusability metrics in Section III. In Section IV we propose our novel software reusability framework for the context of ad-hoc reuse approaches. The paper is finally concluded with a summary of its findings and an outlook on potential future work in Section V.

II. REUSE & REUSABILITY

Literature provides a significant number of definitions for software reuse; probably the most popular one was published by Krueger [1] who has defined software reuse as the use of *"existing software artifacts during the construction of a new software system"*. Software reuse can be embraced in several different ways: on the one hand, it can be practiced in a structured and controlled manner inside organizations, where software artifacts are systematically designed for reuse when created according to a software reusability policy [12]. In this case, the artifacts can be usually reused within a particular domain, which is nowadays well-known as the paradigm of software product line engineering [4,9]. It is based on the presumption that most software systems are not new but rather a variation (or improvement) of already existing systems in a domain [9]. For such software reusability policies to succeed, they have to be systematic [18] and well planned. Thus, such a scenario of reuse is also known as planned reuse which indicates that it requires up-front investments by the organization implementing it, for example, designing the software for potential future reuse, establishment of libraries of reusable components, etc. [3,19]. Such systematic software reuse has been the central topic of a substantial amount of research papers (as, e.g., [12]).

Another interesting scenario is ad-hoc reuse [3, 19]: in this case, the artifacts for reuse are taken from generic libraries or search engines. This usually happens on an individual basis (i.e., per developer) and not per project or company. Here, the role of the libraries and retrieval mechanisms is of high importance [20]. With the fast growth of the World Wide Web and the possibility to store and retrieve large amounts of data online, it has become much easier to distribute reusable assets over the Internet even between organizations [21,22]. According to the body of existing literature, practicing this kind of reuse in software development can bring substantial benefits to organizations as well as the developers [9,19,23,24]. The most widely expressed and discussed benefits of reuse are: a productivity and quality increase, easier maintainability and higher portability.

An important prerequisite for every kind of component reuse is a *"repository for storing reusable assets, plus an interface for searching the repository"* [12]. In case of planned reuse, companies need to implement and maintain an internal repository of reusable components to store the assets produced and keep them ready for reuse [23]. For ad-hoc reuse, the components are usually stored in a publicly available library, accessible over the Internet (cf. e.g. [7] for an overview). However, the issue with the efficient retrieval of components suitable for reuse is still not completely solved. However, our screening of literature did not uncover sufficient empirical evidence of practitioners and organizations actually experiencing the expected benefits. Therefore, it should not be concluded that the mere availability of prerequisites for reuse alone will increase productivity and quality in software development already. The characteristics of the available artifacts also have to be taken into consideration.

As for software reuse, a large number of definitions for *software reusability* can be found in existing literature, e.g., Kim and Stohr [19] have defined software reusability as *"a measure for the ease with which the resource can be reused in a new situation"*. It is important to distinguish between software reuse and reusability as the former is focused on the practice of reuse itself while the latter tries to make the potential of artifacts for being reused measureable. Poulin [25] has stated in this context that knowing what makes software reusable can help us learn how to build new reusable components and to identify potentially useful (and thus reusable) modules in existing programs. The literature lists several characteristics of software, which are believed to determine reusability and are therefore repeatedly referenced in research papers [8,13,20,25]. Such factors are: adaptability, complexity, composability, maintainability, modularity, portability, programming language, quality, reliability, retrievability, size and understandability. Furthermore, the reusability of a component in a certain context should be comparable to the reusability of other – potentially functionally equivalent – software components in the same context. However, most of the existing research is rather incoherent and only covers one or a few of these aspects so that to our knowledge there is no publication that has tried to bring all these aspects together in a single model.

III. EXISTING REUSABILITY DEFINITONS

In order to get an impression of reusability definitions available in the literature, we performed a systematic literature review that identified a number of articles proposing quantitative metrics for assessing the reusability of software. For that purpose, Google Scholar, IEEE Xplore, ACM Digital Library, Citeseer and Springer Link were searched with the keywords "software reusability". Titles and abstracts of delivered publications were read in order to determine whether they could contribute to the aim of our study, which resulted in a total set of 73 papers that were deemed worthwhile to be more closely read and investigated. In general, we found that some of the metrics described were newly developed solely for the purpose of measuring reusability, while others were modified or adapted from other areas (such as maintainability measurement) and have not been initially developed with reusability in mind. For the sake of practicality, we have separated the results of this survey, i.e., the discovered metrics, into two categories: one for white-box (allowing to look into the code of the components) and one for black-box (where usually merely interface and documentation of a component are available) reusability. This separation helps to distinguish the different nature of metrics for these two paradigms. Within these two categories that are presented in the following two subsections, the results of previous research are presented in chronological order to illustrate the development of reusability measurement. Due to limited space, we can only briefly describe most of these contributions; the interested reader is referred to the original sources for more detailed information.

A. *White-Box Reusability*

As early as in 1991, Caldiera and Basili [26] have defined three main (but still relatively abstract) attributes for assessing the reusability of components – reuse costs, functional usefulness and quality of components. These attributes were determined by factors, which are directly or indirectly measured by classic software metrics such as Halstead's Volume [27], McCabe's Cyclomatic Complexity [28] or other metrics such as Regularity and Reuse Frequency [26]. Volume was used in order to estimate two important attributes, namely reuse costs and quality of components. The Cyclomatic Complexity was used to assess all three reusability attributes introduced before,. Regularity was used to assess the former two attributes, while Reuse Frequency was merely used to assess functional usefulness.

Seven years later, Barnard [29] has suggested a composite metric for reusability of object-oriented software, which was derived from two empirical experiments. As foundation, again a variety of readily available software metrics have been used. Based on the experiments, those metrics that were related best to reusability have been selected (with corresponding confidence intervals). In order to come up with this relation, classes from C++, Java and Eiffel libraries have been considered in the experiments, assuming that classes in libraries are more reusable. Barnard's metric suite is focused on the Simplicity, Genericity and Understandability of classes' interfaces, methods and attributes.

Around the same time, Mao et al. [30] have investigated the effects of inheritance, coupling and complexity on the reusability of classes in object-oriented software. Two years later, Lee and Chang [31] proposed another set of metrics for measuring the reusability and maintainability of object-oriented software. The determining criteria here are complexity and modularity. The corresponding metrics are Internal and External Class Complexity (for complexity), and Class Cohesion and Class Coupling (for modularity).

In 2001, Cho et al. [32] have suggested metrics for component complexity, customizability, reusability and reuse. Component Reusability is determined by the functionality that the software components provide for their domain: it is the ratio between the number of interface methods in the component that provide commonality functions in the its domain, and the total number of interface methods in the component. The more commonality functions a component provides, its reusability is considered higher. Additionally to this metric, Cho et al. have suggested metrics for Component Customizability. They state that if the possibility to customize a component is not given, the reusability is low, since developers cannot adapt components for their purpose. Customizability is determined by the metric Component Variability, which is defined as the ratio of the number of customization methods to all methods in a component's interfaces.

Also in 2001 Etzkorn et al. [33] have published a model capturing reusability of object-oriented legacy software. They suggest a comprehensive metric suite covering different aspects of the reusability of individual classes. It is defined as the sum of metrics for Modularity, Interface Size, Documentation and Complexity of a class, each equally weighted.

Four years later, Bhattacharya and Perry [34] have stated that the usefulness of a software component depends not only on its internal characteristics, but also on the context in which it should be integrated. Therefore, they suggested reusability metrics measuring how well a component fits in a predefined architectural context. The prerequisite is that the (potentially reusable) components are adapted to the architectural description of the target system, which includes a description of the services needed by the system. They have proposed two metrics for measuring software reusability, namely Architecture Compliance and Component Characteristics. The Architecture Compliance metric is measured by three different sub-metrics: Architectural Component Service Compliance Coefficient, Architectural Component Attribute Compliance Coefficient and Architectural Component Behavior Compliance Coefficient. A higher value for the Architecture Compliance metrics indicates a more reusable component in a given context. The Component Characteristics metric measures the compliance of a component with regards to the data and functionality requirements of all attributes and services in the architectural description.

In 2008, Gui and Scott [35] have suggested revised formulas for established coupling and cohesion metrics in order to measure the reusability of Java components. They are proposing to measure to which extent classes are coupled together and to which extend their methods are cohesive. Additionally, they have considered transitive relationships and finally defined two metrics for measuring software reusability, based on their own versions of Coupling and Cohesion. The authors admit that additional determinants of a components' reusability exist; however they are not considered in their paper. Only very recently, Gill and Sikka [36] have proposed five new metrics for better assessing reuse and reusability in object-oriented software development. The metrics are Breadth of Inheritance Tree, Method Reuse per Inheritance Relation, Attribute Reuse Per Inheritance Relation, Generality of Class and Reuse Probability.

B. Black-Box Reusability

To our knowledge, the first set of metrics for measuring the reusability of black-box components was proposed by Washizaki et al. [16] in 2003. They proposed to consider three main factors that are expected to affect reusability: understandability, adaptability and portability. Through an empirical analysis of Java Beans components, the authors have established thresholds for each proposed metric. For measuring the overall understandability, the metrics Existence of Meta-Information and Rate of Component Observability are defined. Rate of Component Customizability measures adaptability, while Self-Completeness of Component's Return Value and Self-Completeness of Component's Parameter measure portability.

In 2004, Boxall et al. [37] have proposed that the Understandability of a software component's interface is a major quality factor for determining reusability. To measure this, they have defined a set of metrics, including values such as Interface Size, Identifier Length or Argument Count. The authors have selected 12 components from different software systems in C and C++ to empirically validate their metrics and developed simple tools to automatically calculate them. The derived values have (merely) been compared against the expert knowledge of the authors judging the reusability of these components. Consequently, the authors have stated that more empirical research is necessary.

Again one year later, Rotaru et al. [24] have identified adaptability, composability and complexity of software components as determinants for their measure of reusability. The composability of a component is determined by the complexity of its interface. Adaptability is the ability of a component to handle environmental changes. Although a preliminary metric specification is given for all three aspects, the authors have stated that an empirical calibration is necessary to better understand its effects.

Only recently (in 2009), Sharma et al. [38] have proposed an Artificial Neural Network (ANN) approach to assess the reusability of software components. The authors have considered determining reusability by four factors: Customizability, Interface Complexity, Portability and Understandability. However, only customizability was quantitatively evaluated so far. The other three factors are only to be assessed qualitatively, i.e. merely ranked on a relative scale by experts.

C. Open Issues

For the metric suites reviewed and presented in this section so far, the most evident shortcoming beyond their quite limited scope is that almost all lack a sufficient empirical validation of their prediction capabilities. Their expressiveness originates mainly from expert opinions and evidence mainly derived from small case studies so that it seems that research on reusability is largely underrepresented in empirical software engineering research so far. There may be many reasons for this situation: one possible explanation is that there is no agreement in the research community which software characteristics provide a sufficient basis for determining software reusability and which metrics for these characteristics are sufficient. In other words, there is no common understanding of what software reusability is and how it can be best measured, yet.

Furthermore, as can be seen in the metric sets we presented, they mainly focus on the technical characteristics of software, which may be inconsistent with the expectations of practitioners. Therefore, a more holistic approach to defining and measuring reusability is needed. We intend to address these issues in current research in the next section. Following the Goal-Question-Metric (GQM) approach [39], an initial proposal for a more structured and analytically justified reusability model will be developed. Thereby, the understanding of software reusability can be improved, which on the one hand should encourage researchers to invest more effort in empirically validating this (and similar) models and on the other hand should give practitioners the confidence they need for measuring reusability "in real life".

IV. STRUCTURING REUSABILITY METRICS

In this section, the reusability requirements for software components will be explained and structured in a reusability requirements model. For this purpose the well-accepted Goal-Question-Metric (GQM) paradigm [39], for deriving appropriate measurements and metrics for software reusability will be used. Following this approach, the first step is to define the goal of this research work. It can be expressed as follows:

Improve the reusability assessment of software components in an ad-hoc software reuse scenario from the developer's point of view.

The purpose here is to *improve*, the issue is *reusability assessment*, the objects are *software components* and the viewpoint is that of the developer. Another element – that of the context (*ad-hoc reuse*) is added to the goal definition. It is not explicitly defined in the GQM approach, but it is important for the research presented in this paper and will be relevant for the further elaboration. The next step foreseen in the GQM approach is to define the questions resulting from the goal stated above. They can be expressed as follows:

- which are the requirements to the software components that can determine their reusability in an ad-hoc reuse scenario?
- which are the characteristics of the software component that determine their reusability in an ad-hoc reuse scenario?

Obviously, these questions are not trivial and it is not possible to give an answer to them directly through identifying the appropriate metrics and hence a more sophisticated elaboration is needed in this case. Therefore, it is helpful to look at the following common ad-hoc reuse scenario (sometimes also called opportunistic reuse [3]) to better identify the needs of their users: usually, a developer (e.g., a software developer) would start thinking about the possibility to reuse a software component when he or she receives a task to develop certain functionality in the software system that she or he is working on. He or she would have two possibilities – (1) to develop this functionality from scratch or (2) to reuse already existing code that provides as much of this functionality as possible.

It is also necessary to stress that (2) would be a valid option only if there were no managerial, organizational or other company-internal obstacles preventing people from reusing code [29] that might come from a 3rd party and of course must be accessible in some kind of repository [17]. Looking into these two alternatives, the developer is likely to choose alternative (2) – the reuse of a software component – if the expected effort (or the corresponding costs) in case (2) is less than the effort (or costs) in case (1) [11]. The costs in (1) could be derived by monetizing the effort invested in the short term and middle/long term activities performed by the developer, and the costs in (2) could be derived by the effort invested in the short term and middle/long term activities (such as searching, integrating and testing a component) of the developer plus possible royalties that need to be paid to the creator or vendor of the component.a

To summarize, reusability requirements can be divided into functional and non-functional requirements as presented in the following. The functional requirement is that, in order to be considered for reuse in a particular context, the component(s) need to provide the functionality requested by the developer. There are two possibilities to address this requirement. The first possibility is to consider this a “hard” , i.e. a “yes or no” requirement. This means that a component would be considered for reuse only if it fully satisfies all needs specified by the developer. The second possibility is to consider the functional requirement as “soft”. In this case, also components that do not fully satisfy the requested and specified functionality are included in the consideration set. In this case, the developer has to change/adjust the functionality of the component before reusing it, or to find a workaround, which clearly also influences the perceived degree of reusability.

The non-functional requirements, which determine the reusability of a software component, can be derived from the ad-hoc reuse scenario and the factors affecting the decision on reuse just explained and can be structured as follows:

- Fast and easy to retrieve: in order to consider a software component for reuse, it has first to be found by the developer. The easier and faster a component can be retrieved, the less effort it will take to reuse it.
- Licensed for reuse in the particular context: If the component is readily approved for reuse in the context of the developer, she or he can directly implement it and thus save time. If there are any legal concerns, they have to be clarified and settled first, which will require additional effort.
- Usability: software components, which are more usable from the viewpoint of the developer, will be preferred. This can have several dimensions: satisfying quality of the software component, easy to understand how it is built and structured, guaranteed maintenance of the component in the future etc.
- Inexpensive: if a component is too expensive, this will increase the overall costs (or their equivalent effort expressed in the developer’s overall effort) and thus make the reuse alternative unattractive.
- Easy to adapt to a new usage context: The easier the component can be adapted to the context of the developer, the less effort it will take to implement it.

Overall it can be said that the better a software component meets these requirements, the higher is the probability that the developer will select it for reuse.

A. Core Elements of Reusability

Our reusability measurement model does not aim on identifying new characteristics of software components determining reusability while rejecting the existing ones, but rather focuses on structuring them better and identifying a common superset of these characteristics to determine the reusability of software components. The match between reusability requirements and characteristics is obviously an *n-to-m* relationship. This means that one characteristic can address many requirements, and in the same time one requirement can be addressed through various characteristics. Therefore, the core measurement model is defined based on the following characteristics distilled from the reusability models presented in the last section:

- **Availability:** the availability of a software component can determine how easy and fast (or hard and slow) it is to retrieve it, this is not to be confused with the opertational availability often used in the context of long-running (server) systems.
- **Documentation:** a good documentation can make the software component more reliable since it makes it easier to understand. Furthermore, it should contain the legal terms and conditions and thus make clear if it is licensed for reuse in the context of the developer or if any legal issues may arise.
- **Complexity:** the complexity of a software component determines how usable it is (i.e., if it possesses satisfying quality, if it is easy to understand and to maintain) and how easy it is to adapt the software component in the new context of use. The rationale behind this is that if there are two components which provide the same functionality (which is the prerequisite for assessing their reusability), then a lower component complexity would mean that functionality is implemented more efficiently. Thus, it is likely that the implementation of this functionality in the component is of higher quality, is easier to understand by the developer and easier to maintain in the future, and it will be easier to adapt in a new context.
- **Quality:** the quality of the component directly determines how usable it is in a given context. The quality of a component is regarded as a characteristic which describes how good it fulfils its requirements and also how error- and bug-free it is. This can include a number of sub-characteristics, e.g . whether it often crashes when it is used, whether it is thoroughly tested and whether it provides suitable test cases to be tested when integrated in a new context.
- **Maintainability:** The maintainability of a software component directly determines how usable it is for reuse. After the integration into the new system, the

component should be able to adjust to the changes in the system along with its evolution (e.g., in future versions of the system). This can be facilitated through appropriate methods that the component provides to the developer or simply through providing changeable source code of the component (this is of course only possible for white-box components).

- **Adaptability:** This characteristic directly determines how easy it is to adapt a software component to the context of the developer. Otherwise, the availability of compatible adapters will increase the ease of adapting, compared to components for which such adapters have to be developed from scratch. Apart from the programming language, the design of the component and the availability of appropriate methods and interfaces to modify, adapt and bind the component to the software landscape of the developer are of high importance.
- **Reuse:** The actual reuse of the component can also be used to infer how usable and how easy it is to adapt it. The amount and frequency of reuse, especially in contexts similar to that of the developer can serve as reference points and she or he may select the component with the higher amount and frequency of reuse.
- **Price:** the price of the software component determines how expensive it is to reuse.

An illustrative overview of the elements influencing reusability measurement as just discussed is presented in the following figure, the concrete metrics contained there are briefly discussed afterwards as well as potentially necessary distinctions between white- and black-box reuse (i.e. for source and binary components).

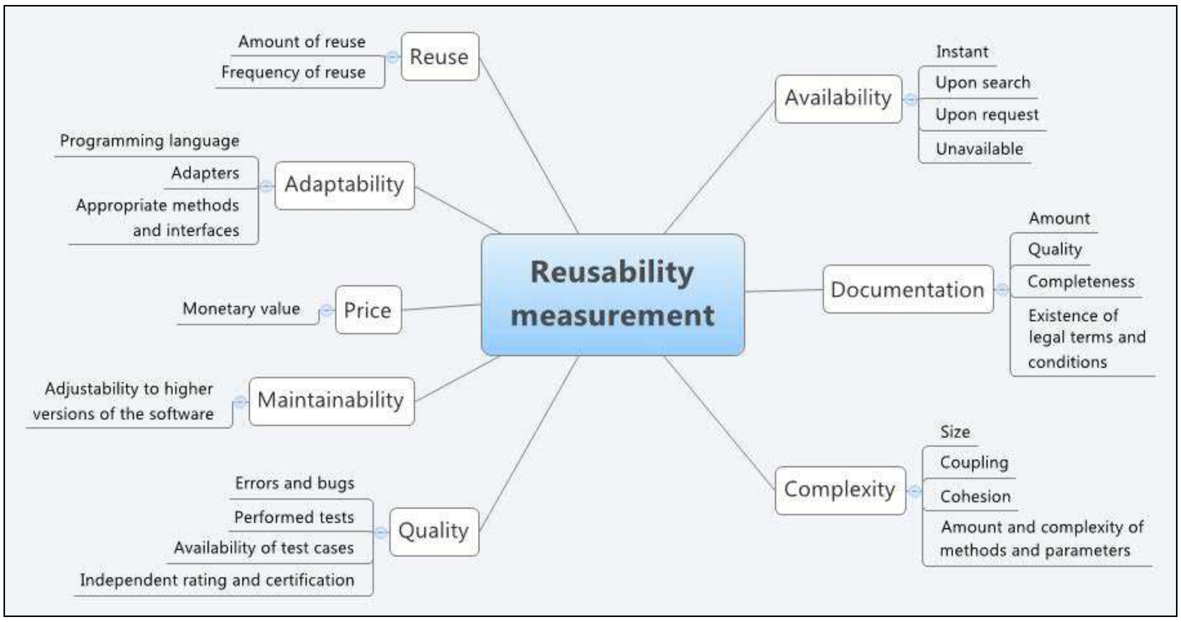


Figure 1. Factors influencing reusability measurement.

In order to calculate the reusability *(R)* of a software component *(c)* in the context of the developer *($_c$)*, the individual parts of the measurement model have to be quantified through metrics first, and then these metrics have to be aggregated in a reusability calculation model. Based on the characteristics introduced above, it can be defined as follows:

$$Rc_c = w_1 \cdot Avail + w_2 \cdot Doc + w3 \cdot Compl + w_4 \cdot Qual + w5 \cdot Maint + w_6 \cdot Price + w_7 \cdot Adapt + w_8 \cdot Reuse \quad (1)$$

w_1 - w_8 are weights and the rest are composite metrics for the attributes from the reusability measurement model. To facilitate the comparison of the reusability of different components in the same context, these values should be adjusted to a common scale, e.g., normalized to the range [0..1] since this is common for software metrics, but not always done for the metrics presented before. The values of the weights determine the importance of each characteristic of the component for its reusability and have to be determined empirically or through expert opinion. Their sum has to be equal to 1 (because of the normalization). As to the other metrics, their absolute values should be calculated first and then normalized such that a minimal and a maximal value need to be found for each metric [38]. These values can be absolute min/max values found by analytical methods or empirical values derived from the components in a large enough consideration set.

B. Concrete Metrics Proposals

Based on the insights gained from surveying numerous reusability definitions, we have been able to distill the following preliminary suggestions for concrete reusability metrics within each characteristic as presented below:

- **Availability**: a generic, qualitative and subjective metric can be used. The alternative values are: *instant*, *search with automatic aid* (e.g., an online library), *search manually* (e.g., via manually screening software systems for suitable components), upon *request* and *not available*. These values have to be placed on an ordinal scale (by, e.g., an expert – therefore the metric is subjective) and normalized (as all other metrics) to fit in the overall calculation of reusability.
- **Documentation**: to be determined by four different attributes*: amount*, *quality*, *completeness* of documentation and *availability of appropriate legal terms* and conditions. The amount is a generic, objective and quantitative metric that can be measured through its size, e.g. in kilobytes (kB). The quality is a generic, subjective and qualitative metric that can be measured on an ordinal scale (from, e.g., poor to very good) set by an expert. The same applies to the completeness. The existence of legal terms and conditions is a boolean metric: either this information is provided, or it is not.
- **Complexity**: The complexity of software is a widely researched topic and numerous metrics have been suggested in the literature. Therefore, it makes sense to use some of them for assessing the complexity of software components. The complexity intended here is a composite metric of the size of the component (e.g., in Lines of Code (LOC), excluding the LOC containing only documentation, i.e. comments) and complexity metrics for the classes, methods and parameters of the component, as well as their coupling and cohesion. It should be noted that the application of these metrics will be different for white-box and black-box components.
- **Quality**: generally, it is difficult to assess the quality of code in software engineering. This may often be subjective and inaccurate. In the narrow understanding of quality in this first version, it can be assessed via four attributes: the *number of bugs*, the *number of tests performed* (and their coverage and outcome), *availability of test cases*, provided along with the component, and an *independent rating* and certification. The first two attributes can only be collected through the lifetime of the component and may not be available. They are generic, objective and quantitatively measurable values. The availability of test cases is a context-based, subjective and qualitative metric. The best option would be to provide ready-to-use test cases which fit to the testing environment of the developer. An ordinal scale set by an expert seems reasonable here. A rating can be provided by experts or by other developers who have already reused the component ("wisdom of the crowd"). Such a rating can be an ordinal value, which is subjective, context-based and qualitative.
- **Maintainability**: The difference between maintainability and adaptability of a component is basically the perspective, the former is more concerned with the source code of the component while the latter is focused on its interface. Otherwise, the idea of both is how easy it is to adjust the component to a new context and hence, metrics related to the maintainability of the component are also included in the adaptability metric below. Therefore, the preliminary maintainability metric presented here will include only one additional aspect: the availability of the source code (as available for white-box components). A boolean metric is thus sufficient for this calculation. In the long run it makes sense to incorporate more detailed characteristics such as changeability etc. from the maintainability research community. However, the effect of the metrics used there (such as LOC, Cyclomatic Complexity, Volume etc.) are not yet well understood so that their impact on reusability is also not clear.
- **Price**: A generic, objective and quantitative metric, expressed through a predefined currency (conversions between currencies are possible).
- **Adaptability**: one important aspect of the adaptability is the programming language, and another is the availability of appropriate methods and interfaces for adapting the component [40]. The first aspect is context-dependent, subjective and qualitative. The possible values are: same programming language of the component and the context of the developer, different language with available and suitable component adapters, different programming language with no available suitable adapters. Again, these values have to be placed on an ordinal scale by an expert, while considering the similarity of the programming languages (e.g., it may be easier to adapt a C component to C++ then to Java). The second aspect should be addressed by some of the metrics in chapter 3 – the adaptability metric Rate of Component Customizability (RCC) from the metric suite of Washizaki et al. [16] seems useful in this case. The different applicability of adaptability metrics to white-box and black-box components has to be considered here, since the latter lack the possibility to change their code.
- **Reuse**: can be determined by the amount and frequency of reuse. Both are generic, objective and quantitative metrics. The amount is the overall

number of reuses of the component, and the frequency is the number of reuses for a certain period of time (e.g., the last week, month, year etc., or since it is available).

As stated before, these metrics are currently only suggestions for quantifying the reusability measurement and could be used as a starting point for quantitative empirical research. Their completeness and accuracy in measuring the reusability characteristics have to be empirically validated and synchronized with insights from other communities (such as those investigating software quality, complexity or maintainability).

C. Discussion

This chapter presented a holistic analytical approach for assessing the reusability of software components in an ad-hoc reuse scenario. The literature survey, which was carried out in the beginning of this research, did not identify any other publication that has conducted such extensive analysis of this topic. The GQM approach was used as a formalization technique for the analysis in order to increase its expressiveness. It was argued that, in order to identify appropriate and reasonable metrics, a reusability requirements model and a reusability measurement model have to be defined first. This also corresponds to the guidelines of the GQM approach.

Clearly, the major drawback of this analysis is the missing empirical validation of the proposed measurements and metrics, since the usefulness and practicability of the suggested models can only be proven by conducting empirical case studies and statistically significant tests using real-life data from existing libraries for reusable components. Therefore, the next logical step would be to implement a reusability model based on these metrics by following the guidelines provided in the previous subsections, and to adjust the calculation model until it satisfies the needs of practitioners. It is also possible that the calculation model has to be adjusted for different implementation scenarios – e.g., for implementation in a company-internal reuse library and a freely available online library. The metric suites described in chapter III should be the first source to look into when searching for alternatives or extensions.

Additionally, empirical investigations need to establish thresholds that the reusability values of the components have to beat. These thresholds should reflect the effort that a developer is willing to invest in the case of developing the functionality in-house (similar to the idea of Halstead’s Effort metric [16]). However, this is a non-trivial task, since the abstract and technical characteristics of the software component (included in the reusability measurement model) need to be translated into time, cost or effort values (e.g., “How much effort will be needed to reuse a component in a particular context, if its complexity has the value *X*?”). Obviously, a lot of further research is needed in this area, but since this clearly creates a considerable amount of effort (that has not even been spent for most other software metrics so far), we are forced to limit ourselves on merely sketching the idea of this model for the time being.

Another issue arises from the white-box and black-box nature of the components. If there is a mix of such components in the consideration set, it might become difficult to compare their reusability. This is an issue of the granularity of measurement – the object of measurement for some of the component’s characteristics (e.g., the complexity or adaptability) is different. In the white-box case, these metrics can be calculated on the basis of the whole component, and in the black-box case, they can be calculated based only on the parts of the components made available to the developer – usually the interfaces with their methods and attributes. Further research is needed to define guidelines for comparing the reusability of white-box and black-box components.

In general, we believe that the reusability models defined here will bring more clarity and better structure to reusability research and have the potential to become a new starting point when it comes to assessing reusability. Once this field has made further progress, it becomes more likely that practicing reuse of software components will increase and be more efficient.

V. CONCLUSION & FUTURE WORK

In this paper we have surveyed the current state of research on software reusability: available metrics for the reusability assessment of code in the object-oriented and component-based software development were presented and evaluated. Moreover, a revised comprehensive definition for the reusability of software components was proposed by following a structured analytical approach.

We have identified that the reusability of a software component is a context-specific characteristic that can vary in different application scenarios. This is an important aspect, which affects the definition of reusability and the implementation of its measurement. The reusability of a software component depends on various non-functional characteristics while fulfilling functional requirements (i.e. providing the desired functionality) is a prerequisite for assessing the reusability at all. It has become evident that reusability is a highly complex characteristic and its quantitative assessment is a non-trivial problem (as is the case for most other software quality characteristics). It may not be possible to fully automate the calculation in the near future so that human intervention may always be necessary.

The proposed software reusability measurement models and metrics in this paper still lack empirical validation so that is a logical next step in reusability research that should be addressed in the future. Moreover, the specific issues of setting practical threshold values when assessing reusability and comparing reusability of different components have to be addressed by researchers. Otherwise, it will be difficult to implement the model in practice. If future research succeeds to overcome these open issues in determining software reusability, it is likely that it will be a major step towards a wider adoption of the component reuse paradigm by both academia and business, which in turn can be seen a cornerstone for the further improvement of recently spreading software search engines and component marketplaces.

REFERENCES

[1] C.W. Krueger, Software reuse, ACM Computing Surveys, vol.24, June 1992, pp.131-183.

[2] M.D. McIlroy, Mass produced software components, Software Engineering: Report on a conference by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968, Naur, P., Randell, B., Eds., NATO Scientific Affairs Division, Brussels, Belgium, 1969, pp.138–150.

[3] R. Prieto-Diaz, Status report: software reusability, IEEE Software, vol.10, May 1993, pp.61-66.

[4] M. Griss, Software reuse: From library to factory., IBM Systems Journal, vol. 32, 1993, pp. 548--566.

[5] O. Hummel and C. Atkinson, Using the Web as a Reuse Repository, Reuse of Off-the-Shelf Components, Lecture Notes in Computer Science, vol. 4039, Springer, 2006, pp.298-311.

[6] A. Ampatzoglou, K. Apostolos, G. Kakaronzzos, and I. Stamelos, An Empirical Evaluation on the Reusability of Design Patters and Software Packages, Journal of Systems and Software, vol. 86, Dec. 2011, pp. 2265-2283.

[7] O. Hummel, W. Janjic, and C. Atkinson, Code Conjurer: Pulling Reusable Software out of Thin Air, IEEE Software, vol.25, Sep./Oct. 2008, pp. 45-52.

[8] G. Kakarontzas and I. Stamelos, Component Recycling for Agile Methods, Sev-enth International Conference on the Quality of Information and Communications Technology (QUATIC), 29 Sept. 2010 – 2 Oct. 2010, pp.397-402.

[9] W.B. Frakes and K. Kang, Software reuse research: status and future, IEEE Transactions on Software Engineering, vol.31, July 2005, pp.529-536.

[10] T.C. Jones, Reusability in Programming: A Survey of the State of the Art, IEEE Transactions on Software Engineering, vol.SE-10, Sept. 1984, pp.488-494.

[11] W.B. Frakes and C. Terry, Software reuse: metrics and models, ACM Computing Surveys, vol. 28, June 1996, pp.415-435.

[12] A. Sharma, R. Kumar, and P.S. Grover, A Critical Survey of Reusability Aspects for Component-Based Systems, World Academy of Science, Engineering and Technology, vol. 33, 2007, pp.35-39.

[13] A. Khoshkbarforoushha, P. Jamshidi, and F. Shams, A metric for composite service reusability analysis, Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics (WETSoM '10), ACM, New York, USA, 2010, pp.67-74.

[14] T. Elr, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice-Hall, 2005.

[15] H. Washizaki, H. Yamamoto, and Y. Fukazawa, A Metrics Suite for Measuring Reusability of Software Components, Proceedings of the 9th International Symposium on Software Metrics (METRICS '03), IEEE Computer Society, Washington, DC, USA, 2003, pp.211-223.

[16] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2002.

[17] W.B. Frakes and S. Isoda, Success Factors of Systematic Reuse, IEEE Software, vol. 11, Sep./Oct. 1994, pp.14-19.

[18] Y. Kim and E.A. Stohr., Software reuse: survey and research directions, Journal of Management Information Systems - vol.14, March 1998, pp.113-147.

[19] D. Merkl, Self-Organizing Maps And Software Reuse (book chapter), Compu-tational intelligence in software engineering, Pedrycz, W., Peters, J.F. (eds.), Singapore, World Scientific, 1998, pp.65-95.

[20] O.P. Rotaru and M. Dobre, Reusability Metrics for Software Componenents, ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'05), Washington DC, USA, 2005, pp.24-31.

[21] J. Poulin, Measuring software reusability, Proceedings of the International Conference on Software Reuse: Advances in Software Reusability, 1-4 Nov. 1994, pp.126-138.

[22] G. Caldiera and V.R. Basili, Identifying and qualifying reusable software components, IEEE Computer, vol.24, Feb. 1991.

[23] M. Halstead, Elements of Software Science, Amsterdam: Elsvier North-Holland, Inc., 1977.

[24] T.J. McCabe, A Complexity Measure, IEEE Transactions on Software Engineering, vol. 2, Sept. 1976, pp. 308-320.

[25] J. Barnard, A new reusability metric for object-oriented software, Software Quality Journal, vol. 7, Jan. 1998, pp.35-50.

[26] Y. Mao, H. Sahraoui, and H. Lounis, Reusability Hypothesis Verification using Machine Learning Techniques: A Case Study, Proceedings of the International Conference on Automated software engineering, IEEE, 1998, pp.84-93.

[27] Y. Lee and K.H. Chang, Reusability and maintainability metrics for object-oriented software, Proceedings of the 38th annual on Southeast regional con-ference (ACM-SE 38), ACM, New York, NY, USA, 2000, pp.88-94.

[28] E.S. Cho, M.S. Kim, and S.D. Kim, Component Metrics to Measure Component Quality, Proceedings of the Eighth Asia-Pacific on Software Engineering Con-ference (APSEC '01), IEEE Computer Society, Washington, DC, USA, 2001, pp.419-426.

[29] L.H. Etzkorn, W.E. Hughes Jr., and C.G. Davis, Automated reusability quality analysis of OO legacy software, Information and Software Techn., vol.43, 2001, pp. 295-308.

[30] S. Bhattacharya and D.E. Perry, Contextual reusability metrics for event-based architectures, Intern. Symp. on Empirical Software Engineering, 17-18 Nov. 2005, pp.459-468.

[31] G. Gui and P.D. Scott, New Coupling and Cohesion Metrics for Evaluation of Software Component Reusability, Proc. of the Intern. Conf. for Young Computer Scientists, 2008, pp.1181-1186.

[32] N. Gill and S. Sikka, Inheritance Hierarchy Based Reuse & Reusability Metrics in OOSD, International Journal on Computer Science and Engineering (IJCSE), vol.3, June 2011, pp.2300-2309.

[33] M.A.S. Boxall and S. Araban, Interface Metrics for Reusability Analysis of Components, Australian Software Engineering Conference (ACWEC'04), Melbourne, Australia, 2004, pp.40-50.

[34] A. Sharma, P.S. Grover, and R. Kumar, Reusability assessment for software components, SIGSOFT Software Engineering Notes, vol.34, No.2, February 2009, pp.1-6.

[35] V.R. Basili, G. Caldiera, and H.D. Rombach, The Goal Question Metric Ap-proach, Encyclopedia of Software Engineering, vol.1, New York, John Wiley & Sons, Inc., Sept. 1994, pp.528-532.

[36] S. Becker, A. Brogi, I. Gorton, S. Overhage, A. Romanovsky, and M. Tivoli, Towards an engineering approach to component adaption. In Architecting Systems with Trustworthy Components, Springer, 2006, pp. 193–215.

Deriving DO-178C Requirements Within the Appropriate Level of Hierarchy

Jamie P. White
Department of Computer Science
University of North Dakota
Grand Forks, USA
Jamie.white@gmail.com

Hassan Reza
Department of Computer Science
University of North Dakota
Grand Forks, USA
reza@aero.und.edu

***Abstract*— In this paper, a set of criterion is proposed that assists an engineer in placing a derived requirement as defined by DO-178C in the proper document within the requirement document hierarchy. The proper documentation of derived requirements has historically posed some issues when it comes to requirements-based testing. For one thing, if the derived requirements are inappropriately documented, then it will be very difficult to establish traceability between individual requirements to the elements of design, implementation, and verification. Consequently, the lack of correlation between elements of requirements, design, code, and verification can jeopardize the safety of systems because it will be impossible to establish forward and backward traceability. To this end, the proposed criteria discussed in this work attempts to improve the visibility of derived requirements to prevent the unwanted consequences of masking required information from developers.**

Keywords-requirement traceability; requirements analysis; safety critical systems; RTCA/DO-178C; software testing; software design

I. Introduction

An aircraft system is a complex system that is composed of a hierarchy of subsystems, which are further decomposed into software and hardware elements. A set of requirement documents describe each level of this requirement hierarchy. The highest layer, the system requirements, specifies the observable requirements at the system level (e.g., The System shall display total fuel quantity on the FMS Departure page). These very high level requirements are allocated to specific subsystems such as a Flight Management System (FMS) (e.g., FMS shall display total fuel quantity in either pounds or kilograms). Subsystem requirements are then further allocated to software and hardware high-level requirements (HLRs) (e.g., FMS shall display total fuel quantity in kilograms when metric units are selected). These HLRs can then be decomposed into low-level requirements (LLRs) at which point they should be specific enough to implement in hardware or software. Figure 1 illustrates a hierarchy of requirements starting at the system level that is decomposed into subsystem requirements and then further decomposed into hardware and software requirements.

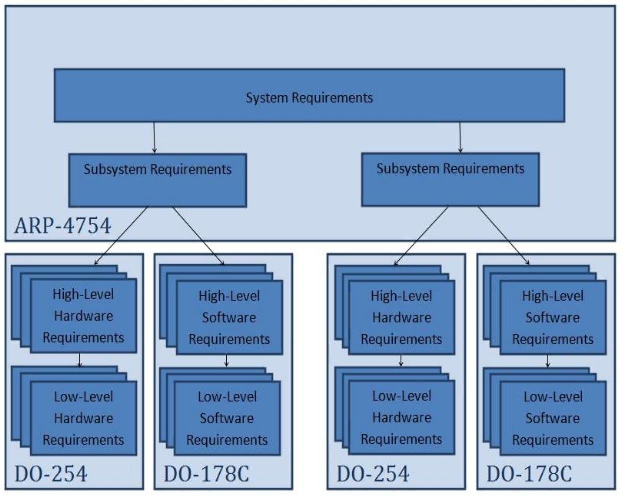

System Requirements
Subsystem Requirements
Subsystem Requirements
ARP-4754
High-Level Hardware Requirements
Low-Level Hardware Requirements
DO-254
High-Level Software Requirements
Low-Level Software Requirements
DO-178C
High-Level Hardware Requirements
Low-Level Hardware Requirements
DO-254
High-Level Software Requirements
Low-Level Software Requirements
DO-178C

Figure 1. Example of a system requirement document hierarchy

High-level requirements do not always contain sufficient details to describe the underlying requirement documents. As such it is necessary to create derived requirements (DR). A DR as defined by DO-178C [10] is a requirement that is not directly traceable to a higher-level source; it is inferred or deduced from a specific source/user. As an example of a derived requirement for ABC system can be read as "The ABC DataFusion subsystem shall write position, velocity, and maneuverability data received from Radar data signal processing site 1 to external storage". Such low-level details may be outside the scope of the SW-HLR document.

Low-level requirements are implementation of high-level requirements; they can be generated differently by different engineers but having the same functionalities. Low level requirements may then be implemented by different programmers in totally different ways, but yet representing the same functionalities [18].

An example of corresponding low-level requirements for ABC system can be read "The ABC DataFusion subsystem shall read the position, velocity, and maneuverability data received from the Radar Subsystem every 60 seconds".

As DO-178C requires the existence of source code is directly traceable to a requirement, it will then become necessary to derive such requirements in a low-level software requirements (SW-LLR) document. Figure 2 shows an example of a software derived requirement (SW-DR) that is derived within a SW-LLR document.

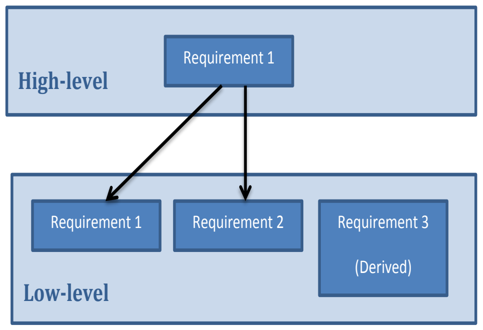


Figure 2. Example of a derived requirement

Deciding at what level a requirement should be derived at is generally done on a case by case basis (e.g., is it more appropriate to derive the requirement at the SW-HLR requirement document versus the SW-LLR document?). Such a decision could have unintended consequences and cost by hiding information applicable to other stakeholders.

As an example, let us assume that a SW-HLR document exists for a multi-threaded FMS application. From the SW_HLR document, multiple SW_LLR documents are created and categorized by independent functions worked on by multiple software engineering teams. In one of the SW_LLR documents a DR is intoduced that impacts a shared resource among the other software functions. Once implemented the other software functions could exhibit unpredicatable behaviors or defects, which may lead to additional time spent debugging the issue. If the defect is identified late in the software development process, then the cost for fixing the defect will become increasing expensive [13].

The paper is organized as follows. Section 2 provides an overview of the certification agencies and DO-178C. Section 3 contains related work. Section 4 defines criteria to serve as a guideline for the appropriate placement of a derived requirement regardless if it impacts safety aspect of a system. Such criteria are based on nonfunctional requirements such as operability, safety, reliability, observability, etc. Criteria for functional requirements are also defined, which include interfaces and configurationable elements. Section 5 shows a simple example of requiremnents for the display of fuel quantity in an aircaft cockpit. Finally Section 6 discusses a conclusion and future work.

II. BACKGROUND

Certification agencies such as the Federal Aviation Administration (FAA), Transport Canada, and the European Aviation Safety Agency (EASA) rely on industry standards to serve as guidelines on how to create aircraft systems that are certifiable, or trusted for use in airborne applications. ARP-4754 [7] serves as the guideline for system and subsystem processes, DO-254 [8] for hardware processes, and DO-178C [10] for software processes.

Avionics systems have contained software since the 1970s. As the certification of avionic systems increased in complexity, additional methods were necessary to achieve the same level of assurance as hardware based systems [11].

Radio Technical Commission for Aeronautics (RTCA) and European Organization for Civil Aviation Equipment (EUROCAE) formed committees to create common certification criteria for software development [11]. The works from these committees were merged, which led to RTCA publishing DO-178 [17] and EUROCAE publishing ED-12 with both documents containing identical content [11]. DO-178 categorized systems as critical, essential and non-essential and defined the rigor needed to develop software to each level [16].

DO-178C, published in December 2011, is the recent standard, which describes the processes in the creation of flight critical software. These processes outline the stages which include the creation of multiple levels of requirements, design, implementation and verification. From the previous version of DO-178B, published in 1992, little has changed from this core document. The changes mostly consist of fixing errors and inconsistencies, word improvements, and adding several clarifications and objectives [11]. The bulk of the work was the creation of supplement documents, referred to by DO-178C, that provide guidance on model-based development, tool qualification, object-oriented technology, and formal methods.

DO-178C describes that system level requirements are decomposed into SW-HLRs. SW-HLRs are defined by DO-178C as being developed from the analysis of system requirements, safety-related requirements and system architecture [10]. SW-LLRs are then created, which further decompose the SW-HLRs. SW-LLRs are software requirements that were developed from SW-HLRs or are derived, which describe in sufficient detail to allow source code to be implemented without additional information [10]. The role of SW-HLRs is to describe the ‘what’ and for the SW-LLRs to describe the ‘how’ [5][6]. SW-DRs can be SW-HLRs or SW-LLRs and are not directly traceable to higher-level requirements. SW-DRs are used to specify additional behavior beyond what is defined in the higher level requirements [10].

Software is implemented from the SW-LLRs and is verified from requirements based testing that verify the correctness of SW-HLRs and SW-LLRs. Finally, traceability is required to be maintained at each stage (i.e., SW-LLRs are traceable to SW-HLRs, code is traceable to SW-LLRs and verification is traceable to both SW-LLRs and SW-HLRs).

The certification standards specify that traceability, both forward and backward, is needed for the allocated requirements at each requirement document level. Requirements traceability describes the ability to describe and follow a requirement in both a forward and backward direction [1]. A requirement management tool such as IBM Rational DOORs supports linking between levels of requirements through the use of requirement attributes [2].

In DOORs, a requirement document is called a module. One way to organize a project would be to create a separate DOORs module for each level of the requirement hierarchy (e.g., system requirements, subsystem requirements, SW-HLRs, SW-LLRs). A DOORs link can then be used to connect multiple requirements. A requirement can be linked with another requirement that exists in the same module or a different module. DOORs links are directional and are categorized as 'in-links' or 'out-links'. In the context of DO-178C, the practice of linking is done to show decomposition of requirements. As an example, a SW-HLR could contain in-links from one or more SW-LLRs and an out-link to a subsystem requirement. Using links in DOORs, forward and backward traceability is maintained (e.g., for a specific SW-HLR, traceability information exists for the source of the SW-HLR as well as the SW-LLRs that decompose it).

Finally, DO-178C requires additional processes for SW-DRs. Rationale must be documented to support the existance of a SW_DR When using DOORs, this documentation can be captured as an attribute attached to the SW-DR. Additional processes must be created to provide DRs to the system process. Typically, this is done by having a safety engineer review all SW-DRs and its rationale to determine if there is an impact to safety (i.e., could the SW-DR cause loss of function resulting in additional pilot workload or provide misleading information to the pilots). SW-DRs that are determined to impact safety are captured by the system safety assessment process. Most DRs typically do not impact safety so these requirements are larged ignored at the system level. Even if a SW-DR requirement that impacts safety is tracked at the system level, these requirements are not easily traced to other functional requirements of the system. Hence, it is still important that SW-DRs that impact safety are derived at the appropiate level.

III. RELATED WORK

Since the 1970s, product organizations have used requirements traceability in order to have complete and consistent information about the product being built [2]. Since this time, much work has been done studying requirements traceability and traceability tool support [3]. Others have proposed frameworks for the organization of traceability information [4]. Studies have been conducted to understand the benefits and costs of traceability [1][2].

In 2011, RTCA (Radio Technical Commission for Aeronautics), Inc. published DO-178C "Software Considerations in Airborne Systems and Equipment Certification", which serves as a guide for the creation of airborne software using traditional methods [5], and which typically utilize the C and ADA programming languages. DO-331[ref] was also published in 2012 which describes how to implement software using model-based development (MBD) [6]. Many companies selling aviation products follow DO-178C or the previous release of DO-178B to prove airworthiness of their software elements. Regulatory agencies such as the Federal Aviation Administration (FAA), Transport Canada, and the European Aviation Safety Agency (EASA) audit these software elements seeking compliance to DO-178C.

Nonfunctional requirements (NFRs) play an important role in the creation of software architecture and are blamed for system re-engineering when not considered when designing the architecture [14]. In an avionics environment, certain NFRs play important role, which include security, maintainability, safety, availability, integrity, and schedulability [15]. These NFRs should be considered when creating the software architecture.

IV. METHOD

In this section, 6 points of criterion have been suggested for the proper placement of derived requirements within a requirement document hierarchy. The criteria are intended to cover certain special cases such as requirements that specify external interfaces, externally observable behavior, configurable elements, etc. Engineers should consider each criterion to determine the most appropriate placement for a derived requirement to ensure information is not hidden from the stakeholders.

As an example, system or subsystem engineers will have little visibility of requirements defined in a SW-HLR or SW-LLR document. If a requirement is derived in a SW-HLR or SW-LLR document, there would be no link for the system or subsystem engineer to follow from their requirements documents. This could result in important information being hidden. On the other hand, putting large amounts of irrelevant details in a high level requirements document could result in the document becoming unmanageable.

The end goal is to place requirements in the requirements document that provides the most visibility to the stakeholders while preserving the scope of the document. Hence, there is a fine line between putting too much information in a high-level requirements document and providing an appropriate amount of visibility to stakeholders. In addition, the CAST-15 position paper provides guidance that for software requirements a high-level requirements document should describe the "what" and a low-level requirements document should describe the "how" [5]. Placing the derived-requirements in the correct requirements document will improve traceability by making the requirements more visible to the stakeholders. Some of the consequences of poor traceability include lower changeability and higher maintenance costs [3].

A. *Requirements that specify an external interface*

Software and hardware elements contain external interfaces. Software elements provide *APIs* allowing communication with other software elements. Hardware can contain external interfaces for data buses (e.g., ARINC 429 connectors) power adapters or other form factors.

The introduction of the derived requirement for such features would depend on the scope or visibility of these interfaces (i.e., the software functions or software applications that have access to these interfaces). For instance, if a software element provides a service to software elements in other subsystems, it would be appropriate to create a high-level parent requirement in the systems requirement document (e.g., "The system shall provide an interface to collect fault information") Such a requirement would be further decomposed in the subsystem and software requirements document until it is specific enough to be implemented. If the software element provides an API that is only visible to software elements in the same subsystem, the parent requirement should be derived in the subsystem requirement document (e.g., FMS shall provide an interface to store FMS faults to the FMS maintenance log).

B. Requirements that describe externally observable behavior

Externally observable behavior includes any set of inputs that yields an output that can be noticeable to the user. The set of requirements in the systems or subsystems documents should provide high-level detail for the capabilities and functional requirements that are observable to the user. The user (i.e., pilot) must have a complete understanding for the controls of the aircraft and in turn the aircraft must respond in a deterministic fashion. Requirements must exist to capture all behavior. Requirements should also be added to cover any corner or fault conditions. For instance, if a FMS software element is placed in a fault state, which produces an error message visible to the user, there should be a basis for that requirement at the system level (e.g., FMS shall display "FMS Unavailable" within the target window when unavailable). Such a requirement at the system level would be more appropriate than at the subsystem level since it is observable to the user. Information should not be 'hidden' from the user. In addition, test procedures should be written based on system or subsystem requirements for anything observable at the system level.

C. Requirements that describe configurable elements

Many components of an aircraft are configurable to support reuse on different aircraft types or for selectable options for a specific aircraft type.

This criterion depends on which level the element needs to be configurable. For example, an aircraft manufacture may sell a common avionics package to its customers, which are allowed to purchase additional features. Features may be enabled through the use of licenses. The basis of license management should be defined at the system level with requirements on how to configure the system. Alternatively, software elements may be reused on many aircraft systems, which contain a single configuration per aircraft type. Describing the configurable elements at a higher level would add no value. As an example, the owner of an aircraft may subscribe to services such as Graphical Weather Radar that would require a key to enable. Maintenance personnel could enable the feature by entering the license key in one of the avionics application maintenance pages. System requirements should capture this capability. Another example is a Radio Interface Unit (RIU), which may be configurable to support multiple aircraft types to support reuse. At the system level, there would be no need to capture such requirements since it was a design decision to make the RIU configurable to support reuse. The implementation details should in turn be hidden from the user.

D. Requirements that describe performance, schedulability, and design margin

For software, especially within an Integrated Modular Avionics (IMA) environment, system requirements describing performance, schedulability, and design margin (e.g., maximum allowed latency, CPU utilization, memory usage, etc) should be defined. This is required so that when the aircraft is integrated each application has sufficient resources. Additional capacity should be left for future growth. In addition, subsystems requirements, SW-HLRs, and SW-LLRs could have derived requirements to account for future growth, reuse, task scheduling, etc. For example, derived SW_HLRs describing how often specific threads should run could be defined. SW_LLRs could contain derived requirements specifying size of buffers.

E. Requirements that describe security features

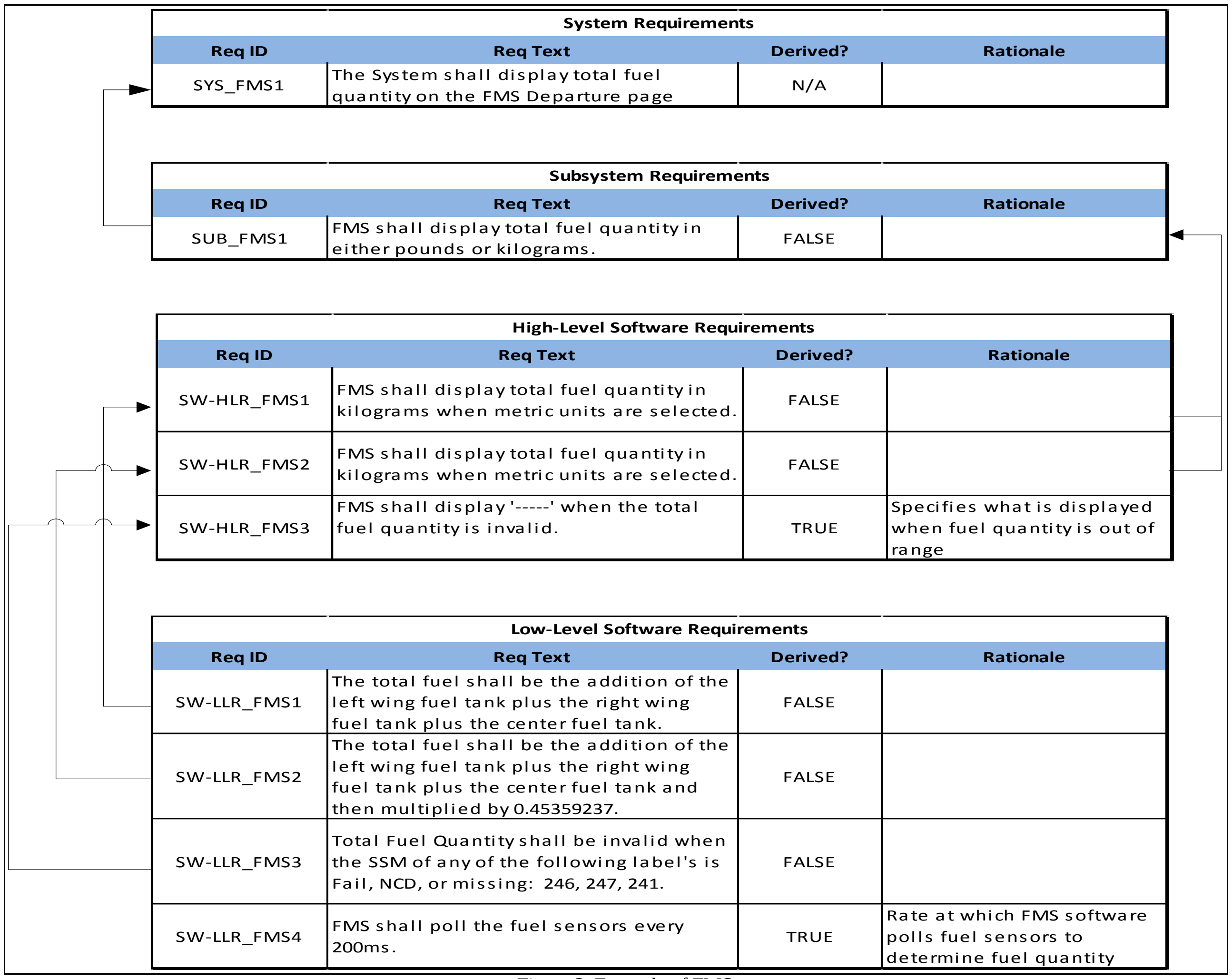
Like external interfaces, requirements describing security features depend on scope. For example, many aircrafts now provide support for ETHERNET for its passengers [15]. The mechanism that isolates ETHERNET traffic from other aircraft communications should be done at the system level or subsystem level. An example of a derived SW_HLR, databases used by an FMS could be encrypted to preserve propriety information. As it is not applicable to the system that such security features are implemented, it would not be appropriate to define such security features in a higher-level document.

F. Requirements that describe system safety availability constraints

Safety requirements concerning the development process include efforts to ensure correctness of the design and correctness of requirements in terms of safety where availability describes the continuity of a function [15]. System safety/availability constraints are expressed in DO-178C through a Design Assurance Level (DAL). DAL A is the most stringent and is designated to functions that could contribute to a catastrophic failure condition (e.g., failure could cause a crash). DAL E is at the other end of the spectrum in which a failure has no impact on the safety of the aircraft. Safety assessments for the aircraft should be conducted and defined at the system level as specified by ARP4754 [7]. Hence, requirements expressing DAL levels should be contained in system documents and not derived in software documents. Derived requirements at any level need to be reviewed by a safety engineer to ensure there is no negative effect on safety or availability of the aircraft (e.g., a failure could cause a loss of the primary flight display).

V. EXAMPLE

Figure 3 illustrates an example of FMS requirements responsible for displaying total fuel quantity beginning at the system level. This example demonstrates some of the difficulty in determining the correct place to capture DRs. SYS_FMS1, a system requirement, is decomposed into one subsystem requirement, SUB_FMS1. This decomposition is represented by an ‘in-link’ into SYS_FMS1. The ‘in-link’ is representted by a directional arrow from SUB_FMS1 to SYS_FMS1.



System Requirements			
Req ID	**Req Text**	**Derived?**	**Rationale**
SYS_FMS1	The System shall display total fuel quantity on the FMS Departure page	N/A	

Subsystem Requirements			
Req ID	**Req Text**	**Derived?**	**Rationale**
SUB_FMS1	FMS shall display total fuel quantity in either pounds or kilograms.	FALSE	

High-Level Software Requirements			
Req ID	**Req Text**	**Derived?**	**Rationale**
SW-HLR_FMS1	FMS shall display total fuel quantity in kilograms when metric units are selected.	FALSE	
SW-HLR_FMS2	FMS shall display total fuel quantity in kilograms when metric units are selected.	FALSE	
SW-HLR_FMS3	FMS shall display '-----' when the total fuel quantity is invalid.	TRUE	Specifies what is displayed when fuel quantity is out of range

Low-Level Software Requirements			
Req ID	**Req Text**	**Derived?**	**Rationale**
SW-LLR_FMS1	The total fuel shall be the addition of the left wing fuel tank plus the right wing fuel tank plus the center fuel tank.	FALSE	
SW-LLR_FMS2	The total fuel shall be the addition of the left wing fuel tank plus the right wing fuel tank plus the center fuel tank and then multiplied by 0.45359237.	FALSE	
SW-LLR_FMS3	Total Fuel Quantity shall be invalid when the SSM of any of the following label's is Fail, NCD, or missing: 246, 247, 241.	FALSE	
SW-LLR_FMS4	FMS shall poll the fuel sensors every 200ms.	TRUE	Rate at which FMS software polls fuel sensors to determine fuel quantity

Figure 3. Example of FMS

SUB-FMS1 is decomposed into two SW-HLRs, SW-HLR-FMS1 and SW-HLR_FMS2. SW-HLR_FMS3 is a SW-HLR DR since it is not traceable to SYS-FMS1. Philisophically, SW-HLR_FMS3 is a DR since it does not decompose SUB-FMS1 but instead describes the behaviour when fuel quantity cannot be displayed. In Figure 3, it is apparent that SW-HLR_FMS3 is a DR since it contains no ‘out-link’ to a higher-level source. In addition, it is a common practice to create a requirement attribute specifying if the requirement is derived or not. This derived attribute is shown in the “Derived?” column for each requirement. Since HLR-FMS3 is a DR, it is also required by DO-178C to include rationale, which is included in the “Rationale” column.

SW-HLR_FMS1, SW-HLR_FMS2, and SW-HLR_FMS3 are further decomposed into SW-LLR_FMS1, SW-LLR_FMS2, and SW-LR_FMS3. Finally SW-LLR_FMS4 and SW-LLR_FMS5 are derived requirements since they are not traceable to a higher source.

Let us now review each of the derived requirements. SW-HLR_FMS3 is an interesting example of a requirement that contains externally observable behaviour’ as described in Section 4. The caveat is, this externally observable behaviour is not based on an explicit input by the user, but instead a minor fault condition.

Many would argue that it would not be appropriate to describe a requirement such as SW-HLR_FMS3 in the systems requirement document, as it would be too detailed. SW-HLR_FMS3 merely captures the behaviour for a corner fault conditon that should not occur in normal operation.

Using the same argument, SW-HLR_FMS3 may also not be appropriated in the subsystem requirement document. There would be value in including a requirement such as SW-HLR_FMS3 in the subsystem requirements document for other reasons. Through subsystem testing, it is quite common to introduce faults into the system that would reach such fault conditions. By not having the information from SW-HLR_FMS3 stored or otherwise traced to the subsystem document, a subsystem engineer may have diffiuclty to understand why the fault occurred. The engineer may need to consult the domain experts or search for this information in software requirement documents.

SW-LLR_FMS4 is another interesting example of a DR, which relates to a requirement that describes performance, schedulability, and design margin (see Section 4). SW-LLR_FMS4 describes how often the FMS task should read fuel quantity information from the fuel sensors. As long as the value chosen does not impact system performance (e.g., sending data requests every millisecond which could overload a data bus), a detail such as this will be insignificant at the system or subsystem level. Since SW-LLR_FMS4 is a DR, a safety engineer is required to review this requirement to determine if it can impact the safety of the aircraft.

For this DR, it is assumed that the fuel sensor information is published throughout the system at some constant rate. Therefore, there should be no real stakeholders of this information. Hence, this requirement could be considered an implementation detail and placed in a SW-LLR document.

On the other hand, if the fuel sensors were a shared resource among multiple FMS threads, there could be contention. In this case, it will be appropriate to put LLR_FMS4 in the SW-HLR document. For a more complicated example, consider if there are primary and secondary fuel sensors. Would it be considered an implementation detail to fail-over to the secondary fuel sensor data when the data from the primary fuel sensor is invalid or missing?

In summary, this work shows that it is not always a trivial problem to determine the proper placement for a DR. In terms of DO-178C, there is no explicit answer beyond putting the “what” in the SW-HLR and the “how” in the SW-LLR. Therefore, other aspects must also be considered such as ensureing traceability and visibility of information.

VI. CONCLUSION AND FUTURE WORK

This study proposes a set of guidance to for the optimal placement of derived requirements as defined by DO-178C. By deriving a requirement in a document that is too low-level may have the unwanted consequences of hiding information from stakeholders (e.g., engineers). This, in turn, may result in forward and backward traceability problem, which can compromise dependability of safety critical systems.

In future work, apart from creating additional detail in the criteria contained in this paper, validation of these criteria will be carried on. Another related topic for future work would be to create criteria to determine if a LLR correctly decomposes a HHR or if it should be considered a DR. There is more effort involved with the creation of DRs (e.g., creation of rationale, review with safety engineer, additional scrutiny, etc.). Because of this, engineers may be more prone to create a trace to a HLR (indicating decomposition) versus specifying as a DR when there is a gray area. Of course, this is especially dangerous (left uncorrected) as this would bypass a review by a safety engineer.

REFERENCES

[1] Gotel, O. C .Z. and Finkelstein, C. W., "An analysis of the requirements traceability problem," Requirements Engineering, 1994., Proceedings of the First International Conference on Software Engineering , pp. 94-101, April 1994.

[2] Kirova, V., Kirby, N., Kothari, D., and Childress, G., ”Effective requirements traceability: Models, tools, and practices,” Bell Labs Technical Journal, vol. 12, no. 4, pp. 143-157, 2008.

[3] Winkler, S. and Pilgrim, J., “‘A survey of traceability in requirements engineering and model-driven development,” Software & Systems Modeling, vol. 9, no. 4, pp. 529-565, 2010

[4] Ramesh, B. and Jarke, M., “Toward Reference Models for Requirements Traceability,” IEEE Transactions on Software Engineering, vol. 27, no.1, pp. 58-93, 2001.

[5] RTCA. DO-178B/ED-12B. Software Considerations in Airborne Systems and Equipment Certification. RTCA, 1992.

[6] Certification Authorities Software Team (CAST) Position Paper.; “CAST-15: Merging High-Level and Low-Level Requirements”, February 2003.

[7] Marques, J. C., Yelisetty, S. M. H., Dias, L. A. V., and da Cunha, A. M., "Using Model-Based Development as Software Low-Level Requirements to Achieve Airborne Software Certification," Information Technology: New Generations (ITNG), pp. 431-436, April 2012.

[8] “Guidelines for Development of Civil Aircraft and Systems”, EUROCAE ED-79A and SAE Aerospace Recommended Practice ARP 4754A, 2010.

[9] RTCA, 2000, DO-254: Design Assurance Guidance for Airborne Electronic Hardware, RTCA, Inc., Washington, DC.

[10] RTCA DO-178C—Software Considerations in Airborne Systems and Equipment Certification, December 2011.

[11] Qualtech Consulting Inc, “Summary of Difference Between DO-178B and DO-178C", http://www.faaconsultants.com/html/do-178c.html.

[12] Taylor, C., Alves-Foss J., and Rinker, B., “Merging Safety and Assurance: The Process of Dual Certification for Software.” Proceeding of Software Technolgy Conference (STC), Salt Lake City, UT, 2002.

[13] Chaar, J. K., Halliday, M. J., Bhandari, I. S., and Chillarege, R., “In-Process Evaluation for Software Inspection and Test,” IEEE Trans. Software Eng., vol. 19, no. 11, pp. 1055-1070, November 1993.

[14] Reza, H., Jurgens, D., White, J., Anderson, J., and Peterson, J., "An architectural design selection tool based on design tactics, scenarios and nonfunctional requirements," Electro Information Technology, 2005.

[15] Paulitsch, M., Ruess, H., and Sorea, M., “Non-functional Avionics Requirements,” Communications in Computer and Information Science, vol. 17, pp. 369–384, 2009.

[16] Johnson, L. A., "DO-178B, Software considerations in airborne systems and equipment certification", http://www.dcs.gla.ac.uk/~johnson/teaching/safety/reports/schad.html.

[17] Radio Technical Commission for Aeronautics, “DO-178 - Software Considerations in Airborne Systems and Equipment Certification”, Washington, United States, 1982.

[18] Hayhurst, K.,Veerhusen, D., Chilenski, J., and Rierson, L. A. Practical Tutorial on Modified Condition/Decsion Coverage. NASA/TM-2001-210876, 2001.

Abstract State Machines Mutation Operators

Jameleddine Hassine
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals, Dhahran, Kingdom of Saudi Arabia
jhassine@kfupm.edu.sa

***Abstract*—Mutation testing is a well established fault-based technique for assessing and improving the quality of test suites. Mutation testing can be applied at different levels of abstraction, e.g., the unit level, the integration level, and the specification level. Designing mutation operators represents the cornerstone towards conducting effective mutation testing and analysis. While mutation operators are well defined for a number of programming and specification languages, to the best of our knowledge, mutation operators have not been defined for the Abstract State Machines (ASM) formalism. In this paper, we define and classify mutation operators for the Abstract State Machines (ASM) formalism. The proposed ASM mutation operators are illustrated using examples written in the CoreASM language. Furthermore, we have developed a tool for automatic generation of mutants from CoreASM specifications.**

***Keywords-Mutation testing*; *specification*; *mutation operator*; *Abstract State Machines (ASM)*; *CoreASM*.**

I. Introduction

Mutation testing [1] is a well established fault-based testing technique for assessing and improving the quality of test suites. Mutation testing uses mutation operators to introduce small changes, or mutations, into the software artifact (i.e., source code or specification) under test. A mutant is produced by applying a single mutation operator, and for each mutant a test is derived that distinguishes the behaviors of the mutated and original artifact.

In a recent survey on the development of mutation testing, Jia and Harman [2] have stated that more than 50% of the mutation related publications have been applied to Java [3], Fortran [4] and C [5]. Although mutation testing has mostly been applied at the source code level, it has also been applied at the specification and design level [6][2]. Formal specification languages to which mutation testing has been applied include Finite State Machines [7][8][9], Statecharts [10], Petri Nets [11] and Estelle [12].

Fabbri et al. [7] have applied specification mutation to validate specifications based on Finite State Machines (FSM). They have proposed 9 mutation operators, representing faults related to the states (e.g., wrong-starting-state, state-extra, etc.), transitions (e.g., event-missing, event-exchanged, etc.) and outputs (e.g., output-missing, output-exchanged, etc.) of an FSM. In a related work, Fabbri et al. [10] have defined mutation operators for Statecharts, an extension of FSM formalism, while Batth et al. [13] have applied mutation testing to Extended Finite State Machines (EFSM) formalism.

Hierons and Merayo [9] have investigated the application of mutation testing to Probabilistic (PFSMs) or stochastic time (PSFSMs) Finite State Machines. The authors [9] have defined new mutation operators representing FSM faults related to altering probabilities (PFSMs) or changing its associated random variables (PSFSMs) (i.e., the time consumed between the input being applied and the output being received).

The widespread interest in model-based testing techniques provides the major motivation of this research. We, in particular, focus on investigating the applicability of fault-based testing (vs. scenario-based testing) to Abstract State Machines (ASM) [14] specifications. We aim at assessing and further enhancing the fault-finding effectiveness of test suites targeting ASM-based models.

While mutation operators are well defined for a number of FSM related paradigms such as EFSM, PFSM and Statecharts, to the best of our knowledge mutation operators have not been defined for the Abstract State Machines [14] paradigm.

This paper serves the following purposes:

- Provide a set of mutation operators for Abstract State Machines [14] formalism.
- Present a classification of the proposed mutation operators into three categories: ASM domain operators, ASM function update operators, and ASM transition rules operators.
- Present a tool for generating and validating ASM mutants.

The remainder of this paper is organized as follows. The next section provides an overview of the Abstract State Machines (ASM) [14] formalism. In Section III, we define and classify a collection of mutation operators for ASM paradigm. Section IV describes the ASM Mutation tool. Finally, conclusions are drawn in Section V.

II. Abstract State Machines

Abstract State Machines (ASM) [14] define a state-based computational model, where computations (runs) are finite or infinite sequences of states $\{S_i\}$ obtained from a given initial state S_0 by repeatedly executing transitions δ_i:

$$S_0 \xrightarrow{\delta_1} S_1 \xrightarrow{\delta_2} S_2 \quad \cdots \quad \xrightarrow{\delta_n} S_n$$

An ASM $\mathcal{A}$ is defined over a fixed vocabulary $\mathcal{V}$, a finite collection of function names and relation names. Each function name f has an arity (number of arguments that the function takes). Function names can be static (i.e., fixed interpretation in each computation state of $\mathcal{A}$) or dynamic (i.e., can be altered by transitions fired in a computation step). Dynamic functions can be further classified into:

- Input functions that $\mathcal{A}$ can only read, which means that these functions are determined entirely by the environment of $\mathcal{A}$. They are also called monitored.
- Controlled functions of $\mathcal{A}$ are those which are updated by some of the rules of $\mathcal{A}$ and are never changed by the environment.
- Output functions of $\mathcal{A}$ are functions which $\mathcal{A}$ can only update but not read, whereas the environment can read them (without updating them).
- Shared functions are functions which can be read and updated by both $\mathcal{A}$ and the environment.

Static nullary (i.e., 0-ary) function names are called constants while Dynamic nullary functions are called variables. ASM *n-ary* functions have the following form: f: T_1 x T_2 x …... $T_n \rightarrow$ T.

Given a vocabulary $\mathcal{V}$, an ASM $\mathcal{A}$ is defined by its program $\mathcal{P}$ and a set of distinguished initial states S_0. The program $\mathcal{P}$ consists of transition rules and specifies possible state transitions of $\mathcal{A}$ in terms of finite sets of local function *updates* on a given global state. Such transitions are atomic actions. A transition rule that describes the modification of the functions from one state to the next has the following form:

if *Condition* **then** <*Updates*> **endif**

where *Updates* is a set of function updates (containing only variable free terms) of form: $f(t_1,t_2,\ldots,t_n)$:= t which are simultaneously executed when *Condition* (called also *guard*) is true. In a given state, first, all parameters t_i, t are evaluated to their values, v_i, v, then the value of $f(v_1,\ldots,v_n)$ is updated to v. Such pairs of a function name f, which is fixed by the signature, and an optional argument $(v_1,\ldots,v_n)$, which is formed by a list of dynamic parameters value v_i, are called *locations*.

Example1: The following rule yields the update-set {(x, 2), (y(0), 1)}, if the current state of the ASM is {(x, 1), (y(0), 2)}:

$$\textbf{if} \quad (x = 1) \quad \textbf{then} \quad \begin{aligned} x &:= y(0) \\ y(0) &:= x \end{aligned}$$

In every state, all the rules which are applicable are simultaneously applied. A set of ASM updates is called *consistent* if it contains no pair of updates with the same locations, i.e., no two elements *(loc,v)* and *(loc,v')* with *v*≠*v'*. In the case of inconsistency, the computation does not yield a next state.

Example2: The following update set {(x, 1), (y, 3), (x, 2)}, is inconsistent due to the conflicting updates for x:

$$\begin{aligned} x &:= 1 \\ y &:= 3 \\ x &:= 2 \end{aligned}$$

For a detailed description of Abstract State Machines, the reader is invited to consult [15].

In what follows, we describe mutation operators for Abstract State Machines. Although, we illustrate the applicability of our approach using features and examples from *CoreASM* [16], our proposed mutation operators can be applied to any ASM-based language, thus maintaining the discussion generic.

III. Abstract State Machines Mutation Operators

We use the following guiding principles, introduced in [17], to formulate our mutation operators:

- Mutation categories should model potential faults.
- Only simple, first order mutants should be generated.
- Only syntactically correct mutants should be generated.

There exist several aspects of an ASM specification that can be subject to faults. These aspects can be classified into three categories of mutation operators:

1) ASM domain mutation operators.
2) ASM function update mutation operators.
3) ASM transition rules mutation operators.

Each category contains many mutation operators, one per a fault class.

A. ASM Domain Mutation Operators

A domain (called also *universe*) consists of a set of declarations that establish the ASM vocabulary. Each declaration establishes the meaning of an identifier within its scope. For example, the following *CoreASM* [16] code defines a new enumeration background *PRODUCT* having three elements (i.e., Soda, Candy, and Chips) and three functions *selectedProduct*, *price*, and *packaging*:

enum PRODUCT = {Soda, Candy, Chips}
function selectedProduct: → PRODUCT
function price: PRODUCT → NUMBER
function packaging: PRODUCT*PRODUCT → NUMBER

ASM domains/universes can be mutated by adding or removing elements. Table I shows examples of the following domain mutation operators:

- Extend Domain Operator (EDO): the domain is extended with a new element.

- Reduce Domain Operator (RDO): the domain is reduced by removing one element.
- Empty Domain Operator (EYDO): the domain is emptied.

Table I
EXAMPLES OF ASM DOMAIN MUTATION OPERATORS FOR *CoreASM* [16]

Mutation Operator	CoreASM Mutant S'
Extend Domain Operator (EDO)	enum PRODUCT = {Soda, Candy, Chips, Sandwich}.
Reduce Domain Operator (RDO)	enum PRODUCT = {Soda, Candy}.
Empty Domain Operator (EYDO)	enum PRODUCT = {}.

B. ASM Function Update Mutation Operators

A function update has the following form:

$$f(t_1,\ t_2,\ \ldots,\ t_n) := \text{value}$$

Depending on the type of operands, the traditional operators [4] such as Absolute Value Insertion (ABS), Arithmetic Operator Replacement (AOR), Logical Operator Replacement (LOR), Statement Deletion (SDL), Scalar Variable Replacement (SVR) etc., can be applied. In addition to these traditional mutation operators, we define:

- *Function Parameter Replacement (FPR)*: parameters of a function are replaced by other parameters of a compatible type.
- *Function Parameter Permutation (FPP)*: parameters of a function are exchanged.

Table II
EXAMPLES OF FUNCTION UPDATE MUTATION OPERATORS FOR *CoreASM* [16]

Mutation Operator	CoreASM Spec S	CoreASM Mutant S'
AOR	x := a + b	x := a - b
ABS	x := a + b	x := a + abs(b)
LOR	y := m and n	y := m or n
SDL	x := a + b	skip
SVR	selectedProduct:= Soda	selectedProduct:= Candy
FPR	price(Soda):=70	price(Candy):=70
FPP	packaging(Soda, Candy):= 1	packaging(Candy, Soda):= 1

Table II describes the proposed function update mutation operators.

C. ASM Transition Rules Mutation Operators

The transition relation is specified by guarded function updates, called *rules*, describing the modification of the functions from one state to the next. An ASM state transition is performed by firing a set of rules in one step.

1) Conditional Rule Mutation Operators:: The general schema of an ASM transition system appears as a set of guarded rules:

$$\textbf{if } \textit{Cond} \textbf{ then } Rule_{then} \textbf{ else } Rule_{else} \textbf{ endif}$$

where *Cond*, the guard, is a term representing a boolean condition. $Rule_{then}$ and $Rule_{else}$ are transition rules.

Many types of faults may occur on the guards of conditional rules [18]. Some of these faults include Literal Negation fault (LNF), Expression Negation fault (ENF), Missing Literal fault (MLF), Associative Shift fault (ASF), Operator Reference fault (ORF), Relational Operator fault (ROF), Stuck at 0(true)/1(false) fault (STF). Table III illustrates the mutation operators addressing the above fault classes. Furthermore, we define three additional conditional rule mutation operators:

- *Then Rule Replacement Operator (TRRO)*: replaces the rule $Rule_{then}$ by another rule.
- *Else Rule Replacement Operator (ERRO)*: replaces the rule $Rule_{else}$ by another rule.
- *Then Else Rule Permutation Operator (TERPEO)*: permutes the $Rule_{then}$ and the $Rule_{else}$ rules.

Table III
EXAMPLES OF CONDITIONAL RULE MUTATION OPERATORS FOR *CoreASM* [16]

Mutation Operator	CoreASM Spec S	CoreASM Mutant S'
LNO	**if** (a **and** b)	**if** (**not** a **and** b)
ENO	**if** (a **and** b)	**if not** (a **and** b)
MLO	**if** (a **and** b)	**if** (b)
ASO	**if** (a **and** (b **or** a))	**if** ((a **and** b) **or** a)
ORO	**if** (a **and** b)	**if** (a **or** b)
ROO	**if** (x >= c)	**if** (x <= c)
STO	**if** (a **and** b)	**if** (*true*)
TRRO	**if** a **then** *R1* **else** *R2*	**if** a **then** *R3* **else** *R2*
ERRO	**if** a **then** *R1* **else** *R2*	**if** a **then** *R1* **else** *R3*
TERPEO	**if** a **then** *R1* **else** *R2*	**if** a **then** *R2* **else** *R1*

2) Parallel and Sequence Rule Mutation Operators:: **Parallel Constructor:** If a set of ASM transition rules have to be executed simultaneously, a parallel rule is used:

$$\textbf{par } Rule_1 \ldots Rule_n \textbf{ endpar}$$

The update generated by this rule is the union of all the updates generated by $Rule_1$ to $Rule_n$.

Sequence Constructor: The sequence rule aims at executing rules/function updates in sequence:

$$\textbf{seq } Rule_1, \ldots, Rule_n$$

The resulting update set is a sequential composition of the updates generated by $Rule_1 \ldots Rule_n$.

We define the following mutation operators for both *Parallel* and *Sequence* constructors:

- *Add Rule Operator (ARO)*: adds a new rule to the parallel/sequence of rules.

- *Delete Rule Operator (DRO)*: deletes a rule from the parallel/sequence of rules.
- *Replace Rule Operator (RRO)*: replaces one of the rules in the parallel/sequence by another rule.
- *Permute Rule Operator (PRO)*: changes the order of the parallel/sequence rules by permuting two rules.

Table IV
EXAMPLES OF THE PARALLEL/SEQUENCE RULE MUTATION OPERATORS FOR *CoreASM* [16]

Mutation Operator	CoreASM Spec S	CoreASM Mutant S'
ARO	**seqblock** *R1 R2* **endseqblock**	**seqblock** *R1 R2 R3* **endseqblock**
ARO	**par** *R1 R2* **endpar**	**par** *R1 R2 R3* **endpar**
DRO	**seqblock** *R1 R2 R3* **endseqblock**	**seqblock** *R1 R3* **endseqblock**
DRO	**par** *R1 R2 R3* **endpar**	**par** *R1 R3* **endpar**
RRO	**seqblock** *R1 R2* **endseqblock**	**seqblock** *R1 R3* **endseqblock**
RRO	**par** *R1 R2* **endpar**	**par** *R1 R3* **endpar**
PRO	**seqblock** *R1 R2* **endseqblock**	**seqblock** *R2 R1* **endseqblock**
PRO	**par** *R1 R2* **endpar**	**par** *R2 R1* **endpar**
SPEO	**seqblock** *R1 R2* **endseqblock**	**par** *R1 R2* **endpar**
SPEO	**par** *R1 R2* **endpar**	**seqblock** *R1 R2* **endseqblock**

In addition to these rules, we define the *Sequence-Parallel Exchange Operator (SPEO)* to exchange a sequence rule with a parallel rule and vice versa. Table IV illustrates the Parallel/Sequence rule mutation operators.

It is worth noting that:

- Applying *SPEO* operator may result into mutants that are syntactically correct but containing inconsistent updates. Table V shows a simple *coreASM* sequence rule and its corresponding mutant after applying *SPEO* operator. The execution of the produced mutant leads to an inconsistent update of variable *a* (i.e., the computation of the rule does not yield a next state).

Table V
APPLYING SPEO OPERATOR THAT LEADS TO AN INCONSISTENT UPDATE

Original Spec S	Mutant Spec S'
rule Main =	rule Main =
seqblock	**par**
a := a + 1	a := a + 1
a := b	a := b
endseqblock	**endpar**

- Applying *SPEO* operator may produce a mutant that is equivalent to the original specification. Indeed, such a case may take place when the rules enclosed within the parallel/sequence blocks do not interfere. Table VI shows a specifications S and its mutant S'. Both specifications are equivalents from an input/output perspective since variables *a* and *b* are updated independently. However, the original specification S produces 2 states (i.e., one *a:= a + 1* and one for *b := b +1*) whereas its mutant S' produces only one single state (i.e., *a:= a +1* and *b := b + 1* are executed in one single step).

Table VI
APPLYING SPEO OPERATOR PRODUCES A MUTANT THAT IS EQUIVALENT TO THE ORIGINAL SPEC

Original Spec S	Mutant Spec S'
rule Main =	rule Main =
seqblock	**par**
a := a + 1	a := a + 1
b := b + 1	b := b + 1
endseqblock	**endpar**

3) Choose Rule Mutation Operators:: The choose rule consists on selecting elements (non deterministically) from specified domains which satisfy guards φ, then evaluates $Rule_1$. If no such elements exist, then evaluates $Rule_2$.

choose x_1 **in** $D_1, \ldots, x_n$ **in** D_n **with** $\varphi\ (x_1, \ldots, x_n)$ **do** $Rule_{do}$ **ifnone** $Rule_{ifnone}$

The **with** and **ifnone** blocks are optional. The guard φ may be a simple boolean expression of predicate logic expressions.

To cover the *choose* rule, we define the following mutation operators:

- *Choose Domain Replacement Operator (CDRO)*: replaces a variable domain with another compatible domain.
- *Choose Guard Modification Operator (CGMO)*: alters the guard φ. In this paper, we consider simple boolean expressions as guards. Predicate logic expressions such as *exists* are left for future work.
- *Choose DoRule Replacement Operator (CDoRO)*: replaces the rule $Rule_{do}$ by another rule.
- *Choose IfNoneRule Replacement Operator (CIRO)*: replaces the rule $Rule_{ifnone}$ by another rule.
- *Choose Rule Exchange Operator (CREO)*: replaces the rule $Rule_{ifnone}$ by another rule.

Table VII
EXAMPLE OF THE CHOOSE RULE MUTATION OPERATORS FOR *CoreASM* [16]

Mutation Operator	CoreASM Spec S	CoreASM Mutant S'
CDRO	**choose** x **in** Set1 **with** (x >= 0)	**choose** x **in** Set2 **with** (x >= 0)
CGMO	**choose** x **in** Set1 **with** (x >= 0)	**choose** x **in** Set1 **with** (x <= 0)
CDoRO	**choose** x **in** Set1 **do** Rule1	**choose** x **in** Set1 **do** Rule2
CIRO	**choose** x **in** Set1 **do** Rule1 **ifnone** Rule2	**choose** x **in** Set1 **do** Rule1 **ifnone** Rule3
CREO	**choose** x **in** Set1 **do** Rule1 **ifnone** Rule2	**choose** x **in** Set1 **do** Rule2 **ifnone** Rule1

4) Forall Rule Mutation Operators:: The synchronous parallelism is expressed by a *forall* rule which has the following form:

forall x_1 in D_1, …, x_n in D_n **with** φ **do** $Rule_{do}$

where x_1, …, x_n are variables, D_1, …, D_n are the domains where x_i take their value, φ is a boolean condition, $Rule_{do}$ is a transition rule containing occurrences of the variables x_i bound by the quantifier.

We define the following mutation operators for the *forall* rule that are quite similar to the ones of the *choose* rule :

- *Forall Domain Replacement Operator (FDRO)*: replaces a variable domain with another compatible domain.
- *Forall Guard Modification Operator (FGMO)*: alters the guard φ using the set of operators introduced in Table III.
- *Forall DoRule Replacement Operator (FDoRO)*: replaces the rule $Rule_{do}$ by any other rule.

Table VIII
EXAMPLES OF THE FORALL RULE MUTATION OPERATORS FOR *CoreASM* [16]

Mutation Operator	CoreASM Spec S	CoreASM Mutant S'
FDRO	**forall** x **in** Set1 **with** (x = 0) **do** R1	**forall** x **in** Set2 **with** (x >= 0) **do** R1
FGMO	**forall** x **in** Set1 **with** (x = 0) **do** R1	**forall** x **in** Set1 **with** (x <= 0) **do** R1
FDoRO	**forall** x **in** Set1 **do** R1	**forall** x **in** Set1 **do** R2

In addition to the proposed *forall* rule mutation operators illustrated in Table VIII, we define the *Choose-Forall Exchange Operator (CFEO)* to exchange a *choose* rule with a *forall* rule and vice versa (See Table IX).

Table IX
EXAMPLES OF THE CHOOSE-FORALL EXCHANGE OPERATOR FOR *CoreASM* [16]

Mutation Operator	CoreASM Spec S	CoreASM Mutant S'
CFEO	**forall** x **in** Set1 **do** R1	**choose** x **in** Set1 **do** R1
CFEO	**choose** x **in** Set1 **do** R1	**forall** x **in** Set1 **do** R1

5) Let Rule:: The *let* rule assigns a value of a term *t* to the variable *x* and then execute the rule *Rule* which contains occurrences of the variable *x*. The syntax of a Let rule is:

let (x = t) **in** $Rule$ **endlet**

We define the following mutation operators (see Table X):

- *Let Variable Assignment Operator (LVAO)*: assigns a different value to *x*, other than *t*, of a compatible type.
- *Let Rule Replacement Operator (LRRO)*: replaces the rule *Rule* by another rule that has occurrences of *x*.
- *Let Rule Variable Replacement (LRVR)*: replaces the variable *x* by another variable.

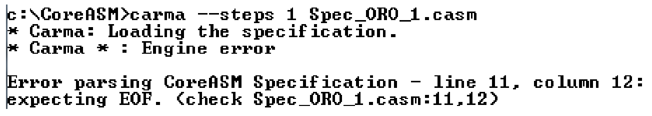
Table X
EXAMPLES OF THE LET RULE OPERATORS FOR *CoreASM* [16]

Mutation Operator	CoreASM Spec S	CoreASM Mutant S'
LVAO	**let** x = 1 **in** R1	**let** x = 2 **in** R1
LRRO	**let** x = 1 **in** R1	**let** x = 1 **in** R2
LRVR	**let** x = 1 **in** R1	**let** y = 1 **in** R1

Other ASM rules such as Case rule, iterate rule, etc. are not covered in this work due to the lack of space.

IV. ASM MUTANTS GENERATION

Figure 1 illustrates the *ASM Mutation Tool* user interface. The user may select one or multiple operators from the three operator categories. The produced mutants are then stored in separate files and run using *carma*, a comprehensive command-line to run *CoreASM* specification, to check their validity. Figure 2 shows an example of the output produced from the execution of *carma*, from the command line, on a syntactically incorrect specification (i.e., the output shows 'Engine Error' and the error location). Note that only 1 execution step is needed to detect syntax errors (i.e., carma –steps 1 MySpec.casm).


```
c:\CoreASM>carma --steps 1 Spec_ORO_1.casm
* Carma: Loading the specification.
* Carma * : Engine error

Error parsing CoreASM Specification - line 11, column 12:
expecting EOF. (check Spec_ORO_1.casm:11,12)
```

Figure 2. Checking the Validity of the Generated Mutant

It is worth noting that the mutation operator EDO (Extend Domain Operator) requires manual definition of the added element.

V. CONCLUSION AND FUTURE WORK

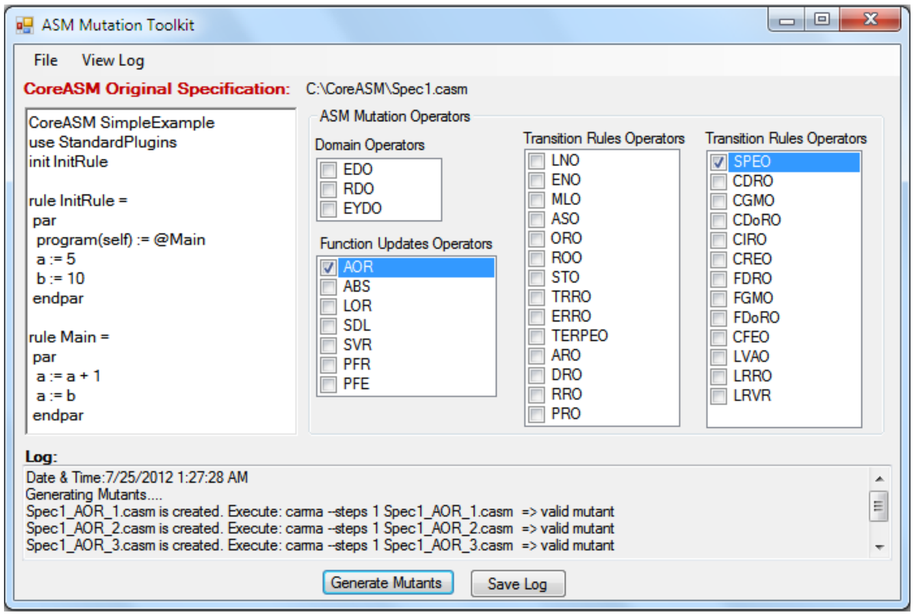
In this paper, we have introduced mutation operators for Abstract State Machines (ASM) formalism. The proposed set of mutation operators are classified into three main categories: ASM domain operators, ASM function update operators, and ASM transition rules operators. Mutants are generated automatically and their syntax are checked for correctness. As a future work, we are planning to conduct an empirical evaluation of the designed operators and to assess their effectiveness and the number of mutants they produce.

REFERENCES

[1] A. P. Mathur, *Mutation Testing*. John Wiley & Sons, Inc., 2002.

[2] Y. Jia and M. Harman, “An analysis and survey of the development of mutation testing,” *Software Engineering, IEEE Transactions on*, vol. 37, no. 5, pp. 649 –678, Sept.-Oct. 2011.

[3] Y.-S. Ma, J. Offutt, and Y. R. Kwon, “Mujava: an automated class mutation system: Research articles,” *Softw. Test. Verif. Reliab.*, vol. 15, pp. 97–133, June 2005.



CoreASM Original Specification: C:\CoreASM\Spec1.casm

```
CoreASM SimpleExample
use StandardPlugins
init InitRule

rule InitRule =
 par
  program(self) := @Main
  a := 5
  b := 10
 endpar

rule Main =
 par
  a := a + 1
  a := b
 endpar
```

Domain Operators: EDO, RDO, EYDO

Function Updates Operators: [✓ AOR], ABS, LOR, SDL, SVR, PFR, PFE

Transition Rules Operators: LNO, ENO, MLO, ASO, ORO, ROO, STO, TRRO, ERRO, TERPEO, ARO, DRO, RRO, PRO

Transition Rules Operators: [✓ SPEO], CDRO, CGMO, CDoRO, CIRO, CREO, FDRO, FGMO, FDoRO, CFEO, LVAO, LRRO, LRVR

```
Date & Time:7/25/2012 1:27:28 AM
Generating Mutants....
Spec1_AOR_1.casm is created. Execute: carma -steps 1 Spec1_AOR_1.casm  => valid mutant
Spec1_AOR_2.casm is created. Execute: carma -steps 1 Spec1_AOR_2.casm  => valid mutant
Spec1_AOR_3.casm is created. Execute: carma -steps 1 Spec1_AOR_3.casm  => valid mutant
```

Figure 1. ASM Mutation Toolkit

[4] A. J. Offutt, VI and K. N. King, “A fortran 77 interpreter for mutation analysis,” *SIGPLAN Not.*, vol. 22, pp. 177–188, July 1987.

[5] H. Agrawal, “Design of mutant operators for the C programming language,” Software Engineering Research Center/Purdue University, Tech. Rep., 1989.

[6] P. E. Black, V. Okun, and Y. Yesha, “Mutation operators for specifications,” in *Proceedings of the 15th IEEE international conference on Automated software engineering*, ser. ASE ’00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 81–88.

[7] S. Pinto Ferraz Fabbri, M. Delamaro, J. Maldonado, and P. Masiero, “Mutation analysis testing for finite state machines,” in *Proceedings of the 5th International Symposium on Software Reliability Engineering*, November 1994, pp. 220 –229.

[8] J.-h. Li, G.-x. Dai, and H.-h. Li, “Mutation analysis for testing finite state machines,” in *Proceedings of the 2009 Second International Symposium on Electronic Commerce and Security - Volume 01*, ser. ISECS ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 620–624.

[9] R. M. Hierons and M. G. Merayo, “Mutation testing from probabilistic and stochastic finite state machines,” *J. Syst. Softw.*, vol. 82, pp. 1804–1818, November 2009.

[10] S. C. P. F. Fabbri, J. C. Maldonado, T. Sugeta, and P. C. Masiero, “Mutation testing applied to validate specifications based on statecharts,” in *Proceedings of the 10th International Symposium on Software Reliability Engineering*, ser. ISSRE ’99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 210–219.

[11] S. C. P. F. Fabbri, J. C. Maldonado, P. C. Masiero, M. E. Delamaro, and E. Wong, “Mutation testing applied to validate specifications based on petri nets,” in *Proceedings of the IFIP TC6 Eighth International Conference on Formal Description Techniques VIII*. London, UK, UK: Chapman & Hall, Ltd., 1996, pp. 329–337.

[12] S. D. R. S. De Souza, J. C. Maldonado, S. C. P. F. Fabbri, and W. L. De Souza, “Mutation testing applied to estelle specifications,” *Software Quality Control*, vol. 8, pp. 285–301, December 1999.

[13] S. S. Batth, E. R. Vieira, A. Cavalli, and M. U. Uyar, “Specification of timed efsm fault models in sdl,” in *Proceedings of the 27th IFIP WG 6.1 international conference on Formal Techniques for Networked and Distributed Systems*, ser. FORTE ’07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 50–65.

[14] Y. Gurevich, “Evolving Algebras 1993: Lipari Guide,” in *Specification and Validation Methods*, E. Börger, Ed. Oxford University Press, 1995, pp. 9–36.

[15] E. Börger and R. F. Stark, *Abstract State Machines: A Method for High-Level System Design and Analysis*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[16] R. Farahbod, V. Gervasi, and U. Glässer, “CoreASM: An Extensible ASM Execution Engine,” *Fundamenta Informaticae*, vol. 77, pp. 71–103, January 2007.

[17] M. Woodward, “Errors in algebraic specifications and an experimental mutation testing tool,” *Software Engineering Journal*, vol. 8, no. 4, pp. 211–224, Jul 1993.

[18] M. F. Lau and Y. T. Yu, “An extended fault class hierarchy for specification-based testing,” *ACM Trans. Softw. Eng. Methodol.*, vol. 14, pp. 247–276, July 2005.

Towards a Knowledge-Based Representation of Non-Functional Requirements

Mohamad Kassab
The Pennsylvania State University
Malvern, Pennsylvania, U.S.A
muk36@psu.edu

Ghizlane El-Boussaidi
École de Technologie Supérieure
Montreal, Canada
ghizlane.elboussaidi@etsmtl.ca

Abstract— **Knowledge-based representation is necessary to support the description of Non-Functional Requirements within a system and to provide practitioners and researchers with a valuable alternative to current requirements engineering techniques. The aim of our research reported in this paper is to systematically develop an ontology which provides the definition of the general concepts relevant to NFRs without reference to any particular application domain. The general concepts can then act as a common foundation for describing particular non-functional attributes as well as providing a conceptual model for NFRs (including, e.g., entity definitions, relations, etc.). The ontology also contains rules which define the semantics of the defined concepts.**

Keywords- non-functional requirements; ontology; software architecture; quality.

I. INTRODUCTION

The IEEE-830: “Guide to Software Requirements Specifications” [1] defines a proper requirements specification as being: unambiguous, complete, verifiable, consistent, modifiable, traceable, and usable during operations and maintenance. To help achieving this, the requirements elicitation process should consider: (1) the functional requirements which are associated with specific functions, tasks, or behavior that the system must support and (2) the non-functional requirements (NFR).

Existing NFRs elicitation methods adopt memo of interview transcripts to collect initial NFRs and then construct systems with the NFRs integrated according to the experience and intuition of the designers [2]. However, empirical reports [3, 4, 5] indicated a number of drawbacks when using these methods. For example, a significant portion of NFRs may be neglected as it is difficult to ask users to provide their NFRs explicitly because they are always related to specific domains and affected by context. Furthermore, NFRs can often interact, in the sense that attempts to achieve one NFR can help or hinder the achievement of other NFRs at certain functionality. Such an interaction creates an extensive network of interdependencies and trade-offs between NFRs which is not easy to describe [6]. In addition, the current methods don’t provide sufficient answers on how the NFRs should be accommodated at later stages of the development (e.g., software architecture).

The growing awareness of these issues among the requirements engineering (RE) community in the last few years led to a heightened interest in NFRs description and modeling and, in turn, to the emergence of several models intended to capture and structure the more relevant concepts defining the NFRs and their relations. Such models are generic ones and must be instantiated to be usable for specific domains or applications. Yet, the instantiation process is not easy to perform since the generic models usually do not contain sufficient information about NFRs interdependencies [7]. Some standards have been proposed in order to unify the definition of subsets of NFRs; e.g., software quality concepts [8]. However, till now there is no clear and coherent generic representation of the NFRs concepts.

On the other hand, the growing interest in ontology-based applications as opposed to systems based on information models have resulted in an increasing interest in the definition of conceptual models for any kind of domain. Software Engineering is one of those domains that have received high attention in that respect [9, 10, 11]. Current research studies by knowledge engineering scholars on requirement acquisition, for example, use domain ontology to support software requirements description [12, 13, 14]. These studies leverage the existing knowledge of the relationship between the software requirements and the information in the related domain. According to this relationship, the domain knowledge influences the result of requirements acquiring [2]. International Software Engineering standards such as IEEE [15] provide a foundation for the development of ontology for software engineering in terms of common vocabulary and concepts. Nonetheless, the process of analysis of the standards to come up with a logical coherent ontology is by no means a simple process [10]. Moreover, NFRs have received little or no attention from the ontology research groups due to inherent challenges imposed by the semantic imprecision of NFRs conceptual schemas [10].

Building on the above discussion, a knowledge-based representation is necessary to support the description of NFRs within a system and to provide practitioners and researchers with a valuable alternative to current requirements engineering techniques. In [16], a systematically developed ontology which provides the definitions of the general concepts relevant to NFRs was presented. The aim of our research reported in this paper is to present an updated version of the NFRs ontology with: 1) updated and more comprehensive rules which define the semantics of the defined concepts; and 2) an extension to the NFRs' refinements relating to software architecture concepts without reference to any particular application domain. The general concepts can then act as a common foundation for describing particular non-functional attributes as well as providing a conceptual model for NFRs (including e.g., entity definitions, relations, etc.).

The paper is organized as follows. Section II provides the background on ontologies and the Web Ontology Language. We describe in details the NFRs ontology in Section III. Section IV evaluates the NFRs ontology. Section V discusses related work and finally, Section VI concludes the paper.

II. ONTOLOGIES IN SOFTWARE ENGINEERING

The software engineering community has recognized ontologies as a promising way to address current software engineering problems. Researchers have so far proposed many different synergies between software engineering and Ontologies. For example, ontologies are proposed to be used in requirements engineering, software modeling, model transformations, software maintenance, software comprehension, software methodologies, and software community of practice.

Ontology can be defined as "*a specification of a* conceptualization" [17]. More precisely, ontology is an explicit formal specification of how to represent the objects, concepts, and other entities that exist in some area of interest and the relationships that hold among them. In general, for ontology to be useful, it must represent a shared, agreed upon conceptualization. The use of ontologies in computing has gained popularity in recent years for two main reasons: i) they facilitate interoperability and ii) they facilitate machine reasoning. In its simplest form, ontology is taxonomy of domain terms. However, taxonomies by themselves are of little use in machine reasoning. The term ontology also implies the modeling of domain rules. It is these rules, which provide an extra level of machine "understanding".

Holsapple [18] describes a number of approaches to ontology design: inspiration, induction, deduction, synthesis and collaboration. We chose to follow the deductive approach. Deductive approach to ontology design is concerned with adopting some general principles and adaptively applying them to construct an ontology geared toward a specific case. This involves filtering and distilling the general notions so they are customized to a particular domain subset. It can also involve filling in details, effectively yielding an ontology that is an instantiation of the general notions.

The constructs used to create ontologies vary between ontology languages. One class of ontology languages is those which are based upon description logics [19]. OWL is one such language. OWL [20] is the Web Ontology Language, an XML-based language for publishing and sharing ontologies via the web. OWL originated from DAML+OIL both of which are based on RDF (Resource Description Framework) triples. There are three 'species' of OWL – but the most useful for reasoning - OWL-DL - corresponds to a description logic. Editing OWL manually can be equally difficult for the very same reason. We used Protégé and its OWL plug-in for NFRs ontology development.

OWL ontology consists of Classes; also referred to by concepts, and their Properties; also referred to by relations. The Class definition specifies the conditions for individuals to be members of a Class. A Class can therefore be viewed as a set. The set membership conditions are usually expressed as restrictions on the Properties of a Class. For instance the *allValuesFrom* and *someValuesFrom* property restrictions commonly occur in Class definitions. These correspond to the universal quantifier ($\forall$) and existential qualifier ($\exists$) of predicate logic. More precisely, in OWL such restrictions form anonymous Classes of all individuals matching the corresponding predicate. A key feature of OWL and other description logics is that classification (and subsumption relationships) can be automatically computed by a reasoner which is a piece of software able to infer logical consequences from a set of asserted facts or axioms. For the purpose of the NFR ontology, we will use a semantic web reasoning system and information repository Renamed Abox and Concept Expression Reasoner (RACER) [21].

III. NFRS ONTOLOGY

Most of the terms and concepts in use for describing NFRs have been loosely defined, and often there is no commonly accepted term for a general concept [22]. As indicated in the Introduction, common foundation is required to enable effective communication and to enable integration of activities within the RE community. This common foundation is realized by developing an ontology, i.e. the shared meaning of terms and concepts in the domain of NFRs.

There are many resources for setting up a glossary for NFRs. In addition, there are many different perspectives from where NFR terms are defined, (e.g., NFRs in product-oriented perspective vs. process-oriented perspective [6]). In this paper, the NFRs glossary is developed based on commonality analysis and generalization from the previous publications in requirements engineering and software engineering communities.

The NFRs ontology has an important core about NFRs model, but also addresses areas such as software architectures. It contains many concepts. In order to cope with the complexity of the model we use views of the model. A view is a model which is completely derived from another model (the base model). A view cannot be modified separately from the model from which it is derived. Changes to the base model cause corresponding changes to the view [23]. Three views of the NFRs ontology are identified: The first view concerns the NFRs relation with the other entities of the software system being developed (intermodel dependency view). The second contains the classes and properties intended to structure NFRs in terms of interdependent entities (intramodel dependency). The third view represents the measurement process and contains the concepts used to produce measures to measurable NFRs. The measurement view will not be discussed in this paper as it maintains the same structure from the earlier version of the NFRs ontology [16].

A. Intermodel Dependency View

Figure 1 illustrates the structure of the NFRs intermodel dependency view by means of a simplified UML class diagram. The core of this structure relies on the fact that NFRs are not stand-alone goals, as their existence is always dependent on other concepts in the project context. If a requirement is a member of the class NonFunctionalRequirement, it is necessary for it to be a member of the class Requirement and it is necessary for it to be a member of the anonymous class of things that are linked to at least one member of the class AssociationPoint through the hasAssociationPoint property. On the other hand, isAssociatingNfrTo links the AssociationPoint to a range of: FunctionalRequirement union Element union Process union Product union Resource. The AssociationPoint can be thought of as an interface from the perspective of the association to the individuals from the above range. Thus, if an individual is a member of the AssociationPoint Class, it is necessary for it to be linked to one and only one individual from: the (FunctionalRequirement class through the isAssociatingNfrTo property) OR (Element through isAssociatingNfrTo property) OR (Process through isAssociatingNfrTo property) OR (Product through isAssociatingNfrTo property) OR (Resource though the isAssociatingNfrTo property).

An individual from AssociationPoint class can be linked to many individuals from the NonFunctionalRequirement class through hasAssociationPoint property.

1) Association to FR (or derived elements)

Functionality-related NFRs refer to individuals instantiated from the NonFunctionalRequirement class and that participate in hasAssociationPoint relation with an individual from the AssociationPoint class which in its turn participates in isAssociatingNfrTo relation with an individual from the FunctionalRequirement class (see Figure 1). In fact, a subset of NFRs, namely functionality quality requirements, is defined with an existential restriction to have at least one association point with FR as it represents a set of attributes that bear on the existence of a set of functions and their properties specified according to the ISO 9126 definition to the functionality quality [8]. Valid example of functionality-related NFRs is: “the interaction between the user and the software system while reading email messages must be secured”.

The FunctionalRequirement class is further specialized into PrimaryFunctionalRequirement and SecondaryFunctionalRequirement . A NFR can be associated to either type of FRs.

Functional Requirement is further realized through the various phases of development by many functional models (e.g., in the object-oriented field, a use-case model is used in the requirements analysis and specification phase, a design class model is used in the software design phase, etc.). Each model is an aggregation of one or more artifacts (e.g., a use-case diagram and a use-case for the use-case model, a domain model diagram and a system sequence diagram for the analysis model, a class diagram and a communication diagram for the design model). The artifact by itself is an aggregation of elements (e.g., a class, an association, an inheritance, etc. for the class diagram). Modeling artifacts and their elements in this way gives us the option of decoupling the task of tracing NFRs from a specific development practice or paradigm.

If an NFR is associated with functionality, then some or all the offspring elements that refine this functionality will inherit this association. More specifically:

$((\text{NFR}_i \text{ hasAssociationPoint AssociationPoint}_j) \wedge (\text{AssociationPoint}_j \text{ isAssociatingNfrTo FunctionalRequirement}_k)) \implies \exists \text{ Element}_n ((\text{NFR}_i \text{ hasAssociationPoint AssociationPoint}_m) \wedge (\text{AssociationPoint}_m \text{ isAssociatingNfrTo Element}_n) \wedge (\text{FunctionalRequirement}_k \text{ FrIsMappedInto Element}_n))$

When hasAssociationPoint property links an individual NFR to an individual AssociationPoint which is further linked to an individual FunctionalRequirement or Element through isAsscoatingNfrTo property, then the AssociationPoint can be further specified through one of three subclasses. These subclasses specify the type of association between an individual from the NonFunctionalRequirement class and an individual from the FunctionalRequirement and Element classes. We adopt the concepts of overlapping, overriding and wrapping, commonly used in various separations of concerns approaches [24] to define these three subclasses:

• Overlapping: the NFR requirement modifies the FRs it transverses. In this case, the NFR may be required before the functional ones, or it may be required after them. For example, the implementation of security requirement (e.g., user’s authorization) needs to be executed before the user can access “read email messages” functionality.

• Overriding: the NFR superposes the FRs it transverses. In this case, the behavior described by the NFR substitutes the FRs behavior.

• Wrapping: NFR “encapsulates” the FRs it transverses. In this case, the behavior described by the FR is wrapped by the behavior described by the NFRs.

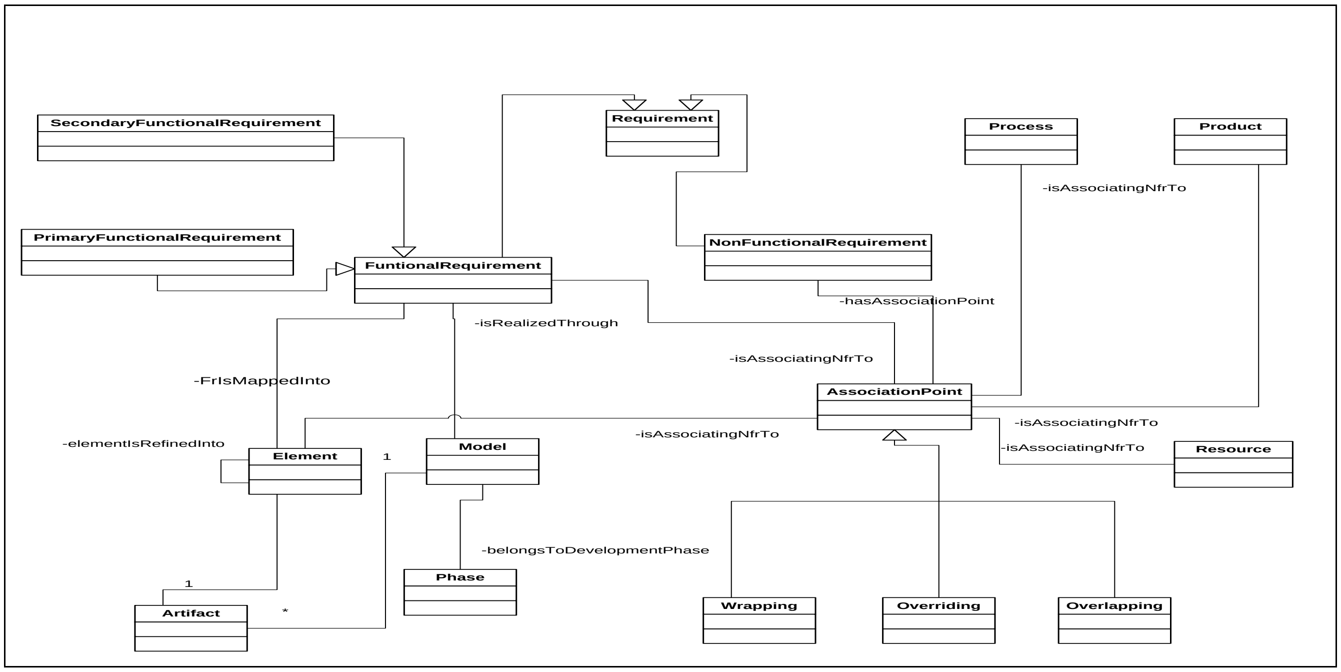


Figure 1. NFRs Intermodel Dependency View.

2) Association to Process

A software development process is a structure imposed on the development of a software product. Synonyms include software life cycle and software process. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

From the above definition to the software process, process-related NFRs specify concerns relative to the scope of the development process. Examples of such NFRs are “The project will follow the Rational Unified Process (RUP)” and “Activities X, Y, Z will be skipped for this project”.

3) Association to Product

Product-related NFRs refer to those NFRs which have a global impact on the system as whole. Example of such NFRs are: “The system should be easy to maintain”.

4) Association to Resource

Resources serve as input to the processes used on a project. They include people, tools, materials, methods, time, money, and skills [25]. An example of an NFR associated with a resource is illustrated through a requirement like “The software maintainers should have at least 2 years of experience in Oracle database.” This is an operating constraint that is associated with candidates for the maintenance position for the system (another type of resources).

It is to be noted that the inter-relationships among the above five concepts (e.g., the relation between the product and the process) is out of the scope of this paper.

B. Intramodel Dependency View

The intramodel dependency view is concerned with the refinement of NFRs into one or more offspring; through either decomposition or operationalization, and the correlation among the concepts of the NFRs model. The view is depicted in the UML class diagram in Figure 2 and it is discussed through the concepts and properties referring to: NFRs type, NFRs decomposition, NFRs operationalization and NFRs interactivity.

1) NFRs Type

Specifying NFR through types is a particular kind of refinement for NFRs [6]. This allows for the refinement of a parent on its type on terms of offspring, each with a subtype of the parent type. Each subtype can be viewed as representing special cases for the NFR. Five subclasses are identified as a candidate for the root node for an NFR type refinement hierarchy; namely, QualityRequirement, DevelopmentConstraint (e.g., implementation language constraint, constraints on system architecture), EconomicConstraint (e.g., allocated budget), OperatingConstraint and PoliticalCulturalConstraint (e.g., law imposing to support bilingual system user interface). These in fact are not mutual exclusive classes.

A special type of Development constraints is the architectural concern which presents an architectural requirement on the system under development. A concern is an area of interest or focus in a system. Concerns are the primary criteria for decomposing software into smaller, more manageable and comprehensible parts that have

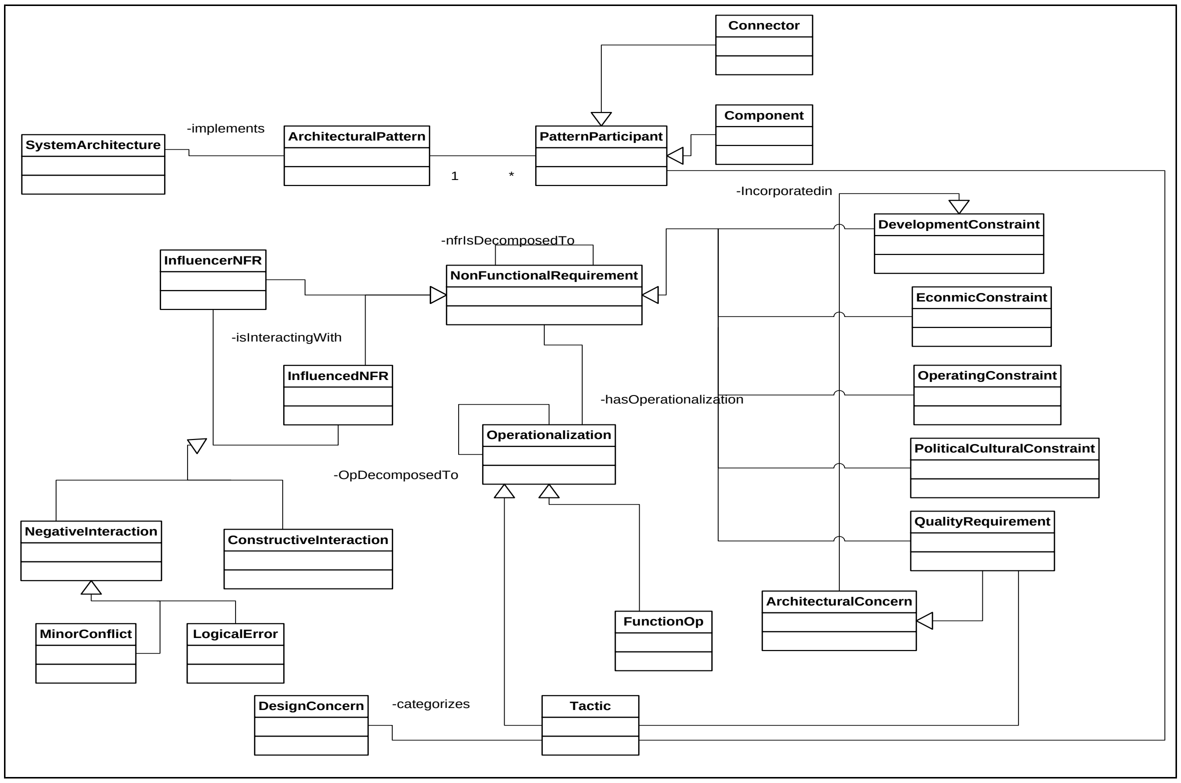


Figure 2. NFRs Intramodel Dependency View.

meaning to a software engineer. From this, architectural concerns are defined as those concerns that significantly influence the architecture [26]. An example of an architectural concern could be the need for coordination between distributed entities within the system.

On other hand; a special type of architectural concern is QualityRequirement [27] (e.g., security guarantees for the system). This implies that quality requirements are in fact development constraints themselves; as the development process should bear in mind the required qualities while taking architectural; design or implementation decisions.

2) *Decomposition*

This refers to the NfrIsDecomposedTo property that decomposes a high-level NFR into more specific sub-NFRs. In each decomposition, the offspring NFRs can contribute partially or fully towards satisfying the parent. NfrIsDecomposedTo is a transitive property. The decomposition can be carried either across the type dimension or the association point dimension. For example, let us consider the requirement "read an email message with high security". The security requirement constitutes quite a broad topic [6]. To deal effectively with such a requirement, the NFR may need to be broken down into smaller component using the knowledge of the NFR type; discussed in the previous subsection, so that an effective solution can be found. Thus, the requirement states as "read an email with a high security" can be decomposed into "read an email with high integrity", "read an email with high confidentiality", and "read an email with high availability". An example of decomposition across the association point is: "read inbox folder messages with high security", "read system-created folder messages with high security". The decomposition can be "ANDed" (all NFR offspring are required to achieve the parent NFR goal) or "ORed" (it is sufficient that one of the offspring be achieved instead, the choice of offspring being guided by the stakeholders).

3) *Operationalization*

This refers to the hasOperationalization property that refines the NFR into solutions in the target system that will satisfy the NFR [6]. One type of operationalizations is "FunctionOp" which corresponds to functionalities to be implemented. For example, "Authorization" and "Authentication" are potential instances of FunctionOp class to implement Security quality. Similar to decomposition, operationalization can be ANDed or ORed.

In the inferred taxonomy; the taxonomy after the reasoner impact, the reasoner classifies FunctionOp based on the imposed assertions as a subclass for FunctionalRequirement. This classification is consistent with many arguments in the requirements engineering community on the tight link between the FRs and NFRs [28]. The ontology brings formalism and a concrete understanding to this link.

The second type of operationalizations is “Tactic” which represents design decisions aiming at satisfying some quality requirements. Indeed, when designing software, an architect relies on a set of idiomatic patterns commonly named architectural styles or patterns. A software architectural pattern defines a family of systems in terms of a pattern of structural organization and behavior [29]. More specifically, an architectural pattern determines the vocabulary of components and connectors that can be used in instances of that style, together with a set of constraints on how they can be combined [30]. Common architectural patterns include Layers, Pipes and Filters and Model View Controller (MVC). As shown in Figure 2, an instance of SystemArchitecture class is an implementation of one or more instances of ArchitecturalPattern class. While architectural patterns embody high level design decisions, an architectural tactic [31] is a design strategy that addresses a particular quality attribute. Tactics are a special type of operationalization that serves as the meeting point between the quality attributes and the software architecture. Tactics are the building blocks of patterns [31] and implementing a tactic within a pattern may affect the pattern by modifying some of its components, adding some components and connectors, or replicating components and connectors [32]. An instance of class Tactic is linked to an instance of classes Component / Connector through one of the following properties which define the semantics of impact of incorporating the tactic into the pattern (adopted from [27]):

- Implemented in: The tactic is implemented within a component of the pattern. Actions are added within the sequence of the component.
- Replicates: A component is duplicated. The component’s sequence of actions is copied intact, most likely to different hardware.
- Add, in the pattern: A new instance of a component is added to the architecture while maintaining the integrity of the architecture pattern. The new component comes with its own behavior while following the constraints of the pattern.
- Add, out of the pattern: A new component is added to the architecture which does not follow the structure of the pattern. The added actions do not follow the pattern.
- Modify: A component’s structure changes. This implies changes or additions within the action sequence of the component that are more significant than those found in “Implemented in”.
- Delete: A component is removed.

Tactics which have relatively a similar impact can be grouped together into categories which are instances of DesignConcern class. For example, a design concern towards the architectural concern “high performance for the system” is how to “manage resources demands”. This design concern is a group of four tactics that aim to improve the performance quality: increase computation efficiency, reduce computational overhead, manage event rate and control frequency of sampling. It’s worth to point out that FunctionOp and Tactic are not mutual exclusive classes.

4) Interactivity

An individual NFR may participate in *isInteractingWith* property which links it to another NFR. This refers to the fact that the achievement of one NFR; *InfluencerNfr*, at a certain association point can hinder (through *isNegativelyInteractingWith* property) or help (through *isPositivelyInteractingWith* property) the achievement of other NFR; *InfluencedNfr*, at the same association point, e.g., *security* and *performance* at “*read an email message”* functionality. *isInteractingWith* is not a symmetric property.

The negative interaction is further specialized through the two sub-properties, which help classifying the negative interaction into: *hasLogicalErrorWith* and *hasMinorContradictionWith*.

Logical Error: This is a fundamental conflict which must be resolved immediately. It occurs when the achievement of NFR_1 will prevent the achievement of NFR_2. This is expressed by means of the proposition LogicalError (NFR_1, NFR_2) $\Leftrightarrow$ NFR_1 $\rightarrow$ NOT NFR_2. Logical Error demonstrates a direct contradiction between two requirements. For example, NFR_1 is stated as “Security has to be high at *read email* functionality”; while NFR_2 is stated as “There should be no security constraints at *read email* functionality”!

Minor Contradiction: This is one of the best-known cases of conflict [6]. Associating a win condition with an NFR (say NFR_1) triggers a search of the operationalization that has positive and/or negative effects on NFR_1. For example, the Portability NFR, the win condition of which is “portable to Windows”, has positive effects (i) on the portability layers and separation of data generation and (ii) on the presentation, but has negative effects on the use of fast platform-dependent user interface functionalities that would be affected with the layering strategy. The operationalizations, that are found to have negative effects on other NFRs sharing the same association points with their parents NFRs, are used to identify potential conflicts.

IV. EVALUATION

We evaluated our ontology according to three criteria: 1) is it generally acceptable? 2) is it consistent? and 3) is it accurate?. ‘Generally accepted’ means that the knowledge and practices described are applicable to most projects most of the time, and that there is widespread consensus about their value and usefulness. ‘Generally accepted’ does not mean that the knowledge and practices described are or should be applied uniformly on all projects [33].

Clearly, the evaluation of the acceptance and the accuracy of the ontology as such ultimately rely upon its application by the research community. For the purpose of this evaluation, we have used our ontology within three different projects. These projects helped refining the initial NFRs ontology. Indeed we have instantiated the ontology against the set of requirements from the settings of the NOKIA Mobile Email Application System and the IEEE Montreal Website. Further, we worked closely with experts from SAP-Montreal to use the NFRs Ontology as a repository for the requirements of some of the projects which are under development. From the experiences and the participants' feedback developed from instantiating the NFRs Ontology against the three real-life projects (the Nokia project, the IEEE Montreal website project and the SAP project), the ontology has proven to be easy to instantiate and links the concepts efficiently. Each individual captured NFR was instantiated from its corresponding concept in the Ontology. We make the note here that we did not meet the case in which an individual NFR was not instantiated from a corresponding concept. Finally the consistency of this ontology has been demonstrated through the usage of a semantic web reasoning system and information repository RACER [21].

V. RELATED WORK

Even though there is no formal definition of the term 'NFR', there has been considerable work on characterizing and classifying NFRs. In a report published by the Rome Air Development Center (RADC) [34], NFRs ("software quality attributes" in their terminology) are classified into consumer-oriented (or software quality factors) and technically-oriented (or software quality criteria). The former class of software attributes refers to software qualities observable by the consumer, such as efficiency, correctness and interoperability. The latter class addresses system-oriented requirements such as anomaly management, completeness and functional scope.

Earlier work by Boehm et al. [35] structured quality characteristics of software within a quality characteristics tree of 25 nodes, noting that merely increasing designer awareness would improve the quality of the final product. A well-known and more recent approach to representing NFRs using a graphical method is the NFRs framework by Chung et al [6]. A cornerstone of the framework is the "softgoal" concept for representing the NFR. A softgoal is a goal that has no clear-cut definition or criteria to determine whether or not it has been satisfied. The operation of the framework can be visualized in terms of the incremental and interactive construction, elaboration, analysis and revision of a softgoal interdependency graph (SIG). High-level softgoals are refined into more specific subgoals or operationalizations. In each refinement, the offspring can contribute fully or partially, and positively or negatively, towards satisfying the parent. However, the particular graphical notations make it difficult to coordinate with mature UML tools and be integrated with existing models of FRs. This integration has been tackled in [24, 36, 37] by extending UML models to integrate NFRs to the functional behavior. Although the integration process must be considered at the meta-level, these approaches only model certain NFRs (e.g., response time, security) in a way that is not necessarily applicable for other requirements.

On a different track, Hauser et al. [38] provide a methodology for reflecting customer attributes in different phases of design. Dobson et al [23] describe an approach to specifying the Quality of Service (QoS) requirements of service-centric systems using an ontology for Quality of Service. The above approaches address only a subset of NFRs; namely quality requirements, and sometimes within a specific context; (e.g., service computing in [24] and automotive industry in [38]). On contrast, our work aims at providing a more generic solution to all types of NFRs with independence from any context.

Al Balushi and Dabhi [39] used an ontology-based approach to build NFR quality models with the objective to gather reusable requirements during NFR specification. We agree with these authors on the usefulness of ontology, however, the research objectives of their research efforts and ours differ, which in turn, leads to essential difference in the research outcomes. While the conceptual model in [1] is geared towards solving requirements reuse problems, our ontology covers a broader spectrum of NFR issues. This is achieved by using multiple views, which explicate requirements phenomena by complementing the strengths of multiple conceptualizations of NFRs.

VI. CONCLUSION AND FUTURE WORK

Although non-functional requirements are receiving more and more attention in the requirement and software engineering communities, little progress has been made in using ontologies for NFRs. This is mainly because NFRs are too abstract and affected by a large number of subjective factors, which makes it difficult for users to describe their own NFRs accurately and precisely. In this paper, we proposed a NFRs ontology that we developed by analyzing and generalizing concepts from the literature. We used a disciplined approach to ontology development, with explicit requirements, ontology design, and implementation. This ontology describes glossaries and taxonomies for NFRs. We used these glossaries for generalization to the common NFRs concepts. To evaluate the ontology, we have used it within the context of three projects. This initial evaluation proved that the ontology is consistent and easy to use.

Clearly, the evaluation of the acceptance and the accuracy of the NFRs ontology, as such, ultimately rely upon its application by the research community. The authors of this are hoping to soon benefit from interaction with a number of interested parties in this topic. In particular, we plan to explore the way in which NFRs ontology could be further leveraged in more complex requirements specification scenarios in real-life settings. In order to ground the concept further, we plan to develop tools to leverage the benefits of ontology for NFRs and evaluate our results against scenarios designed to test the capabilities of the ontology. One potential tool of our interest will aim at facilitating the investigation of studying the impact of incorporating the quality tactics into the software architectural patterns. In addition, we will investigate further to which degree having the NFRs ontology adopted in the requirements engineering activities guarantees the compliance of the final product with the captured NFRs.

REFERENCES

[1] IEEE Std. 830-1998. (1998), "IEEE recommended practice for software requirements specifications", IEEE Transactions on Software Engineering.

[2] T. Jingbai, H. Keqing, W. Chong, and L. Wei, "A Context Awareness Non-functional Requirements Metamodel Based on Domain Ontology", IEEE International Workshop on Semantic Computing and Systems, 2008, Huangshan, China, pp.1-7.

[3] K. K. Breitman, J. C. S. P. Leite, and A. Finkelstein, "The World's Stage: A Survey on Requirements Engineering Using a Real-Life Case Study", Journal of the Brazilian Computer Society, 1(6), 1999, pp. 13-37.

[4] A. Finkelstein and J. Dowell, "A Comedy of Errors: The London Ambulance Service Case Study", proceedings of the 8th International Workshop Software Specifications and Design, 1996, pp. 2-5.

[5] L. Leveson and C. S. Turner, "An Investigation of the Therac-25 Accidents", IEEE Computer, 26(7), 1993, pp. 18-41.

[6] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, "Non-functional Requirements in Software Engineering", Kluwer Academic Publishing, 2000.

[7] P. M. O. O. Sack, M . Bouneffa, Y. Maweed, and H. Basson, "On Building an Integrated and Generic Platform for Software Quality Evaluation", 2nd IEEE International Conference on Information and Communication technologies: From Theory to Applications. April 24 - 28, 2006, Umayyad Palace, Damascus, Syria.

[8] International Standard ISO/IEC 9126-1. Software engineering – Product quality – Part 1: Quality model. ISO/IEC 9126-1:2001, 200.

[9] O. Mendes and A. Abran, "Software Engineering Ontology: A Development Methodology", Metrics News, 9, 2004, pp. 68-76.

[10] M. A. Sicilia and J. J. Chadrado-Gallego, "Linking Software Engineering concepts to upper ontologies", Proceedings of the First Workshop on Ontology, Conceptualizations and Epistemology for Software and Systems Engineering, 2005, Alcalá de Henares, Spain.

[11] C. Wille, A. Abran, J. M. Desharnais, and R. R. Dumke, "The quality concepts and subconcepts in SWEBOK: An ontology challenge", In Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering, 2003, Taipei, Taiwan.

[12] K. Haruhiko and S. Motoshi, "Using domain ontology as domain knowledge for requirements elicitation", proceedings of the 14th IEEE International Requirements Engineering Conference, 2006, Minneapolis, USA, pp. 186 – 195.

[13] Z. Jin, "Ontology-based requirements elicitation automatically", Chinese J. Computers, Vol.23, No.5, 2000, pp. 486 – 492.

[14] H. Kaiya and M. Saeki M, "Ontology based requirements analysis: lightweight semantic processing approach", proceedings of the 5th International Conference on Quality Software (QSIC), 2005, Melbourne, Australia, pp. 223 – 230.

[15] IEEE (1990). Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12-1990.

[16] M. Kassab, O. Ormandjieva, and M. Daneva, "An Ontology Based Approach to Non-Functional Requirements Conceptualization", Proceedings of the 4th International Conference on Software Engineering Advances, ICSEA 2009, September 20-25, 2009 - Porto, Portugal.

[17] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications", Knowledge Acquisition, 1993, pp. 199 – 220.

[18] C. W. Holsapple and K. D. Joshi, "A Collaborative Approach to Ontology Design", Communication of the ACM, February 2002, Vol 45, No 2, pp. 42 - 47.

[19] F. Baader, I. Horrocks, and U. Sattler, "Description logics as ontology languages for the semantic web", in Lecture Notes in Artificial Intelligence. Springer, 2003, http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/BaHS03.pdf. [retrieved: 11,2012].

[20] W3C, "Web Ontology Language (OWL)", http://www.w3.org/2004/OWL.

[21] Racer: Renamed Abox and Concept Expression Reasoner. http://www.sts.tu-harburg.de/~r.f.moeller/racer/

[22] M. Glinz, "On Non-Functional Requirements", 15th IEEE International Requirements Engineering Conference (RE 2007), 2007, Delhi, India, pp.21-26.

[23] R. Lock, G. Dobson, and I. Sommerville, "Quality of Service Requirement Specification using an Ontology", Conference Proceedings 1st International Workshop on Service-Oriented Computing: Consequences for Engineering Requirements (SOCCER'05), Paris, France, 30th August 2005.

[24] J. Araujo, A. Moreira, I. Brito, and A. Rashid, "Aspect-Oriented Requirements With UML", Workshop on Aspect-Oriented Modeling with UML (held with UML 2002).

[25] S. Whitmire, "Object Oriented Design Measurement", John Wiley & Sons, 1997.

[26] N. Bouck´e and T. Holvoet, "Dealing with concerns ask for an architecture-centric approach", In European Interactive Workshop, 2005.

[27] N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? A model and annotation", Journal of Systems and Software, vol. 83, Issue 10, pp. 1735-1758, 2010.

[28] B. Paech, A. Dutoit, D. Kerkow, and A. von Knethen, "Functional requirements, non-functional requirements and architecture specification cannot be separated – A position paper", 8th International Workshop on Requirements Engineering: Foundation for Software Quality, 2002, Essen, Germany.

[29] D. Garlan and M. Shaw, "An Introduction to Software Architecture", Technical Report, CMU, Pittsburgh, PA, USA, 1994.

[30] Microsoft Application Architecture Guide: Patterns & Practices, 2nd Edition, http://msdn.microsoft.com/en-us/library/ff650706.aspx.

[31] L. Bass, P. Clements, and R. Kazman, "Software architecture in practice", Addison-Wesley, 2003.

[32] N. B. Harrison, P. Avgeriou, and U. Zdun, "On the Impact of Fault Tolerance Tactics on Architecture Patterns", In proceedings of 2nd International Workshop on Software Engineering for Resilient Systems (SERENE 2010), London, UK, 2010.

[33] PMBOK (2000). Project Management Body of Knowledge Guide 2000. See http://www.cs.bilkent.edu.tr/~cagatay/cs413/PMBOK.pdf. [retrieved: 11, 2012].

[34] T. P. Bowen, G. B. Wigle, and J. T. Tsai, "Specification of Software Quality Attributes", Volume 2, Software Quality Specification Guidebook, 1985.

[35] B. W. Boehm, J. R. Brown, M. Lipow, "Quantitative Evaluation of Software Quality". In proceeding of the 2nd Int. Conference on Software Engineering, San Francisco, CA, Oct. 1976. Long Branch, CA: IEEE Computer Society, 1976, pp. 592-605.

[36] A. Moreira, J. Araujo, and I. Brito, "Crosscutting Quality Attributes for Requirements Engineering", In 14th Int. Conf. on Soft. Eng. and Knowledge Engineering, Ischia, Italy, 2002, pp. 167-174.

[37] D. Park, S. Kang, and J. Lee, "Design Phase Analysis of Software Performance Using Aspect-Oriented Programming", In 5th Aspect-Oriented Modeling Workshop in Conjunction with UML 2004, Lisbon, Portugal, 2004.

[38] J. R. Hauser and D. Clausing, "The House of Quality", Harvard Business Review, May – June 1988, (pp. 63- 73).

[39] T. H. Al Balushi, P. R. Sampaio, D. Dabhi, and P. Loulopoulos, "ElicitO: A Quality Ontology-Guided NFR Elicitation Tool", Proc. Of REFSQ 2007, Requirements Engineering: Foundations for Software Quality, Trondheim, Norway, June 11-12 2007, pp. 306-319.

Towards Better Comparability of Software Retrieval Approaches Through a Standard Collection of Reusable Artifacts

Oliver Hummel, Werner Janjic
Software Engineering Group
University of Mannheim
Mannheim, Germany
{oliver, werner}@informatik.uni-mannheim.de

***Abstract* — The idea of component-based software reuse as a cornerstone of a more engineering-like approach to software development has been around for more than four decades. Since software and its building blocks represent an important and valuable intellectual asset for most companies, researchers have been struggling for nearly the same time to get their hands on a substantial amount of reusable material to experiment with. Only the advent of the open source movement mitigated this problem considerably and hence inspired interesting new research in this area within the last decade. However, basically all novel software retrieval solutions of that period have been developed and evaluated independently from each other and are thus by no means comparable with one another. To address this flaw, an initiative was started to foster the creation of a reference reuse collection for software search and retrieval, which is intended as a common baseline for future comparison of software retrieval systems. In this paper we explain the motivation for this initiative, identify and discuss important foundations as well as open issues and present an initial sketch of architecture, content and practical prerequisites of such a collection.**

Keywords-component-based software development; software reuse; software search; software retrieval; reference collection.

I. Introduction

Despite the immense benefits that are attributed to the reuse of software [1] and a large number of seminal approaches (such as by Zaremski and Wing [2], [3]; see e.g. Mili et al. [4] for a comprehensive overview) developed in recent decades, Douglas McIlroy's initial idea of setting up market places with reusable components [5], [6] still has not lived up to its full potential [7]. Nevertheless, given today's exploding amount of potentially reusable (open source) software, freely available on the Internet, the need for effective software search and retrieval solutions – not only as an enabling factor for reuse – is more apparent than ever: open source repositories such as Sourceforge are hosting tens of thousands of software projects with millions of artifacts and even the version control systems of larger companies contain more files than a human can ever overlook.

Consequently, the so-called reuse repository problem [8] of not having enough material to fill repositories and market places with reusable components is no longer an issue since the Internet and the World Wide Web can be used as a source for harvesting reusable material [9]. However, in order to use this "megastore" of information for systematic software reuse, sufficiently sophisticated software retrieval approaches and tools are necessary. Especially when existing material was not initially intended for reuse (as is the case with most open source software today), this will only become accepted if developers are able to find and access useful components quick and easy. Consequently, this change of prerequisites has not only triggered a new wave of interesting academic research to better deal with search and retrieval of software artifacts (e.g. [10], [11], [12]), but has also created a new interest of commercial search engines (such as Koders, Krugle or formerly Google Codesearch) in searching for source code and software. Although all approaches available today are certainly important and have brought a new momentum to the community, they share one significant problem: to date, their evaluations, if existing at all, are largely based on different and/or proprietary datasets and thus it is impossible to objectively compare their performance on a common basis. Since even researchers are not able to assess the existing solutions and to understand their strengths and limitations, it is no surprise that software search and component reuse are still not widely adopted in industrial practice.

Interestingly, this evaluation challenge is not limited to component reuse alone; it is rather a problem that has been plaguing computer science (and especially software engineering) for a while. As observed by Tichy [13], computer scientists perform relatively little evaluations of their approaches so that the experimental paradigm is not as well established in our world as, for example in medicine or physics. In other words, computer scientists often focus too much on the development of new approaches and too little on their systematic evaluation, which makes it hard if not impossible to judge whether a new approach is really better than the previous ones. Certainly, the development of new approaches is important, but nevertheless, repeatable evaluations of new developments are at least as important for good research, as e.g. stressed by Basili [14] about twenty years ago: *"Proposing a model or building a tool is not enough. There must be some way of validating that the model or tool is an advance over current models or tools"*. In order to overcome this unsatisfying situation in the area of software search and retrieval, the creation of a reference collection of reusable artifacts was proposed recently [15]. The main motivation for this effort is to simplify the comparison of software retrieval systems. Furthermore, as already experienced in the text retrieval community such a collection will offer a good starting point for the development of new and innovative tools as it freely provides the data necessary for initial experiments [16].

The remainder of this paper, discussing motivation and early experience from setting up such a collection, is organized as follows: First, we introduce the foundations of retrieval techniques and their evaluation, which is required to better understand the contribution of this paper and the usage scenarios in which the proposed reference collection can be used (Section 2). Subsequently, we briefly survey existing tool evaluations and identify their common weakness, before we shed some light on reference collections from related areas and how their ideas can be transferred to a standard collection of reusable components (Section 3). Section 4 introduces our approach for tackling this challenge and gives an example supposed to illustrate the usability of the approach before we conclude our paper in Section 5.

II. FOUNDATIONS

The origins of software search and retrieval can clearly be seen in "classic" text information retrieval [4] and therefore most early approaches for the former simply applied techniques from document retrieval to software artifacts (cf. [17]). Software retrieval, however, is potentially a far more complex undertaking than pure text retrieval since software does not only contain linguistic semantic information, but syntax and functional semantics as well. Zaremski and Wing were amongst the first researchers that elaborated on signature [2] and semantics specification [3] matching as a way of identifying reuse candidates. About ten years ago, Mili et al. [4] have presented a well-known survey that identifies five general groups of techniques applicable for the retrieval of software artifacts, namely –

1. Information retrieval methods
2. Descriptive methods
3. Operational semantics methods
4. Denotational semantics methods
5. Structural methods

The original listing contains a sixth group, called topological approaches, which from today's point of view is rather an approach for the ranking of search results than a retrieval approach itself so that we have left it out in the enumeration. It obviously makes sense to reuse methods from information retrieval to perform simple textual analyses on software assets. Descriptive methods go one step further and require additional textual descriptions of the asset like a set of keyword or facet [18] definitions. Operational semantic methods rely on the execution or so-called sampling [19] of the assets. Denotational semantics methods use signatures (see e.g., [2]) or specifications [3] of artifacts for matching, while structural methods do not deal with the code of the assets directly, but with program patterns or designs. Overlap between these classifications can occur at various places, e.g., between (3), (4) and (5) as "behaviour sampling" [19] of components typically needs a specific signature to work on.

Based on the numerous results that had been presented in the late 1990s some researchers were even convinced that the most important software retrieval challenges have already been solved [20]. Existing prototypes were able to deal with the artifact collections available at that time easily (containing, however, often merely a few dozen elements). On the contrary, other researchers were convinced that the existing techniques would not be precise and usable enough when the amount of reusable material grows larger [4], which has been commonly seen as a required condition for successful marketplaces with reusable artifacts [7]. The latter assumption has at least preliminary been proven right almost ten years later when initial experiments [9] with "internet-scale" software collections have shown that the usage of merely one of the above mentioned retrieval techniques is usually not precise enough to deliver practically usable results. These experiments showed, e.g., that the precision of signature matching quickly drops to under one percent in collections with millions of artifacts. Consequently, in recent years, there has been an increasing interest in improving software retrieval approaches that led to a number of interesting approaches (as well as a number of high-profile publications [10], [11], [21]). Although their documentations include reasonable evaluations that demonstrate the prototypical applicability of the underlying approaches, it is impossible to compare them with each other as they were developed independently and evaluated with totally different methods and test collections. Even worse, the examples used to experiment with the prototypes are usually not publicly available and hence it is extremely difficult to judge the actual effectiveness of the evaluations and basically impossible to replicate the experiments performed.

Due to the conceptual proximity to information retrieval it is no surprise that common evaluation techniques from classic information retrieval are widely applied in the context of software retrieval. The two most prominent measures to assess the quality of retrieval systems are Precision *P* (measuring the fraction of relevant results D_r amongst all delivered results *D*) and Recall *Re* (the fraction of delivered relevant results D_r amongst all relevant results *R*):

$$P(D_r, D) = \frac{|D_r \cap D|}{|D|}$$

$$Re(D_r, R) = \frac{|D_r \cap D|}{|R|}$$

Further well-known but not so commonly used measures include Fallout (the fraction of non-relevant documents that is retrieved from all non-relevant documents) and the F1 measure (the weighted harmonic mean calculated from Precision and Recall) [16].

Recall is typically more important on small collections or on large collections with very specialized queries (where one assumes to have only few useful results per query), while Precision becomes more important on large collections with potentially numerous results. In this context, a tool should clearly minimize the amount of false positives since delivering only few relevant results amongst thousands of irrelevant candidates will not only result in a poor precision, but also in a low user satisfaction. It is obvious that such a behavior is not tolerable in the area of software reuse where a careful assessment, selection and integration of potential reuse candidates may demand significant effort from the developer. Hence, a thorough assessment of a retrieval system typically requires the combination of at least two measures, since otherwise a system can be optimized for one and may fail in practice. The common approach to combine e.g., Recall and Precision in such evaluations is through so-called Recall/Precision curves [16] in which a large area under the curve indicates a well performing retrieval system. If other practical aspects are of interest other measures can be derived as well, including the search execution time of a query, for example.

A. Assessing Software Retrieval Tools

Although this general approach for the evaluation of software retrieval tools is undisputed, one aspect that complicates the evaluation is the challenge of defining the actual relevance of a reusable artifact. While determining the relevance of natural language documents is pretty straightforward for a human (e.g., does a document tell you how high Mount Everest is or not?), this task is much more challenging for software artifacts. As discussed before, the latter typically have three facets that can be used for retrieving them, namely linguistic information, the syntax of their interfaces and their semantics, i.e. their concrete functionality. As already observed by Mili et al. [4], the evaluation of software retrieval tools and algorithms is faced with a serious problem when it needs to find a good criterion that determines the practical relevance of a delivered result. Usually none of the three facets mentioned before is sufficient to achieve this on its own, as, e.g., text extracted from a component not necessarily describes its functionality in a precise and unambiguous manner; and even if a component with matching functionality has been found, a wrong interface might make its integration into a given environment hard or even impossible. In other words, a reusable component delivered by a state of the art software search engine might still require a significant amount of effort from a developer in order to finally determine whether it provides the desired functionality and is usable in the environment at hand. We believe, the ultimate *relevance criterion* for determining the reusability of a component and solving the make or buy dilemma [25] in favor of buy (reuse) is clearly the question whether a reusable artifact can be integrated into a system under development “as is”, i.e., with virtually “zero effort” and deliver the required functionality.

To our knowledge, however, this relevance criterion has rarely been consequently defined in the literature so far and thus, most previous evaluations have been relying on a kind of surrogate, namely the so-called *matching condition* that simply determines whether a search engine considers a document as relevant or not. Obviously, this does not reveal much useful information about the reusability of a component in a given context.

B. Usage Scenarios

Software development is a continuous and complex process that can benefit from software search at various occasions, which makes it important to identify and to bear in mind which usage scenarios exist for software retrieval tools within the software development lifecycle. Obviously, the process of “reusing” an artifact as an inspiration during the design or implementation phase of a software system is totally different to the actual reuse of a concrete component that needs to adhere to a given specification. While a stakeholder may be satisfied with relatively “blurry” results for the former, the latter requires a perfect match in order to make reuse more worthwhile than building the component from scratch, as explained before. Figure 1, taken from our earlier work [22], summarizes various archetypal usage scenarios for software retrieval systems and identifies the development activities where they are likely to be most useful. We used different shapes of lines, to illustrate distinction between more *speculative* (dashed) and *definitive* (solid) searches. The most important usages scenarios in the context of this paper are additionally highlighted in bold typeface.

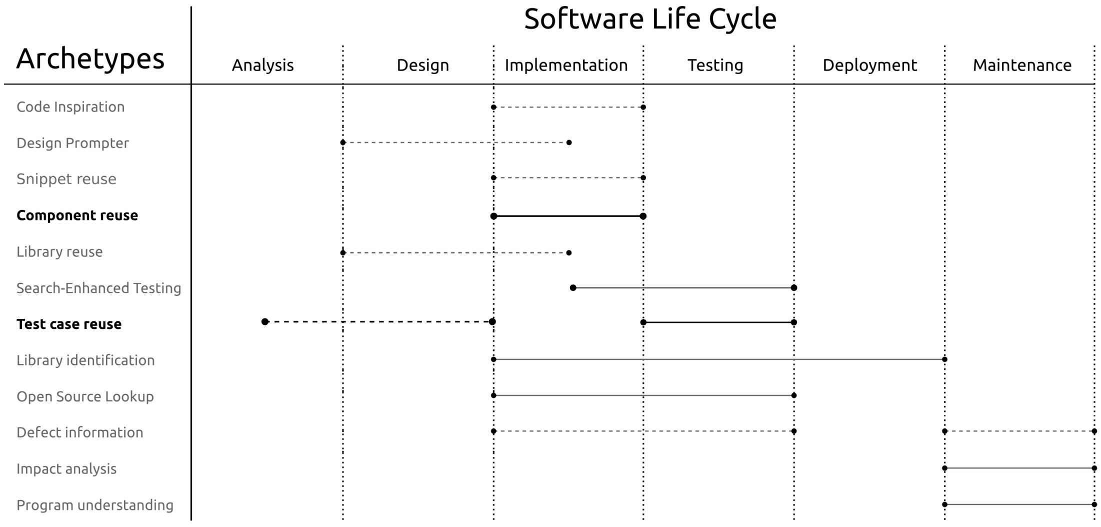


Figure 1. Overview of software retrieval usage scenarios and their possible times of application in the software development life cycle.

The motivation for definitive searches is always the concrete need for a specific artifact; let it be a reusable component described by a specification or a missing library that is required to overcome Java's infamous ClassNotFoundException. Both require a search engine to deliver as exact matches as possible and even near misses are usually not perceived helpful by the user, since they can often only be integrated with a substantial amount of modification, if they can be integrated at all. On the other hand where speculative searches are driven more by general information need, it is usually of interest whether any component or service is available for a given task at all. The exact shape of the desired artifact is typically not important in the context of such a speculative search, since the design of the application it should be integrated into is still moldable [23]. Due to a lack of space, we have to refer the interested reader to the original publication for more details on this topic; the focus of this paper and our initial work for a reference collection is on definitive searches used for the retrieval of well-defined reusable components.

III. Software Retrieval Evaluations so far

As indicated in the introduction, most software reuse approaches that have been published so far contain a reasonable evaluation that demonstrates their feasibility and leaves the interested reader at least with an idea of their potential and of potential problems. However, as can be seen in Table 1, that summarizes some of the best known reuse tools of the last 20 years, most of these evaluations (i.e., those that were performed on a component collection with more than just a few hundred elements) were incomplete from the perspective of classic information retrieval as they usually only calculated some kind of "top n Precision".

TABLE I. Overview of evaluations of previous component reuse systems.

Tool	**No. of Artefacts**	**Content**	**Input**	**Relevance Criterion**	**Measures**
Proteus [17]	~ 100	Unix commands	Keyword-based	Expert judgement	Precision, Recall, search time
CodeBroker [28]	~700	Java Classes	Signature and keywords	Expert opinions	Precision, Recall
SparsJ [10]	~180,000	Java Source Classes	Keywords	Expert opinions	Top n Precision
Maracatu [[29]]	~4,000	Java Source Classes	Keywords, facets	Expert's opinion (based on text matching)	Precision, Recall (only for subset of 200 artefacts)
Merobase [11]	~ 4 M	Java Source Classes	Test Cases	Passing of test cases	Top n Precision
Sourcerer [12]	~ 250,000	Java Source Classes	Keywords	Expert judgement	Hits per result page

This kind of "crippled" precision measure is typically used for search engines that operate on very large collections where it is not feasible to determine the relevance of all (potentially thousands of) results that may be returned for a query. Instead, human experts revise only the, e.g., 20 highest ranked results (n = 20) for their relevance. Further results and the Recall (for which knowledge of all relevant elements in a collection is required) are simply ignored. This procedure is usually justified by the habit of human users of internet-scale (commercial) search engines that typically do not consider more than roughly the first 20 results. However, for a scientific comparison of search engines this is obviously neither sufficient nor satisfying, especially in the area of software retrieval where both, high precision and high recall are essential as explained before.

A. *Reference Collections so far*

Tool evaluations in computing are often challenging, as they typically require expensive empirical investigations to demonstrate that a tool is better than other tools available before [14]. However, software engineering is certainly not the only discipline in computer science that has to deal with somewhat fuzzy requirements to its tools. Therefore, the idea of creating reference collections that allow benchmarking of tools is certainly not new. Take, for example, the "Siemens Testing Suite" [30], a popular collection of programs containing known errors, which was widely used during the 1990s to evaluate the effectiveness of test cases and test case creation strategies. More specifically, the challenge of evaluating retrieval approaches is clearly known in related disciplines as well. First and foremost, it is certainly the information retrieval (IR) community [16] that found itself in trouble how to evaluate their emerging text retrieval algorithms some twenty years ago. At that time there were a lot of new and exiting ideas as well as prototypes around in this community, but the proprietary (and often very expensive) evaluations performed on them individually were usually not very helpful and especially not comparable with each other. Fortunately, the IR community was able to overcome this challenge by defining so-called reference collections comprising a large set of documents, a substantial number of tasks for retrieval systems and the expected solutions for them. The most prominent one is probably the Text REtrieval Collection (TREC) [16] that has been considered as a major success fostering IR research since its creation tremendously. Although TREC as a text-based collection is not of direct use for the retrieval of software artifacts, it can still be used to learn about some basic principles how to define and built such a reference collection. Furthermore, in the long term, the results gained with it might also be helpful by giving some insights on heuristics that can help to improve text-based retrieval algorithms for software retrieval tools with techniques such as stemming or the use of thesauri [16].

A second group that has been struggling with the comparability of its tools is the rather young community trying to match and orchestrate (semantic) web services. As it is dealing with executable artifacts as well, it is obviously more closely related to the retrieval of software components than pure text retrieval. Given the enormous amount of money that for instance was recently spent by the European union (Küster and König [24] talk about 70 million Euros) to support the research on semantic web services, it is not a surprise that especially European researchers came up with the idea of setting up a reference collection of semantic services to evaluate matching tools and have been driving this idea ever since. The so-called S3 (for Semantic Service Selection) collection is the initial result of these endeavors. The current version of S3 contains 1.083 semantically (with 38 ontologies) annotated web services and a set of 42 queries for them. Various participants of the S3 contests and related workshops have manually identified services in the collection they considered relevant for each query in order to create a set of relevant answers. To our knowledge together with the OPPOSUM portal [24] (that subsumes S3 and a few significantly smaller collections) it forms the only baseline that allows systematic comparison of (ontology-based) software retrieval algorithms so far. To our knowledge, there exists no similar undertaking for a specific reference collection in the reuse area for the time being.

Limitations of Web Service Reference Collections

However, although this can also be seen as a first step towards a better evaluation of software retrieval algorithms, its applicability in the context of software reuse is questionable for a number of reasons. First and foremost, the introduction of a graded relevance scheme and the revision of the relevant results in the 2010 version of the S3 collection changed the perception of relevance considerably and it seems that there is still a large degree of subjective judgment that influences the understanding of relevance here. Thus, the risk that even this sophisticated collection does not contain a clear notion of relevance, as we demanded it for the evaluation of software reuse tools, is high. Second, most of the existing software retrieval and reuse approaches operate on source code, which is by definition not available from web services, while vice versa, source code available in open source repositories is usually not annotated with any kind of ontological information. Moreover, the size of the existing S3 collection is still rather limited (compared with current software reuse collections as introduced in Table 1) and the chance of substantially increasing it seems low, as the definition of relevant results and the annotation of the indexed services with ontological information is effort manual activity. Given the size of state of the art reuse collections that already goes into the millions, it is not clear whether the results obtained from such a small collection can be scaled up to internet-scale search engines. Finally, the current S3 collection is focused on speculative searches that are supposed to deliver all services that can be (remotely) helpful for a query; the actual syntax of a service is currently not taken into account. In other words, a definitive match between query and result is highly unlikely in this collection and a composition of various results may be required to create a service that is finally able to satisfy a concrete request.

B. Requirements for a Reuse Reference Collection

Küster and König [24] identified a number of desirable characteristics for a semantic web service collection in their publication and obviously it makes sense to revisit their work as a starting point for a reference collection for software reuse. In total they list the following five major points:

1. Expressivity & Usability: contained elements need to be described as precisely as possible in order to avoid room for interpretation of the results.
2. Scope: the collection should comprise elements from as many different domains as possible in order to maintain a high diversity and to allow making statements, which approaches work under which circumstances.
3. Scalability & Size: since large testbeds are required to properly evaluate retrieval approaches, the collection must be kept scalable.
4. Automation: obviously, the use of the collection should be automated as much as possible.
5. Decoupling: as many people as possible should contribute to the endeavor in order to avoid unintended bias in the collection.

In general we can accept this list of requirements as helpful for a reuse reference collection as well, although requirements 1) and 3) are clearly contradicting each other in the context of a very large collection. This fact makes a precise relevance criterion even more important because it is not possible to manually investigate millions of artifacts for their relevance. But nevertheless, it is important to preserve as much information as possible when content for the collection is harvested, as different usage scenarios for software search engines may require slightly different information to evaluate the retrieval algorithms.

Special Requirements in the context of Component Reuse.

As discussed before, the main motivation for the use of a component collection from a reuse point of view is to find a concrete artifact that definitively fills an existing gap. Besides other factors, it has been mentioned numerous times in the literature [25] that a reusable component must be large enough so that reusing it is cheaper and easier than self-implementing it (often called the “make or buy decision”). Otherwise the incentive for a developer to reuse is obviously low. While a component was initially seen as function by McIllroy [5] in his seminal reuse paper, the granularity of components has continuously been growing since then and today a component is typically seen as an independently deployable part of a system [6], comprising numerous classes (if developed in an object-oriented language) behind a well-defined interface. At the same time, research has started to investigate automated adaptation [26] of components and automated orchestration [24] of (usually semantically annotated) services or in other words: automated "glue coding". As the use of clever glue coding is likely to increase the haul of matching components from a collection (and thus to influence the number of relevant components) in the future, we believe it makes sense to consider the following three categories of automated glue coding when competing search engines are to be evaluated in the context of a reuse reference collection:

1. No glue coding at all: only direct matches are allowed, no changes beyond simple path and package configurations can be made to reuse candidates.
2. Adaptational glue coding: adapters [27] that wrap a single component in a 1:1 fashion (or change them internally) are allowed.
3. Compositional glue coding: the 1:n orchestration of multiple sub-components behind a newly created interface in the sense of the facade pattern [27] is allowed.

IV. Proposed Approach

The two most important "ingredients" for a reuse reference collection are certainly a large collection of reusable material and a large enough collection of (at least some) non-trivial queries that can be used to challenge search engines and is not under the suspicion of being biased for a particular engine. Moreover, a good way of determining the relevance of retrieved candidates needs to be found. Since we have already faced this challenge during the evaluation of our Merobase search engine [11], we believe a good starting point for this is the collection of test cases (e.g., written in JUnit) that can be used to doubtlessly judge whether a delivered result is relevant or not. Ideally, a search engine would directly support the use of such test cases for automating this assessment, as Merobase does, for example. The technique behind such a feature is known in the literature as test-driven reuse [11] [21]. The following two subsections go into more detail on this before we present the results of an exemplary query that demonstrates the practical usability of our approach and conclude this section with a brief discussion of our preliminary findings.

A. Data Sets

In the context of the ICSE workshop on Search-driven development: Users, Infrastructure, Tools and Evaluation (SUITE) in 2010 a working group was formed with the goal to evaluate the feasibility of creating a reuse reference collection. As a result, the groups of Christina Lopez at the University of California in Irvine and our group at the University Mannheim have agreed to make the collections forming the backbone of the software search engines Sourcerer ([12], http://sourcerer.ics.uci.edu) respectively Merobase ([11], http://merobase.com) available on the Web so that they can be downloaded via http://resuite.org. Currently, several hundred gigabytes of data are available there and hence processing and indexing these collections is certainly a matter of weeks if not months: Irvine's collection comprises about 500,000 java source files from roughly 13,000 open source projects, while our collection consists of about 3 million java files harvested from nearly 50,000 open source projects. To our knowledge these two packages form the largest body of open source software freely available today. Given its large size, it is likely that it will not only be facilitating experiments for software reuse, but in related areas (such as the community organizing the Mining Software Repositories conference) as well.

B. Queries

As briefly mentioned before, we plan to start the creation of queries for the reference collection based upon the test cases we created as input for evaluating our earlier work [8] to which we have to refer the reader for further details due to the limited space of this paper. To our knowledge, test cases are currently the best available technique that allows formulating a semantically precise and automatically checkable specification for reusable software components. A further advantage of test cases is that they can easily be "translated" into input for other retrieval approaches as well. Consider the following simple JUnit test case that is supposed to test an equally simple Stack data structure:

```
public class StackTest extends TestCase {
     public void testStack() {
       Stack s = new Stack();
       assertTrue(s.isEmpty());
       s.push((Object)"Object1");
       s.push((Object)"Object2");
       s.push((Object)"Object3");
       assertFalse(s.isEmpty());
       assertEquals(s.pop(), (Object)"Object3");
       assertEquals(s.pop(), (Object)"Object2");
       assertEquals(s.pop(), (Object)"Object1");
       assertTrue(s.isEmpty());
     }
   }
```

From this piece of code it is, for example, possible to extract keywords (such as stack, push, pop, isEmpty) or the complete interface of a stack required to satisfy this test case without much ado. Moreover, even the extraction of a simplified description of the Stack's behavior is contained in this test case. In addition to the above mentioned set of test cases we are aware of two other recent publications that used test cases (or at least test data) for a similar purpose and contain further evaluation challenges (cf. [21] & [12]). We have recently made all test cases that have been used for test-driven reuse available as JUnit test cases via resuite.org as another pillar for the reference reuse collection described in this paper. Since all three approaches are currently in a prototypical stage, there is no precision recall analysis available. Nevertheless, a sufficient quality of the test cases guarantees that retrieved candidates are able to deliver the desired functionality. Reusable components that have actually been retrieved beyond simple data structures such as stacks or binary trees, include a validator for credit card numbers, spreadsheet calculation and Blackjack logic, a comprehensive overview can be found in the mentioned publications ([11], [12], [21]).

C. *Examplary Results*

Based on the test case we just introduced, we carried out an exemplary analysis of various retrieval techniques in order to show the expressivity improvement that we were able to achieve in comparison to the simple top 20 precision determination used in earlier evaluations [11]. In particular, we analyzed interface-based (only classes with identical interfaces, i.e. all names, parameter and return types had to match), name-based (only names of classes and methods had to match) and signature-based (only the parameter and return types had to match, names were ignored [2]) searches for their Recall and Precision as shown in the following figure:

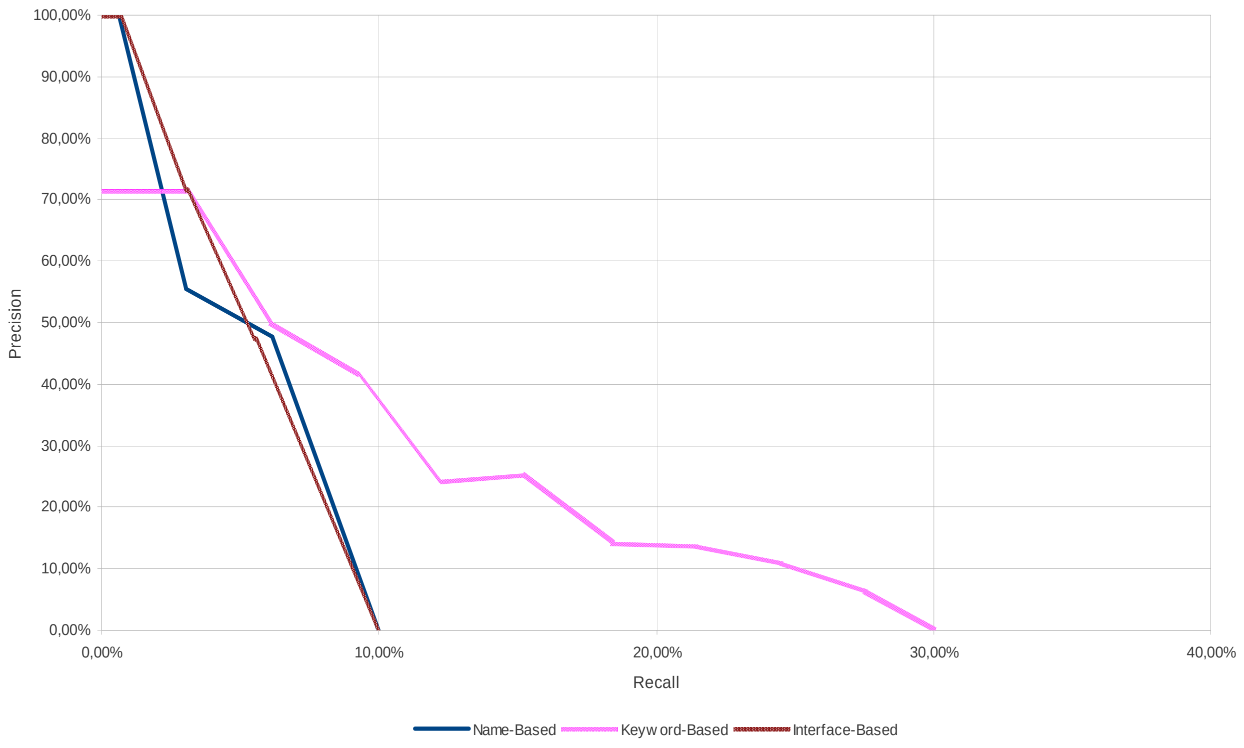


Figure 2: Recall/Precision curve of different retrieval algorithms for the Stack test case from above.

In total, the signature required by the above test case yielded a pool of 454,541 classes from the Merobase collection that at least contained the three method signatures defined and thus theoretically had the potential for being usable as stacks. In practice, only a small fraction of them – namely 163 – have been successfully tested and delivered this functionality with the current version of our testing tool (supporting adaptation and rudimentary dependency resolution, but no composition). Thus, 163 was used to calculate Recall and Precision. As visible in Figure 2, the relatively simple retrieval algorithms used for this experiment suffer from either a low recall or a low precision as summarized in the subsequent table.

TABLE II. COMPARISON OF RETRIEVAL ALGORITHMS.

	Name	**Interface**	**Signature**	**Keyword**
Max. Recall	8.0 %	5.5 %	100 %	28.2 %
Precision at max. Recall	50.0 %	47.4 %	< 0.1 %	5.8 %
No. Relevant / Candidates	13 / 36	9 / 19	163 / 454,541	46 / 3,000

D. *Discussion and Forthcoming Steps*

The approach we have just described already forms a useful core for a reference collection of reusable artifacts. Our preliminary results indicate that an evaluation based on such a collection with results known as relevant is feasible for internet-scale software repositories as well and delivers significantly better results than the top n precisions usually calculated for such repositories. However, as long as only one tool with potentially imperfect adaptation has been used to identify the relevant results for a query, it is not sure that all relevant results have actually been discovered. Only a combination of various tools and approaches can guarantee a (nearly) perfect coverage of relevant results and thus create a valid baseline for the calculation of Recall and Precision.

Therefore, one central prerequisite for the creation of a viable reference collection is to have a large body of researchers and working groups contributing their ideas and tools. We would like to invite the community to challenge, discuss and extend the requirements and the contents of the collection in its current state. Although contact requests via email are always welcome, we believe it makes sense to discuss the further proceeding personally with as many people as possible and hence plan to organize a workshop on this topic soon. Another important future step is the creation of a web portal (similar to the one described by Küster and König [24] for semantic web services) that offers easy access to data sets and queries.

E. Open Questions

Compared with a simple text reference collection or even a web service collection, a generic reuse collection is faced with a number of additional questions we briefly need to mention at this occasion.

Although web service descriptions are by definition programming language independent, the elements of a collection of reusable components, however, can be created in any arbitrary programming language. In other words, their evaluation would require support for test-driven reuse in each of these languages. For the time being only three different prototypes of a test-driven reuse system in the Java programming language exist. Clearly, it makes sense to set up similar collections for other languages in order to study whether a different language will affect the performance of retrieval algorithms in any way.

Another issue closely related with the programming language is the question whether an artifact is compilable and executable at all. Often source files have dependencies on other source files and will not be testable without either complex dependency resolution algorithms as available in the Eclipse framework and used by e.g., Code Genie [12] or without the complete metadata (build path, etc.) of the original project. Since most software search engines today still focus on individual classes (cf. table 1) we rely on the simple dependency resolution mechanisms contained in our tool right now and bear in mind that they are not perfect. Hence, it is likely that other tools might discover additional relevant results in the future through the use of better dependency resolution. However, a similar progression of relevant results has been observed during the creation of the TREC collection, so that this is perfectly acceptable.

The TREC collection has another advantage over a software reference collection, namely the one that texts that are once written (such as newspaper or research articles) typically are not changed later. However, in the context of software retrieval it is very likely that the projects forming the collection will be updated over time and hence the question arises whether and how updates can be performed. Updating the collection itself is essentially noncritical as it just calls for replacing, adding or removing files; the actual challenge is to identify all results that may have become relevant or irrelevant after such an update.

A final issue to deal with is the question how to decide what makes a component or project elevated enough to become part of a reference collection? We are well aware that one may allege a certain bias for elements included e.g., in our Merobase collection so far. However, we believe that the sheer size of about 3 million Java source files is already large enough to mitigate such allegations. Furthermore, it only contains open source projects harvested from popular open source hosters (such as SourceForge) and no specifically tailored projects that would harden this suspicion. Moreover, the idea is that a future collection is extensible so that everyone interested is able to contribute his own material to it.

V. Conclusion and Future Work

Component-based software reuse is by no means a new concept and as widely demonstrated in the literature a sophisticated software retrieval tool is an essential building block to make reuse work. However, despite decades of intensive research and the significant progress made in software retrieval in recent years, it is still hard to compare existing reuse approaches, as there exists no common testbed for this purpose. As we have discussed in this paper, recent efforts to set up a semantic web service reference collection are certainly a step in the right direction, however, since the prerequisites and goals of this community are different to those of the component reuse community, it is unlikely that results gained with this collection can be transferred to software component retrieval.

Thus, we have proposed to create a reference collection with reusable components based upon two recently published collections of files from more than 50,000 open source projects. Our proposal includes creating definitive queries for concrete reusable artifacts in the form of test cases that can be used to determine free of doubt whether a delivered candidate will be usable in a given context specified by the test case. Such test cases may be seen as a rather harsh relevance criterion for reusable software components, but ultimately they are the only way to establish the fitness for purpose of a component and in our understanding this is the only way to lower the threshold currently still hindering systematic reuse in practice. Moreover, we defined three classes of adaptation approaches that may be used to classify the contestants that should be compared with a reuse reference collection.

Such a collection will not only be applicable for comparing existing tools with full Recall / Precision curves, it is also likely that it will simplify the creation of and initial experimentation with other innovative tools in the future. Furthermore, there is a high chance that the data sets will be useful for other communities (such as the one that is mining software repositories, for example) as well and hence we invite researchers from all related areas to contribute to the efforts in setting up this collection as well.

References

[1] Krueger, C.W.: Software Reuse, ACM Computing Surveys, Vol. 24, Iss. 2, 1992.

[2] Zaremski, A.M. and Wing, J.M.: Signature Matching: A Tool for Using Software Libraries. ACM Transactions on Software Engineering and Methodology, Vol. 4, Iss. 2, 1995.

[3] Zaremski, A.M and Wing, J.M.: Specification Matching of Software Components, ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 4, 1997.

[4] Mili, A., Mili, R., and Mittermeir, R.: A Survey of Software Reuse Libraries. Annals of Software Engineering 5, 1998.

[5] McIlroy, D.: Mass-Produced Software Components. Software Engineering: Report of a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 1969.

[6] Szyperski, C.: Component Software, 2nd Edition, Addison-Wesley, Amsterdam, 2002.

[7] Ravichandran, T. and Rothenberger, A.: Software reuse strategies and component markets. In Communications of the ACM, 46, 8, 2003.

[8] Hummel, O.: Semantic Component Retrieval in Software Engineering, PhD Dissertation, University of Mannheim, Germany, 2008.

[9] Hummel, O. and Atkinson, C.: Using the Web as a Reuse Repository. In: Morisio, M. (ed.) Proceedings of the International Conference on Software Reuse, LNCS 4039, Springer, Heidelberg, 2006.

[10] Inoue, K., Yokomori, R., Fujiwara, H., Yamamoto, T., Matsushita, M., and Kusumoto, S.: Ranking Significance of Software Components Based on Use Relations. IEEE Transactions on Software Eng., Vol. 31, Iss. 3, 2005.

[11] Hummel, O., Janjic, W., and Atkinson, C.: Code Conjurer: Pulling Reusable Software out of Thin Air. IEEE Software, Vol. 25, Iss. 5, 2008.

[12] Bajracharya, S., Ossher, J. and Lopes, C.: Sourcerer: An internet-scale software repository. Int. Workshop on Search-Driven Development, SUITE 2009.

[13] Tichy, W.: Should computer scientists experiment more? IEEE Computer, Iss. 5, 2002.

[14] Basili, V.: The Experimental Paradigm in Software Engineering. Experimental Software Engineering Issues: Critical Assessment and Future Directions, Springer, 1993.

[15] Hummel, O.: Facilitating the Comparison of Software Retrieval Systems through a Reference Reuse Collection. Int. Workshop on Search-Driven Development, SUITE 2010.

[16] Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval, Addison-Wesley, 1999.

[17] Frakes, W.B. and Pole, T.P.: An Empirical Study of Representation Methods for Reusable Software Components. IEEE Transactions on Software Engineering Vol. 20, Iss. 8, 1994.

[18] Prieto-Díaz, R.: Implementing faceted classification for software reuse. Communications of the ACM, Volume 34, Issue 5, 1991.

[19] Podgurski, A. and Pierce, L.: Retrieving Reusable Software by Sampling Behavior, ACM Transactions on Software Engineering and Methodology, Vol. 2, Iss. 3, 1993.

[20] Poulin, J.: Reuse: Been There, Done That. Commun. of the ACM, Vol. 42, Iss. 5, 1999.

[21] Reiss, S.P.: Semantics-based Code Search. Int. Conf. on Software Engineering, 2009.

[22] Janjic, W., Hummel, O., and Atkinson, C.: More Archetypal Usage Scenarios for Software Search Engines. Int. Workshop on Search-Driven Development, SUITE 2010.

[23] Crnkovic, I., Chaudron, M., and Larsson, S.: Component-based Development Process and Component Lifecycle, Proc. of the Intern. Conf. on Software Engineering Advances, 2006.

[24] Küster, U. and König-Ries, B.: Towards standard test collections for the empirical evaluation of semantic web service approaches. Int. Journal Semantic Computing, Vol. 2, Iss. 3, 2008.

[25] Clements, P.: From Subroutines to Subsystems: Component-Based Software Development. in Heineman, G., Councill, W. (eds..: Component-based Software. Eng. Ad.-Wesley, 2001.

[26] Hummel, O. and Atkinson, C.: Automated Creation and Assessment of Component Adapters with Test Cases. Intern. Symposium on Component-Based Software Engineering, 2010.

[27] Gamma, E.; Helm, R.; Johnson, R., and Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software, Addison-Wesley, Amsterdam, 1995.

[28] Ye, Y. and Fischer, G.: Reuse-Conducive Development Environments. Journal of Automated Software Engineering, Vol. 12, No. 2, Kluwer, 2005.

[29] Garcia, V., Lucrédio, D., Durão, F., Santos, E., Almeida, E., Fortes, R., and Meira, S.: From Specification to Experimentation: A Software Component Search Engine Architecture, International Symposium on Component-Based Software Engineering, CBSE 2006.

[30] Hutchins, M., Foster, H., Goradia, T., and Ostrand, T.: Experiments on the effectiveness of dataflow- and control flow-based test adequacy criteria. International Conference on Software Engineering, 1994.

Improving IT Infrastructures Representation: A UML Profile

Luís Ferreira da Silva[1]
luis.alexandre@campus.fct.unl.pt

Fernando Brito e Abreu[2,1]
fba@iscte-iul.pt

Victor Moreira[2]
vitor_hugo_moreira@iscte.pt

[1]QUASAR Group, CITI, FCT/UNL
Universidade Nova de Lisboa
2829-516 Caparica, Portugal

[2]DCTI, ISTA, ISCTE-IUL
Instituto Universitário de Lisboa
1649-026 Lisboa, Portugal

***Abstract*— IT infrastructures are most times informally modeled. The resulting models are ambiguous to stakeholders, cannot be checked for validity, and therefore are unable to play their important role in design, deployment and maintenance activities. The main reason for such a poor state-of-the-art lies mainly in the absence of a modeling language capable of representing IT infrastructures at the required level of abstraction. Indeed, existing candidate languages are too abstract, as shown in this paper by reviewing their metamodels. The present paper mitigates this problem by proposing a UML profile to describe the semantics of an IT infrastructure.**

Keywords – Information Technology; IT Infrastructures; UML Profile; Modeling; Design Patterns

I. Introduction

An *Information Technology Infrastructure (ITI)*, also known as *Technology Architecture* in most Enterprise Architecture frameworks, is the foundation on which business processes that drive the success of an organization are based [1] and has been defined as "*all hardware, software, networks and facilities, etc. that are needed to develop, test, deliver, monitor, control or support IT services*" [2]. Some of the unique characteristics of the ITI layer are:

- Is the foundation for all the other architecture layers, meaning that a problem at this layer can influence all the layers above;
- Provides services that are used by multiple applications, processes, and users;
- Is usually not perceived as a layer that offers financial benefits, but rather as an utility that enables business processes to be performed.

The ability to streamline the conception, deployment and maintenance of ITIs depends largely on our ability to model them, as in most engineering endeavors, to produce complex artifacts. The use of models was introduced in Computer Science in the seventies to simplify complexity in Software Engineering. Models convey a simplified representation of part of the real world, which can be useful for analytical purposes. The use of models provides a way to view specific aspects of a system with multiple levels of abstraction within different contexts.

To produce models, we require a modeling language providing a set of composable constructs. The use of informal modeling notations creates communicational problems among stakeholders and ultimately makes ITIs suffer the same problems of legacy software: undocumented decisions, redundancy, inconsistencies and increased cost of ownership. To mitigate these issues, we should adopt a well-formed graphical notation, based on a formal grammar, usually called a *metamodel*. The latter includes precise definitions of constructs and their relationships, along with composition rules that must be fulfilled for creating valid models. Models are said to be metamodel instances since they conform to it. Metamodels allow the development of syntax checking editors and validation tools. With such a support, models are then prone to provide a less ambiguous and shared meaning to all relevant stakeholders, and therefore play their expected role in ITI engineering.

Despite the aforementioned benefits on using a precise modeling language, our experience in the field has shown that most organizations depict their ITIs informally, either using some ad-hoc templates or informal notation not supported by a standard or framework, resulting in ambiguous models without any kind of traceability features like the one represented in Figure 1. Such ad-hoc models frequently lead to discussions as each stakeholder has its own interpretation.

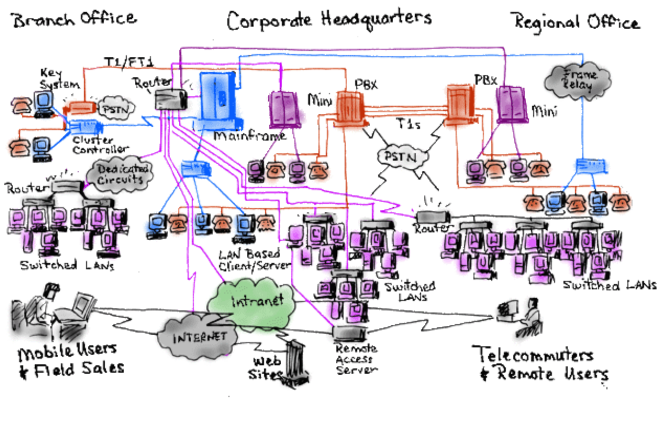


Figure 1. Model of an IT infrastructure (source: [3])

In this paper, we propose an extension to the UML2 metamodel, provided as a profile, to describe the semantics of ITI modeling constructs.

This paper is organized as follows: in the next Section we review related work; in Section III we present the UML profile itself, by overviewing its structure and stereotypes; then, in Section IV, we briefly describe how we have deployed the proposed profile in a professional modeling tool; finally, in Section V, we present some conclusions and future work.

This paper extends (by presenting related work and a modeling example using the profile) and improves (by providing more detail on the profile and on the future work) a previous short paper of ours [4].

II. Related Work

Several modeling languages provide constructs for modeling ITIs with some precision. In this section, we will overview those constructs, as defined in the metamodels of three of the most well-known languages that can be used in the context of IT infrastructures: *UML*, *TOGAF* and *ArchiMate*.

A. UML Metamodel

UML is a general-purpose modeling language embodying a collection of best engineering practices that have proven successful in the modeling of large and complex systems of a wide range of domains. Under the stewardship of the Object Management Group (OMG), UML has emerged as the software industry's dominant modeling language. IT infrastructures are modeled in UML with *Deployment Diagrams*. The latter allows representing the hardware for a system, the software that is installed on that hardware, and the middleware used to connect machines.

Since UML version 2 (UML2) has thirteen different types of diagrams, our first endeavor was assessing their relative usage at a global scale, based upon the hits provided by several web search engines, either general purpose, or academic / research oriented. Plotted values in Figure 2 are represented in percentage of total hits. When available, we split textual search hits from image search hits, but the ranking in both cases does not differ significantly.

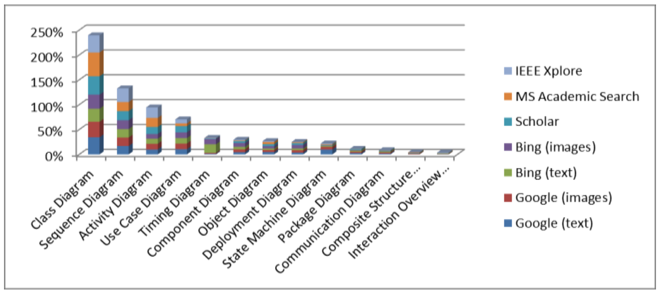


Figure 2. Ranking the use of the thirteen UML2 diagrams

As it can be observed in Figure 2, *Deployment Diagrams* are among the less used UML2 diagrams. A possible interpretation for this phenomenon is that UML2 offers limited modeling constructs (e.g., nodes, components and associations), that do not cope "as is" with the required diversity for modeling ITIs.

There are four modeling elements in UML deployment diagrams, represented as metaclasses in the corresponding UML metamodel extract, as shown in Figure 3 and Figure 4. Those constructs are: *Nodes* to represent a hardware component, *Components* to represent software, *Dependencies* to show that one component relies upon another component and *Links* to connect nodes.

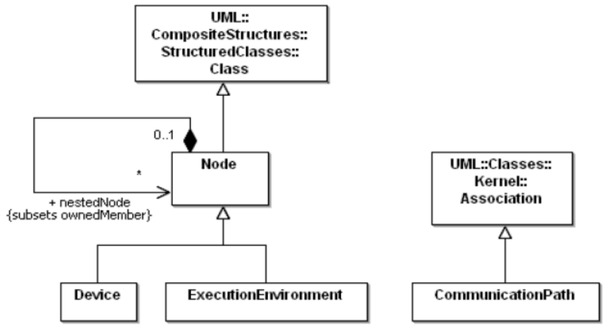


Figure 3. UML metamodel extract corresponding to deployment diagrams.

As can be seen in Figure 3, *Node* is a central modeling element. It can have other elements of type *Node* and represents the environment in which a component or a set of components execute. A *Node* is a generic concept and can represent several things such as a physical hardware device, an operating system or infrastructure software (e.g., database server, web server, application server) and is connected through communication paths.

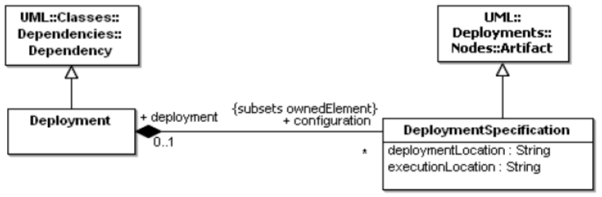

UML::Classes::Dependencies::Dependency
UML::Deployments::Nodes::Artifact
Deployment
+ deployment
0..1
{subsets ownedElement}
+ configuration
*
DeploymentSpecification
deploymentLocation : String
executionLocation : String

Figure 4. Metaclasses used to define the deployment component.

UML2 has a comprehensive coverage of the whole lifecycle in software development. As a result, its large specification, spanning more than 900 pages [5, 6] is in some aspects too abstract. This is well the case of *Deployment Diagrams*, and as a result they are not widely used as other UML2 diagrams, as corroborated by our survey. In short, "plain vanilla" UML provides no specialized stereotypes for the many concepts and association types used in any IT infrastructure, what makes it a weak candidate for modeling ITIs.

B. TOGAF Content Metamodel

The *Open Group Architecture Framework* (TOGAF) is a framework for enterprise architecture developed by the *Open Group Architecture Forum* in the United States, which provides a comprehensive approach for designing, planning, implementation, and governance of an enterprise information architecture. TOGAF is a high level and holistic approach to design, which is typically modeled at four levels: Business, Application, Data, and Technology. It tries to give a well-tested overall starting model to information architects, which can then be built upon. It relies heavily on modularization, standardization and already existing, proven technologies and products. Its latest release, at the time of writing, also includes a metamodel called *Content Metamodel*, that defines all types of building blocks that exist within architecture and how they are related to each other to allow architectural concepts to be captured, stored, filtered and queried in a structured and consistent manner. The IT infrastructure (called technology architecture in TOGAF terminology) is part of the *Content Metamodel* and its direct relationships are shown in Figure 5.

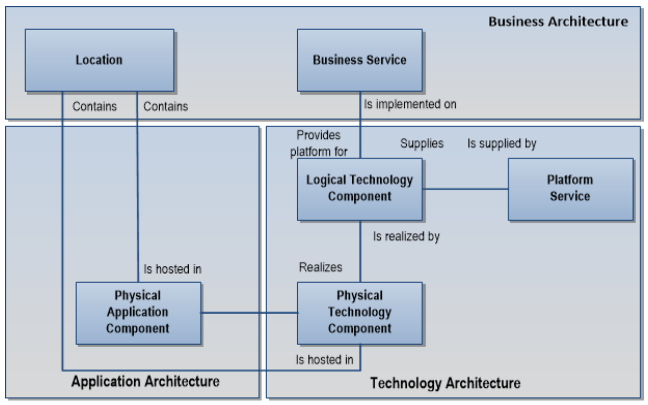


Figure 5. TOGAF 9.1 metamodel extract

From an IT infrastructure perspective, there are few concepts in the content metamodel: the *Platform Service* that represents the support for delivering applications, the *Logical Technology Component* used to represent a class of technology products and *Physical Technology Component* to represent specific technology products. As with UML, these general concepts from TOGAF's *Content Metamodel* allow, in theory, to model IT infrastructures. The lack of specialized stereotypes and relationships appears to be a serious hindrance for its effective adoption. Furthermore, some authors argue that TOGAF's *Content Metamodel* lacks a formal ontology to mitigate its ambiguities and inconsistencies [7].

C. *Archimate Metamodel*

ArchiMate is an open architecture modeling standard with focus on the visualization of viewpoints and notations on models. The metamodel encompasses several enterprise architecture domains (*Business, Application, Information, Technology*).

The *ArchiMate* metamodel was inspired in the UML 2.0 standard [5, 6]. As seen in Figure 6, *Node* is also the main structural concept and is specialized in *Device* (e.g., servers) and *System Software* (e.g., operating system called "execution environment" in UML).

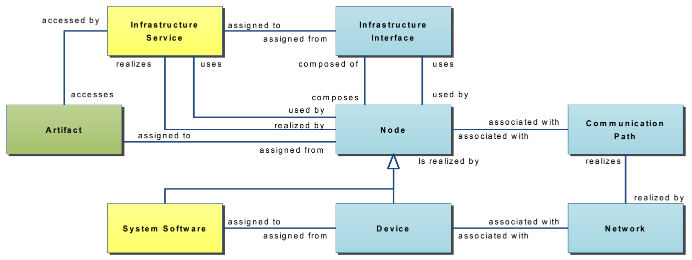


Figure 6. *ArchiMate* metamodel extract corresponding to IT Infrastructure modeling

The *Infrastructure Interface* is the "logical" location where the *Infrastructural Services* offered by a *Node* can be accessed by other *Nodes*. The *Communication Path* and *Network* are used to connect interrelated components in the technology layer. The *Artifact* (also taken from UML 2.0) represents a physical piece of information and can be deployed to a *Node*.

ArchiMate's technology architecture metamodel extract is more detailed than the corresponding TOGAF extract, namely by allowing to model the hardware platforms and communication infrastructure. However, it is still too generic and with more focus on describing the relationships between layers than providing clear guidelines and rules on how to model the various components of the technology architecture. It is argued in the *ArchiMate* [8]specification that modeling infrastructure components such as routers or database servers would add a level of detail that is not useful at the enterprise level of abstraction .

III. UML PROFILE FOR IT INFRASTRUCTURES

UML makes provisions for its own extension, by allowing "customization" to a specific area or domain, with a so-called *UML Profile*. The latter is a coherent collection of UML extensions (stereotypes, tagged values, and constraints) that allows refining the standard semantics in strictly additive manner (i.e. without contradicting it). For instance, a profile may use a stereotype to refine the concept of *Node*.

Several UML profiles have been proposed in the literature and some of them have been endorsed by the OMG itself. Examples include a profile for aspect-oriented software development [9], a profile for requirements management of software and embedded systems [10], a profile for business process modeling [11] and a profile for modeling real-time embedded systems [12].

According to Frank Ulrich, the existing tools and methods for IT management are not suitable because they focus on issues such as hardware and operational metrics. This author claims further that there is a gap between the technical level and IT management. He points out that a mitigating strategy to cope with the complexity of this task would be providing adequate support for the analysis and communication among the various stakeholders. He corroborates our observation that existing approaches are too generic, therefore not providing the required level of detail and stresses that the lack of formality not only leads to communication problems among stakeholders, but also limits the use of automatic problem detection or validation techniques in existing infrastructures [13].

Due to the aforementioned limitations of existing modeling languages, we have developed an UML profile for IT infrastructures. The decision for extending UML, instead of developing a domain specific language (DSL) from scratch, was based on the following rationale:

- There is a large community, both in industry and academia, that understands and actually uses the UML language;
- Extending UML allows reusing existing UML modeling elements, with well-defined syntax and semantics;
- There are tools that support the development of UML profiles.

A. *ITI Profile Structure*

Figure 7 presents a conceptual view of the ITI profile, where the software and hardware layers are represented with different colors.

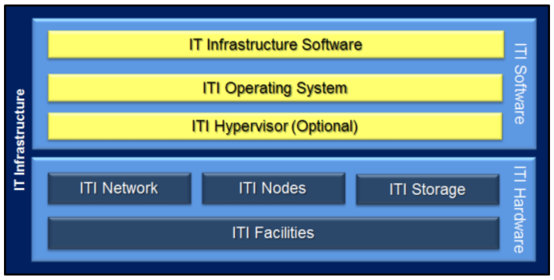


Figure 7. Layered structure of the ITI profile

The ITI Software Layer (Yellow) has three packages: *ITI Hypervisor*, *ITI Operating System* and *ITI Software,* while the ITI Hardware layer (Blue) has four packages: *ITI Facilities*, *ITI Network, ITI Nodes* and *ITI Storage.*

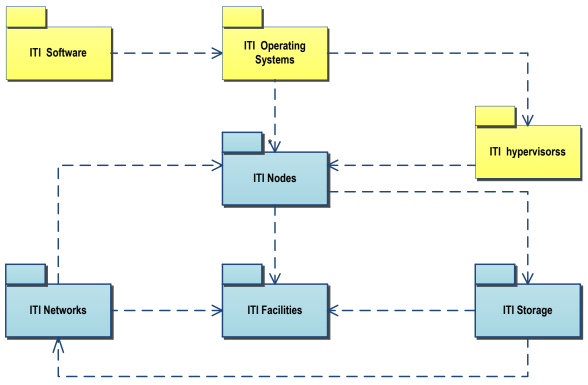


Figure 8. ITI packages and their relationships

Figure 8 provides an overview of the ITI packages and their relationships in the ITI profile. In the software layer the package *ITI Software* models software platforms such as antivirus, application servers, backup, collaboration servers, database servers, directory, and email servers, among others. These ITI platforms execute upon an *ITI Operating System* such as Windows, Linux, AIX. The operating system may be deployed directly on hardware or it can be deployed on top of an *ITI Hypervisor* such as XEN, Hyper-v or VmWare, that executes directly on the *ITI Nodes* hardware.

The package *ITI Nodes* is a very important one since it contains the constructs used to model systems such as servers and their components. The metaclass *Host* inherits the properties of an UML2 *Node* and was created with a set of stereotypes to allow the representation of physical and virtual servers, mainframes or supercomputers. The metaclass *Device* is similar to *Host* but is used to represent other equipment such as phones, tablets, slates, laptops, or PDAs. The *Peripheral* metaclass represents the components that may be connected to a *Host* or *Device* and includes monitors, keyboards, mice, printers or smartcard readers, among others. A *Port* is a built-in component in a *Host* or *Device* such as a host-based adapters or a network card.

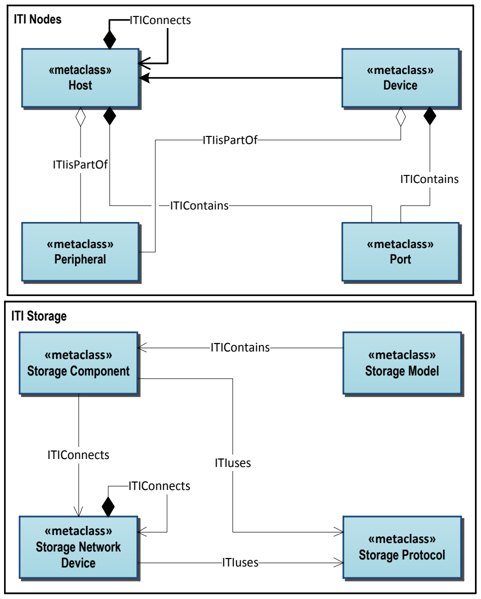


Figure 9. ITI Nodes and ITI Storage

The package *ITI Storage* represents the multiple *Storage Components* such as storage LUNs, storage arrays and pools, storage controllers and they may be configured in different *Storage Models* such as Storage Area Network (SAN) or Network Access Storage (NAS). These storage components are connected to *Hosts* and *Devices* trough *Storage Networks* using fiber channel or Ethernet routers or switches that use specific *Storage Protocols* such as ISCSI, Fiber Channel or Fiber Channel over Ethernet (FCoE) among other protocols. Both *ITI Nodes* and *ITI Storage* packages and their relationships are represented in Figure 9.

Hosts and *Devices* from the *ITI Nodes* package can be interconnected by *Network Devices* available in the *ITI Network Package*. *Network Devices* includes, among others, devices such as access points, firewalls, hubs, routers and switches. Those devices are used to create different *Network Zones* such perimeter networks, intranets or extranets and they communicate using a specific *Network Protocol* such as frame relay or Ethernet. The *Network Devices* may be configured in multiple *Network Types* such as LANs, WANs or Wi-Fi. All aforementioned components reside in the *ITI Facilities* package. Both packages are shown in Figure 10.

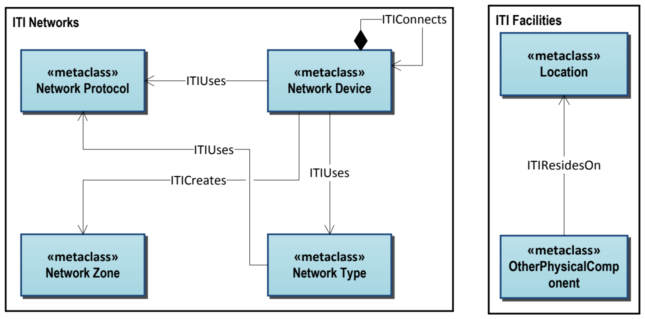


Figure 10. ITI Network and ITI Facilities

B. *ITI Profile Metaclasses and Stereotypes*

To connect multiple ITI components, we require ITI connectors that extend the metaclass *Association* or the metaclass *Composition*, as represented in Figure 11.

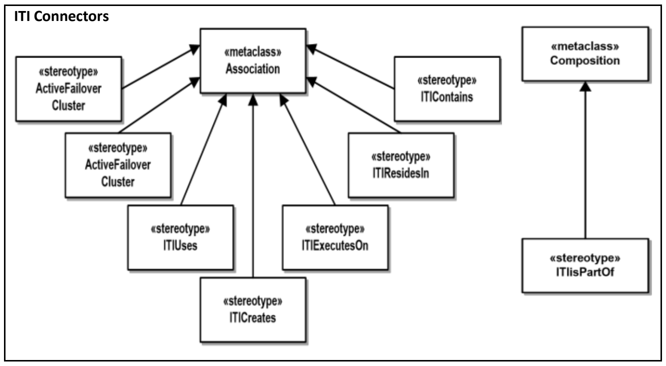


Figure 11. IT Infrastructure Connectors Package.

Besides these two connector metaclasses, we also extended UML2 metaclasses elements such as *Class, Location, Boundary, Device* and *Node*. An example is the package *ITI Facilities,* composed by the metaclass *Location* and the metaclass *OtherPhysicalComponents*. A location can be the *Headquarters*, a *Datacenter, a Branch Office,* or a *Regional Office. OtherPhysicalComponents* includes *Cables* to connect hosts, *Racks* to attach servers, *Power* supplies and *Cooling* systems. Both metaclasses and their extending stereotypes can be seen in Figure 12.

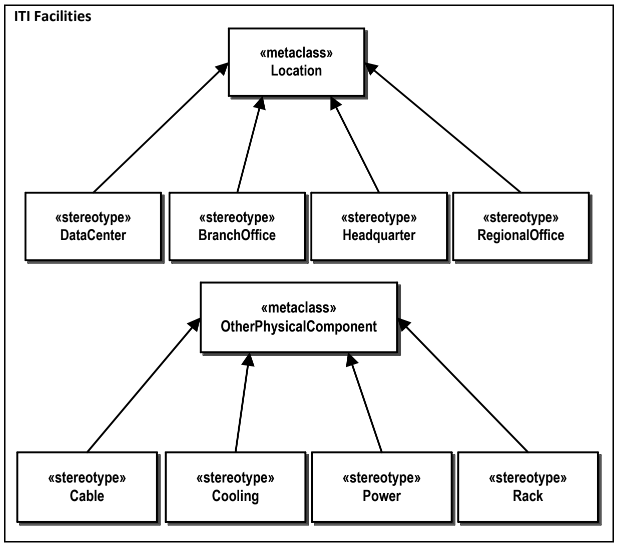


Figure 12. Package Facilities

To allow expressing as much information as desired in the ITI domain, we enriched each stereotype with additional attributes (called "tagged values" in earlier UML versions). The attributes chosen for each stereotype were based on our field experience and inspired on the standard *Common Information Model* (CIM) [14] created by the *Distributed Management Task Force* (DMTF). The latter is a worldwide initiative spearheaded by industry-leading technology companies such as AMD, Broadcom Corporation, CA, Cisco, Citrix Systems, EMC, Fujitsu, HP, Huawei, IBM, Intel, Microsoft, NetApp, Oracle, RedHat, SunGard and VMware.

CIM was created to provide a common approach to the management of systems, networks, applications and services and enable multiple vendors to exchange semantically rich management information between systems throughout the network. This paper only includes a subset of the stereotypes. The complete set of stereotypes, tagged values and constraints will be available as a technical report on the QUASAR group website [15].

IV. DEPLOYING THE PROFILE

We have deployed the proposed ITI profile in a widely used modeling tool: *Sparx Systems' Enterprise Architect* [16] that supports the definition of profiles.

Figure 13 represents an ITI model produced with our deployed profile. This example provides a first evidence that our proposal reduces the ambiguity in modeling ITIs, while providing the recurrent ITI concepts used by ITI architects, such as data centers, servers, network types such as perimeter, intranet, extranet, firewalls, routers or switches. The increased preciseness facilitated by the use of a formal metamodel is rendered possible by specifying well-formedness rules upon it using OCL clauses.

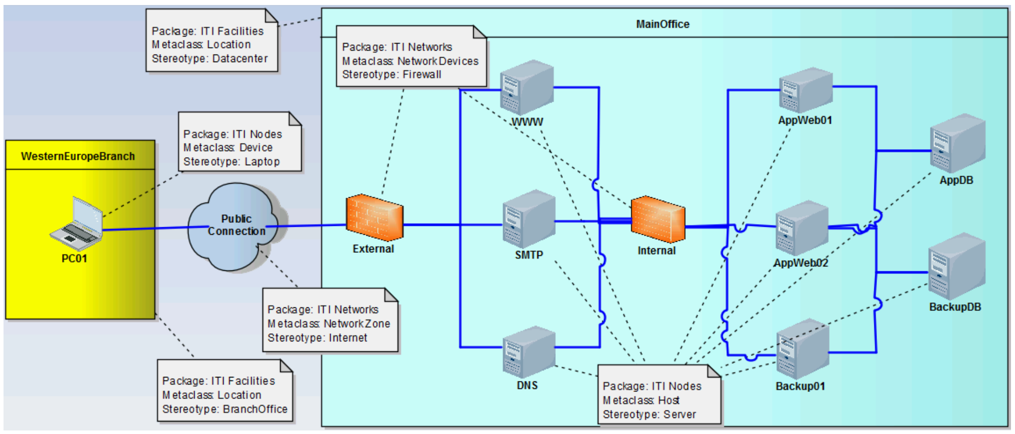


Figure 13. Simple ITI model using the proposed profile

V. Conclusion and Future Work

This paper introduces a UML profile for modeling IT infrastructures, covering constructs at all required abstraction levels (hardware, middleware, network and software). Since the full profile has many stereotypes and each stereotype is described by means of several attributes, only a subset of the profile could be presented here. Nevertheless, we present some evidence that this profile mitigates the problems identified while reviewing existing approaches for modeling IT infrastructures:

i) The widely used ad-hoc approaches produce ambiguous models, do not facilitate knowledge reuse and cannot support validation approaches;

ii) The existing formal approaches for modeling ITIs, such as *UML Deployment Diagrams, TOGAF* or *ArchiMate*, do not provide the required abstractions and, probably due to that, are not used in practice.

Several future research threads relate to the availability of this profile:

ITI models capture – Organizations often have IT service management tools (e.g., CMDB – Configuration Management Data Base) that store information on the ITI elements. Being able to import that information and generate preliminary layouts is a major research concern.

ITI models scalability - Models of real-world infrastructures, even in medium-sized companies, can easily reach hundreds or even thousands of modeling elements, especially when software components are considered. In such a case, a model can easily be rendered useless due to excessive detail. We plan to mitigate this problem by using zooming facilities like those available in GIS.

Reuse ITI modeling best practices: We have proposed elsewhere the concept of *ITI patterns* [17, 18]. The availability of this profile will allow granting more preciseness to the formalization of those patterns.

Acknowledgment

This work is partly supported by grant PEst-OE/EEI/UI0527/2011 of Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL).

References

[1] A. Gunasekaran, H. J. Williams, and R. E. McGaughey, "Performance measurement and costing system in new enterprise," *Technovation,* vol. 25, pp. 523-533, 5// 2005.
[2] OGC, *IT Infrastructure Library (ITIL) - Service Design (Version 3)*. London: The Stationery Office, 2007.
[3] *http://www.oracle11grelease2.com/services/infrastructure-velolux/*.
[4] L. Ferreira da Silva, F. Brito e Abreu, and V. Moreira, "A UML Profile for Modeling IT infrastructures," presented at the INFORUM'2012, Caparica, Portugal, 2012.
[5] OMG, "Unified Modeling Language (UML) Specification: Superstructure (version 2.3)," ed, 2010.
[6] OMG, "Unified Modeling Language (UML) Specification : Infrastructure, version 2.3," 2010.
[7] A. Gerber, A. Van der Merwe, and P. Kotze, "Towards the Formalisation of the TOGAF Content Metamodel using Ontologies," presented at the Proceedings of the 12th International Conference on Enterprise Information Systems, Funchal, Madeira, Portugal, 2010.
[8] The Open Group, "Archimate 2.0 Specification," ed. Zaltbommel: Van Haren Publishing, 2012.
[9] T. Aldawud, A. Bader, and T. Elra, "UML profile for aspect-oriented software development," presented at the The Third International Workshop on Aspect-Oriented Modeling, 2003.
[10] T. Arpinen, T. Hamalainen, and M. Hannikainen, "Meta-Model and UML Profile for Requirements Management of Software and Embedded Systems," *EURASIP Journal on Embedded Systems,* 2011.
[11] B. List and B. Korherr, "A UML 2 Profile for Business Process Modelling," presented at the Perspectives in Conceptual Modeling (ER 2005 Workshop), Klagenfurt, Austria, 2005.
[12] OMG, "A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems," vol. ptc/2008-06-09, ed: Object Management Group, 2008.
[13] U. Frank, D. Heise, H. Kattenstroth, D. Ferguson, E. Hadar, and M. Waschke, "ITML: A Domain-Specific Modeling Language for Supporting Business Driven IT Management," presented at the 9th OOPSLA workshop on domain-specific modeling (DSM), Helsinki, Finland, 2009.
[14] DMTF, "Common Information Model (CIM) Infrastructure," November 2007.
[15] *http://ctp.di.fct.unl.pt/QUASAR (accessed in 5/9/2012).*
[16] *http://www.sparxsystems.com (accessed in 5/9/2012).*
[17] L. Ferreira da Silva and F. Brito e Abreu, "Software distribution to remote locations," presented at the Proceedings of the 15th European Conference on Pattern Languages of Programs, Irsee, Germany, 2010.
[18] L. Ferreira da Silva and F. Brito e Abreu, "An IT Infrastructure Patterns Approach to Improve IT Service Management Quality," presented at the 7th International Conference on the Quality of Information and Communications Technology (QUATIC'2010), Porto, Portugal, 2010.

An Investigation into Reference Architectures for Mobile Robotic Systems

Daniel Feitosa and Elisa Yumi Nakagawa
Dept. of Computer Systems, University of São Paulo - USP
PO Box 668, 13560-970, São Carlos, São Paulo, Brazil
{feitosa, elisa}@icmc.usp.br

***Abstract*—Currently, robotic systems have been more and more required for a diversity of new products, such as in domestic robots and in robots for dangerous environments. As a consequence, an increase in the complexity of these systems is observed, requiring also considerable attention to their quality and productivity. In another perspective, reference architectures have emerged as a special type of software architecture that achieves well-recognized understanding of specific domains, facilitating the development, standardization, and evolution of software systems. In this perspective, reference architectures have also been proposed for the robotic domain and they have been considered an important element to the development of systems for that domain. However, there is a lack of work that present an panorama about these architectures; furthermore, there exists no support to choose a reference architecture when developing or evolving robotic systems. Thus, the main contribution of this paper is to present a panorama about reference architectures of the robotic domain, in particular, for mobile robots. It is worth highlighting that we used the systematic review technique to identify and investigate these architectures. We have found that these architectures have in general become consolidated and have already contributed to the industry during the development of robotic systems. Besides that, results of our investigation could support the decision about which architecture to adopt aiming to develop a new software. Also, our analysis could help to create new reference architectures. However, there are still important perspectives of research that need to be investigated.**

***Keywords-robotic system*; *robot*; *reference architecture*; *systematic review*.**

I. Introduction

The field of Robotics has presented an increasing growth in the last years, impacting various sector of the society and opening new, important application areas [1]. A number of different types of robots has been developed and used and, in particular, mobile robots have currently detached by their relevance and range of applications. Good examples of mobile robots are vacuum cleaners, vigilant robots, and mappers, including robots developed to be operated in dangerous environments, sometimes, not accessible by human beings. Furthermore, according to Graaf et al. [2], in the next years, the market for these robots is forecasted to exponentially grow and they will have more and more important roles. In this perspective, the complexity of these mobile robots has been increasing, creating a considerable challenge for their development. Thus, both academic and industrial research have focused on their development, aiming at achieving quality in such systems and timely delivery [2]. It is also worth highlighting that robots are basically composed by mechanical devices (such as sensors and actuators) and software systems (i.e., robotic systems) and the development of these systems have been perhaps the major challenge.

In another perspective, software architectures have been increasingly investigated as the main artifact that plays a pivotal role in determining system quality, forming the backbone of any successful software-intensive system [3]. More specifically, a reference architecture has achieved the status of a special type of software architecture that captures the essence of the system architectures of a given domain, i.e., it encompasses the knowledge about how to design, standardize, and evolve the system architectures of a specific domain. Considering the relevance of reference architectures, various application domains have proposed, used, and reused the knowledge contained in such architectures. It is worth highlighting that these architectures have been sometimes developed by consortia that involve academy and major industrial players (such as manufacturers and suppliers). Thus, reference architectures have been considered as a quite important element to improve productivity and quality of the software systems.

Regarding robotic domain, several reference architectures have been also proposed for the development and evolution of mobile robots [4], [5], [6]. Each reference architecture has its particular characteristics; besides that, they have been successfully used in specific projects. However, in the most cases, robotic systems for mobile robots have been almost always developed and evolved without using reference architectures, i.e., these systems are not taking advantage of the knowledge contained in such architectures in order to be more easily developed and evolved. This fact can be the result of the difficulty to select an more adequate architecture. Thus, a detailed, comprehensive panorama about these architectures will be in fact important. However, there is a lack of such panorama about reference architectures for the robotic domain.

In this scenario, the main contribution of this paper is to present a broad panorama of the reference architectures for robotic systems, as well as an more detailed investigation among these architectures. In the context of this work, we have focused in reference architectures for mobile robots. It is important to say that in order to find these architectures, we adopted and conducted a systematic review [7], i.e., an efficient and effective technique to summarize, assess, and interpret all evidence related to a specific question, topic area, or phenomenon of interest. Besides that, we complemented our investigation using additional documents related to each reference architecture found. As main results, we have observed, in the last years, an increase in the number of reference architectures for robotic systems. These facts have evidenced a real interest by both academy and industry. Moreover, our investigation could be considered as a valuable element during the selection of a more adequate architecture for the design and evolution of robotic systems for mobile robots. Moreover, our results could be used as a start point to the proposal of new reference architectures for mobile robots domain. Finally, this work could make it possible to identify interesting and important research topics for further investigations.

This paper is organized as follows. In Section II, we present an overview on mobile robots and reference architecture, since these topics are important to understand our analysis. In Section III, we present the methodology used to systematically identify reference architectures. In Section IV, we describe each reference architecture found. In Section V, we present a comparison among these architectures. In Section VI, we discuss about the achieved results. Finally, in Section VII, we summarize our contributions and discuss perspectives for further work.

II. Background

The field of Robotics encompasses several types of robotic applications, such as robotic arms for assembly lines, household robots, and military robots. In this context, an important type is mobile robots, which also are basically composed by software and hardware projects. The software implements the robotic control system (i.e., robotic system), which is responsible for analyzing the sensor signals, planning and decision-making, and control of actuators. In parallel, the hardware is responsible for the physical implementation of sensors, actuators (responsible for robot's movement and actions), and data processing (the basic components for partial or total autonomy of the robots) through the use of an embedded processor. In particular, sensors are the hardware components responsible to give the robot the "vision" of the world, representing different senses, such as vision and hearing, and allowing the robot to interpret the environment. Otherwise, actuators enable the robot to interact with the environment, allowing the robot to move and perform actions, such as picking or pushing objects.

Regarding robotic system, it enables the robot to develop essential, complex, and intelligent activities. For instance, it also determines the robot autonomy level, from tele-operation to autonomous behavior. Thus, the main activities performed by a robot are: *navigation*, *localization*, and *mapping*. *Navigation* is the act of controlling the movement of the robot from an initial position to a target position. The task of navigation is usually implemented through the use of a control architecture which represents how the robot behaves. *Localization* consists of estimating the position of the robot in the environment. This activity is essential and basic for the robot navigation. If a robot determines exactly its position, it will be capable of planning a path to its destination and will fulfill adequately the tasks allocated to it [8]. *Mapping*, i.e., the act of getting data on the environment and the construction of maps, is an estimation problem which is a major task for the development of autonomous mobile robots. A correct map of the environment is fundamental to find the most efficient path. Through the map, the robot checks the possible paths that lead to the desired position and the obstacles that must be avoided [9].

In another perspective, considering the relevance of reference architectures, a diversity of them for various domains can be found, such as for automotive (e.g., the AUTOSAR [10]), and commerce (e.g., Microsoft Reference Architecture for Commerce [11]). They have served as an important basis for the software systems development, since these architectures have proved their efficiency regarding improvement in productivity during software development. Reference architectures have also been built for different purposes [12]: (i) improvement of interoperability among different components of software systems; (ii) standardization of software systems of a given domain or of a company; and (iii) reuse of knowledge from domain experts regarding development of systems for that domain.

In this context, the robotic community has also noticed that the establishment of reference architectures for robotic systems of mobile robots is also quite interesting. Thus, several architectures can be also found. However, selecting a more adequate architecture aiming at using this one as basis of the development of new robotic systems, as well as evolution of existing systems, is still a hard task. There is not a complete panorama of these architectures and also information or guidelines that support selection of such architectures. This scenario has therefore motivated this work.

III. Finding the Reference Architectures

In order to specifically find reference architectures which could be applied to the development of robotic systems for mobile robots, we performed an exhaustive search conducting a systematic review. Our systematic review was conducted from December/2011 to January/2012, following the process proposed by Kitchenham [7]. In short, this process presents three main phases: (i) Phase 1 - Planning: In this phase, the research objectives and the review protocol are defined. The protocol constitutes a pre-determined plan that describes the research questions and how the systematic review will be conducted; (ii) Phase 2 - Conduction: During this phase, the primary studies are identified, selected and evaluated according to the inclusion and exclusion criteria established previously. For each selected study, data are extracted and synthesized; and (iii) Phase 3 - Reporting: In this phase, a final report is formatted and presented.

In short, we established one research question: "Which existing reference architectures could be applied to develop robotic systems to mobile robots?" We then identified the main keywords — "reference architecture", "robot", and "unmanned ground vehicle" — and we established a search string considering these keywords and their possible synonymous: `((``reference architecture'' OR ``reference model'') AND (``robot'' OR ``robotic'' OR ``unmanned ground vehicle'' OR ``UGV'' OR ``intelligent vehicle''))`. We added "reference model" in the search string, since it is sometimes used to refer to reference architecture. We also added "intelligent vehicle" to refer to mobile robots. Finally, to find the works (also primary studies in the systematic review context), we used main publication databases: ACM Digital Library [13], IEEEXplore [14], ISI Web of Knowledge [15], and Scopus [16]. As a result of our search in these databases using the search string, a total of 409 primary studies were discovered. Removing the repeated studies, we had 371 unique studies. The title and abstract sections of each study were read and a total of 14 studies were selected for further reading. Next, these studies were read in full and, finally, seven reference architectures that are applicable to mobile robots were identified: 4D/RCS [17], ACROSET [18], AIS [6], JAUS [5], Robot Teleoperation [4], Servicebots [19], and SMAS [20]. For the selection of these architectures, we considered reference architectures that filled three main requirements: (i) the architecture presents a set of pre-defined functionalities that could be contained in robotic systems; (ii) the architecture explains the interaction among these functionalities; and (iii) the architecture permits the derivation of software architectures and their respective systems. It is important to say that these requirements are essential to reference architectures, if it is intended in fact to use them to the robotic system development.

It is important to say that the objective of the systematic review was then to identify all reference architectures applicable to mobile robots. Furthermore, a second search for specific information of these architectures was conducted considering additional documents, such as the web sites, books, papers in conferences and journals, and any other documents that could support to conduct our analysis.

IV. Description of the Reference Architectures

Based on the seven reference architectures that are applicable to the mobile robots, we conducted a detailed investigation on each one and developed a comparison among them. In Table I, each reference architecture is presented together with name and type of the mobile robots. Below, we present an overview of each reference architecture.

The **4D/RCS** reference architecture provides a theoretical foundation for designing, engineering, integrating, and testing intelligent software systems for unmanned vehicle systems [17]. It consists of a multi-layered multi-resolutional hierarchy of computational nodes, each containing elements of sensory processing (SP), world modeling (WM), value judgment (VJ), and behavior generation (BG). From high levels to low levels, the reference architecture contains functionalities that permit goal definition, going to perception, cognition, and reasoning, involving sensors and actuators. According to its author, this architecture enables precise and fast responses in lower levels while it formulates plans and abstracts concepts in higher levels.

The **JAUS** [21] reference architecture uses a message passing protocol to provide interoperability among subsystems and components that compose systems resulting from this architecture [5]. JAUS presents a service-oriented approach to enable distributed command and control of these systems. The reference architecture provides information about how to enable online interoperability of unmanned systems and their components. For that, JAUS defines a set of basic services which are required by most higher level components, and they are defined in JAUS Core Service Set (JSS Core).

The **ACROSET** is a component-oriented reference architecture for teleoperated service robots [18]. Its main characteristic is the reuse of components from different systems. The reference architecture is composed by subsystems: *Coordination, Control and Abstraction Subsystem* (CCAS); *Intelligence Subsystem* (IS); *User Interaction Subsystem* (UIS); and *Safety, Management and Configuration Subsystem* (SMCS). The CCAS abstracts and encapsulates the functionality of the physical devices of the robots. This subsystem is composed by virtual components that can be implemented in either software or hardware. Besides that, in order to deal with operator-driven, semi-autonomous systems, the IS was inserted in this architecture. This subsystem permits to have different types of user (and even an autonomous subsystem). The UIS is responsible for interpreting, combining, and arbitrating among orders that may come simultaneously from different users. Finally, the SMCS presents two main functionalities: (i) the monitoring of functionalities from other subsystems; and (ii) management and configuration of the initialization of the application.

The **Servicebots** reference architecture was designed to service robots operating in indoor environments [19]. In this context, service robots are those supposed to perform tasks (like mail delivery, tourist guide, etc) in buildings of a whole variety of characteristics [19]. The reference architecture is composed by three subsystems that use the IT (Information Technology) backbone (i.e., the local area

Table I
REFERENCE ARCHITECTURES FOR MOBILE ROBOTS

Name	Type
4D/RCS [17]	Unmanned ground vehicles
JAUS [5]	Unmanned systems
ACROSET [18]	Teleoperated service robots
Servicebots [19]	Indoor service robots
SMAS [20]	Situated multi-agent systems
Robot Teleoperation [4]	Robots with many controllers
AIS [6]	Adaptive intelligent systems

network) to communicate and complete the task; these three subsystems or types of robots are: *servicebots*, *fixbots*, and *softbots*. The type *servicebots* is a robot capable of driving autonomously only with sensorial information in an average complex environment (e.g., corridors). The *fixbots* present sensor and actuators distributed all over the environment, having their own intelligence and communication channel. The *softbots* refer to the software agents executing various tasks for the requesting user, fixbot or servicebot. Thus, the reference architecture is concerned with performance, configuration, problem, human-interface, and security management.

The **SMAS** reference architecture provides a blueprint for architectural design of multi-agent system applications [20]. It is composed by two subsystems: the *agent* and the *application environment*. The first one comprises three modules: (i) perception through getting information from the environment; (ii) decision making, selecting the agent action; and (iii) communication, responsible for interactions with other agents. The second subsystem comprises seven modules: (i) representation generator, perceiving the environment; (ii) interaction, dealing with agents' influences in the environment; (iii) communication service, collecting messages and delivering messages to the appropriate agents; (iv) observation, observing the deployment context; (v) synchronization, monitoring domain-specific parts of the deployment context and keeping the corresponding representation in the state of the application environment up to date; (vi) dynamics, maintaining processes in the application environment that happen independent of agents or the deployment context; and (vii) translation of influences and messages into low-level interaction primitives with the deployment context, and low-level formatted messages into messages for agents.

The **Robot Teleoperation** reference architecture is concerned with robots having different controllers [4]. To achieve the objective, the reference architecture was proposed according the definition of Bass [22]. Thus, a domain analysis was made to identify the set of components, followed by the domain design to make the generic design, where patterns and common models could be used. The main identified components were: *graphical representation*, *collisions detection*, *user interface*, *communications* and, the most important, *controller*. To specify how the different components are going to interact, two architectural styles were selected: (i) client-server, used in the interactions between *graphical representation* (client) and *collisions detection* (client) with *controller* (server); and (ii) communicating processes, used in the interactions between *user interface* and *communications* with *controller*, because all of them can take the initiative to send data.

Finally, the **AIS** reference architecture is a heterogeneous mixture of common architectural styles [6], permitting the creation of various adaptive intelligent systems. It is divided hierarchically into layers for different sets of computational tasks. Properties of pipe and filter style architectures are provided by the layers and relations among them. Thus, the reference architecture has two layers (or levels): the *physical level*, responsible for action and perception in external behaviors; and the *cognitive level*, responsible for more abstract reasoning activities (e.g., planning, problem solving, etc). The components comprising each layer are organized in a blackboard style, allowing a range of potentially complex behaviors, since basic functionalities provided in each level can work together to perform more complex functionalities.

After investigating each reference architecture, a comparison among them was developed, aiming at providing information in order to better support selection of one or more architectures when developing new robotic systems or evolving existing ones. Also, we expect this analysis supports the proposal of new reference architectures, since we present a set of main features present in the analyzed architectures. Moreover, the comparison will not point out which reference architecture is better, because each one has its specific requirements and application environment.

V. Analysis of the Reference Architectures

Regarding analysis of the reference architectures, we also adopted the systematic guidelines proposed by systematic review. For this, we defined three perspectives and compared these architectures: context of application, maturity, and functionalities. To the first perspectives, we analyzed each reference architecture determining if they were developed in an *academic* context or in a *industrial* context. The result of this analysis is presented in Table II. We observed that both industry and academy are interested in proposing reference architectures.

Considering other domain where the success of a reference architecture depends on involvement of the industry, we can observe that reference architectures for mobile robots present good perspective of success, since more than half of the architectures present efforts from industry. Besides that, if selection of an architecture is necessary, it is more interesting to select architectures that have efforts from industry.

Table II
CONTEXT OF DEVELOPMENT OF THE REFERENCE ARCHITECTURES

Reference Architecture	Development Context
4D/RCS	Industrial
JAUS	Industrial
ACROSET	Industrial
Servicebots	Academic
SMAS	Academic
Robot Teleoperation	Industrial
AIS	Academic

Thus, 4D/RCS, JAUS, ACROSEFT, and Robot Teleoperation could be first considered. However, each architecture has its characteristics and knowledge aggregated; hence depending on the purpose of the robotic system to be develop a specific architecture could be more adequate than another one.

Aiming at determining the maturity level of the architectures (i.e., how much they are evaluated), we established three levels: (i) Architectural instantiation: the reference architecture was only instantiated, i.e., the design of robotic systems was developed, but no implementation is presented; (ii) Implementation: at least a robotic system was implemented based on the reference architecture, through, for instance, a case study; and (iii) Use in real situation: the reference architecture is in fact already used in real situations, in particular, in the industry. This fact shows a higher level of maturity of the architecture. Table III shows the result of our investigation. As result, we have observed that reference architectures for mobile robots are in general mature, since three architectures present implementation of robotic systems based on the architecture; besides that, the most of them (i.e., four architectures) have been already used in the industry. Therefore, 4D/RCS, JAUS, ACROSET, and Robot Teleoperation seem to be the best choices to be considered in the adoption of a reference architectures, considering their maturity.

Table III
MATURITY LEVEL OF THE REFERENCE ARCHITECTURES

Reference Architecture	Maturity Level
4D/RCS	Use in real situation
JAUS	Use in real situation
ACROSET	Use in real situation
Servicebots	Implementation
SMAS	Implementation
Robot Teleoperation	Use in real situation
AIS	Implementation

The analysis of the last perspective was the most difficult to be conducted, since we needed to determine a set of functionalities that comprise all reference architectures and, frequently, we found sub-functionalities inside other one. Thus, we defined 10 functionalities and indicated, for each reference architecture, if each functionality is present partially, completely, or not explicitly present. The result of this investigation is presented in Table IV, where an “X” indicates a functionality completely present, an “X*” refers to a functionality partially present, and a blank space indicates functionality not present. Therefore, these architectures have presented a range of functionalities, partially or completely, and, in general, ACROSET seems to be the most complete architecture. Furthermore, our investigation could provide important information to guide the selection of a more adequate reference architecture. For instance, if functionality “Decision judgement” is required in the robotic system to be developed, it is interesting select ACROSET than other architectures. Besides that, this table provides important information about which functionalities could be inserted in the architectures in order to become them more complete. Considering the set of functionalities identified in this work, “owners” of these architectures could have a direction about how to evolve their architectures, if desired. This set can also support the proposal of new reference architectures, since they can be considered as a basic set of functionalities because they are present in a significant number of reference architectures. Moreover, each functionality can be explored in depth, if necessary. In order to use these architectures, a detailed study could be necessary to understand specific points of them.

It is important to highlight that these results do not indicate if a reference architecture is better than another one. Besides that, there exists also functionalities or aspects of reference architectures which were not discussed in this work, because they are not important for mobile robots.

VI. DISCUSSION

The investigation presented in this work intends to support selection of reference architectures for new projects, to the evolution of existing ones or to the proposal of new reference architectures; thus, productivity and quality of the mobile robotic systems could be possibly improved. In general, robotic domain presents good perspectives regarding reference architectures, mainly because industry have been involved in the establishment of such architectures. Regarding their documentation, in general, these architectures are well-documented; however, several of them, for instance, that presented in [18] and [4], could present a more comprehensive representation, if it is intended an adequate dissemination of their architectures. In this perspective, these architectures will have more chances to provide an effective contribution to the robotic area.

Besides that, it was clear the potential of these architectures, since they permit to derive as many components.

Table IV
FUNCTIONALITIES CONTAINED IN THE REFERENCE ARCHITECTURES

Functionality	**Reference Architecture**						
	4D/RCS	**JAUS**	**ACROSET**	**Servicebots**	**SMAS**	**Robot Teleoperation**	**AIS**
Sensorial processing	X	X	X	X	X*	X*	X
Controlling	X	X	X	X	X	X	X
Collision detection	X	X*	X*	X*		X	X*
World mapping	X*			X*			X
Action planning	X	X*	X*	X*	X		X
User interfacing		X	X	X	X*	X	
Communication	X	X		X	X	X	X*
Security		X	X	X			
Decision judgement			X			X*	X*
Multi-robotic interaction	X*	X*		X	X		X*

However, we noticed that some reference architectures, in particular [19], could have their components and functionalities presented in more detailed way.

Based on our investigation, new topics of research can be identified. Thus, the most important ones that we have observed are:

- Establishment of a general, unique, and complete reference architecture for mobile robotic systems, containing possibly all functionalities, constraints, other important information related to robotic system development. This architecture could facilitate therefore the development of any robotic systems;
- Proposal of mechanisms (such as frameworks and components already implemented, as well as support tools) to easily use the reference architectures, since architectures found in this work do not provide in general an adequate automated support; and
- Proposal of complete guidelines to use the reference architectures for mobile domain, since most studies make it more succinctly. However, there are other studies presenting well-specified, detailed guidelines, including the entire architecture. Thus, it is possible to identify this topic as a trend in reference architectures for robotic systems.

Regarding the limitation of this work, other sources of information could be used, possibly resulting in more reference architectures to be studied and, as consequence, achievement of a more comprehensive investigation. In another perspective, we considered only reference architectures applicable to mobile robots; however, this experience could be extended to other types of robots, such as production line robots. Furthermore, the conduction of a systematic review involving a new research area — in our case, mobile robotic systems — is not easy, since there is not a consensus in concepts/terms used by different reference architectures; thus, sometimes, it was necessary to infer a conclusion to make some decisions.

VII. CONCLUSION AND FUTURE WORK

The main contribution of this work is to present an panorama of the reference architectures which could be applied in the development of mobile robotic systems. In general, good initiatives can be found, including architectures that have been used in the industry context. Furthermore, besides our suggestion of future research topic in this area, we believe that this investigation could contribute to the robotic community to open other research fields in mobile robots and related areas.

Motivated by the achieved results, we intend to conducted a detailed, comprehensive investigation involving other types of robotic systems, for instance, production lines robots, intending to contribute to a more effective development of robotic systems.

In a parallel work, we are proposing a new reference architecture aiming at the development of multi-robotic systems for indoor service robots. Thus, we hope that the emerging field of robotics can be supported by our reference architecture.

ACKNOWLEDGMENT

This work is supported by Brazilian funding agencies FAPESP and CNPq, and Capes. Also, the authors would like to thank to the National Institute of Science and Technology on Critical Embedded Systems (INCT-SEC) (Grant N.: 573963/2008-8 and 2008/57870-9).

REFERENCES

[1] E. Ruffaldi, E. Sani, and M. Bergamasco, “Visualizing perspectives and trends in robotics based on patent mining,” in *ICRA’10*, Anchorage, Alaska, 2010, pp. 4340–4347.

[2] B. Graaf, M. Lormans, and H. Toetenel, “Embedded software engineering: The state of the practice,” *IEEE Software Engineering*, vol. 20, no. 6, pp. 61–69, 2003.

[3] P. Kruchten, H. Obbink, and J. Stafford, “The past, present, and future for software architecture,” *IEEE Software*, vol. 23, no. 2, pp. 22–30, 2006.

[4] B. Álvarez, A. Iborra, A. Alonzo, and J. A. De la Puente, “Reference architecture for robot teleoperation: Development details and practical use,” *Control Engineering Practice*, vol. 9, no. 4, pp. 395–402, 2001.

[5] M. N. Clark, “JAUS compliant systems offers interoperability across multiple and diverse robot platforms,” in *AUVSI’2005*, Baltimore, USA, 2005, pp. 249–255.

[6] B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, and M. Balabanovic, “A domain-specific software architecture for adaptive intelligent systems,” *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 288–301, 1995.

[7] B. Kitchenham, “Procedures for performing systematic reviews,” Keele University, Tech. Rep. TR/SE-0401, Jul. 2004.

[8] D. Fox, W. Burgard, and S. Thrun, “Markov localization for reliable robot navigation and people detection,” in *International Workshop on Sensor Based Intelligent Robots, LNCS: 1724*, Dagstuhl Castle, Germany, 1999, pp. 1–20.

[9] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*, ser. Intelligent robotics and autonomous agents. The MIT Press, Aug. 2005.

[10] AUTOSAR, “AUTOSAR (AUTomotive Open System ARchitecture),” [*On-line*], *World Wide Web*, 2011, available in: http://www.autosar.org/ (accessed 02/25/2012).

[11] Microsoft, *Microsoft Reference Architecture for Commerce Version 2.0*, ser. Patterns & practices. Microsoft Press, 2002.

[12] E. Y. Nakagawa, F. Oquendo, and M. Becker, “Ramodel: A reference model of reference architectures,” in *WICSA/ECSA’2012*, Helsinki, Finland, 2012, pp. 1–5.

[13] ACM, “ACM Digital Library,” [*On-line*], *World Wide Web*, available in: http://portal.acm.org (accessed 01/28/2012).

[14] IEEE, “IEEE Xplore,” [*On-line*], *World Wide Web*, available in: http://ieeexplore.ieee.org (accessed 01/26/2012).

[15] ISI, “ISI Web of Knowledge,” [*On-line*], *World Wide Web*, available in: http://apps.isiknowledge.com (accessed 01/24/2012).

[16] Elsevier, “SciVerse Scopus,” [*On-line*], *World Wide Web*, available in: http://www.scopus.com (accessed 01/28/2012).

[17] J. S. Albus, “4D/RCS - a reference model architecture for intelligent unmanned ground vehicles,” *Unmanned Ground Vehicle Technology*, vol. 4715, pp. 303–310, 2002.

[18] F. Ortiz, D. Alonso, B. Alvarez, and J. Pastor, “A reference control architecture for service robots implemented on a climbing vehicle,” in *Ada-Europe’10, LNCS: 3555*, York, UK, 2005, pp. 13–24.

[19] L. Peters, M. Pauly, and A. Arghir, “Servicebots - a scalable architecture for autonomous service robots,” in *FUZZ-IEEE’00*, vol. 2, San Antonio, USA, 2000, pp. 1013–1016.

[20] D. Weyns and T. Holvoet, “A reference architecture for situated multiagent systems,” in *E4MAS’06, LNCS: 4389*, Hakodate, Japan, 2006, pp. 1–40.

[21] JAUS, “JAUS (Joint Architecture for Unmanned Systems),” [*On-line*], *World Wide Web*, 2012, available in: http://www.openjaus.com/ (accessed 01/17/2012).

[22] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley, 2003.

Human Computer Interaction Teaching Method to Encourage Creativity

Deller James Ferreira
Informatics Institute
Federal University of Goiás
Goiânia, Brazil
deller@inf.ufg.br

***Abstract*—Scarce attention has been given to Human Computer Interaction teaching methods to promote creativity and innovation. Standard approaches to teaching interface design include the use of design aids such as usability guidelines, interaction design patterns and anti-patterns. These approaches do not, however, encourage creativity properly. Interaction design space is usually limitedly and unsystematically explored during student designs. In this paper, we propose a pedagogical framework for design exercises for use in the teaching of Human Computer Interaction and present some examples of its usage. The use of the pedagogical framework makes it possible to teachers create significant Human Computer Interaction experiences to students, fostering them to activate mental processes underlying creativity.**

Keywords-creativity; human computer interaction; collaborative learning

I. Introduction

New information technologies have revolutionized the way people work, study, socialize, access and transmit information, have fun, communicate and access services. This fact calls attention to the problems caused by software with poor user interfaces, making computer science researches and software developers very aware of Human Computer Interaction (HCI) as a driving force within software development practice and usability as an essential aspect of HCI.

A bad user interface for a Website or desktop application has a huge social cost. A badly designed user interface can detrimentally influence millions of people and consequently cause users to make expensive errors [1]. Although today there are lots of good solutions that can be reused by developers, many user interfaces are bad designed. That leads us to believe that one possible reason of the problems of HCI is an absence of appropriate and efficacious education. Undeniably, an effective way to improve HCI is by improving HCI education. Questions about education and understanding HCI must be addressed in the academic or research literature about this subject.

Indeed, computer science research has wide interest in effective teaching methods for the discipline. The field of HCI recognizes a more or less standard process of how to teach and practice interaction design, that is summarized the following phases: problem definition, user study, iterative prototyping, and evaluation [2]. This point of view is adopted in HCI as an engineering discipline. In reflecting on key objectives of engineering, primary concerns are with efficiency and reliability.

In an engineering approach of HCI, practitioners must be able to build interfaces quickly and consistently. Engineering as a discipline that seeks procedures to systematize and operationalize best practices, allowing others to create usable interfaces and lending analytical structures to guide analysis within diverse contexts [1]. Teaching engineering practices involve the engagement of HCI students in problem-solving by using procedures and analytical methods. Software engineering, computer science, and information systems students are taught structured methods to analyse, decompose, and to develop systems. Such structures can hinder creativity, which is generally a much less structured activity.

User-centered design approaches do not necessarily lead to good designs either [3]. User-centered design helps the designer focus on the user and the context of their work. Creative and innovative solutions require an extra effort, which the designer must then build upon to deliberately and consciously to devise them.

As a result from engineering emphasis, most HCI courses have a significant focus on teaching the students to evaluate interfaces usability. This sparked the advent of a new industry in usability evaluation services [3]. Despite much more research is still needed for methods that improve usability, we believe that another very important problem is a lack of methods for inventing better solutions and designs in IHC. If there are better solutions in the first place, there will be a lesser need to make tests and redesigns to software that has been delivered.

Creativity needs to be more valued in HCI. Researchers in HCI education must investigate more what is involved in inventing creative solutions, and therefore, how we might be able to teach this to the students. They should adapt and repurpose existing tools and methods, and orchestrate them in a way that would boost creativity, in order to scaffold students to gain the insights that can lead to creative solutions.

The challenge for educators then is to move from these highly organized structures, to organized, but yet creative, structures that can facilitate creative invention [4]. To achieve this aim, teachers needs to understand the nature of creativity and inventiveness and therefore how it can be fostered in the light of HCI context, without disregarding the systemic HCI methods available for the search of well-crafted designs.

In this paper we propose a framework for design exercises for use in the teaching of HCI aiming at positioning teachers and students with regards to both adaptive and innovative creativity. The use of the pedagogical framework elaborated in this research makes possible to teachers create significant HCI experiences to students, allowing them to activate mental processes underlying to both innovative and adaptive creativity. Innovative creativity is related to the original, transformational, and expressive, while adaptive creativity linked to logical, adequate, and well-crafted solutions.

First, in section II, we present a framework for design exercises in HCI. In section III, we discuss the application of the framework in HCI teaching. In section IV, we describe a preliminary case study that was conducted during one semester of an HCI course. In section V, we present and discuss some preliminary results. Finally, in section VI we describe further investigations concerning the framework presented.

II. Framework for Design Exercises in HCI

Collaborative learning is the pedagogical method that provides the fundamentals for the proposed framework. Collaborative learning has been proved efficacious when the teacher helps the students to develop the collective ability to use dialogs for learning, fostering productive interactions during argumentation in instructional settings. Discourse must be facilitated aiming creative and innovative processes and products.

This work introduces a framework to nourish creative discussions during collaborative problem solving in HCI. We consider here that invention occurs at different levels of innovation. The framework contains seven collaborative and creative dimensions. The dimensions are: immersing, unpacking opportunities, exploring complementary paths, overcoming boundaries, expanding, discovering unpredictable places and developing. Each dimension contains dialogic processes. Dialogic processes are dialogs aligned with mental creative processes associated to both adaptive and innovative creativity. Dialogic processes afford ideas build upon other ideas while people collaborate. Here we say that the students widen the design space when a new idea emerges and that they deepen the design space when an idea is evolved. Following, we present the dialogic framework dimensions.

A. Dimension 1. Immersing in the Design Space

Students can widen the design space while discuss having in mind search information having an objective in mind and search information for inspiration, detect relevant and irrelevant information, recognize familiar information and cope with new information, reapply techniques and adapt techniques, experience having an open mind and experience having an objective, state goals and brainstorm, adapt hypothesis and make conjectures, are aware of generalities and specificities, and explore similarities and differences of problems.

According to Jonassen [5], when students scrutinize similar problems for their structures, they gain more robust conceptual knowledge about the problems, constructing stronger problem schema. This dimension concerns with the enhancement of the analogical thinking. Analogical reasoning involves the transfer of solutions from previously known problems to novel ones and the ability to abstract similarities and apply previous productive experiences to new situations. This dimension is also concerned with the search for information. To be successful at discovery and innovation students should be aware of previous and related work and should be aware of principles and techniques to be applied in the development of their work. The more diverse your knowledge, the more interesting the interconnections.

*B. Dimension 2. U*npacking Opportunities *of the Design Space*

Students can deepen the design space when discuss while collaboratively look for attributes and relationships among concepts and new ideas, and try to organize the information, recognize dependence and independence relations, necessary and sufficient conditions, causes and effects, similarities and differences, correspondences and oppositions, class inclusion and exclusion, associations and dissociations, hierarchy ascendant and descendant relations, order and disorder, abstract and concrete features, potential and non-potential uses/functions, examples and counter-examples, and make an interplay between concrete and abstract features.

Guilford advocates that elaboration and fluency are two fundamental components of the creative process [6]. This dimension embraces the divergent thinking abilities elaboration and fluency. The teacher can boost the students´ improvement of these abilities to explicit what is already there but hidden and also to deal with the who, what, why, and how elements of solution ideas.

C. Dimension 3. Exploring Complementary Paths in the Design Space

This dimension involves complementarities. Here, we elaborate dialogic processes based on Ponty´s [7] notion of “chiasm”. In Ponty´s notion of “chiasm,” two concepts emerge as complementary ways of referring to an idea. For example, both sides, figure and ground, depend upon each other and can reverse around each other. This divergence is considered to be a necessary and constitutive factor in allowing subjectivity to be possible at all. However, he suggests that rather than involving a simple dualism, the divergence between touching and being touched, or between the sentient and the sensible, mind and body, subject and object, self and other also allows for the possibility of overlapping and encroachment between these two terms. For all dialogic process we address its complementary to address more possibilities of exploration of an idea and to achieve better awareness of students. The teacher is also called to elaborate the students´ tasks based on complementary concepts.

D. Dimension 4. Overcoming Boundaries of the Design Space

Students can widen the design space while discuss and jointly situate ideas in a bigger, smaller or different context, performing contextual shifting, search for relationships with "neighbour" ideas outside a given context, deals with scope and limitations, and deals with constraints relaxation or imposition.

Generating alternative designs is an essential aspect of the interaction design process [2]. Considering alternatives and thinking about different perspectives can provide the designer with considerable insight into the problem space. Considering alternatives is important to "think out of the box". Students as future designers and software developers must be trained to consider alternatives.

E. Dimension 5. Expanding the Design Space

The students can widen the design space making together recombination and combination of similar or distinct concepts and ideas, building on other´s ideas, decomposing and composing ideas, re-thinking their previous ideas, and rebutting ideas. The students also try to make combinations of possible disparate or unconnected ideas. They derive new knowledge on the basis of a lack of similarity between two or more past constructs or elements from domains which are far apart.

This dimension entangles constructive interactions among students related to innovative construction of a complex system of ideas. The main premise in this dimension is that unexpected and new arrangements and other´s interpretation trigger new interpretations and ideas. Previous opinions and concepts are co-constructed and students´ understandings expanded. Students integrate answers from many places in diverse ways, in a process of transcending and exchanging different perspectives and constructing new ideas.

Here, it is evoked Dewey´s notion of transactional inquiry to elaborate the creativity concept in a dialogic way. Dewey defined inquiry as a set of operations by which an indeterminate situation is rendered determinate [8]. When participants engage in inquiry together, new meanings are created as a co-production. For Dewey, the term transaction emphasizes the transformational aspects of interaction [9]. A mutual exchange is a transaction whenever a response to another's act involves contemporaneous response to a thing as entering into other's behaviour, and this upon both sides.

F. Dimension 6. Discovering Unpredictable Places in the Design Space

Students can widen the design space, when they have the opportunity to explore a bad idea. They do not only reflect about positive impacts, relevant implications or good features, but also reflect upon why a failure occurred, about negative impacts, features and implications, why an idea did not have impact, and problems created. They do not just eliminate the wrong paths, but reflect and take advantage of it. Students turn ideas and concepts in new interpretations, also thinking about misconceptions.

This dimension capitalizes on often way in which bad ideas become beneficial detours to good ideas. The exploration of good ideas allows a local exploration of the design space, which leaves unexplored large areas of this space [10]. The exploration of bad ideas pulls the students to new unpredictable places, facilitating a movement to far away places, which thus allows students to overcome the limitation of exploration that good ideas entail.

G. Dimension 7. Developing the Design Space

Students can deepen and widen the design space evaluating, comparing, selecting concepts and ideas, considering different alternatives, pointing positive and negative outcomes based in criteria application, starting a search for a more adequate cognitive perspective, reasoning process aiming to resolve conflict and uncertainty, identifying best solutions, and removing inconsistencies.

This dimension encompasses the evaluation, critics, and bringing together of ideas. By means of evaluations of ideas students are able to carry out decision-making processes based on criteria application and improve ideas considering its bad features. One important aspect of this dimension is that when students evaluate and critique different perspectives and ideas they must be confronted with uncertainty and conceptual conflict. Both are states of disequilibrium that activate a process of conflict resolution and a quest for certainty [8]. Besides, interaction criticism is a design practice that enables design practitioners to engage with the aesthetics of interaction, helping practitioners cultivate more sensitive and insightful critical reactions to designs and exemplars [11].

III. Applying the Framework in HCI Teaching

According to Preece et al. [2] interaction design is about designing interactive products to support people in their everyday working lives and interactive experiences that enhance and extend the way people communicate, interact and work. Interface design comprises determining how content is organized and presented, choosing appropriate design metaphors and affordances, and providing effective interaction and techniques.

However, teaching students how to develop interactive experiences is not an easy task. There are a myriad of aspects involved that should be considered. It is not an easy task to create a meaningful experience for others. The students must first understand your audience, their needs, abilities, interests, and expectations, and then have the ability to conceptualize and refine effective solutions. The same apply to teaching students to be creative and allowing them to experience such skills.

Teaching attention is mainly focussed on usability principles and guidelines, but although principles and guidelines are extremely important, they are not the only aspect to be considered. Principles and guidelines do not suggest solutions, although they can be used to guide design. Principles and guidelines of usability provide clues to the designer about what to do but not about how to do.

Patterns are increasingly being used in software engineering education. Many pattern libraries have been published [12][13] and more are appearing every year. Alexander introduced the notion of patterns [14]. A pattern focuses on the relationship between problem, solution and context. The solution can be realized in different ways but has an invariant core, which captures all the possible solutions to the problem given. Solutions described in patterns are proved to work in practice, they are a proven solution for a common user interface or usability problem that occurs in a specific context of work. Patterns communicate insights into design problems capturing the essence of problems and designs in a compact form. They describe the problem in depth, the rationale for the solution, how to apply the solution and some of the trade-offs in applying the solution [15].

Some research shows that the use of interaction patterns is successful [16]. It was shown that designers who made more use of the available interaction patterns were able to produce better results than those not using the patterns. Anti-patterns are also used to convey the knowledge in HCI. An anti-pattern is a solution that seems like a good idea, but backfires badly when applied, and can cause an interface to fail. Anti-patterns are literature written in pattern form to encode practices that do not work or that are destructive.

Guidelines, patterns, and anti-patterns exist to capture experts' expertise and to communicate knowledge. Design guidelines and patterns can be used to help the interface design, being used to aid the production of usable design solutions. These design aids however, encourage and foster creativity or the generation of new metaphors or alternative designs in a limited way.

A trade-off exists between enforcing the use of standard design guidelines or patterns and encouraging the development of creative design solutions. This paper explores the possibilities for developing a combined approach to teaching creative interface design. This approach proposes the combined use of guidelines, patterns, and anti-patterns and the proposed framework with the aim to produce usable and creative design solutions. Following we provide some possible instructional uses of this integrated approach.

A. *Examples of Tasks in View of the Framework*

Example 1. This task is based on Dimension 1. In this task, the teacher establishes an HCI subject and patterns related to the chosen subject. Afterwards, the teacher uses the dialogic processes from the Dimension 1 to scaffold students' collaborative activities and dialogues to promote knowledge creation. This task focuses on Websites navigation and the discussion is centered on the fat menu pattern [12].

Considering the dialogic process "Search information having an objective in mind" and "Specification", the teacher asks students to search Websites to discuss different adaptations of the Fat menu pattern. Also, the teacher ask the students to discuss if the Websites found are strongly related to the problem described in the pattern. The problem is: the designer deals with many categories, possibly a hierarchy with three or more levels.

Having in mind the dialogic processes "Recognize familiar information" and "Generalization", the teacher asks the students to discuss trying to figure out what kind of Websites is best designed by fat menus based on the examples provided in the pattern description and the Websites found.

Paying attention to the dialogic process "Adapt techniques", the teacher asks the students to jointly choose a Website to adapt the Fat menu pattern, regarding that users must focus their attention on the available navigation options with no distractions. The teacher asks the students to discuss ways to use headers, dividers, white space, and how to take advantage of horizontal space. Also, the teacher asks the students to discuss the uniformity and regularity of the pattern adaptation, designing to fit well into colour scheme and other aspects on the page.

Regarding the dialogic processes "Make conjectures" and "Search information for inspiration", the teacher asks students to discuss the viability to include graphics elements in the fat menu pattern adaptation. The student can look at similar systems or look at very different systems.

Example 2. The teacher asks the students to make connections between the Fat menu pattern and other patterns, between Fat menu pattern and guidelines, and among elements and other patterns inside the pattern. Questions formulated by the teacher are used to lead the discussions. The questions are based on the dialogic processes from Dimension 2 "Recognizing associations", "Being aware of concrete and abstract features", "Recognizing order", "Recognizing class inclusion and exclusion", "Recognizing dissociations", "Exploring differences and similarities", and "Recognizing associations".

1. The Fat menu pattern can include others to complete him?
2. There must be order inside the Fat menu pattern?
3. How can be organized the categorizations inside Fat menu pattern?
4. How can be managed the vertical separation of categories in the Fat menu pattern?
5. How are the differences and similarities between Fat menu pattern and Menu page pattern [12]?

6. Using the Fat menu pattern the user enters directly in the content. This feature can be associated to a guideline?
7. Using the Fat menu pattern the information is hidden until the user looks for it. This feature can be associated to a guideline?

Example 3. In this task, the teacher takes into account Dimensions 3 and 7. In view of Dimension 3, students are supposed to explore complementary paths and in light of Dimension 7 students are supposed to criticize and build on other's ideas.

The dialogic processes from Dimension 7 considered in this task were: "Point negative and positive outcomes", "Compare ideas", and "Select concepts and ideas".

The teacher chooses three Websites and asks the students to point in each Website positive and negative outcomes of the following Website complementary features and to justify their answers. The complementary features to be considered by the students are:

1. Visibility and constraint;
2. Delimiters and white spaces;
3. Figure and background;
4. Good and bad usage of a metaphor;
5. Simplicity and complexity of the information;
6. Breadth and depth in navigation.

Afterwards, the teacher asks the students to compare and select the best Website designs.

Example 4. This task pays attention to Dimensions 1, 2, 4 and 5. In this example, the teacher follows the steps:

1. Choose a Web or desktop interface;
2. Ask the students to discuss while collaboratively decompose the interface, detecting the patterns used in its design;
3. Change the application context;
4. Ask the students to discuss while collaboratively integrate, adapt, and elaborate the patterns to create a new design in the new application context.
5. The dialogic processes took into account were "Decompose ideas", "Combine ideas", and "Recombine ideas" from Dimension 5; "Perform contextual shifting" from Dimension 4; "Elaborate an idea" from Dimension 2, and "Adapt techniques" from Dimension 1.

Example 5: In this task is regarded Dimension 6. Here, the teacher utilizes anti-patterns to provide opportunities for students to explore a bad design in order to better understand good designs. Anti-patterns capture poor or sub-optimal software development practices. So, the students have the opportunity to analyse why an apparently good idea did not have a positive impact and also to investigate how they can provide an alternative for a poor design. First, the teacher presents a set of anti-patterns, each one possessing a bad characteristic, such as: external inconsistency, internal inconsistency, dialog box without cancel button, go overboard in selection during data entry, badly designed affordances, badly designed metaphors, and complex or extremely deep navigation. Second, the teacher asks the students to discuss about what is bad concerning the anti-pattern, why this feature is illogical, inadequate, and ill-crafted, if there is a good design possessing this feature, if so what is the difference, and if there is a situation where it could be considered well-crafted.

IV. Preliminary Case Study Description

A preliminary case study was conducted during one semester of an HCI course. One class containing forty-eight under graduate students from software engineering course was subdivided in 8 (eight) groups, each group containing 6 (six) students. The students were analyzed considering interaction and participation patterns in online discussion forums in Moodle Platform. There were assigned 7 (seven) collaborative tasks to the students. The tasks are described following.

Task 1. The students were asked to analyze 16 (sixteen) Websites, considering good and bad usages of affordances and metaphors. The students were also invited to evaluate and critique other students' ideas.

Task 2. The students were asked to analyze 7 (seven) Websites, considering usability guidelines. The students were also invited to evaluate and critique other students' ideas.

Task 3. First, each group must choose a good and bad Website. Second, the groups must justify your choices, taking into account good and bad usages of affordances, good and bad usages metaphors, and usability guidelines.

Task 4. The teacher presented usability guidelines and interaction patterns for mobile applications. The students were asked to create a mobile version for a given Website, obeying the usability guidelines and performing patterns adaptations and combinations.

Task 5. Make a collaborative paper interrelating the usability guidelines presented. Discuss in your group forum.

Task 6. Criticize the Website of the Institute of Informatics. Discuss in your group forum.

Task 7. Make a re-design of the Institute of Informatics' Website. Consider the interaction patterns presented and discuted in class.

Interaction and participation patterns were analyzed based on Newman, Webb and Cochrane's adapted model [17] described in 10 categories. This model was chosen because it covers key aspects of the proposed framework. The framework application aims profitable students' interactions that result in a deeper and wider design space. Following we describe Newman, Webb and Cochrane's adapted model.

Category 1. Relevance: Relevant states or diversions.

Category 2. Importance: Important points and issues or unimportant points and trivial issues.

Category 3. Novelty, new info, ideas, and solutions: New problem related information or repeating what has been said.

Category 4. Bringing outside knowledge or experience to bear on problem: Drawing on personal experience or sticking to prejudice or assumptions.

Category 5. Ambiguities; clarified or confused: Clear statements or confused statements.

Category 6. Linking ideas, interpretation: Linking facts, ideas and notions or repeating information without making inferences or offering an interpretation.

Category 7. Justification: Providing proof or examples or irrelevant or obscuring questions or examples.

Category 8. Critical assessment: Critical assessment or evaluation of own or others' contribution or uncritical acceptance or unreasoned rejection.

Category 9. Practical utility (grounding): Relate possible solutions to familiar situation or discuss in a vacuum.

Category 10. Width of understanding (complete picture): Widen discussion or narrow discussion.

V. Preliminary Results

Successful interactions require broad and active students' participation. The total number of students' posts was 773. So, there was a substantial participation of the students.

Considering Category 8, the results indicated that online interactions were cohesive. The students engaged critically or constructively in other students' ideas in 50% of the posts. They asked and clarified doubts in 40% of the posts and acted solo in 10% of the posts.

Regarding Category 9, the students used the guidelines as criteria to judge what is good and bad. In Task 1 the students did not know how to judge what was a good or bad usage. In tasks 2, 3, and 6, the students justified their ideas by means of the guidelines.

Considering Category 5, 6 and task 5, each group provided a distinct integration scheme for the guidelines. Each group had a different interpretation. However, they provided designs as examples for their connections. It proofs that IHC is a complex and ill-structured subject. The teacher should had confronted the different schemes and promoted discussions involving the whole class in order to converge to a solution.

Taking into account Categories 1, 2, and 7, there was 70% of relevant posts and 30% of diversions. Also, in the relevant posts, were discussed important points and issues.

Regarding Category 4, task 4 was successfully and easily performed by the students. Due to the fact that the great majority of students possessed i-phones or android interface mobile phones, the students could draw on personal experience to design the mobile interface.

Considering Category 3 and Task 7 the students successfully adapted and combined interaction patterns, being able to apply previous information to solve a problem during the Website re-design.

Taking into account Category 10, there were widen discussions, containing many different points and aspects being analyzed.

VI. Conclusion and Future Work

The main contribution of this paper is a novel framework to boost creativity in interaction design. The framework provides a structure for the elaboration of design exercises. Teaching interaction design is approached under a dialogic point of view taking advantage of dialogic processes. Dialogic processes are mapped in creative ways of thinking, so they serve to scaffold students to productive discussions. Dialogic processes are underneath creative dimensions that reveal forms of interaction design space exploration. The proposed framework provides broad and systematic interaction design space exploration and is theoretically supported by collaborative learning, many researches that address idea generation and researches involving interaction design space exploration.

The preliminary results indicated that the proposed framework has a great potential to help teachers to mediate students' creativity, through facilitating students' involvement in productive discussions, which in turn increases the quality of the design process. Students benefits from activating creative mental process during interaction design discussions and performing a better exploration of design space both in breadth and in depth, while teachers benefits from many strategies to elaborate design exercises involving usability guidelines, interaction design patterns and anti-patterns in order to boost discussions.

To ensure that the framework combined to usability guidelines, interaction design patterns and anti-patterns can indeed provide an effective connected approach to teaching HCI, we intend to further investigate its application as future work.

For at least four semesters of an HCI undergraduate course, there will be one class taught considering collaborative learning, but not the dialogic framework (control group) and another one taught by the proposed method (treatment group). Students performance will be investigated by discourse analysis in order to check if there is knowledge co-construction and advancement as well as the achievement of a deeper and wider knowledge in the collaborative settings. The discourse analysis will focus on analysis of interaction and participation Patterns [18]. We will also analyze and compare the students´ designs by means of an instrument to measure Website creativity [19] and indicators of creativity in solutions [20].

Acknowledgment

This work is supported by FAPEG - Fundação de Amparo à Pesquisa - Goiás - Brazil.

References

[1] H. Thimbleby, "Teaching and Learning HCI", Proceedings HCI International, Part I, Universal Access, HCII 2009, Lecture Notes in Computer Science, edited by C. Stephanidis, San Diego, Springer Verlag, 2009, pp. 625–635.

[2] J. Preece, Y. Rogers, and H. Sharp, Interaction Design: Beyond Human- Computer Interaction. 2nd Edition. New York, NY: John Wiley & Sons, 2007.

[3] W. Wong, P. Kotzé, J. Read, L. Bannon, and E. Hvannberg, From inventivity in Limerick to creativity in Aveiro: Lessons learnt, in: Creativity and HCI: From Experience to Design in Education - Selected Contributions from HCIEd 2007 (this book), edited by P. Kotzé, W. Wong, J. Jorge, A. Dix, and P.A. Silva, IFIP Series, Springer, 2008, pp. 19-29.

[4] P. Kotzé and P. Purgathofer, "Designing Design Exercises – from Theory to Creativity and Real-world Use", HCIEd 2009. Aveiro, Portugal, 2009, pp. 29 – 30.

[5] D. H. Jonassen, "Research Issues in problem Solving", The 11th International Conference on Education Research New Educational Paradigm for Learning and Instruction, 2010.

[6] J. P. Guilford, The nature of human intelligence. New York: McGraw-Hill, 1967.

[7] M. Ponty, The Visible and the Invisible. Evanston: Northwestern U Press, 1968, pp. 267.

[8] J. Dewey, The Quest for Certainty: A Study of the Relation of Knowledge and Action. London: George Allen e Unwin, 1929.

[9] J. Dewey, Logic: The theory of inquiry, in J. A. Boydston (Ed.) John Dewey: The Later Works, 1925-1953, Volume 12, Southern Illinois University Press, Carbondale, IL, 1938, pp. 1-5.

[10] A. Dix, T. Ormerod, B. Twidale, B. Michael, C. Sas, A. P. G. Silva, L. McKnight, "Why Bad Ideas are a Good Idea", Proceedings of the HCI Educators' Workshop HCIEd.2006-1 Inventivity: Teaching theory, design and innovation in HCI, 2006, pp. 24-28.

[11] J. Bardzell, "Interaction Criticism: An Introduction to the Practice", Interacting with Computers, 2011, pp. 604-621.

[12] J. Tidwell, Designing Interfaces. 2nd Edition. O'Reilly, 2010.

[13] V. Welie, Patterns of Interaction Design. http://www.welie.com/. [retrieved: January, 2012].

[14] C. Alexander, A Pattern Language: Towns, Buildings, Construction. Oxford: Oxford University Press, 1977, pp. 1216.

[15] W. J. Brown and R. C. Malveau, Architectures. Anti-patterns: Refactoring Software and Projects in Crisis. John Wiley and Sons, 1998.

[16] S. R. Schach, Object-oriented and Classical Software Engineering. New York: McGraw Hill Higher Education, 2005.

[17] C. C. Sing and M. S. Khine, An Analysis of Interaction and Participation Patterns in Online Community. Educational Technology & Society, 2006, pp. 250-261.

[18] C. C. Sing and M. S. Khine, "An Analysis of Interaction and Participation Patterns in Online Community", Educational Technology & Society, 2006, pp. 250-261.

[19] D. Cropley and A. Cropley, "Recognizing and Fostering Creativity in Technological Design Education", International Journal of Design and Education, 2010, pp. 345-358.

[20] L. Zeng, G. Salvendy, and M. Zhang, "Factor Structure of Website Creativity", Computers in Human Behavior, 2009, pp. 568-577.

Future Chances of Software Customization: An Empirical Evaluation

Michaela Weiss, Norbert Heidenbluth
Inst. for Applied Information Processing,
University of Ulm,
89069 Ulm, Germany,
Email: {michaela.weiss, norbert.heidenbluth}@uni-ulm.de

***Abstract*—Customization is an important market trend because companies can only survive when they focus on their customers' needs. In order to offer demand-driven customization options, it is necessary to empirically analyze the benefit of the various adaptations from a customer's point of view. In the software sector, only a few surveys have been conducted that exceed technical aspects. Moreover, existing studies are limited to the overall acceptance and benefits of customization, but draw no conclusions on different adaptation options. Thus, we present a large study that analyzes the starting points of software customization and deals with general questions of customization. The results indicate that software customization increases the willingness to pay (WTP) by about 15%. The survey points out that especially customization options, which adapt the functionality, increase the usability, and enable parental controls are of great importance for future software implementation. Hence, our results enable competitive advantages by implementing customization options that meet customer needs.**

Keywords-customization; adaptability; tailoring; user study; human-computer interaction.

I. Introduction

Today, competitive pressure and customer empowerment change selling conditions. Customers are no longer willing to accept the *customer sacrifice* [1], [2], the gap between products offered and customer needs. That is why the *long tail phenomenon* [3], [4] starts to rule the market and a multitude of tiny niche markets replaces traditional mass markets. Hence, producers turn from selling off-the-shelf products to offering customization. Åhlström and Westbrook [5] have shown that the demand for non-standard products is even growing and producers plan to increase customization.

This trend can also be seen in the software sector. *Software product lines (SPL)* help to build software that satisfies a specific market segment on the basis of a common set of core assets [6]. However, accessibility movements and the regulation by law demand an even stronger focus on the individual (cf. Section 508 of the *Rehabilitation Act* [7] and the *German Equality Law For Disabled People* [8]). However, accessibility is not the only reason for software customization. The *International Standard on Ergonomics of Human System Interaction* (ISO 9241/110) indicates that customization is an important principle to design a dialog. The *Technical Report on Software Engineering* (ISO/IEC TR 9126-2) even says that customization is a requirement of software quality that helps to meet the user's needs. Software customization could also enhance the customer experience [9]. Thus, customization is a crucial part in current software engineering.

There are various ways to customize software. The *DUFS* customization classification [10] organizes this richness and helps us to outline software customization. This categorization subdivides *design customization*, *usability customization*, *functionality customization*, and *customization of service and communication*. In this context, design customization means an adaptation of the appearance of the *Graphical User Interface* (GUI). As companies can only prosper if they focus on their customers, software developers need to know which customization features are in demand. A previous empirical study [5] cites this lack of knowledge of customer needs as the major difficulty in customization.

Nevertheless, existing studies often only focus on non-software vendors and software customization surveys are limited to technical aspects. Thus, we conduct a comprehensive survey. This paper elaborates on the small excerpt presented in a previous paper [10]. The detailed results on customization enable an in-depth analysis of customer evaluation. Hence, customer opinions of customization in the non-software sector as well as in the software sector are considered and future chances are identified. To our knowledge this survey is the largest one in terms of software customization and the only one that considers customization starting points. Thus, our study helps software developers to decide on customization implementation and provide adaptations that are valuable for their customers.

In the following, Section II summarizes previous surveys on customization and explains why analyses of non-software customization provide valuable insights that could be used for software customization. Section III introduces our study and illustrates the methods used. Afterwards, Section IV presents the non-software analysis and spotlights customization usage. The results of the software customization investigation are presented in Section V. Before concluding, Section VI critically examines the survey.

II. Background and Related Work

The beginnings of customization go back to 1987, when Stanley Davis introduced a business strategy to implement customization called *Mass Customization* (MC) [11]. In 1993, Joseph Pine made this strategy popular [12]. From then on, the trend of customization spread. The following lists the most important studies on customization in the non-software sector as well as in the software sector. Additionally, the section shows why findings of non-software products could be valuable for software products.

A. Non-Software Products vs. Software Products

Kotler and Armstrong described a product as ”anything that can be offered to a market for attention, acquisition, use, or consumption that might satisfy a want or need” [13]. Additionally, Peter Dracker stated that a company can only prosper if it focuses on its customers and their needs [14]. Consequently, the customer's perception of the product and its value determines the company's success.

According to Kittlaus and Clough [15], the value that comes out of the intangible product software can only be realized in its functionality. However, the emergence of appearance customization in software, e.g., the tailoring of forum appearances, shows that software is more than something to get things done. Thus, there are similarities in customer perception between non-software and software products, even though they differ greatly in characteristics.

Due to the importance of customer perception, we believe that findings from the non-software area in terms of perceived customization could be valuable for the software sector. This practice is useful since the non-software sector has a much longer history and many people see software as incomprehensible ”black magic” [16]. Thus, non-software products are well-known to a wider audience whereas software knowledge could still be limited.

In contrast to non-software products, many software characteristics support its customizability. Software has no physical form and belongs to the economic factor of *knowledge* [15], [16]. This makes later adaptations easier and enables repeated customization. As Frederick Brooks said, this easy adaptability obliges software vendors to offer adaptation options [17]. Besides, the delivery of software is simple, fast and could be made on an individual basis. These facts facilitate software customization.

B. Previous Studies on Non-Software Customization

To address customer needs it is important to know which product features create value from a customer's point of view. As only customers can answer this question properly, many surveys have been conducted. Unfortunately, many of them only consider non-software products [18], [19], [20]. Piller et al. [21] listed existing studies of MC and highlighted especially the additional contribution that could be achieved with the help of customization. By offering shoes that are adapted in terms of fit, function and design, the sporting goods producer *Adidas*, e.g., achieved 30 to 50% higher prices [21], [22]. However, in 1998 Huffman and Kahn [23] empirically documented problems in information retrieval.

C. Previous Studies on Software Customization

Despite the fitness of software for customization few studies analyze general aspects of software customization. The most important one was made in 1991 by Mackay [24]. She observed the triggers and barriers of software customization. According to her, the main triggers are the reusing of repeated patterns, the retrofitting after a system change, and the avoidance of annoying behavior. In contrast, barriers are a lack of time and knowledge. In 1996, these results were proven by Page et al. [25].

Most existing studies on software customization only consider technical aspects. Many authors compared the three methods of software customization. *Adaptable initiatives* are based on the self-customization of the user. Moreover, in the non-software area it is also quite common for manufacturers to adapt the product to the customer's needs. In the software sector this can be done by the software itself. This method is known as *adaptive initiative*. Additionally, software could use a *mixed initiative* which combines both aspects. Thus, several studies tried to identify the best practices for designing menus [26], [27] or GUIs [28], [29], [30]. Furthermore, research is done to analyze accessibility aspects [31], [32]. These studies verified the benefits of software customization. With regard to quantitative aspects, an increase in performance and decrease in error rates could be measured. Moreover, improvements in qualitative aspects, such as usability, stress in usage, and individual preferences, became visible. The increase in user satisfaction, a software quality requirement (cf. ISO/IEC 25051:2006), was empirically verified in a study on *Apache Security Software* [33]. However, its validity was limited by only interviewing skilled users.

The financial effects of software customization have been studied by Oliver et al. [34]. They found that 5 to 10% higher profit margins and doubled revenues could be realized.

All existing surveys on the subjective advantages of software customization focused on overall feelings but no conclusions on the acceptance and benefit of particular customization features could be drawn. Thus, we conducted a large study that used the DUFS customization classification [10] to analyze customer perception on different customization options. DUFS sub-divides software customization into four categories. *Design customization* sums up all options that help adapt the interface's appearance according to the customer's preferences. *Usability customization* refers to adaptations which make the software more effective, efficient and task satisfying (cf. DIN EN ISO 9241 Part 11). All customization options that help close the gap between offered and needed functionalities belong to the category of

TABLE I. PROFESSIONAL DISTRIBUTION

Profession	Participants	
Job Applicant	8	2.92%
Scholar	26	9.49%
Trainee	12	4.38%
Student	59	21.53%
Employee	94	34.31%
Operative	12	4.38%
Executive	36	13.14%
Self-employed	20	7.30%
Senior Citizen	7	2.55%
Sum	274	100%

functionality customization. Customized auxiliary services and customized software messages and greetings are part of *service and communication customization*. By using this categorization and involving heterogeneous user groups our survey gives detailed insights into the appraisal of software customization. Additionally, it deals with general questions on customization to form a basis for in-depth research.

III. METHODS

Our empirical study analyzes customization from a customer's point of view. All values in the text are rounded off to two digits after the decimal point.

A. Data Collection and Sample

As previously outlined [10], the study was conducted in 2010 in South Germany. The answers of ten interviewees could not be used because of missing data. Thus, the study includes the answers of 274 participants. 43.43% of them are female. The sample includes heterogeneous participants (cf. Fig. 1 and Table I). The survey could be completed either electronically (20.44%) or in paper form (79.65%).

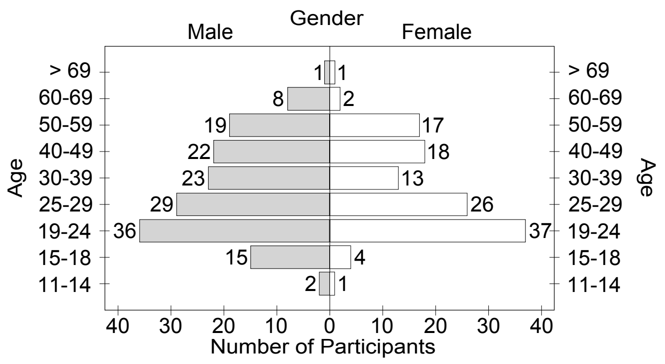


Figure 1. Age Distribution.

The sample is slightly different to the whole German population because of the proximity to the University of Ulm. It includes comparatively many young, well-educated participants and male opinions are somewhat overrated. However, the large and heterogeneous sample allows conclusions to be drawn on perceived customization.

B. Participant Grouping

Two participants had never used a computer before, so the sample size for the computer-related questions is 272. For analysis purposes these participants were divided into *frequent users* that use their computer daily and perform at least five different tasks and *non-frequent users*. The study contains data of 175 frequent and 97 non-frequent users.

IV. CUSTOMIZATION USAGE AND REASONS

The survey started with non-software customization to facilitate access and ensure the quality of the given answers. Additionally, it helps to get elementary insights into customization. These results are pointed out in the following.

A. Customization Usage

In order to analyze customization usage we subdivided non-software products into the categories *clothing and shoes*, *vehicles and bikes*, *grocery and beverages*, *stationery*, *souvenirs*, *jewelry*, *health care*, as well as *electronic articles*. The survey gave examples of customization in the different categories. This helped to ensure that all participants could properly answer if they had ever used customization in these categories. Fig. 2 illustrates the results.

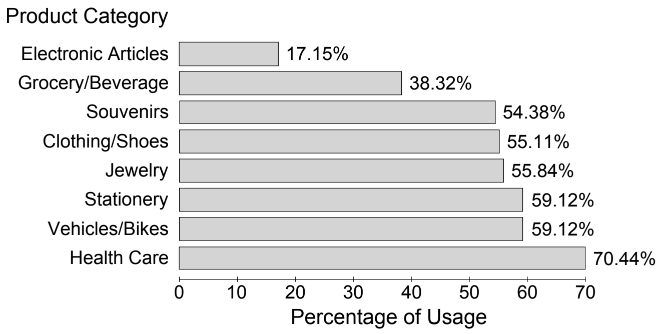


Figure 2. Customization Usage.

The overall usage rate of 51.19% supports customization research. Besides of adaptations of electronic articles, grocery and beverage, customization usage is quite common.

Analyses of the adaptation starting points with the help of the *DEFS* customization categories *design*, *ergonomics and fitting*, *functionality*, as well as *services* [10] show that in most of the product categories adaptations are generally minor variations in design. These adaptations are used with clothing and shoes, vehicles and bikes, grocery and beverage, stationery, souvenirs, jewelry, and electronics. Nevertheless, products with adaptations to increase the ergonomics and fitting are also available. In the health sector these adaptations are necessary to cope with the customer-specific body. However, even clothes or shoes are adaptable in terms of ergonomics. Thus, e.g., the sporting goods producer *Adidas* offers ergonomically adaptable shoes. Above all, the categories vehicles and bikes as well as electronic articles offer ways of adapting the functionalities by adding adequate modules. In contrast, service customization is only rarely available in the *Business-to-Consumer* (B2C) sector.

B. Reasons for Customization Usage

The survey tried to get an insight into the reasons for the named customization usage. The participants had to state if they use customization to *adapt the functionality*, *adapt the design*, *highlight individuality aspects*, or *participate in a future trend*. They also got the chance to say they see *no benefit* in customization. This question was semi-open and multiple answers were allowed. Fig. 3 lists the results.

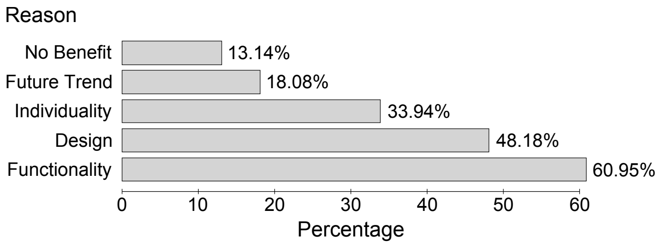


Figure 3. Reasons for Customization Usage.

According to the participants, the main reason for customization is to adapt the functionality. Another frequently named reason is design adaptation. The enhancement of one's individuality which rests on a human basic need was also seen as a major factor. Moreover, only 13.14% of the participants see no benefit in customization. In contrast, 18.08% judge customization as an important future trend. This justifies the importance of this paper's topic.

The free-text answers showed that customization is used to possess something unique, increase the imaginary value, differ from the mass, and give presents. The participants also revealed the importance of value for money aspects.

Since the female and male answers differ, we did a Chi-Square Test ($\alpha = 0.05$) to analyze this. Table II shows that only the fact that the women use customization more often to highlight their individuality (41.18% vs. 28.39%) could be traced back to gender.

TABLE II. CHI-SQUARE TEST
H0: The occurrence of a customization reason is independent of gender.

Topic	Pearson's Chi-Square	Degrees of Freedom	Asymptotic Significance (2-sided)	Fisher's Significance (2-sided)
Functionality	3.538	1	0.060	0.063
Design	0.254	1	0.615	0.628
Individuality	4.911	1	0.027	0.029
Future Trend	0.526	1	0.468	0.526
No Benefit	0.017	1	0.895	1.000

V. SOFTWARE CUSTOMIZATION

The second part of the survey examined the usage of existing software customization options, financial benefits for software vendors, and future chances.

A. Usage of Software Customization

In order to get meaningful results on the perception of existing software customization options, we chose features that are well-known to participants of all age categories. The participants rated 15 adaptation options in the areas *operating systems*, *Office products* and *world wide web*. All but one are adaptable options because users take more notice of self-made changes than of automatic ones. This increases the reliability of the findings. With regard to usability, we listed the *creation of links*, *creation of bookmarks*, the *quick launch bar* of the operating system, and the *tool bars* of Office programs which all help to get quick access. Usability aspects can also be customized by tailoring the *update handling* and *choosing one's native language*. Existing software offers many features to adapt the design, i.e. the GUI appearance. We chose the adaptation of *fonts*, *colors and contrasts*, *icon size*, *desktop background*, *mouse pointer*, and the *screen saver*. Furthermore, the participants judged the functional customization offered by *iGoogle* and *Windows gadgets*. Since service customization is only rarely available in the B2C sector, we limited our survey to *adaptive purchase proposals* in online shops.

The survey explained each feature to ensure that all participants understood the questions. Afterwards, the participants stated if they knew the customization option and if they had ever used this adaptation. Additionally, they rated the benefit of each option. According to Schwarz et al. scales with zero-to-positive-values should be used to measure the intensity of a single attribute [35]. Thus, we used a scale from *0* to *5* to evaluate the feature's benefit. A value of *5* indicates that it is very useful. In contrast, a benefit of *0* shows that this feature has no benefit at all. Fig. 4 illustrates the results and maps them to the DUFS software customization categorization. It also shows the usage of the customization options. Even though most of the options are well known, some participants did not know several features. We excluded these participants to calculate a meaningful percentage of usage. The strong correlation (Pearson's r = 0.87) between benefit values and usage highlights that the customer's view is of crucial importance in terms of customization research.

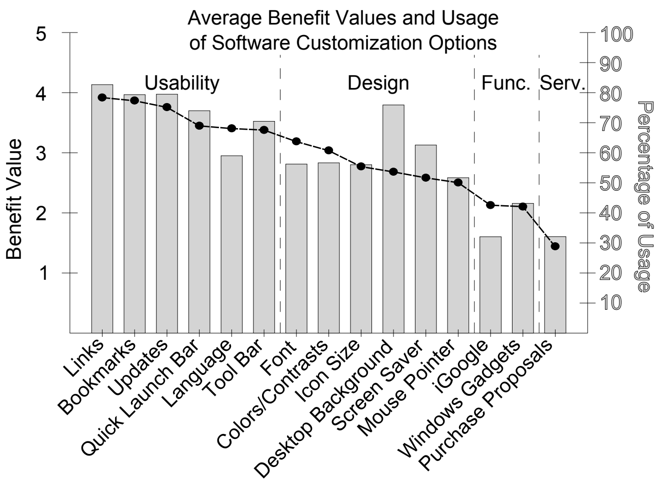


Figure 4. Benefit and Usage of Software Customization.

The mapping of the named customization options to the DUFS categories shows that usability adaptations in particular achieved high benefit values. Design customization options are rated with high benefit values, too. In contrast, functionality adaptations as well as adaptations of service and communication are rated particularly low. This might be based on some external factors. In contrast to the long-established examples in the categories usability and design, the specified functionality adaptations are quite new. This negatively influences the knowledge, acceptance and also the perceived benefit. The results of customization usage in the non-software sector verify this argumentation. Functionality adaptations are the main reason named for customization. Furthermore, it has to be pointed out that service adaptations have only been rarely available to date in the B2C sector.

Additionally, our findings show that all evaluated design customization options achieved lower benefit values than usability adaptations. All functionality customization options achieved lower benefit values than design adaptations and service and communication customization options are left far behind. This proves that the DUFS classification categories reflect customer perception adequately.

B. Willingness To Pay

We differ between non-software and software products to examine the *Willingness To Pay* (WTP) and evaluated the additional contribution in comparison to an off-the-shelf product. As our goal was to analyze the general WTP rather than the WTP for a specific product, we chose an hypothetical approach. The analyses of Miller et al. [36] showed that hypothetical approaches could generate mean WTP estimations that do not significantly differ from the actual WTP and could be used to make meaningful management decisions. Our participants had to state if they would pay *not more*, *10%*, *25%*, *50%* more, *double the price*, or *more than double the price* to get a customized product. The latter was quantified by a contribution of 150%. Table III summarizes the results – specified by age and user-group. The WTP of the participants younger than 15 years and older than 69 years are not representative but are listed for completeness.

TABLE III. ADDITIONAL CONTRIBUTION

Age (in Years)	Non-Software frequent users	non-frequent users	overall	Software frequent users	non-frequent users	overall
11-14 Years	-	*10.00%*	*10.00%*	-	*11.67%*	*11.67%*
15-19 Years	9.55%	26.88%	14.21%	13.18%	18.13%	15.26%
20-24 Years	15.76%	24.29%	17.40%	12.80%	13.93%	13.01%
25-29 Years	20.24%	13.08%	17.64%	21.79%	16.15%	19.55%
30-39 Years	13.64%	12.31%	13.06%	14.32%	11.92%	14.14%
40-49 Years	11.00%	11.75%	11.38%	9.25%	8.75%	9.00%
50-59 Years	13.53%	17.22%	15.28%	17.35%	18.61%	18.00%
60-69 Years	27.50%	7.50%	15.50%	33.75%	11.67%	20.50%
> 69 Years	-	*0.00%*	*0.00%*	-	*0.00%*	*0.00%*
Average	15.69%	15.52%	15.22%	15.69%	13.87%	14.97%

The overall average willingness for an additional contribution in the non-software sector is 15.22% and indicates that there is a broad-mindedness for customization. Other surveys in the mass customization area document an even higher WTP of about 30% and partly 100% [21], [37].

The average contribution of 14.97% in the software sector is lower. The reason for this could be that software is not a daily product for everybody and could seem to be less important. This argumentation is verified by the fact that the average contribution of non-frequent users is 14.90% in terms of non-software products but 13.79% with regard to software. In contrast, the average contribution of the participants with frequent computer usage is in both product categories exactly the same (15.46%). These findings could encourage the assumption that in a world where computer usage becomes increasingly common the WTP for software customization will become more similar to non-software products. Thus, the analysis of non-software products could be valuable for software vendors.

C. Future Chances

In order to evaluate the potential of software customization, the survey contained an appraisal of the perceived future chances. We used the DUFS classification to make a distinction. By adding the subcategory *parental controls* we extended the DUFS category *functionality*. The category *usability* was also further divided by using the subcategories *intuitiveness* and *language*. This enables detailed analyses.

In every category the participants rated the future chances of software customization by evaluating their need for future implementation. The survey highlighted that participants should incorporate adaptable and adaptive adaptations. Once again we used a *0*-to-*5* scale to achieve consistency and support the reliability of the results. A value of *0* marks *no future requirements* whereas a value of *5* characterizes *great future requirements*. Fig. 5 illustrates the mean future requirements and their variances in a simplified *Software Customization Chart* (SCC) [10]. In contrast to a full SCC, we make no distinction between adaptable, adaptive or mixed initiatives.

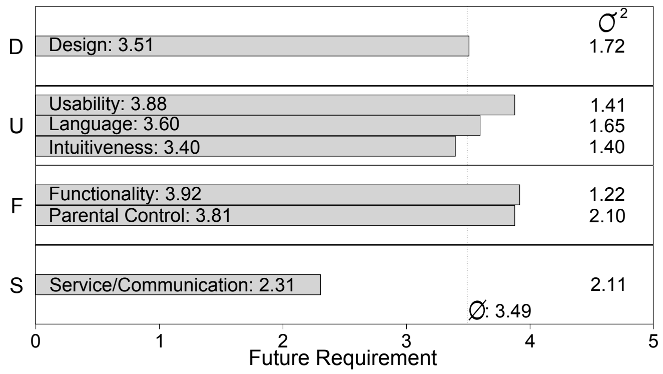


Figure 5. Future Requirement of Software Customization.

The results show that functionality customization in particular achieves high rates and should be increasingly implemented in future software. The participants also highlighted

TABLE IV. T-TESTS ON DIFFERENCES IN FUTURE CHANCES

DUFS Categories	Gender Differences						User Group Differences					
	Leverne-Test		T-Test for a Mean *H0: The average values in the male and female group are the same.*				Leverne-Test		T-Test for a Mean *H0: The average values of the frequent and non-frequent users are the same.*			
	F	**Siffgnificance**	**Uniform Variances** $\alpha = 0.05$	**T**	**Degrees of Freedom**	**Significance (2-sided)**	**F**	**Significance**	**Uniform Variances** $\alpha = 0.05$	**T**	**Degrees of Freedom**	**Significance (2-sided)**
Design	5.491	0.020	false	-4.355	266.393	0.000	2.213	0.138	true	2.367	270	0.019
Usability	1.333	0.249	true	-1.518	272	0.130	0.953	0.330	true	0.249	270	0.803
Language	2.528	0.113	true	-1.170	272	0.243	1.166	0.281	true	-2.312	270	0.022
Intuitiveness	0.003	0.958	true	-0.573	272	0.567	12.726	0.000	false	3.266	169.419	0.001
Functionality	2.411	0.122	true	-0.613	272	0.540	8.671	0.004	false	1.305	175.428	0.194
Parental Ctrl.	8.780	0.003	false	-3.067	270.760	0.002	0.039	0.844	true	0.125	270	0.900
Service/Com.	0.862	0.354	true	-2.951	272	0.003	0.133	0.716	true	-0.061	270	0.952

great future chances in usability customization. Customization options which enable parental control by limiting the available functionality achieved high support, too. Moreover, the results support the importance of adapting the language of the GUI. The participants also called for an improvement in design adaptations and the intuitive software handling. In terms of service and communication customization the estimations were only moderate. This might be based on the fact that the participants could not evaluate the benefits of service customization because of the low availability in the B2C sector. In summary, an overall value of 3.49 emphasizes the future chances of software customization and highlights the need for further research. Furthermore, the differences in the starting-point evaluation support the DUFS classification.

We performed t-tests to verify the differences in evaluations of female and male participants. The results (cf. Table IV) indicate that the differences in the categories design, communication, and parental controls depend on gender. With regard to parental controls the female rated the future requirement very high (4.10) in comparison to a moderate rate in the male group (3.58). The women also rated the future chances of design adaptations (3.89) as well as adaptations in services and communication (2.61) higher than the men did (3.23 resp. 1.90). In all other categories the differences could not be tracked back to gender.

Further t-tests analyzed the higher valuation of the frequent computer users. Table IV illustrates that the differences in the rating of intuitiveness, design, and language depend on the frequency of computer usage. Fig. 6 shows these differences.

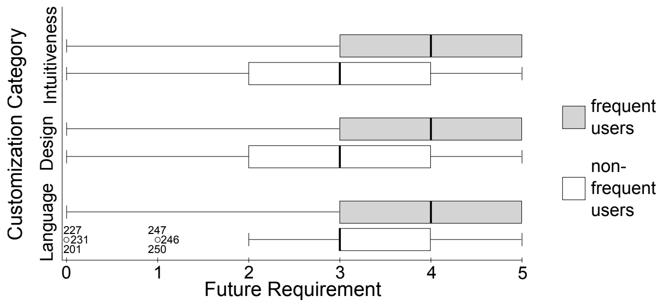


Figure 6. Future Requirement divided by user group.

VI. DISCUSSION

In the following, some threats to validity are considered and the importance of this paper's topic is proved.

A. Threats to Validity

The chosen questions could have an impact on the results and be a threat to internal validity. However, we controlled this by carefully designing the questionnaire, avoiding ambiguous questions, keeping consistency, and using well known examples within the survey. Moreover, we checked the survey in a pretest with five persons.

A potential threat to external validity might be the representativeness. However, the age, the profession, and the expertise of the participants differ significantly and the data set is sufficiently large. To our knowledge, this is actually the largest study of customer acceptance and benefits of software customization. Moreover, it is the only one that evaluates software customization starting-points. Hence, we judge the reported results to be meaningful.

The large sample size also supports the validity of the Chi-Square Test in Section IV. Although, there is only one degree of freedom, the minimal expected frequencies (15.64 to 59.07) are far away from the critical border of ten.

B. Validation of the Need for Customization

The results show that some answers are related to being part of a specific user group, e.g., men or women or frequent or non-frequent users. Yet, many evaluations are based on the user's experience, characteristics, and preferences. This diversity can be seen in the listed variances (cf. Fig. 5). Only customization could cope with these individual aspects.

VII. CONCLUSION AND FUTURE WORK

In this paper, we analyzed the acceptance and benefits of customization in the non-software as well as in the software sector. For this reason, we conducted a comprehensive study with 284 participants. Our results showed that customers want customization and are willing to pay a contribution of about 15% for adaptable or adaptive software. The results revealed an overwhelming approval of software customization but indicated unused capabilities in existing systems. The participants use the existing software customization options. However, they highlighted that the importance of the adaptation depends on the starting point of the customization. The participants especially emphasized the benefits of adaptations to increase the usability. With regard to future software engineering, the participants called for the improvement of functionality and usability customization options. Moreover, they requested further adaptations in parental controls. Additionally, the findings supported the DUFS customization classification. In summary, the study gave insight into the perception of software customization from a customer's point of view. This could help software developers to detect valuable software customization and provide software that exactly meets customer-specific needs.

For future work, the enlargement of the study could identify cultural impacts on customization perception.

References

[1] C. W. Hart, "Mass customization: conceptual underpinnings, opportunities and limits," *Int. J. of Service Industry Management.*, vol. 6, no. 2, pp. 36–45, 1995.

[2] A. Bardakci and J. W. AND, "How "ready" are customers for mass customisation? an exploratory investigation," *European J. of Marketing*, vol. 38, no. 11/12, pp. 1396 – 1416, 2004.

[3] C. Anderson, "The long tail," *Wired Magazine*, vol. 12, no. 10, pp. 170–177, 2004.

[4] C. Anderson, *The Long Tail: How Endless Choice is Creating Unlimited Demand*. Random House, 2010.

[5] P. Åhlström and R. Westbrook, "Implications of mass customization for operations management," *Int. J. of Operations and Production Management*, vol. 19, no. 3, 1999.

[6] P. Clemens and L. M. Northrop, *Software Product Lines: Practices and Patterns*. Addison Wesley, 2002.

[7] *Rehabilitation Act*. United States of America, 1973.

[8] *Gesetz zur Gleichstellung behinderter Menschen (BGG)*. Bundesrepublik Deutschland, 2002.

[9] S. Marathe and S. Sundar, "What drives customization? control or identity?" in *Proc. of the 2011 annual conf. on Human factors in computing systems*, CHI'11, 2011, pp. 781–790.

[10] M. Weiss and N. Heidenbluth, "A two-dimensional overall software customization classification and visualization," in *Proc. of the 2012 Int. Conf. on Software Engineering Research & Practice*, SERP'12, 2012, pp. 293–299.

[11] S. M. Davis, *Future Perfect*. Addison-Wesley, 1987.

[12] B. I. Pine, "Mass customization: The new frontier in business competition," 1993, Harvard University Press.

[13] P. Kotler and G. Armstrong, *Principles of Marketing*. Pearson Education, 2009.

[14] P. F. Drucker, *The Practice of Management*. HarperBusiness, 2006.

[15] H.-B. Kittlaus and P. N. Clough, *Software Product Management and Pricing*. Springer Verlag, 2009.

[16] M. Dorfman and R. H. Thayer, Eds., *Software Engineering*. IEEE Computer Society Press, 1997.

[17] F. P. Brooks, "No silver bullet," *Computer*, vol. 20, no. 4, pp. 10–19, 1987.

[18] E. Consortium, *The Market for Customized Footwear in Europe*, F. T. Piller, Ed., 2002.

[19] C. Kieserling, "Mass customization in the shoe industry," 1999, survey conducted by Selve AG, Munich.

[20] Outsize, *Problems and Needs of Customers when buying Clothes and Shoes*, R. Duwe, Ed., 1998.

[21] F. T. Piller, K. Moeslein, and C. M. Stotko, "Does mass customization pay?" *Production Planning & Control*, vol. 15, no. 4, pp. 435–444, 2004.

[22] C. Berger and F. Piller, "Customers as co-designers," *IEE Manufacturing Engineer*, pp. 42–45, August/September 2003.

[23] C. Huffman and B. Kahn, "Variety for sale: mass customization or mass confusion?" *J. of Retailing*, vol. 74, pp. 491–513, 1998.

[24] W. E. Mackay, "Triggers and barriers to customizing software," in *Proceedings of the SIGCHI Conf. on Human Factors in Computing Systems*, CHI'91, 1991, pp. 153–160.

[25] S. R. Page, T. J. Johnsgard, U. Albert, and C. D. Allen, "User customization of a word processor," in *Proc. of the SIGCHI conf. on Human factors in computing systems*, SIGCHI'96, 1996, pp. 340–346.

[26] L. Findlater and K. Z. Gajos, "Design space and evaluation challenges of adaptive graphical user interfaces," *AI Magazine*, vol. 30, no. 3, pp. 68–73, 2009.

[27] L. Findlater and J. McGrenere, "A comparison of static, adaptive, and adaptable menus," in *Proc. of the SIGCHI conf. on Human factors in computing systems*, SIGCHI'04, 2004, pp. 89–96.

[28] A. Bunt, C. Conati, and J. McGrenere, "What role can adaptive support play in an adaptable system?" in *Proc. of the 9th int. conf. on Intelligent user interfaces*, IUI-CADUI'04, 2004, pp. 117–124.

[29] A.Bunt, C. Conate, and J. McGrenere, "Supporting interface customization using a mixed-initiative approach," in *Proc. of the 12th int. conf. on Intelligent user interfaces*, IUI'07, 2007, pp. 92–101.

[30] J. McGrenere, R. M. Baecker, and K. S. Booth, "An evaluation of a multiple interface design solution for bloated software," in *Proc. of the SIGCHI conf. on Human factors in computing systems*, SIGCHI'02, 2002, pp. 164–170.

[31] K. Z. Gajos, J. O. Wobbrock, and D. S. Weld, "Automatically generating user interfaces adapted to users' motor and vision capabilities," in *Proc. of the 20th annual ACM symposium on User interface software and technology*, UIST'07, 2007, pp. 231–240.

[32] G. Z. Krzysztof, J. Wobbrock, and D. S. Weld, "Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces," in *Proc. of the twenty-sixth annual SIGCHI conf. on Human factors in computing systems*, SIGCHI'08, 2008, pp. 1257–1266.

[33] N. Franke and E. von Hippel, "Satisfying heterogeneous user needs via innovation toolkits: The case of apache security software," *Research Policy*, vol. 32, no. 7, pp. 1199–1215, 2003.

[34] K. Oliver, L. Moeller, and B. Lakenan, "Smart customization: Profitable growth through tailored business streams," *strategy+business*, vol. 34, pp. 34–45, 2004.

[35] N. Schwarz, B. Knuper, J.-J. Hippler, E. Noelle-Neumann, and L. Clark, "Rating scales: Numeric values may chance the meaning of scale labels," *Public Opinion Quarterly*, vol. 55, pp. 570–582, 1991.

[36] K. M. Miller, R. Hofstetter, H. Kromer, and Z. J. Zhang, "How should consumers' willingness to pay be measured? an empirical comparison of state-of-the-art approaches," *J. of Marketing Research*, vol. 48, no. 1, pp. 172–184, Feb. 2011.

[37] F. T. Piller and M. Müller, "A new approach to mass customization," *Int. J. of Computer Integrated Manufacturing*, vol. 17, no. 7, pp. 583–593, 2004.

Experimentation Package for Evaluation of Problems Applied to the Software Project Subject Using PBL

Jacson Rodrigues Barbosa*, Fabrizzio Alphonsus Alves de Melo Nunes Soares*, Auri Marcelo Rizzo Vincenzi*
**Instituto de Informática*
Universidade Federal de Goiás, UFG
Goiânia-GO, Brazil
E-mail: {jacsonbarbosa, fabrizzio, auri}@inf.ufg.br

***Abstract*—This paper presents an experimentation package that compares the traditional and problem-based learning (PBL) approaches in the Software Project academic subject. The package was applied in a controlled experiment in a Computer Science class of a higher education institution, having message-oriented middleware as a case study. The case study enabled us to validate the experimentation package and to collect initial data to investigate the advantages and disadvantages of PBL against traditional learning. Even though the statistical analysis failed to show differences between the two approaches in view of the data collected, students' answers to a questionnaire enable us to verify how PBL may be used to increase their motivation and interest in the subject.**

***Keywords*-Message-oriented middleware; software project teaching; problem-based learning.**

I. Introduction

In the teaching process of almost all fields of knowledge, a constant and crucial feature is problem solving or the preparation for problem solving. If the teacher provides facts and procedures to his/her students without giving them the chance to carry out investigations on their own and to formulate questions, they may memorize the subject but be unable to understand it in depth or to apply it [1].

Problem-based learning (PBL) offers a structure that helps students understand a given subject in more detail. According to this method, problems must challenge the students to reach higher levels of knowledge [2]. A possible way to verify the cognition levels that a certain problem must reach in PBL consists in applying Bloom's revised [3] taxonomy, which classifies the cognitive abilities of individuals according to six levels, as is shown in [4], and summarized below:

- Remember: to produce correct information from memory;
- Understand: to provide a meaning to educational material or experiences;
- Apply: to use a procedure;
- Analyze: to break down a concept into parts and report on their relation with the whole;
- Evaluate: to carry out inferences based on criteria and patterns;
- Create: to link pieces of data in order to create something new.

Therefore, this paper proposes an experimentation package which assesses the cognitive levels reached by students enrolled in the Software Project subject when instructed by PBL, as well as to identify the quality of the software they developed, based on standard OO metrics [5].

This paper is structured as follows: Section II presents the problems that were investigated in Software Project. Sections III and IV describe the experimentation package applied and the questionnaire given to the students, respectively. Section V displays the results obtained from applying the package. Finally, Section VI offers conclusions and suggestions for future research.

II. Problems applied to the Software Project subject

Message-oriented middleware (MOM) and software coupling were selected for the case study.

Message-oriented middleware is a communication method between software components used in distributed systems. A client may send and receive asynchronous messages to and from any other client, connected to a special agent that provides facilities to create, send, receive and read messages. Software coupling is a measure of the interconnection between theses classes or subsystems. Strong coupling means that the related classes need to know internal details from each other, that changes spread throughout the system and that the system is potentially more difficult to understand. Thus, loose coupling is linked to the considerable need for flexibility required by great distributed systems, in addition to failure tolerance. This means that the dependencies must be kept to a minimum as much as possible so that, in case of failure or unavailability of a system and/or service, the others remain available and working. MOM allows for loose coupling. Sender and receiver do not need to be synchronized or previously known. It is an alternative to synchronized distributed methods which may resort to blocking during communication.

In the PBL approach used in this research, the notions of MOM and coupling are presented in three interdependent problems.

The first problem discusses the main concepts regarding MOM and coupling, as well as presents two message exchange models. The first model is based on producer/consumer problems and is also known as point-to-point. It follows the concept of queues that are fed with messages, which in turn are then removed by a single consumer. The producer sends messages to a queue and the consumer reads them. In this case, the producer knows the message's destination and sends it directly to the consumer's queue. This model comprises the following features:

- Only one consumer reads the message;
- The producer does not have to be executed while the consumer reads the message, just as the consumer does not need to be executed while the producer sends the message;
- Once a message is successfully read, the consumer acknowledges the message to the producer.

The second model selected is the publisher/subscriber model. It is based on the concept of topic, according to which messages are listed in topics that are received by one or more subscribers [6]. It supports publishing messages to a given topic of messages. The subscriber(s) may register interest in receiving ("subscribing") messages on a particular topic. According to this model, neither the publisher nor the subscriber knows about each other. Its features are:

- Several consumers may read the message;
- There is a timing dependency between publishers and subscribers of a given topic. A publisher must create a subscription for subscribers to receive messages. The topic subscriber must be continuously active in order to receive messages.

Once the problems were discussed with the students, the latter were asked to research two real-life problems that may be solved by using each of the models presented in class. They were given two days to solve this first problem.

As regards the second problem, two brief descriptions of systems were presented. The first is used for selling cinema tickets (Software 1), whereas the second records students' enrolments in the subjects of a given university (Software 2). Students were asked to define the most suitable architecture (publisher/subscriber or point-to-point) for both software, as well as create a class diagram and a prototype of graphical user interface. They were given two weeks to complete this task.

Finally, the third problem involved asking students to build Software 1. In view of the fact that they already had knowledge of Java^{TM}, they were asked to implement the software via Java 6, Java Message Service (JMS) - Version 1.1 and Oracle^{TM}GlassFish Server Open Source Edition 3.1.

The tasks were given in this order to promote more effective learning, taking into consideration the cognitive levels explored in each problem proposed. Further details regarding these levels are provided in the following section.

A. Problem analysis according to Bloom's taxonomy

Tables I, II and III show the cognitive levels for Problems 1, 2 and 3, respectively, in accordance with Bloom's taxonomy.

The first problem aims to assess the students' abstraction abilities, i.e. to verify their ability to choose real-life applications of the recently learnt tools. For instance, this problem verified their ability to choose real-life examples that are linked with each of the recently discussed MOM architectural models. Therefore, this problem intended to explore the most superficial levels of Bloom's taxonomy, having provided the students with an initial contact with MOM-related concepts. Hence, the concepts under study were those of remembering, understanding and applying.

Table I
Analysis of Problem 1

Cognitive level	Characterization
Remember	MOM software architectural concepts.
Understand	Understanding of MOM concepts.
Apply	Identification of real-life examples linked with both kinds of architecture.
Analyze	Not explored.
Evaluate	Not explored.
Create	Not explored.

The results of this stage are shown as a table of confusion, also known as a confusion matrix. It consists of a table with two rows and two columns that reports the number of false positives, false negatives, true positives and true negatives.

The second problem also aimed to assess the students' abstraction abilities, but unlike the previous problem, now they had to choose the tool i.e. the model most suited to solve the problem in question. Table II shows the cognitive levels explored in this problem.

The third problem aimed to assess the students' technical abilities. During this stage they had to show the extent of their ability to encode software by using the MOM models learnt. The cognitive levels explored are shown in Table III.

Table II
Analysis of Problem 2

Cognitive level	Characterization
Remember	Use of MOM concepts acquired in problem 1 and use of representational form of classes and associations in UML.
Understand	Understanding the representational form of classes and associations in UML in order to meet the corresponding functional requirements and architectural restrictions.
Apply	Use principles and techniques of software project modelling.
Analyze	Identification of parts of the problem that are modellable based on informal textual specification.
Evaluate	Design the software to meet the quality attribute of usability.
Create	Creation of conceptual model (UML class diagram) and graphical user interface project.

Table III
Analysis of Problem 3

Cognitive level	Characterization
Remember	Use of software project concepts acquired in problem 2.
Understand	Understanding of MOM to solve the problem.
Apply	Use of Java for error treatment, database access and message exchange.
Analyze	Identification of parts which must be implemented in the program based on the models.
Evaluate	Choose specific data structures to provide an effective solution.
Create	Development of software for selling/booking cinema tickets.

III. Experimentation package for assessing PBL in the Software Project subject

This section presents the experimentation package which was adopted during the experimental tasks and result analysis. Its detailed description makes it possible for this study to be replicated in future research. The experimentation package set for the analysis of PBL in the Software Project subject was organized following [7], Its stages were:

A. Definition of experiment

To analyze PBL in the teaching of Software Project.

With the purpose of assessing PBL in the teaching of Software Project.

As regards factors that contribute to the quality of teaching, such as the ability to remember, to understand, to apply, to analyze, to evaluate and to create.

In the context of undergraduate students of Computer Science enrolled in Software Project.

B. Selection of context

The context chosen for the experiment is the teaching and application of MOM-related concepts. The experiment involved the use of academic software developers whose time and tools were controlled for teaching and the designing of a software for cinema ticket selling via JMS.

C. Formulation of hypotheses

A two-way analysis of variance (ANOVA) was performed to verify statistical differences between PBL and traditional approaches in Software Project.

ANOVA defines the null hypothesis (H_0), according to which there are no statistical differences between the methods under analysis. When the probability (p) found is lower than 0.05, the null hypothesis may be rejected, otherwise it will not be possible to state that there are statistical differences between the methods.

D. Selection of variables

The variables analyzed in the experiment are divided into two types: dependent and independent. The former were the cognitive levels reached in the problems, and the latter were the problems' cyclomatic complexity and size.

E. Selection of participants

All students enrolled in the Software Project subject were selected ($N = 14$). Observe that, even though N is small, an experimentation package allows us to replicate this same experiment several times, and new collected data can be added to this one, thus increasing confidence on the obtained results. This first replication aimed at validating the proposed package.

F. Experimental project

Firstly, the teacher presented the major theoretical concepts regarding MOM and the two message exchange models: point-to-point (queue model) and publish/subscribe. No examples were given regarding the models.

G. Quality assessment

A control sample was also used to validate the experiment internally. This sample consisted of a task set to the students regarding the traditional learning approach (first part of the subject); according to the task, students had to define a class diagram based on a context previously specified by the teacher. The external validation, as the students' profiles in Table VII confirm, was based on academic professionals who were then enrolled in the third semester of the course and whose average work experience in the field amounted to 7.5 months.

H. Preparation

Students were not aware of the reasons behind the research being carried out, but were informed of the steps to be taken to solve each of the three problems proposed.

I. Application

The experimentation package was applied during one semester (five months) in the Software Project subject, part of the Computer Science course. The first part of the subject involved the application of traditional methodological strategies (first two months), and the second part involved PBL (last three months).

J. Descriptive statistics

Ordinal and interval scales were used for the statistical analysis of the experiment to verify students' performance under both traditional and PBL approaches. Once the data had been collected, the sample was assessed to establish whether it showed normal or non-normal distribution. In case of the former, parametric statistics was required; in case of the latter, non-parametric statistics was required [8].

K. Data reduction

The criteria for data reduction were only required in case of dropouts, i.e. students who did not carry out any of the tasks set. About 21.4% of data (three students dropped out of the university) was left out so as not to jeopardize the experiment results.

IV. Students' views on PBL

After having solved Problem 3, the students were given a questionnaire on the experience of problem-based learning. The following questions in Table VII were adapted from those found in [9].

V. Results and discussion

A. Characteristics of Problem 1

In Table IV each column represents the student's option, whereas each row yields the correct option. For instance, line 2 and column 2 show that 100% of the students managed to correctly identify real-life problems that may be solved through the point-to-point model.

As Table of Confusion shows, solving this problem proved quite easy for the students. Of the solutions provided for the Topic model, only 18.18% did not comply with its definitions.

Table IV
Table of confusion of Problem 1

	Topic	Point-to-point
Topic	81.82%	0%
Point-to-point	18.18%	100%

B. Characteristics of Problem 2

As Table V shows, all the students successfully selected the architecture model most suited to each of the software descriptions presented.

Table V
Selection of architecture model

Software	Coherent	Incoherent	Total
Software 1	100%	0%	100%
Software 2	100%	0%	100%

C. Characteristics of Problem 3

The estimate for the necessary effort to develop the software for this problem was calculated by the function point analysis (FPA) described in [10]: 1,949 lines of Java.

Table VI shows some metrics collected from the solutions given by the students with the aid of JaBUTi (Java Bytecode Understanding and Testing) [11]. The first metric regards the size of the programs in terms of non-comment source lines of code (NCLOC). Following are two metrics related to maximum and average cyclomatic complexity [12] of the methods from each program, CC-MAX and CC-AVG, respectively. The remaining metrics are part of C&K metrics [5], such as: weighted method count via cyclomatic complexity (WMC-CC), depth of inheritance tree (DIT), lack of cohesion in methods (LCOM), response for class (RFC) and coupling between objects (CBO).

Regarding the LOC metric, the mean size of implementations was 1,700 LOC. Half of them showed values above average and closer to the estimate given by FPA.

The analysis of metrics related to McCabe's cyclomatic complexity revealed that, even though CC-MAX shows methods with maximum cyclomatic complexity of about 30 (the recommended yield would not be greater than 10 [13]), this only occurs in some isolated methods which, despite showing several conditions, are simple from the standpoint of programming logic. Thus, in general, as the remaining complexity-related metrics (CC-AVG and WMC-CC) attest, class methods are described as simple and devoid of considerable risk [14], this is confirmed by AMZ-LOCM, which shows that the average size of the methods of each implementation ranged from 6.55 LOC in Implementation 1 to 2.59 in Implementation 5, which results in an overall size average of 3.95 LOC.

DIT shows that inheritance was little explored in all projects. The DIT limit was restricted to one, without considering the calculations of Java's API classes.

LCOM is important to estimate the degree of cohesion in a given software. For instance, in an object-oriented software, it may be used to measure the cohesion of each software class. A high LCOM value may suggest that the class project is poor, as was proved in a class of Implementation 5 whose LCOM value was 366.

CBO is useful in showing the potential reuse of classes, given the fact that loosely-coupled classes are "more independent". However, strongly-coupled classes are also very complex and more sensitive to changes in a project, which makes maintenance difficult and requires more rigorous tests. But as Table VI shows, the solutions provided by the students generally showed low coupling.

D. Students' views

Table VII gathers information collected from the students after the academic subject had come to an end. All the students who answered Question 7 (62.5%) declared positive experiences with PBL. Some of the comments were "The method promoted opportunities to increase knowledge", "It makes the students responsible in the teaching-learning process" and "Motivation for students".

As regards Question 8, 71.43% of the students consider the use of PBL in Software Project a "good" opportunity to solve real-life problems. However, they also consider both methods (traditional and PBL) as important in the subject in question (refer to answers to Questions 9 and 10).

E. Statistical analysis

Table VIII and Figure 1 show statistical data that refer to students' performance in the tasks they were assigned, in accordance to the teaching methodology adopted. It is important to point out that, for both methodologies, the same group of students was used in all the problems.

The Lilliefors and Cochran test revealed the normality and homogeneity of the variances ($p < 0.05$); as the problems taken into account in the experiment are also independent, then ANOVA may be used to carry out data analysis.

ANOVA obtained $p = 0.75$ for the set of collected data. The result was higher than 0.05, therefore the null hypothesis (H_0) cannot be rejected i.e. from a statistical perspective and in view of the range of problems explored, there are no statistical differences between the traditional methodology and PBL.

Table VIII
Descriptive statistics

Teaching method	Mean	Median	Standard Deviation
Traditional	7.16	7	0.75
PBL	7.29	7.33	0.98

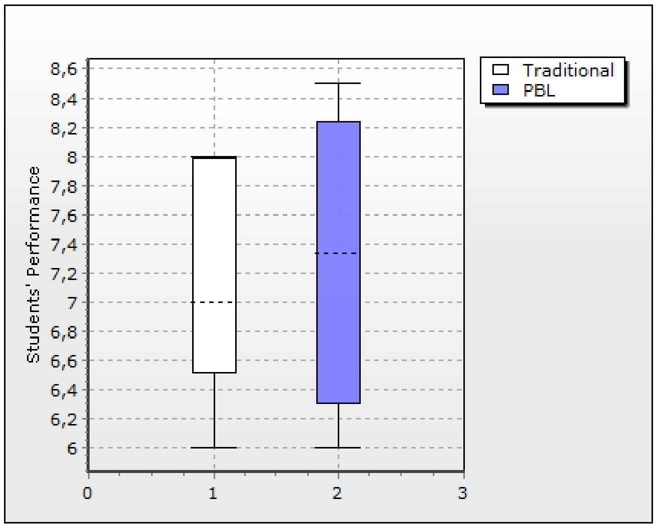


Figure 1. Boxplot of Students' Performance

VI. Conclusion

This paper presented the experimental package and results of a controlled experiment which aimed to compare students' performance in view of the application of two teaching methodologies (traditional and PBL) within the Software Project subject. However, this paper's main objective was to validate the experimentation package based on its first application in the Software Project subject.

The statistical analysis revealed that there are no differences between the teaching methods adopted. Nevertheless, the assessment of the mean differences showed that PBL's was 0.13 higher than the traditional methodology's; furthermore, the analysis of students' answers to the questionnaires revealed that they had positive experiences when using PBL.

In future research, we intend to apply this experimental package to other university classes in order to compare the differences. Furthermore, we intend to repeat the experiment with students enrolled in the following year. Further detailed information on this topic can be found in [15]. We also intend to carry out a statistical analysis (logistic regression) of the metrics collected by JaBUTi to identify patterns associated with cognitive levels.

Acknowledgments

The authors would like to thank the Brazilian Funding Agencies - CNPq and Capes - for their partial support to this work.

References

[1] R. Delisle, *How to use problem-based learning in the classroom.* Alexandria, Virgina, USA: ASCD, 1997.

Table VI
Direct metrics of solutions presented

Implementation	LOC	CC-MAX	CC-AVG	WMC-CC	AMZ-LOC	DIT	LCOM	RFC	CBO
1	2,007	33	1.44	5.63	6.55	1	3.16	18.77	6.16
2	804	27	1.13	7.31	3.32	1	5.25	19.22	6.34
3	1,838	33	1.12	5.16	3.56	1	2.52	15.19	4.52
4	3,579	37	1.24	5.84	4.7	1.06	3.35	20.81	6.97
5	1,238	46	1.06	5.55	2.59	1	6.51	16.88	4.85
6	805	7	1,08	4.23	2.98	1	3.13	14.7	5.49
Average	1,711.83	30.50	1.18	5,62	3.95	1.01	3.99	17.60	5.72
Standard deviation	1,045.02	13.11	0.14	1.01	1.46	0.02	1.55	2.41	0.94

Table VII
Information obtained after questionnaire application

Question	Answer	%
How old are you?	Average=20.26 years	-
Do you work with software development (internship, contract etc.)?	Yes	57.14
	No	42.86
If you answered "yes" to the previous question, how long have you worked in the field (specify the period in months or years, e.g. six months)?	Average=7.5 months	-
Why have you chosen this university to study Computer Science?	Quality of teaching at the university	12.5
	Visibility of the university	12.5
	Duration of the course	12.5
	No particular reason	12.5
When you made your decision, were you aware that the university's Computer Science Department had adopted a new teaching method?	Yes	0
	No	100
If you answered "yes" to the previous question, do you think this influenced your decision?	Yes	0
	No	0
The university decided to apply active teaching-learning methodologies in the Computer Science curriculum. How would you describe your experience as an active participant of such methodologies in this department?	Offered conditions to increase knowledge	37.5
	Makes student responsible in the teaching-learning process	12.5
	Students' motivation	12.5
	Question not answered	37.5
Based on your learning process, how do you rate the use of problem-based learning (PBL) in designing software projects to solve real-life problems?	Excellent	14.29
	Good	71.43
	Fair	14.29
	Poor	0
Do you prefer a traditional teaching method, in which the teacher presents the contents to be learnt and you study them?	Yes	0
	No	28.57
	Both	71.43
You have designed/implemented a software for cinema ticket selling as part of a PBL-based case study. Do you prefer this teaching-learning strategy to the traditional method?	Yes	42.86
	No	0
	Both	57.14

[2] J. A. M. Santos and M. F. Angelo, "Análise de problemas aplicados em um estudo integrado de programação utilizando pbl," *WEI - Workshop sobre Educação em Computação, Anais do XXIX Congresso da SBC*, pp. 519–522, 2009.

[3] B. J. Duch, S. E. Groh, and D. E. Allen, *The Power of Problem-Based Learning: a practical how to for reaching undergraduate courses in any discipline.* Virginia: Stylus Publishing, LLC, 2001.

[4] L. W. Anderson, D. R. Krathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayer, P. R. Pintrich, J. Raths, and M. C. Wittrock, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Complete Edition.* Allyn & Bacon, 2001.

[5] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, June 1994.

[6] I. Gorton, *Essential Software Architecture.* Berlin, Germany: Springer, 2006.

[7] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction.* Springer, 2000.

[8] D. C. Montgomery, *Design and Analysis of Experiments*, 5th ed. John Wiley & Sons, 2001.

[9] J. R. B. Costa, V. F. Romano, R. R. Costa, A. P. Gomes, and R. S. Batista, "Active teaching-learning methodologies: Medical students views of problem-based learning," *Revista Brasileira de Educação Médica*, pp. 13–19, 2011.

[10] E. J. Braude and M. E. Bernstein, *Software Engineering Modern Approaches*, 2nd ed. John Wiley Sons, 2011.

[11] A. M. R. Vincenzi, W. E. Wong, M. E. Delamaro, and J. C. Maldonado, "JaBUTi: A coverage analysis tool for java programs," in *XVII Simpósio Brasileiro*

de Engenharia de Software (SBES 2003), Manaus, AM, October 2003.

[12] T. J. McCabe, “A complexity measure,” in *Proceedings of the 2nd international conference on Software engineering (ICSE ’76)*, October 1976.

[13] R. S. Pressman, *Software Engineering – A Practitioner’s Approach*, 7th ed. McGraw-Hill, 2009.

[14] E. Vandoren and K. Sciences, “Cyclomatic complexity,” june 2000, available at: http://www.sei.cmu.edu/str/. Accessed on: 03/06/2012].

[15] J. R. Barbosa, F. A. A. de Melo Nunes Soares, and A. M. R. Vincenzi, “Problems applied to the software project subject using pbl,” Web page, july 2012. [Online]. Available: http://www.inf.ufg.br/˜auri/pbl-en/. Accessed on: [07/18/2012].

Improving Undergraduate Students' Programming Skills

Sukhamay Kundu
Computer Science Department, Louisiana State University
Baton Rouge, LA 70803, USA
e-mail: kundu@csc.lsu.edu

Abstract – Two key factors for the failure of undergraduate students in creating high quality programs are their inability to: (1) identify the basic steps in building a high-level solution algorithm, and (2) convert these steps into an elegant and efficient program implementation. Both these tasks require creative thinking and a systematic approach with emphasis on the programming process. We believe that with proper training the students can improve their programming skills and create high-quality programs. We describe a new approach to programming, which can be viewed as a refinement of the well-known "stepwise refinement" method. We use the notion of work-breakdown structure to address factor (1) and suggest a few coding techniques to address factor (2). Our initial experience in using the new approach has been very positive in terms of improved quality of student programs.

Keywords: work-breakdown structure; stepwise refinement; programming process.

I. INTRODUCTION

The low success rate in the first and second year undergraduate computer programming courses continues to be a serious problem even today after the decades of many advances in Programming Language designs (including object-oriented languages) and Software Engineering methods. A drop-out rate of 30% in the first year programming courses is not uncommon. We often see another 20-30% of the remaining students having difficulties with programming. This means 45-50% of the incoming computer science undergraduates either change their major or struggle with their computer science degree program. Advances in the programming methods from structured programming and stepwise refinement to object-oriented (OO) programming has not led to significant improvements in the undergraduate students' programming ability [2]. For some students, the problem persists beyond their undergraduate years.

The key element of our new approach to programming is the use of work-breakdown structure (WBS) in building the high-level pseudocode or algorithm. The notion of WBS is routinely used by business analysts as a tool in planning and scheduling. The connection of WBS to the stepwise refinement method (SRM) of program development is that both use a top-down approach, where a task is successively decomposed into disjoint subtasks until we arrive at subtasks that can be readily solved. Because WBS is not concerned with the control-logic (if-then-else and loops), it is simpler than SRM where one develops subtask decomposition in parallel with the refinement of control-logic in the form pseudocode. This makes WBS a good intermediate step and one can add the control-logic later in a bottom-up or top-down fashion at each subtask-decomposition step. We can say that WBS helps to operationalize SRM in the same way that SCRUM software development method helps to operationalize agile programming. Both SRM and WBS can accommodate refinements of data, operation, and control-logic. There have been many advances in SRM-based formal techniques for program development [5-7], but these techniques are not suitable for teaching undergraduate-level programming.

We limit ourselves here to non-OO programming. Indeed, one cannot create a high quality OO-program, which involves the added complexity of class-subclass considerations, if one cannot create a small high-quality non-OO program. For the present discussion, a high quality program means simple and clear logic in the basic algorithm, clean implementations for good computational efficiency and memory usage, and simple and clear input/output interfaces.

A. *Work-Breakdown Structure*

We build a WBS in a top-down fashion and represent it as a tree, where the children of a node shows the decomposition of the node into two or more subtasks. The terminal nodes, called the *work-units*, represent the actual tasks performed in solving the original problem given by the root of WBS-tree. The intermediate nodes represent a hierarchical grouping of work-units into larger conceptual "chunks" and they help us in arriving at the work-units. Creating a WBS-tree is often a non-trivial task and requires much thought and insight into the problem at hand. Although, in principle, a node with $n \geq 3$ children can be replaced by a chain of $n-1$ nodes each with 2 children, this is not always easily done and nor it is advisable to force this because it can lead to artificial subtasks. A WBS represents a solution approach in a more abstract form than a pseudocode because it does not show the control-logic. The simultaneous development of control-logic and the bodies of loops and the then/else parts of if-statements makes SRM harder to apply than our two-step method: build a WBS first and then add the control-logic.

B. Two Demonstration Problems

We consider two problems to show the usefulness of WBS in developing complex pseudocodes and their efficient implementations. The solution of the first problem, the triangle classification (TRC), involves a complex nested if-then-else and the solution of the second problem, the constrained binary string generation (CBSG), involves several interacting pairs of loops and if-statements. In both cases, we want simplest logic and the maximum efficiency. The use of WBS helps to create a proper high-level design for the solution algorithm, and then a proper choice of data-structures leads to an efficient implementation. The importance of the CBSG problem is that many other problems can be either formulated in this form or can be solved by generalizing or modifying the method used for solving CBSG. For example, we can associate the subsets of size m of a set of size n with the binary strings of length n with m ones and thus we can generate those subsets using the solution of the CBSG problem. As another example, we can generate all permutations of $n \geq 2$ items by modifying the method for solving CBSG.

Sections II and III present the solutions of the TRC-problem and the CBSG-problem. Some important coding techniques are described in Section IV. Section V gives the conclusion and the future work.

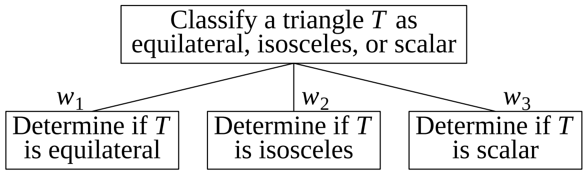
II. TRC-PROBLEM

Here, we want to classify a triangle T as one of equilateral, isosceles, and scalar given the lengths a, b, and c of its sides which are assumed to satisfy the triangle inequalities: $a + b > c$, $a + c > b$, and $b + c > a$. Table 1 shows the conditions for classifying T. Note that we don't need the condition "$a = c$" in C_1 and likewise the condition "$b \neq c$" in the first and-combination of C_2, etc.

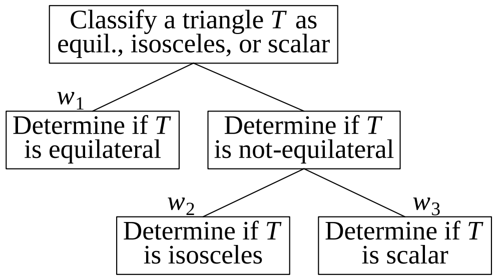
Table 1. Conditions for triangle classification.

Type	Condition
Equilateral	C_1: $(a = b) \wedge (b = c)$
Isosceles	C_2: $[(a = b) \wedge (a \neq c)] \vee [(a = c) \wedge (a \neq b)] \vee$ $[(b = c) \wedge (a \neq b)]$
Scalar	C_3: $(a \neq b) \wedge (a \neq c) \wedge (b \neq c)$

Figures 1(i)-(ii) show two WBSs for the TRC-problem, each with three work-units w_1 to w_3. There are two more WBSs similar to Figure 1(ii) based on decomposing the root with respect to isosceles triangle and scalar triangle. Clearly, the WBSs for any classification problem can be mapped in an one-to-one fashion to the classification trees for that problem if we do not include additional computation details in the WBSs for the classification problem.


Classify a triangle T as equilateral, isosceles, or scalar
w_1
w_2
w_3
Determine if T is equilateral
Determine if T is isosceles
Determine if T is scalar

(i) A simple WBS.


Classify a triangle T as equil., isosceles, or scalar
w_1
Determine if T is equilateral
Determine if T is not-equilateral
w_2
w_3
Determine if T is isosceles
Determine if T is scalar

(ii) A multi-level WBS with more structure.

Figure 1. Two alternate WBS for the TRC-problem.

Figure 2 shows several alternate pseudocodes for the TRC-problem. The one in Figure 2(i) comes directly from the WBS in Figure 1(i) and is the least efficient; it does not take advantage of the disjointness of conditions C_1-C_3. The pseudocode in Figure 2(ii) is from [3]; it is little better than the one in Figure 2(i) but it fails to fully exploit the relationships among the equalities and inequalities in C_1-C_3. Figure 2(iii) is obtained from Figure 2(ii) by replacing C_3 by its negation (thereby avoiding negations like "$\neq$") and interchanging the associated then-else parts; this is better than using C_2 in terms of the number of evaluations of various "=" and "$\neq$". Figure 2(iv) gives the most efficient and elegant solution. In terms of E = the average number of evaluations of "=" or "$\neq$", it has the minimum E = 12/5. If we are given the additional information that $a \leq b \leq c$, then Figure 2(v) is the most efficient solution.

Table 2 shows the number of boolean conditions like "$a = b$" or "$a \neq b$" evaluated for different triangle types for the pseudocode in Figure 2(iv). Here, the notation T(5) in column "$b = c$" means the condition "$b = c$" evaluates to T (true) in line 5 of Figure 2(iv). This gives E = (2+2+2+3+3)/5 = 12/5. A similar analysis of the pseudocodes in Figures 2(ii)-(iii) gives E = 16/5, which is 30% higher than 12/5. The pseudocode in Figure 2(i) has E = 35/5. The pseudocode in Figure 2(v) is a slight simplification of that in Figure (iv) and has E = 9/4; the only two cases of isosceles triangles are now: "$(a < c) \wedge (a = b)$" and

"$(a < c) \wedge (b = c)$". Since sorting a, b, c takes at least 2 comparisons, Figure 2(iv) is still the best and many students fail to obtain this solution.

Table 2. Computation of $E = 12/5$ for Figure 2(iv).

Triangle	Conditions			Number of
Type	$a = b$	$a = c$	$b = c$	evaluations
Equilateral	T(1)	T(2)		2
Isosceles: case ($a = b$)	T(1)	F(2)		2
case ($a = c$)	F(1)	T(5)		2
case ($b = c$)	F(1)	F(5)	T(5)	3
Scalar	F(1)	F(5)	F(5)	3

```
if C1 then output("equilateral")
if C2 then output("scalar")
if C3 then output("isosceles")
```

(i) A very inefficient pseudocode.

```
if C1
then output("equilateral")
else if C3
then output("scalar")
else output("isosceles")
```

(ii) A better but not-so-good pseudocode.

```
if (a=b) AND (a=c)
then output("equilateral")
else if (a=b) OR (a=c) OR (b=c)
then output("isosceles")
else output("scalar")
```

(iii) An slightly better variant of (ii), with the nested then/else parts interchanged.

```
if (a=b)
   if (a=c)
   then output("equilateral")
   else output("isosceles")
else if (a=c) OR (b=c)
then output("isosceles")
else output("scalar")
```

(iv) A more efficient variation of (iii).

```
if (a=c) //assuming a <= b <= c
then output("equilateral")
else if (a=b) OR (b=c)
then output("isosceles")
else output("scalar")
```

(v) Another variation, assuming $a \le b \le c$.

Figure 2. Several alternate pseudocodes for the TRC-problem having different efficiency in terms of E.

III. CBSG-PROBLEM

Let $B(n, m)$ = the set of binary strings of length $n \ge 1$ and having m ones, $0 \le m \le n$. We want to generate the strings in $B(n, m)$ one by one in lexicographic order, say, to avoid generating a string more than once and missing some strings altogether. The lexicographic order $s < s'$ between two strings $s, s' \in B(n, m)$ means s represents a smaller integer than s'. For $B(4, 2)$, the lexicographic ordering is: 0011 < 0101 < 0110 < 1001 < 1010 < 1100 and thus we want to generate these strings in that order.

A. WBS for CBSG problem

Figure 3 shows a WBS for the CBSG problem, with work-units w_1 to w_4. The use of "leftmost" in w_2 is critical in two ways: (1) we sometimes cannot determine the "rightmost" position of change in s without determining the leftmost position of change, and (2) the corresponding new w_3 and w_4 do not give us any computational advantage. The leftmost position of change in s is the position of '0' in the rightmost "01" in s. If there is no "01" in s, then there is no string $s' > s$ and $s = 1^m 0^{n-m}$, the last string in $B(n, m)$. In this case, w_3 and w_4 will not be done. (This illustrates an important point about WBS: the child-nodes of a node do not always form an and-decomposition; they have to be, however, disjoint and one or more of them must always suffice to complete the parent-task. A program that implements a WBS must thus contain code for each child-subtask of a node because for some input situations the work for that child-subtask has to be done.)

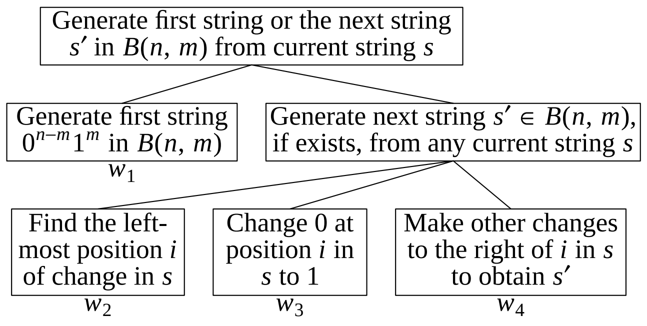

Generate first string or the next string s' in $B(n, m)$ from current string s
Generate first string $0^{n-m}1^m$ in $B(n, m)$
w_1
Generate next string $s' \in B(n, m)$, if exists, from any current string s
Find the left-most position i of change in s
w_2
Change 0 at position i in s to 1
w_3
Make other changes to the right of i in s to obtain s'
w_4

Figure 3. A WBS for CBSG problem.

B. From WBS to Pseudocode

To transform a WBS into a pseudocode with "stubs" for the work-units, one may need to add a good amount of control logic. We may also need to choose parameters of the functions for the main-task (root node of WBS) and some other tasks, and add some variables for use in the control-logic. Building a WBS first makes building a pseudocode easier. Figure 4 shows the pseudocode for the WBS in Figure ; "stubs" for the work-units are shown in bold. We use n and m instead of "length" and "numOnes" because of the small column-width of the printed text. The nextBinString-function returns NULL if there is no next binary string; otherwise, it returns a pointer to the binary string generated. The static-variable binString holds the most recent string (a beginning programmer may make binString a global variable instead).

```
char *nextBinString(int n, int m)
{ static char *binString = NULL;
  if (!binString) {
     allocate storage for binString;
     generateFirstBinString; }
  else {
     find leftmost position i of change in binString;
     if (i not found) binString = NULL;
     else { change 0 at position i to 1;
            make other changes in binString;
          }
  }
  return(binString); //NULL, if no next string
}
```

Figure 4. A pseudocode for WBS in Figure 3.

Figure 5 shows a pseudocode for generating all strings in $B(n, m)$. We can start a new cycle for strings in $B(n, m)$, perhaps with one or both of n and m changed, and call nextBinString(n, m) repeatedly once the previous cycle ends, indicated by the return-value NULL.

```
void genAllBinStrings()
{ char *binStr;
  int maxLength = 100;
  prompt("Enter string length n >= 1
          and <= maxLength");
  read(n);
  prompt("Enter number of 1’s m >= 0
          and <= length");
  read(m);
  do { binStr = nextBinString(n, m);
       if (binStr) write(binStr);
  } while (binStr);
}
```

Figure 5. A pseudocode to generate all strings in $B(n, m)$.

We now present three ways of improving the efficiency of computing the next binary string s' from s.

C. Implementation/optimization issue: IS.1

A close look at the successive strings in $B(n, m)$ shows to get the next string s' from an s most of the time the changes to s are made to few of its rightmost bits. This means the search for the rightmost "01" in s should be done from the right to left. A naive code to find the position of ’0’ in rightmost "01" in s is shown below.

```
for (i=n-2; i>=0; i--)
    if ((’0’ == s[i])&&(’1’ == s[i+1]))
       break;
```

The problem with this code is that it looks at some of the trailing 0’s in s more than once. For example, each of the three underlined zeros in $s = 00111\mathbf{0}111110\underline{000}0$, where the desired ’0’ in s is shown in bold, are looked at twice by the above code. A better way to find the rightmost "01" in s is shown below; it looks at the same positions in s as before but they are now looked at only once. In both cases, if $i \geq 0$ on termination of the for-loop(s), then it gives the position of ’0’ of the rightmost "01" in s.

```
for (i=n-1; i>0; i--) //find rightmost ’1’
    if (’1’ == s[i]) break;
for (i=i-1; i>=0; i--) //skip preceding 1’s
    if (’0’ == s[i]) break
```

This code is an example of a general principle "Do A Little and Take Advantage of it" (in short, DALTA, to be read as "delta"). Another applications of this principle was in Figure 2(iv) in the construction of nested if-then-else. To force the better implementation of w_2 shown above, we decompose w_2 in Figure 3 as shown below.

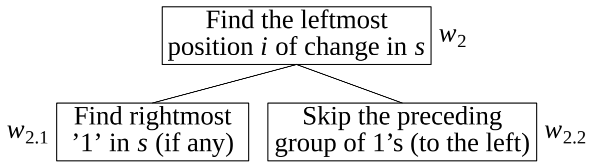

Find the leftmost position i of change in s
w_2
$w_{2.1}$
Find rightmost ’1’ in s (if any)
Skip the preceding group of 1’s (to the left)
$w_{2.2}$

Figure 6. A decomposition of w_2 in Figure 3.

D. Implementation/optimization issue: IS.2

A further analysis of the relationship between a string $s \in B(n, m)$ and its next string s' shows that we can avoid the for-loop for skipping the ending group of 0’s in s (if any) and, moreover, we need the for-loop to skip the preceding group of 1’s only in one case. For this, we keep track of $z(s)$ = size of the ending group of 0’s in s, $k(s)$ = size of the preceding group of 1’s in s (which is the same as the rightmost group of 1’s), and update them as follows:

Case $k(s) = 1$:	$z(s') = z(s) + 1$, $k(s') \geq 1$.
Case $k(s) > 1$:	$z(s') = 0$, $k(s') = k(s) - 1$.

E. Implementation/optimization issue: IS.3

Now, consider the work-unit w_4 in Figure 3. First, we change ’1’ of the rightmost "01" (at position $i + 1$) in s to ’0’. Next, we move the remaining $(k(s) - 1)$ many 1’s of the rightmost group of 1’s to the extreme right in s and bring the ending groups of 0’s in s (if any) to the left of these 1’s. A naive approach for this would be to interchange left half of the last $n - i - 2 = k(s) + z(s) - 1$ positions in s with its right half. But a slightly more efficient method is to simply interchange $p = \min\ \{k(s) - 1,\ z(s)\}$ rightmost items in s with p leftmost items among the last $n - i - 2$ positions. Thus, for $s = \cdots\mathbf{01}\underline{111}0000\underline{000}$, with $k(s) = 4$, $z(s) = 7$, and $s' = \cdots\mathbf{10}\underline{000}0000\underline{111}$, we need to interchange only $p = 3$ positions (shown underlined in s) among the last $10 = 4 + 7 - 1$ positions, instead of interchanging $10/2 = 5$ positions. Since the rightside p items will be made ’1’ and leftside p items will be made ’0’, we don’t need to read these $2p$ items for the exchange.

F. Final code and its performance

Figure 7 shows a complete *C*-code for nextBinString-function based on Figure 4 and the optimizations in (IS.1)-(IS.3). This corresponds to version V.1 in Table 3; Table 3 also briefly describes three other versions. Table 4 shows some performance data from the actual measurements of average (over 5 runs) execution time for generating the strings in $B(n, m)$, without printing. Note that $|B(n, m)| = C(n, m)$ is largest when $m = n/2$. The version V.1 has the best performance. The left-to-right search for rightmost "01" in V.4 contributes most to its inefficiency. The recursive version V.3 has no search for "01" and the issues (IS.1)-(IS.3) are not relevant; it is less efficient than V.2 due to the cost of recursive calls. In V.3, we successively fill the positions $i = 0, 1, \cdots, (n-1)$ in the binary string in that order as follows using two recursive calls: fill position i by ’0’ (if $m < n$) and call nextBinString($n-1$, m) to fill the remaining positions on right, and fill position i by ’1’ (if $m > 0$) and call nextBinString($n-1$, $m-1$) to fill the remaining positions on right.

Table 3. Brief description of versions V1 to V4.

V.1:	Searches the rightmost "01" from right to left and adopts optimizations for (IS.1)-(IS.3); see Figure 7.
V.2:	Searches the rightmost "01" from right to left but does not adopt optimization for (IS.1)-(IS.3).
V.3:	Uses recursion to generate strings in $B(n, m)$.
V.4:	Searches the rightmost "01" from left to right and does not adopt optimization for (IS.1)-(IS.3).

Table 4. Aver. #(ticks over 5 runs in generating all strings in $B(n, m)$; a tick = 1/128 sec. The numbers in parentheses give aver. #(accesses to items of a string in $B(n, m)$) in search of the rightmost "01", if any.

Version	$n = 20$ $m = 10$	$n = 30$ $m = 10$	$n = 30$ $m = 15$	$n = 30$ $m = 20$
V1	0.6	83.6	419.6	93.6
	(1.0)	(1.0)	(1.0)	(1.0)
V2	0.8	156.0	739.8	164.0
	(4.2)	(5.4)	(4.3)	(4.4)
V3	0.8	168.2	783.6	172.2
	(3.8)	(4.3)	(3.9)	(4.3)
V4	3.8	895.4	4578.2	823.4
	(28.5)	(48.3)	(43.5)	(38.7)

IV. SOME ELEGANT CODING TECHNIQUES

We have seen in Sections II-III that going from an algorithm to an efficient implementation is a non-trivial task. We now briefly describe a few techniques to go a

```
#include<stdio.h>

char *nextBinString(int length, int numOnes)
{ static char *binString;
  static int i, //posn of ’0’ in rightmost "01"
                //= -1 if there is no "01"
             k, //size(rightmost group of 1’s)
             z, //size(group of 0’s at length-1)
             firstCall=1;
  int j, min;
  if (1 == firstCall) {
     firstCall = 0;
     binString = (char *)malloc((length+1)*
                                sizeof(char));
     for (j=length-1-numOnes; j>=0; j--)
         binString[j] = ’0’;
     for (j=numOnes; j>0; j--)
         binString[length-j] = ’1’;
     i = length - 1 - numOnes; //= -1 if numOnes
                               //= length
     k = numOnes; z = 0;
  } else if ((-1 == i) || (0 == k)) {
            free(binString); binString = NULL;
            firstCall = 1;
  } else { //this part may set i = -1 or k = 0
           binString[i] = ’1’;
           binString[i+1] = ’0’;
           k--;
           if (k > 0) { //move 1’s to right; set
                        //i, z for new binString
              if (z < k) min = z;
              else min = k;
              for (j=0; j<min; j++) {
                  binString[length-1-j] = ’1’;
                  binString[i+2+j] = ’0’;
              }
              i = length - 1 - k; z = 0;
           } else //no move of 1’s needed; set
                  //i, k, z for new binString
                  { z = length - 1 - i;
                    for (i=i-1; i>=0; i--)
                        if (’0’ == binString[i])
                           break;
                    k = length - 1 - i - z;
                  }
         }
  return(binString);
}
```

Figure 7. Final code for nextBinString-function.

step further in creating a high quality elegant code, without sacrificing the efficiency, simplicity, and clarity. This involves some post-processing (cleaning) of the code, a step often ignored by students. One such technique is code-folding; we describe below two types of code-folding. See [1][4] for other techniques of good programming. In Section III.C, we have seen an example of the opposite process "code-unfolding", where we replaced a loop by two loops to reduce computation.

A. Folding nested if-then-else

We often see student-codes that do not make proper use of else-statements. For example, the two if-statements "if (x > y) x = y; if (x <= y) y = x" can be simplified to "if (x > y) x = y; else y = x" to avoid the unnecessary test "x <= y". An extreme case of code-folding is replacing an if-then-else statement by a simple statement as in replacing "if (0 == x) y = x; else y = 2*x" by "y = 2*x". Figures 8(i)-(ii) show two slightly more complex cases of code-folding, where the code on the right gives the simplified form. The best solution to the TRC-problem in Figure 2(iv) uses both folding and unfolding of if-conditions.

```
(i)  if (x)                 if (x && y)
        if (y) s = t;          s = t;
        else u = v;         else u = v;
     else u = v;

(ii) if (x)                 if (x && !y)
        if (y) u = v;          s = t;
        else s = t;         else u = v;
     else u = v;
```

Figure 8. Nested if-then-else simplified by code-folding.

B. Folding else-part

Given an array of non-zero numbers, suppose we want to compute posCount = #(positive items) and negCount = #(negative items). Figures 9(i)-(ii) show two inelegant student-codes for this problem. The first one ignores that the array-items are non-zero, and both of them ignore the property "posCount + negCount = #(items)".

```
(i)   for (posCount=i=0; i<n; i++)
          if (nums[i] > 0) posCount++;
      for (negCount=i=0; i<n; i++)
          if (nums[i] < 0) negCount++;

(ii)  for (posCount=i=0; i<n; i++)
          if (nums[i] > 0) posCount++;
          else negCount++;
```

Figure 9. Two inelegant solutions to a simple counting problem, where each nums[i] $\neq 0$.

The second solution is more efficient than the first, but the best solution shown below is obtained by the DALTA-principle. It is missed even by some graduate students.

```
for (posCount=i=0; i<n; i++)
    if (nums[i] > 0) posCount++;
negCount = n - posCount;
```

C. Folding or merging of loops

Consider computing the variance V of nums[i], $0 \leq i < n$. To use the formulas $V = [\sum_{i=0}^{n-1}(\text{nums}[i] - a)^2]/n$, where $a = (\sum_{i=0}^{n-1} \text{nums}[i])/n$, we need two separate loops. If we use the formula $V = (\sum_{i=0}^{n-1} \text{nums}[i^2]) - a^2$, then a novice programmer may also use the two loops shown below to compute $\sum_{i=0}^{n-1} \text{nums}[i]^2$ and a.

```
for (sum=i=0; i<n; i++)
    sum += nums[i];
for (sumOfSquares=i=0; i<n; i++)
    sumOfSquares += nums[i]*nums[i];
```

A better solution is to merge the two loops. This code is shorter and runs a little faster; it performs half as many tests "$i < n$" and half as many increments "i++".

```
for (sum=sumOfSquares=i=0; i<n; i++) {
    sum += nums[i];
    sumOfSquares += nums[i]*nums[i];
}
```

V. CONCLUSION AND FUTURE WORK

We have presented here a new approach to teaching undergraduate-level programming by using the notion of work-breakdown structure (WBS) as an intermediate step in applying the well-known stepwise-refinement method (SRM). Our four-step approach consists of: (1) creating a WBS of tasks in solving the problem, (2) adding control logic at various levels of the WBS to build a high-level pseudocode, (3) creating an efficient implementation of each work-unit (lowest level tasks in the WBS) based on a detailed analysis and its alternative implementation choices, and finally (4) applying certain code-transformations to obtain a more elegant program without loss of efficiency and logical clarity. Our initial experience in using this approach has been very positive. The students designed more efficient and elegant code using less time and with fewer errors. Our future work will involve a more extensive study of this new approach on a larger student population and extending the new approach to OO-programming.

REFERENCES

[1] J.L. Bentley, *Programming Pearls* (2nd ed.), Addison-Wesley, 2008.

[2] D. Gries, What have we not learned about teaching programming. *IEEE Computer,* Oct. 2006, pp. 81-82.

[3] P.C. Jorgensen, *Software Testing: a craftsman's approach* (3rd ed.), Auerbach Publ., 2008, pp. 22.

[4] A. Hunt and D. Thomas, *The Pragmatic Programmer,* Addison-Wesley, 2000.

[5] V. Preoteasa and R.-J. Back, Invariant diagrams with data refinement, *Formal Aspects of Computing, 24(2012), pp. 67-95.*

[6] R.-J. Back and J. von Wright, *Refinement Calculus: a systematic introduction,* Springer Verlag, 1998.

[7] C. Morgan, *Programming from specifications (2nd ed.),* Prentice-Hall, 1994.

Requirements Engineering: A Process Model and Case Study to Promote Standardization and Quality Increase

Jose Andre Dorigan
Computer Science Department
State University of Londrina
Londrina, PR, Brazil
jadorigan@gmail.com

Rodolfo Miranda de Barros
Computer Science Department
State University of Londrina
Londrina, PR, Brazil
rodolfo@uel.br

Abstract—**In Requirements Engineering, it is very common for requirements to be poorly specified, inconsistent with the client needs or badly written. Based on these problems, this paper presents a model of Requirements Engineering Process for description standardization, through the reuse of words, seeking to improve the specification quality. We present a Case Study to evaluate and identify benefits of its use in an academic software development. Also, a comparative study between processes that deal with requirements quality assurance was developed showing the works difference.**

Keywords-Requirements Reuse; Quality on Requirements Description; Requirements Standardization.

I. Introduction

Requirements are linked to the main problems of software development; correct requirements gathering is one of the most important tasks in software development. In most cases, they do not reflect the real needs of users, because they are incomplete or inconsistent [14][17].

A major difficulty is to ensure that the requirements specification is in accordance with the client ideas [15][18]. Often, there are misinterpretations by the Requirements Engineer (we will use the abbreviation *REng* in this work) or the clients cannot clearly express their real needs [15]. These specification problems create non-standardized and inconsistent requirements.

IEEE standards 830 [4] and 1233a [5] indicate several properties for software and system requirements specification to obtain a good quality level. Some examples of these properties are: how to avoid ambiguity, use of natural language to describe requirements and the possibility of requirements compliance verification.

The process model proposed in this paper is justified on the ideas of requirements description standardization presented in [8]. Utilizing these, we expected a decrease in development time, during the Requirements Specification, an increase in quality description and confirmation of the client needs by validating the requirements created.

Within this scenario, the objective is to propose a model of Requirements Engineering Process to help with description standardization and reuse of words used in requirements description, in order to increase the requirements specification quality and to be compliant with the client needs.

The structure of this paper is as follows: Section II exposes the proposed model of Requirements Engineering Process, in Section III, we present a Case Study to evaluate and validate the process identifying benefits and considerations about its use. Section IV presents the related work regarding processes that deal with requirements quality assurance and a Comparative Evaluation of the proposed process. Finally, in Section V, we present our Conclusion and Future Work.

II. The Proposed Process

In this section, we present the proposed process model to provide assistance to the REng in requirements analysis, specification and validation. The process seeks to increase the description standardization and minimizes the inconsistencies occurrence in requirements.

The process model is divided into three phases: Analysis, Specification and Validation. We also present a description for each phase, showing what is covered in that phase, the goals and its input and output artifacts.

A. *Analysis*

Input Artifacts: Description of client needs.

Description: **<u>1st. Step</u>**: The process begins when the client and the REng interact in iterative and incremental meetings, debating the system requirements. These meetings may be held where the client deems necessary; in most cases they happen at the client's company. Every new meeting resumes the issues discussed and the needs already identified in an incremental manner, until a final consensus is established.

<u>2nd. Step</u>: Next, the REng transcribes the needs, passed by the client, and identified as possible system requirements.

Goals: Previous works show that this is the moment with a strong probability that the information being passed by the client is inaccurate or it does not truly represent the client needs [10][15][18]. So, in this phase, it is extremely important that the REng gets as much information as possible about the client needs for the system, so that he can translate these needs into requirements.

Output Artifacts: Description of needs transcribed by the REng.

B. *Specification*

Input Artifacts: Description of needs transcribed by the REng.

Description: **1st. Step**: Having knowledge about client needs, the REng can identify, or create, what we call General Context. These are words that will identify where the specified requirement will be contained. It is used, for example, for a project developed in various modules, as the requirements of each module will be separated from the General Context identifying them, facilitating future search.

The Specific Context has the same function as the General Context, but makes the area where the requirement will be included more specific. Using the same idea from the example above, in a project with several modules, the Specific Context will point within each module, where the requirements being specified will be contained.

2nd. Step: After the contexts creation, the REng describes into requirement, using words in natural language, the need already identified by the client.

The process model does not use a treatment for synonymous words. Each new word typed, if it has not previously been validated, is classified as new word and not reused, even if it is a synonymous of a word already validated and stored in RHBD.

In Figure 1, we show an example of the beginning of a requirement description. We are also presenting some examples of suggested words.

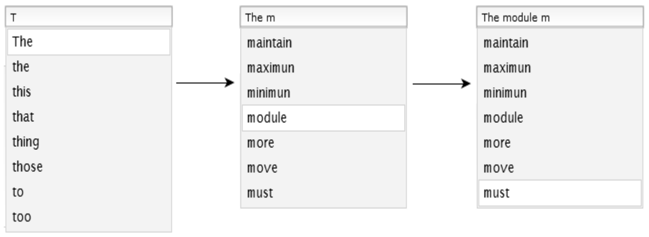


Figure 1: Requirement description example

3rd. Step: Following the description, our process has a requirement classification by functionality. According to [17], they are treated as Functional Requirements and Non-Functional Requirements.

The differential of the proposed process occurs during the 1st. and 2nd. steps of this phase. As the words are described, both in General and Specific Context and in the requirement description, we propose an aid to the REng when the requirement is being described. Words suggestions, previously used in another specification, will be offered to the REng, which he can use or not to continue describing the requirement.

It is important to point out that, even the reuse of words proposed by the model being optional to the REng, if the words are not reused this will reflect in the requirement validation. The expected quality will not be achieved, resulting in a negative validation.

For the proposed model, a requirement will be complete only when connected to a General and Specific Context, its description is complete and classified as Functional or Non-Functional Requirement.

Goals: In this phase, the process has four goals:

- Facilitate the requirements separation, using General and Specific Contexts, based in the [2][3][10] works;
- Clarify ambiguities that may occur in a larger project, when some requirements become much like others;
- Description Standardization, since the REng has available words suggestions;
- Reuse of words, differently of the proposed reuse in [11], our process proposes the reuse of already used words, and points the possibility of partial or total requirement reuse.

Output Artifacts: Requirement Specification.

C. Validation

Input Artifacts: Requirement Specification.

Description: **1st. Step**: According to the rigorousness levels pre-defined by the REng, the description of General and Specific Contexts and the requirement description are evaluated and validated.

The validation occurs together with the client, based on what we call Requirements History Database (RHBD). It contains all the words utilized in any requirements specification in any project within the organization.

When we deal with requirements validation based on RHBD, some care is needed to ensure the new specified requirement, which uses words from others validated requirements, has the desired quality [13]. For this reason, the process model proposes a complete requirement validation (General and Specific Context, Description and Classification), with the client, checking the consistency of all these items.

2nd. Step: If REng and client understand the validation as negative, we encourage the change of items (general and specific context and requirement description) that contains inconsistencies. After these changes are made, a new validation may occur. A negative validation means that the client need was not correctly translated in the form of requirement. This may occur because he has passed incorrect information or the REng cannot correctly identify the requirement.

3rd. Step: If REng and client understand the validation as positive, we propose that all items are stored. They will be in a list we call Requirements List, containing all the requirements identified, and then stored in the RHBD. A positive validation represent that the client was able to read, understand and validate the requirement specified by the REng.

Goals: Validate if the described requirement reflects the needs described by the client, and if it contains the proposed description standardization and expected quality. At this moment, the client may realize the translation of his need, previously expressed, to a requirement for the system. Thus, his analysis, along with the REng, can confirm that it is actually the desired requirement.

Output Artifacts: Requirements List containing all the specified requirements.

We exemplified, in Figure 2, the process model proposed by this paper, with its phases and input and output artifacts.

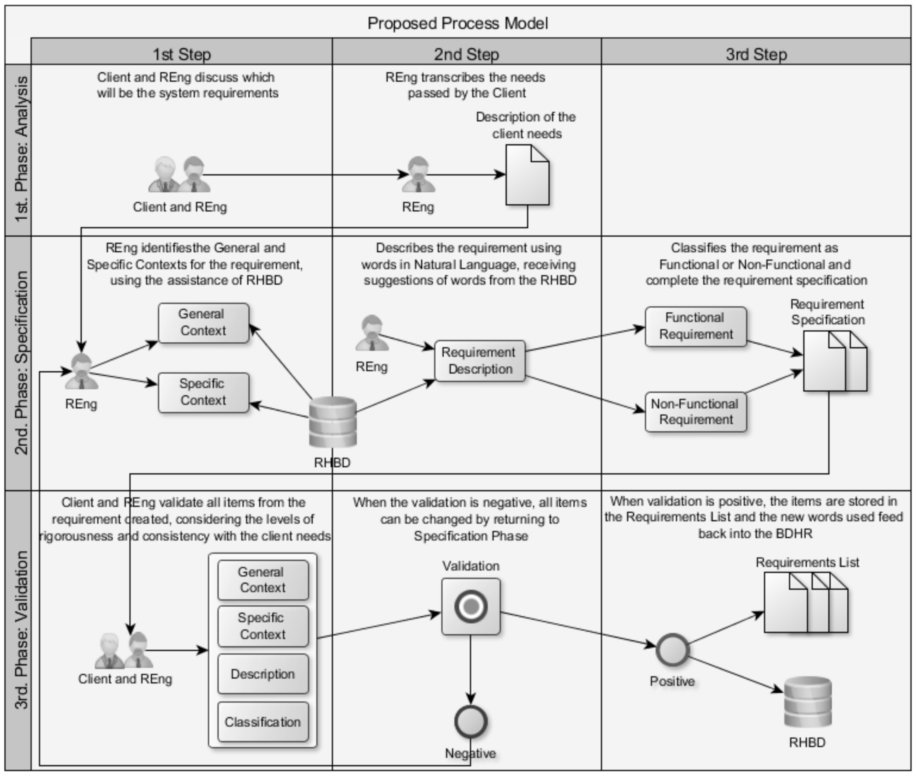

Proposed Process Model
1st Step
2nd Step
3rd Step
1st. Phase: Analysis
2nd. Phase: Specification
3rd. Phase: Validation
Client and REng discuss which will be the system requirements
REng transcribes the needs passed by the Client
Description of the client needs
Client and REng
REng
REng identifiesthe General and Specific Contexts for the requirement, using the assistance of RHBD
Describes the requirement using words in Natural Language, receiving suggestions of words from the RHBD
Classifies the requirement as Functional or Non-Functional and complete the requirement specification
General Context
Specific Context
REng
RHBD
Requirement Description
Functional Requirement
Non-Functional Requirement
Requirement Specification
Client and REng validate all items from the requirement created, considering the levels of rigorousness and consistency with the client needs
When the validation is negative, all items can be changed by returning to Specification Phase
When validation is positive, the items are stored in the Requirements List and the new words used feed back into the BDHR
General Context
Specific Context
Description
Classification
Client and REng
Validation
Negative
Positive
Requirements List
RHBD

Figure 2: Proposed Process Model

At the end of all meetings, after all requirements are defined and validated, it would be possible to create a Software Requirements Document using the Requirements List the proposed process. Our process does not specify a model or a default template, because it is not the intention of this work. We only seek to offer all requirements previously validated and stored, assigned to a software project under development.

III. CASE STUDY

For this case study, we selected two modules of an academic project of the Software Factory – GAIA. It is located in the Computer Science Department at the State University of Londrina. The modules have been developed in partnership with the University Dental Clinic (UDC), an agency owned by the University. The project was to deploy an electronic health record supporting the activities developed in the Odontology Department.

The objective of this case study is evaluate the quality increase in the requirements description, using the process model proposed, and minimizes the occurrence of non-standardized requirements.

We use the concepts presented in [6][16] to report, quantify and evaluate the results obtained during this study.

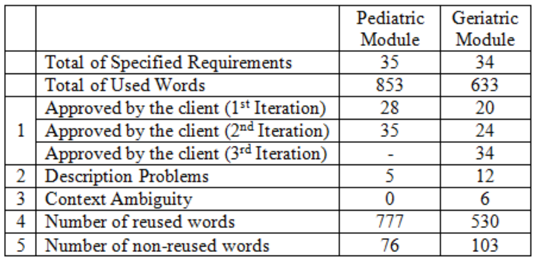
The development team was composed of four graduate students in the role of the implementation team, and two master students in the role of the REng. In this study, the role of the Client was played by teachers responsible for the clinical (disciplines) of Pediatric Dentistry and Geriatric Dentistry attended by UDC.

In this case study, the master students playing the role of REng have only academic experience, so the proposed process was applied without the experience of an expert in the software development industry. However, the analysis and results of the case study were consistent and within the expected ranges.

Both modules were located in the same project, but they have been developed separately, such that the results of the requirements specification could be compared. The Pediatric module used the proposed process and the Geriatric module was developed without the use of the proposed process.

A. Data Analysis

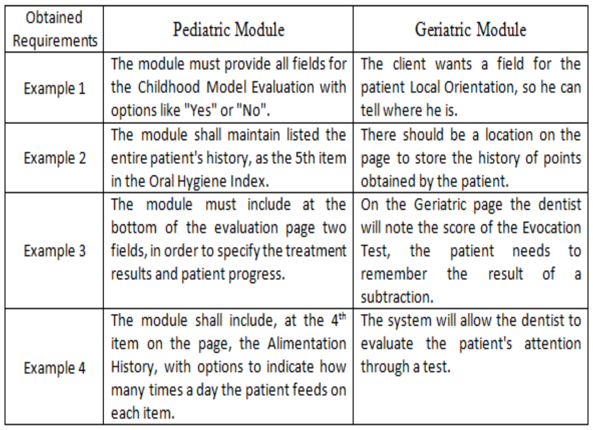
For the data analysis, five items were chosen, reflecting the metrics that describe the data for the two modules. Thus, we could evaluate the effectiveness of the proposed process. In Figure 3 we present these metrics.



		Pediatric Module	Geriatric Module
	Total of Specified Requirements	35	34
	Total of Used Words	853	633
1	Approved by the client (1st Iteration)	28	20
	Approved by the client (2nd Iteration)	35	24
	Approved by the client (3rd Iteration)	-	34
2	Description Problems	5	12
3	Context Ambiguity	0	6
4	Number of reused words	777	530
5	Number of non-reused words	76	103

Figure 3: Quantitative Data Analysis

We present in Figure 4, four examples of requirements for the Pediatric module obtained using the proposed process model, and for the Geriatric module without the use of the model.



Obtained Requirements	Pediatric Module	Geriatric Module
Example 1	The module must provide all fields for the Childhood Model Evaluation with options like "Yes" or "No".	The client wants a field for the patient Local Orientation, so he can tell where he is.
Example 2	The module shall maintain listed the entire patient's history, as the 5th item in the Oral Hygiene Index.	There should be a location on the page to store the history of points obtained by the patient.
Example 3	The module must include at the bottom of the evaluation page two fields, in order to specify the treatment results and patient progress.	On the Geriatric page the dentist will note the score of the Evocation Test, the patient needs to remember the result of a subtraction.
Example 4	The module shall include, at the 4th item on the page, the Alimentation History, with options to indicate how many times a day the patient feeds on each item.	The system will allow the dentist to evaluate the patient's attention through a test.

Figure 4: Requirements obtained in the Case Study

These examples were chosen to demonstrate the results qualitatively. In addition, they have been specified in the first iteration of each module, i.e., some of them have inconsistencies which will be discussed below.

1) *Requirements Approved by the Client*

This metric consisted in the analysis of the entire description of the client needs, specification of these needs in the requirement form, and finally, the validation of the requirements created by the REng and client. In this case study, 2 meetings were necessary until all requirements were validated. The 1st. meeting consisted in one iteration and the 2nd. meeting in two iterations, with total of three iterations as indicated in Figure 3.

Analyzing the data presented, in the first iteration the process achieve 80% of requirements approval against 58% without the process. In the second iteration, the proposed process model has reached 100% of requirements approved against 70%, requiring a third iteration, to achieve 100% of requirements approved.

This metric analysis allows us to identify that using the proposed process model has increased by 22% the number of requirements approved in the first iteration and by 29% the number of requirements approved in the second iteration.

Another important fact is the reduction in the number of iterations in the meetings with the client. This reduction was achieved through the reuse of words based on RHBD, and also the experience of the REng. These make it possible to translate the client needs more quickly, increasing the specification quality.

2) *Description Problems*

Here, we present the number of requirements that have experienced problems in their description. They could be writing errors or requirements that do not consistently represented the need described by the client.

In the requirement of Example 4, Geriatric module, there is an inconsistency which generated a negative validation by the client. The description:

"The system will allow the dentist to evaluate the patient's attention through a test."

presents an error that was corrected in the 2nd iteration, since the client need was:

"The module must allow the dentist to evaluate and record, through a field, the degree of the patient attention, when asked to recall the name of a shown object."

In the same Example 4, now in Pediatric module, the requirement was positive validated in the first iteration. The description:

"The module shall include, as the 4th item on the page, the Alimentation History, with options to indicate how many times a day the patient feeds on each item."

correctly denoted the client need, and have the expected standardization the model proposes.

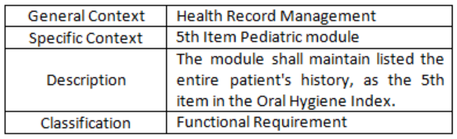
From the 35 requirements specified in the Pediatric module, 5 requirements were identified with some of the problems cited, while in the Geriatric module, the number of problematic requirements was 12 out of 34 requirements. This data indicates a decrease in the amount of requirements description problems by 21% when using the proposed process model.

3) *Context Ambiguity*

This metric was used to evaluate the effectiveness of the process model 2nd Phase. With the identification and allocation of General and Specific contexts for the specified requirement, this became distinguishable from other similar requirements, making it clear where in the project the requirement specified is contained.

To demonstrate the results of this metric we used the requirement from Example 2, which shows how General and Specific contexts helped to minimizes the ambiguity occurrence in the requirements.

In Figure 5, we exemplify how the requirement was treated using the ideas proposed by the process model.



General Context	Health Record Management
Specific Context	5th Item Pediatric module
Description	The module shall maintain listed the entire patient's history, as the 5th item in the Oral Hygiene Index.
Classification	Functional Requirement

Figure 5: Use of Requirement Contexts

In comparison, the same Example 2, but now the specified requirement for Geriatric module presents a context ambiguity inconsistency. In the description:

"There should be a location on the page to store the history of points obtained by the patient."

it is not clear where this requirement is inserted into the project, much less what's the module it belongs to or what would this "location on the page" the client expects. We could easily transport this requirement to another project module with high consistency possibility within this module as well.

There was no context ambiguity in the requirements specified using the proposed process model. However, there were 6 inconsistencies with context ambiguity in the requirements specified in the Geriatric module. This data allows us to identify an improvement by 17% concerning problems of context ambiguity of the specified requirements.

4) Number of reused words

The number of reused words in the requirements description was also chosen as a metric, for evaluating the effectiveness of the proposed process model.

All words that were reused at least once in the requirements specification were identified in this analysis. There were also words initially contained in RHBD, so if these words were used they also are identified as reused.

Another important item is the initial word amount contained in RHBD. For both modules, there were 187 available words in RHBD. As the requirements of Pediatric module were specified using the proposed process model these words were used in 2nd. Phase of the process. In the Geriatric module, which did not use the process, we verified if these words were used.

In the Pediatric module 853 words total were used to specify 35 requirements; 777 words were reused, generating a reuse percentage of 91%. In the Geriatric module, 633 words were used to specify 34 requirements; out of these, 530 words were reused, generating a percentage of 83% reuse. Through these data, we can confirm an increase of 7%, by using the proposed process model.

5) Number of Non-Reused Words

We also use a metric to evaluate the number of non-reused words identified. Thus, we could confirm how the reuse of words affects the requirements description.

To calculate this metric, all words used in the requirements specification have been checked, and if the word was used only once it was identified as not reused.

In the Pediatric module, 76 words were used only once, making 8.9% of the total words. In the Geriatric module, 103 were used only once, making 16% of the total. These data shows a reduction of 7% in non-reused words.

The results of the last metrics are shown in Figure 6.

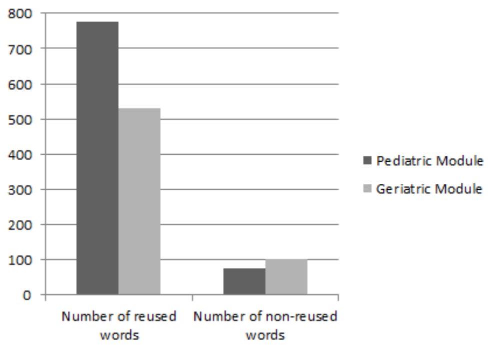


Figure 6: Analysis of Reused and Non-Reused Words

They demonstrate that the relationship is equal to an inverse proportionality, with 7.4% increase in the reuse and 7.3% decrease in non-reuse.

Finally, we emphasize that, this case study objective was to evaluate the quality increase in the requirements description by using the proposed process model, reducing the occurrence of non-standardized requirements.

The requirements specified using the proposed model had a better description, because they were based on words already used and validated by the client, allowing its standardization. The requirements specified without the help of the model were dependent on the REng knowledge and experience, confirming what had already been cited in [14][15][18].

IV. RELATED WORK

In the literature, there are several scientific papers and books dealing with the requirements quality assurance. They point out benefits and provide experiences that help us better understand how to create and maintain a requirements specification with quality.

Thus, through a literature review, three processes dealing with requirements quality assurance were selected. Others could be chosen, which directly address the phases of requirements engineering process, but the purpose of this section is to present processes using divisions in phases, contexts and perspectives in order to improve the requirements treatment receive during their specification.

In their work, Chen et al. [2] described a technique that uses a pre-processing of natural language in software requirements creation. This pre-processing makes use of general and specific fields to separate the requirements. After this, the technique does a search for words, called "objective" by the author, which are described as the central part of the requirement.

According to Cabral et al. [1], the application of systematic reading techniques such as Perspective-Based Reading (PBR) and nonsystematic as Checklist during the requirements analysis has brought good results. In these techniques, several inspectors inspect a software context document looking for errors or inconsistencies before transcribing the requirements document. These errors are then evaluated to compare the two techniques.

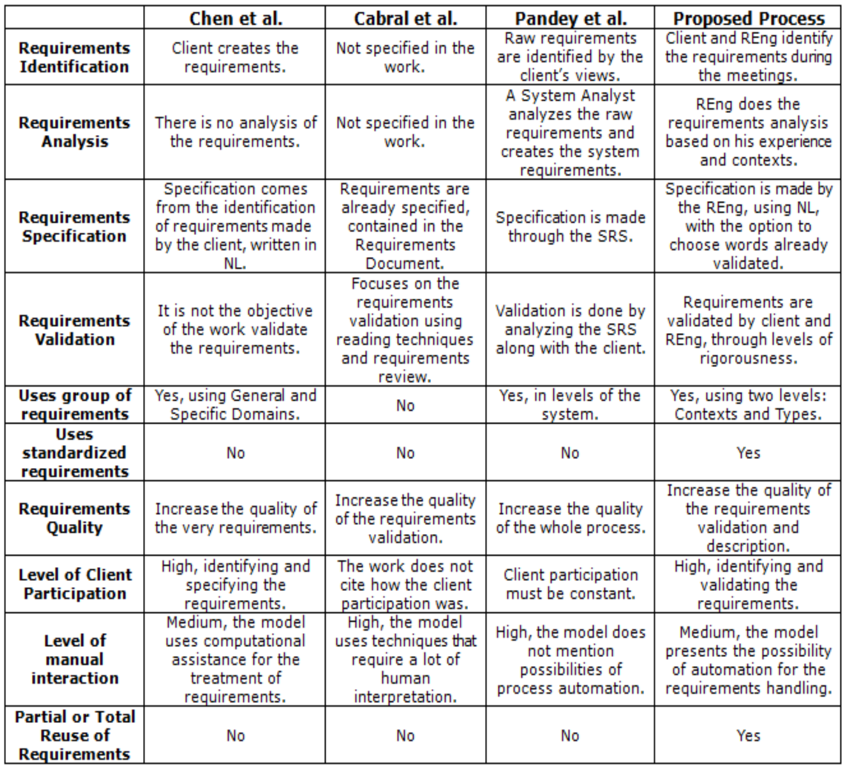
A model of Requirements Engineering Process has been proposed in the work of Pandey et al. [12]. The authors cover the entire area of requirements engineering, proposing the division into four phases: Requirements Elicitation and Development, Requirements Documentation, Requirements Verification and Validation and lastly Requirements Planning and Management. The requirements are stored in a Software Requirement Specification (SRS) and the authors point out that the differential of their work is, besides covering all areas, enabling the Changing Management in requirements already agreed.

A. Comparative Evaluation

This subsection presents a comparative evaluation of the proposed process model and the works discussed. Figure 7 shows this comparison between processes.

It is also important to note that the comparison made sought papers that deal with quality assurance requirements through processes and techniques to achieve a better quality specification.

With this comparison, we want to reinforce the ideas already identified in [8][9], and served as base for creating the process model presented in this paper. These ideas concentrate on the use of specific areas to treat and group requirements, using explicit contexts to prevent inconsistencies and reduce redundancies in the reuse of conflicting requirements. They also present a writing standardization through the reuse of words that will form the requirements.



	Chen et al.	Cabral et al.	Pandey et al.	Proposed Process
Requirements Identification	Client creates the requirements.	Not specified in the work.	Raw requirements are identified by the client's views.	Client and REng identify the requirements during the meetings.
Requirements Analysis	There is no analysis of the requirements.	Not specified in the work.	A System Analyst analyzes the raw requirements and creates the system requirements.	REng does the requirements analysis based on his experience and contexts.
Requirements Specification	Specification comes from the identification of requirements made by the client, written in NL.	Requirements are already specified, contained in the Requirements Document.	Specification is made through the SRS.	Specification is made by the REng, using NL, with the option to choose words already validated.
Requirements Validation	It is not the objective of the work validate the requirements.	Focuses on the requirements validation using reading techniques and requirements review.	Validation is done by analyzing the SRS along with the client.	Requirements are validated by client and REng, through levels of rigorousness.
Uses group of requirements	Yes, using General and Specific Domains.	No	Yes, in levels of the system.	Yes, using two levels: Contexts and Types.
Uses standardized requirements	No	No	No	Yes
Requirements Quality	Increase the quality of the very requirements.	Increase the quality of the requirements validation.	Increase the quality of the whole process.	Increase the quality of the requirements validation and description.
Level of Client Participation	High, identifying and specifying the requirements.	The work does not cite how the client participation was.	Client participation must be constant.	High, identifying and validating the requirements.
Level of manual interaction	Medium, the model uses computational assistance for the treatment of requirements.	High, the model uses techniques that require a lot of human interpretation.	High, the model does not mention possibilities of process automation.	Medium, the model presents the possibility of automation for the requirements handling.
Partial or Total Reuse of Requirements	No	No	No	Yes

Figure 7: Comparison between Processes

This analysis helps us realize that the requirements description standardization, as well as their total or partial reuse are significant issues in the search for quality increase.

Even when studies address requirements quality assurance in a software specification, these items have a very favorable area for research and development.

The proposed process model differs from the works presented by the following:

- In the requirements specification, when it is proposed the suggestion of validated words for the requirements description;
- In the requirements validation, with client and the REng validating the requirements, using levels rigorousness, so the words used and also the requirements created can be reused;
- In the grouping requirements, offering the division into general and specific contexts and also functionality;
- In the description standardization, through the reuse of words, always seeking to achieve a requirements quality increase.

V. Conclusion and Future Work

The description, documentation, and requirements reuse were the main focus of the paper presented. Also, problems encountered in the requirements description, non-standardization and inconsistencies with client needs, were some of the issues that we tried to expose and propose a way to improve.

The proposed process model improves the requirements written standardization and minimizes the chance of description inconsistencies.

The case study showed that it is possible to obtain quantitative and qualitative improvements at the time of specification, corresponding to 22%; in the reduction of requirements description and ambiguity problems, with gains of 21% and 17% respectively; and finally allowing an increase of 7% in the reuse of words that compose the requirements.

As future work, we seek to improve the requirements documentation proposed by the process model, provide an alignment between the proposed process and the Requirements Management, so that, in addition to being treated, requirements can also be managed.

The case study presented allowed the evaluation of the process model using two different modules, which is comparable with a software development. Our next step is to apply the process model in a same module or software, and evaluate the results, obtaining even more data to support the process model effectiveness.

We also seek to develop and use a CASE tool that implements all concepts presented in the process, so we can use computer assistance and further increase the benefits obtained. In addition, we can evaluate the performance of the RHBD when operating with a very large number of words.

References

[1] M. S. Cabral, F. Alencar, J. Castro, O. Pastor, J. Sánches, "Aplicação de técnicas de leitura durante a análise de requisitos", WER08 - Workshop em Engenharia de Requisitos, pp. 193-204, Barcelona, Catalonia, Spain, 12-13 September 2008.

[2] H. Chen, H. Keqing, P. Liang, R. Li, "Text-based requirements pre-processing using nature language processing techniques". International Conference on Computer Design and Applications (ICCDA), pp. 14-18, Qinhuangdao, Hebei, China, 25-27 June 2010.

[3] J. Cybulsky, and K. Reed, "Requirements classification and reuse: crossing domains boundaries". 6th International Conference on Software Reuse, pp. 190-210, Viena, Italy, 2000.

[4] IEEE 830: Recommended Practice for Software Requirements Specification, http://ieeexplore.ieee.org/xpls/abs_all.jsp?isnumbber=15571&arnumber=720574&count=1&index=0, retrieved: July, 2012, 1998 (R2009).

[5] IEEE 1233a: IEEE Guide for Developing System Requirements Specifications, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=741940&contentType=Standards, retrieved: July, 2012, 1998.

[6] B. Kitchenham, et al.: Evaluating guidelines for reporting empirical software engineering studies. Empirical Software Engineering 13 (1), October 2007, pp. 97-121, doi:10.1007/s10664-007-9053-5.

[7] A. V. Knethen, B. Paech, F. Kiedaisch, F. Houdek, "Systematic requirements recycling through abstraction and traceability". RE - Requirements Engineering, pp. 273-281, Essen, Germany, 2002.

[8] G. Kotonya, and I. Sommerville, Requirements Engineering: Processes and Techniques. 1 ed. Wiley, 1998.

[9] W. Lam, T. A. McDermid, A. J. Vickers, "Ten steps towards systematic requirements reuse". Third IEEE International Symposium on Requirements Engineering, pp. 6-15, 1997.

[10] A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications. 1 ed. Wiley, 2009.

[11] B. Moros, C. Vicente-Chicote, A. Toval, "Metamodeling variability to enable requirements reuse". EMMSAD - Exploring Modeling Methods for Systems Analysis and Design, pp. 140-154, Montpellier, France, 16-17 June 2008.

[12] D. Pandey, A. K. Ramani, U. Suman, "An effective requirement engineering process model for software development and requirements management". International Conference on Advances in Recent Technologies in Communication and Computing, IEEE Press, pp. 287-291, 2010.

[13] K. Pohl, Requirements Engineering: Fundamentals, Principles, and Techniques. 1 ed. Springer, 2010.

[14] R. S. Pressman, Software Engineering - A Practitioner's Approach, 7 ed. McGraw-Hill, 2011.

[15] S. Robertson, and J. Robertson, Mastering the Requirements Process. 2 ed. Addison Wesley, 2006.

[16] P. Runeson, and M. Höst, Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14 (2), April 2009, pp. 131-164. doi=10.1007/s10664-008-9102-8.

[17] I. Sommerville, Software Engineering, 8 ed. Addison Wesley, 2007.

[18] I. Sommerville, and P. Sawyer, Requirements Engineering: A Good Practice Guide. 1 ed. Wiley, 1997.

Annotated Component-Based Description for Application Composition

Christian Brel, Philippe Renevier-Gonin, Anne-Marie Pinna-Déry, Michel Riveill
Laboratoire I3S - UMR7271 - UNS CNRS
2000, route des Lucioles - Les Algorithmes
BP 121 - 06903 Sophia Antipolis Cedex – France
{christian.brel, philippe.renevier, anne-marie.pinna, michel.riveill}@unice.fr

***Abstract* – One possible way of developing applications faster is by composing existing applications. In order to support developers this way, we propose a composition approach manipulating both Functionalities and User Interfaces. We present a model of annotation for describing Component-Based applications. By tagging the components with their "ui" and functional concerns, we take into account the UI part of application at a same level as business part. Thanks to such annotations, we define a substitution between components in order to merge controls, inputs or outputs.**

Keywords–application composition; ontology; component-based architecture

I. INTRODUCTION

Nowadays, the trend in software usage is to consume specialized applications. End-users can use the same functionality in several situations, i.e., with several applications. For example, Google Maps is often integrated for geo-localization. In an idealistic way, developers must be able to reuse functionalities without (or with minor) developments. To support developers in their task of combining features from several component-based applications, we contribute towards reducing developers' efforts. We propose an application composition through its User Interface (UI). Our composition preserves the functional linking between components of the applications. Considering a UI as an assembly of components, we explore the composition via their ports. Using the fact that a port can be provided by a component or required by it, we add some annotations about the role the port plays for its attached unit. The added information lets the different components to be combined to obtain a running application.

Section II presents a description of related work. Section III describes the model that an application has to respect in order to be composed. A case study to illustrate our proposed model is shown in Section IV. After the presentation of the composition by substitution in Section V, Section VI details the substitution between two elements. The paper finished with a discussion about our work in Section VII and a conclusion in Section VIII.

II. RELATED WORK

The described problem is naturally related to the state-of-the-art in software composition and UI composition. For software composition, "Composition can be defined as any possible and meaningful interaction between the software constructs involved" according to [5] where a taxonomy of composition mechanisms (e.g., orchestration, aspect oriented programming, etc.) is defined. When the application code is available, solutions, such as aspect oriented programming, are meaningful. On the contrary, when the application code is not available, we can only access to published interfaces and we have to use connectors [7] to perform the composition.

For UI composition, we identify two different approaches. In the first approach, the UI composition is based on abstract description, like in UsiXML [6], in the ServFace project [8], Alias [10] and in Transparent Interface [4]. Those models are defined by XML languages. Final UI are obtained thanks to transformations of those models. In the second approach, the UI composition is based on "UI Components". These ones are reusable high-level widgets, available in repositories. "UI components" are reused by applying design pattern (code level) and detecting pattern of use (UI level). Compose [3], COTS-UI [1], CRUISe [9], WinCuts [12], UI façades [11] and on-the-fly mashup composition [13] illustrate such kind of UI composition.

From the analysis of these works, we note that we can compose the UI (respectively, the functional parts) of former applications, but the other side (respectively, the UI) has to be built again. Moreover, none of these works allows the reusing of former applications with supporting replacement of UI parts. Our goal is to compose applications and in particular their UI, not only by juxtapositions, but also by substitutions between former components of the UI. To obtain a functional application, we also want to preserve former functional links between components of application, in particular between the UI and the business part.

Our proposition is based on applications made of black-box components. We propose a composition model based on roles and ports of those components. The roles are expressed as annotations. The UI are also represented as component assemblies. The composition will be performed by transforming the manipulation on an abstract representation to manipulation on components.

III. APPLICATION MODEL BASED ON PORTS AND ROLES

In order to be compliant with our composition method, the existing applications must follow a clear separation between the functionalities (business part) and the UI (*separation of concerns*). A Component may belong to the two parts. Each Component is described with its ports that we can tag with one of the application concerns, e.g., a port used for a UI concern will be tagged as "UI". Each Component may have required ports (ports required to obtain desirable behavior of another Component) and may have provided ports (ports that Component can provide to other Components). Moreover, each port of a Component must be annotated with a "role" representing the involved behavior of the Component. This role can be Trigger, Input or Output. **Trigger** describes the fact that through its attached port, the Component can call another Component. It can be the button to trigger a particular action or it can be an observable "Component" notifying its observers. **Input** is used to describe a port to get some data. The Component with an Input port can provide data to other Components (like an "input text" in UI or any "Getter" facet of a Component). **Output** is used to describe a port to set some data. The Component can receive data to store or to display (like a "list" or a "label" in UI or any "Setter" facet of a Component). An application to compose must be provided with the annotations of ports of its Components. Those annotations are about roles (trigger, input, output / provided or required) or kind ("UI" or not, i.e., Business). In the remainder of the paper, we use the following acronyms: **rt** - "required-trigger", **pt** - "provided-trigger", **ri** - "required-input", **pi** - "provided-input", **ro** - "required-output" and **po** - "provided-output".

IV. Illustration with a Case Study

We consider two applications:

1. "Movie Theaters", shown in Figure 1, an application displaying movies played in a cinema. There are a text field to entering the name of the cinema (E1), a "get played movies" button (E2) and a display area to list the played movies (E5). The list can also be obtained by validation with "Enter" key in text field (E1).
2. "Cinema Localization", shown in Figure 2, an application displaying the location of a cinema on a map (e.g. using Google Maps). Its UI is also simple: a text field to entering the name of the cinema (E6), a "show cinema place" button (E7) and a map to show the localization of the cinema (E10).

These two applications could be composed in a different way, in order to obtain new applications. A possibility is to have at the same time both the list of the displayed movies and the localization of the cinema. A result of the composition's UI is shown in Figure 3.

V. Composition By Substitution

An application app_i is a set of Components $\{E_n\}$. We define "Ports", the set of ports of a component, and "UsedPorts", the set of used ports:

$\forall Ej \in appi$, Ports(Ej) $=\{ Pn \}$ is the set of the n ports of Ej
UsedPorts(E_j, app_i) $= \{ P_k \}$ is the set of used ports of E_j in app_i

First, we define the role of the ports and their connectivity as the Linkable property, independently of the application in which components are used:

$\forall E_j \in app_{i1}, \forall P_m \in$ Ports(E_j), Role(P_m) $\in$ {pi, po, pt, ri, ro, rt}
isProvided(P_m) $\Leftrightarrow$ Role(P_m) $\in$ {pi,po,pt}
isRequired(P_m) $\Leftrightarrow$ Role(P_m) $\in$ {ri,ro,rt}
$\forall E_k \in app_{i2}, \forall P_m \in$ Ports(E_j), $\forall P_n \in$ Ports(E_k)
We denote r_m = Role(P_m) and r_n = Role(P_n)
Linkable(P_m, P_n) $\Leftrightarrow$ (r_m = ro and r_n = po) or (r_n = ro and r_m = po) or (r_m = ri and r_n = pi) or (r_n = ri and r_m = pi) or (r_m = rt and r_n = pt) or (r_n = rt and r_m = pt)

We define a link between two components through two connected ports in an application as a property Link:

Link((E_j, P_m), (E_k, P_n), app_i) is true if E_j and E_k are linked in a app_i through the ports P_m and P_n.

Such link is possible only if E_j belongs to app_i , E_k belongs to app_i, P_m belongs to UsedPorts(E_j, app_i), P_n belongs to UsedPorts(E_k, app_i) and Linkable(P_m, P_n). For each Component E_j, we define the set Links(E_j, P_m, app_i):

Links(E_j, P_m, app_i) = { (E_k, P_n), $E_k \in app_i$,
$P_n \in$ UsedPorts(E_k, app_i) / Link((E_j,P_m),(E_k,P_n),app_i) }

For all ports, we define a property "*isUIPort*" indicating if the port has a "UI" concern and a function "*isUIPortInApp*" for contextual "UI" concern:

$\forall E_j, \forall P_m \in$ Ports(E_j):

- isUIPort(P_m) $\Leftrightarrow$ isProvided(P_m) and P_m is tagged "UI"
- isUIPortInApp(P_m, app_i) $\Leftrightarrow$ ($P_m \in$ UsedPorts(E_j, app_i) and isUIPort(P_m)) or (isRequired(P_m) and $\exists$(E_k,P_n) / isUIPort(P_n) and Link((E_j, P_m),(E_k, P_n), app_i))

Our composition is made through the construction of a new application, *app$_r$*, initially defined as the union of all former applications:

$app_r = \cup_1^{nb} app_i$ where nb is the number of applications being composed. $\forall E_j \in app_i$:

- $E_j \in app_r$
- UsedPorts(E_j, app_r)= UsedPorts(E_j, app_i)
- $\forall E_k \in app_i, \forall P_m \in$ Ports(E_j), $\forall P_n \in$ Ports(E_k), Link((E_j, P_m), (E_k, P_n), app_r) = Link((E_j,P_m),(E_k,P_n), app_i)
- $\forall P_m \in$ Ports(E_j), isUIPortInApp(P_m, app_r) = isUIPortInApp(P_m, app_i)

The new application *app$_r$* will change with the successive substitutions. A substitution is made between a selection of pairs $\{(E_j, P_m)_i\}$ and a conserved pair (E_k, P_k). We define a "*subst"* function to operate substitution. In a few words, the substitution creates several connectors [7] in order to replace previous links involving substituted pairs by the kept one.

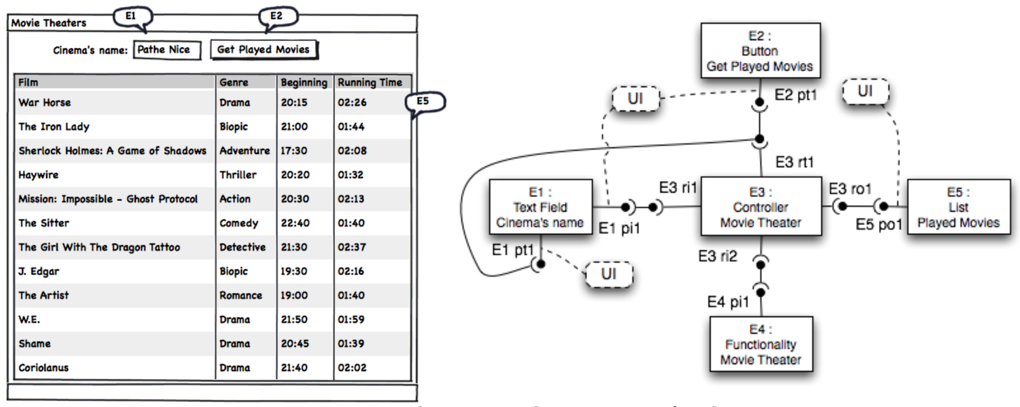
We denote $PreLinks_k$ the value of Links(E_k,P_k,app_r) before the substitution. We denote card($PreLinks_k$) the number of Components in $PreLinks_k$, i.e., the number of Components linked with E_k through P_k.

For each pair $(E_j, P_m)_i$, we denote $PreLinks_i$ the value of Links(E_j, P_m, app_r) before the substitution. We denote card($PreLinks_i$) the number of Components in $PreLinks_i$ i.e., the number of Components linked with E_j through P_m. We denote sel the set of the substituted pairs:

$$sel = \{(E_j, P_m)_i, i \in \{1 \cdots z\}\}.$$
$$\forall (E_j, P_m)_i \in sel, P_m \in UsedPorts(E_j, app_r) ;$$
$$isProvided(P_m) = isProvided(P_k)$$

We denote nk(i) :

- nk(i) = 0 if the substitution doesn't impact previous link with (E_k, P_k) , i.e., the new link and previous links are independent.
- nk(i) = card($PreLinks_k$) if the substitution impacts previous link with (E_k, P_k), i.e., the new link and previous links are dependent (merged).

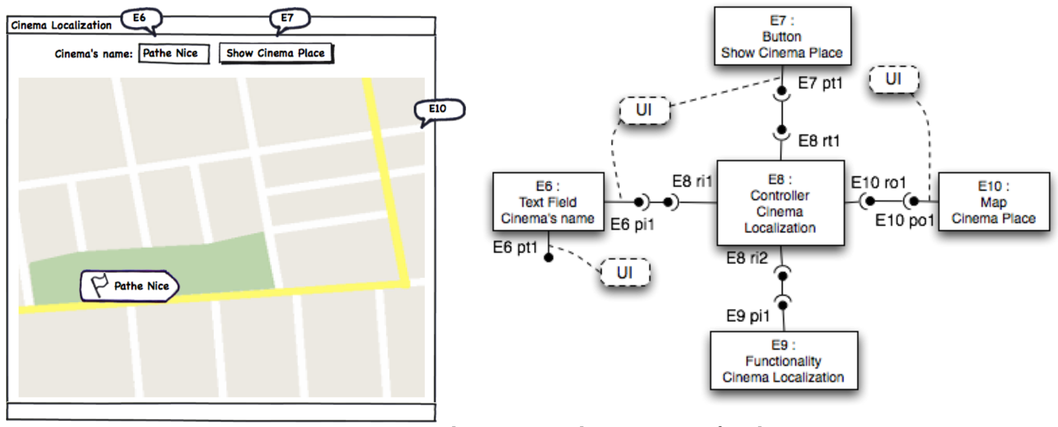

Movie Theaters

Cinema's name: Pathe Nice   Get Played Movies

Film	Genre	Beginning	Running Time
War Horse	Drama	20:15	02:26
The Iron Lady	Biopic	21:00	01:44
Sherlock Holmes: A Game of Shadows	Adventure	17:30	02:08
Haywire	Thriller	20:20	01:32
Mission: Impossible - Ghost Protocol	Action	20:30	02:13
The Sitter	Comedy	22:40	01:40
The Girl With The Dragon Tattoo	Detective	21:30	02:37
J. Edgar	Biopic	19:30	02:16
The Artist	Romance	19:00	01:40
W.E.	Drama	21:50	01:59
Shame	Drama	20:45	01:39
Coriolanus	Drama	21:40	02:02

E1
E2
E5
E2 : Button Get Played Movies
E2 pt1
UI
UI
E3 rt1
E1 : Text Field Cinema's name
E1 pi1
E3 ri1
E3 : Controller Movie Theater
E3 ro1
E5 po1
E5 : List Played Movies
E1 pt1
UI
E3 ri2
E4 pi1
E4 : Functionality Movie Theater

Figure 1. Movie Theater, UI and Components of application.


Cinema Localization

Cinema's name: Pathe Nice   Show Cinema Place

E6
E7
E10
Pathe Nice
E7 : Button Show Cinema Place
E7 pt1
UI
UI
E8 rt1
E6 : Text Field Cinema's name
E6 pi1
E8 ri1
E8 : Controller Cinema Localization
E10 ro1
E10 po1
E10 : Map Cinema Place
E6 pt1
UI
E8 ri2
E9 pi1
E9 : Functionality Cinema Localization

Figure 2. Cinema Localization, UI and Components of application.

If (nk(i) > 0 $\forall (E_j, P_m)_i$) then $PreLinks_k \cap Links(E_k, P_k, app_r) = \emptyset$, i.e., all connectors also replace the previous links involving the conserved pair (E_k, P_k) like in Figure 4. For each Component in *sel*, *UsedPorts*, *Link* and *isUIPortInApp* are impacted by substitution.

subst : $\mathcal{P}(PAIRS) \times PAIRS \rightarrow \mathcal{P}(PAIRS)$

$$subst(sel,(E_k, P_k)) = \{ (Ec_{j,a}, Pm_{j,a})_i \times (Ec_{j,a},Pn_{j,a})_i \times \cup (Ec_{j,a},Pk_y)_i,$$
$$i \in \{1...z\},\ y \in \{1...nk(i)\}, a \in \{1...card(PreLinks_i)\} \}$$
$$\forall (E_j, P_m)_i \in sel, Pm \notin UsedPorts(E_j, app_r)$$
$$\forall (E_j, P_m)_i \in sel, Links(E_j, P_m, app_r) = \emptyset$$
$$\forall (E_j, P_m)_i \in sel, \forall a \in \{1...card(PreLinks_i)\}, Ec_{j,a} \text{ is a new}$$
$$\text{Component of } app_r \ / \{Pm_{j,a},Pn_{j,a}\} \subset Ports(Ec_{j,a}) \text{ and}$$
$$\{Pm_{j,a},Pn_{j,a}\} \subset UsedPorts(Ec_{j,a},app_r) \text{ and } Linkable(P_m,Pm_{j,a}) \text{ and}$$
$$Linkable(Pn_{j,a}, P_k)$$

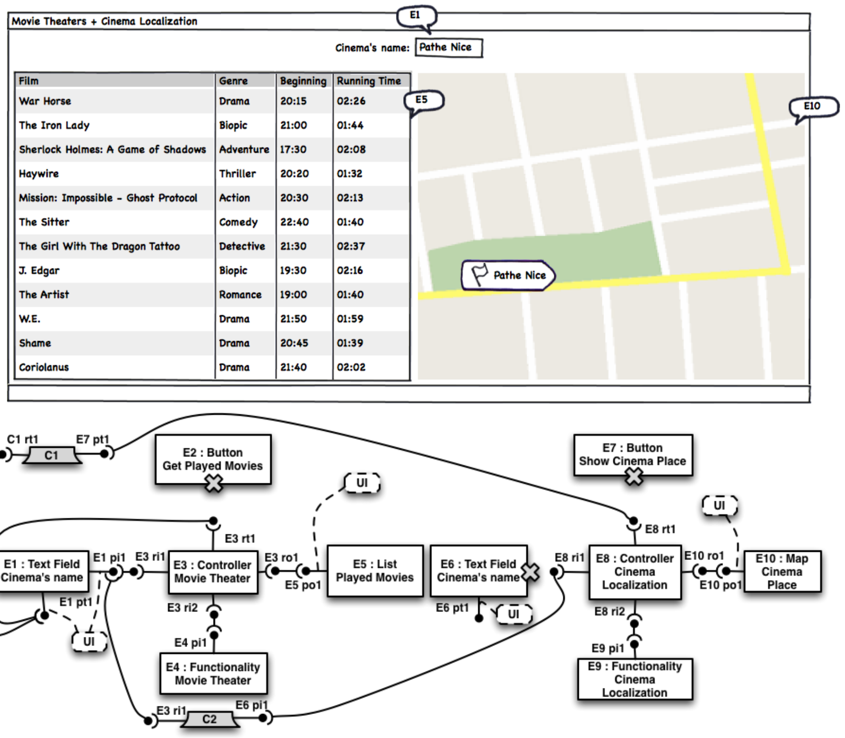
$$\forall Ec_{j,a}, (Ec_{j,a}, Pn_{j,a}) \in Links(E_k, P_k, app_r)$$
$$\forall (E_j, P_m)_i, \exists (Ec_{j,a},Pm_{j,a}) \ / \ Links(Ec_{j,a},Pm_{j,a}, app_r) = PreLinks_i$$
$$\forall (Ec_{j,a},Pm_{j,a}), \exists (E_j, P_m)_i \ / \ Links(Ec_{j,a},Pm_{j,a}, app_r) = PreLinks_i$$
$$\forall (E_j, P_m)_i, nk(i)>0 \Rightarrow \forall y \in \{1..Card(PreLinks_k)\}, Ec_{j,a} \text{ is a new}$$
$$\text{Component of } app_r \ / \{Pk_y\} \subset Ports(Ec_{j,a}) \text{ and}$$
$$\{Pk_y\} \subset UsedPorts(Ec_{j,a},app_r) \text{ and } \exists (E,P) \in PreLinks_k \ /$$
$$(Ec_{j,a}, Pk_y) \in Links(E,P,app_r)$$

As a result, Components no longer involved in links left are removed. The substitution of any pair (E_j, P_m) by a pair (E_k, P_k) is based on the annotations. The role of a port is used to define possible substitutions and the way connectors are used. This is explained in the Section VI. The use of the kind of ports ("UI" or not) is used as following. If before the substitution isUIPortInApp(P_m,app_r) is different from isUIPortInApp(P_k,app_r), then the substitution changes the concern implied in the link, i.e., an input field may be replaced by a data coming from a “Business” Component. That is possible but in such case we could emit a notification.

VI. SUBSTITUTING TWO PAIRS

We now consider substitution between two pairs: a replaced pair and a conserved pair. The function “subst” can replace *n* pairs, but it is just n substitutions performed in parallel. We present the compatibility between the two pairs according to the role of the conserved pair.

In order to perform a substitution between two pairs (Component, Port with a role), we need to add a connector between the substituted pair and the conserved one [7]. Connectors may have several uses: (i) adapting formats of the data or (ii) defining a policy of substitution or (iii) adding a role when the new role makes the Component the “caller”. Thanks to the identification of the Connector and its roles, we can know define the “subst” function for two pairs. Indeed, in Sections VI.A, VI.B, VI.C and VI.D, we describe both the definition domain for two pairs and the results.


Movie Theaters + Cinema Localization

Cinema's name: Pathe Nice

Film	Genre	Beginning	Running Time
War Horse	Drama	20:15	02:26
The Iron Lady	Biopic	21:00	01:44
Sherlock Holmes: A Game of Shadows	Adventure	17:30	02:08
Haywire	Thriller	20:20	01:32
Mission: Impossible - Ghost Protocol	Action	20:30	02:13
The Sitter	Comedy	22:40	01:40
The Girl With The Dragon Tattoo	Detective	21:30	02:37
J. Edgar	Biopic	19:30	02:16
The Artist	Romance	19:00	01:40
W.E.	Drama	21:50	01:59
Shame	Drama	20:45	01:39
Coriolanus	Drama	21:40	02:02

E1
E5
E10
Pathe Nice
C1 rt1
C1
E7 pt1
E2 : Button Get Played Movies
E7 : Button Show Cinema Place
UI
UI
E3 rt1
E8 rt1
E1 : Text Field Cinema's name
E1 pi1
E3 ri1
E3 : Controller Movie Theater
E3 ro1
E5 po1
E5 : List Played Movies
E6 : Text Field Cinema's name
E8 ri1
E8 : Controller Cinema Localization
E10 ro1
E10 po1
E10 : Map Cinema Place
E1 pt1
E3 ri2
E6 pt1
UI
E8 ri2
UI
E4 pi1
E9 pi1
E4 : Functionality Movie Theater
E9 : Functionality Cinema Localization
E3 ri1
C2
E6 pi1

Figure 3. A Result for composition of “Movie Theater” with “Cinema Localization”.

A. *Keeping a Provided Output*

When keeping an output, there is no constraining on the role of substituted ports. By placing a connector before the Component having port playing the Output role, the substitution can be performed. This is a case in the “subst” function where nk(i) > 0 $\forall(E_j, P_m)_i$.

First, the connector may be used to adapt the format of data to display if the substituted role is also Output (a Conversion Connector in [7]) or to define a policy of displaying data if the substituted role is also Output (a mix between a Conversion Connector and a Data Access Connector in [7]). Such policy may be displaying all data, the last received data, etc. Secondly, the connector may also be used to store displayed data and can restitute them when asked if the substituted role is an Input (see Figure 4) (a Data Access Connector in [7]). Thirdly, the connector may also be used to generate an event when the output is updated if the substituted role is a Trigger (an Event Connector in [7]).

In Figure 4, the connector C1 can store displayed data from (E3,ro1) and can restitute them to (E8,ri1) when asked. With that solution, E5 doesn’t need to have a port playing a role of Input, but the Connector has both provided port with Output role for (E3,ro1), required port with Output role for (E5,po1) and required port with Input role for (E8,ri1).

B. *Keeping a Provided Trigger*

As “Trigger” is the only one port’s role that makes the associated Component a “caller”, the role of the port in substituted pair must be also a “Trigger”. We place a connector after the kept “Trigger” for two reasons: (i) adapting the format of the “event” and (ii) defining the policy of the substitution (a mix between an Event Connector and a Procedure Call Connector in [7]). The connector can proceed a sequence between the two triggered actions or put them in parallel etc. This is a case in the “subst” function where nk(i) > 0 $\forall(E_j, P_m)_i$.

C. Keeping a Provided Input

An "Input" cannot replace an "Output" because of the direction of the data. Conversely, an "Input" may replace a "Trigger" (see Figure 5). The connector placed before the kept port can provide on demand (a Data Access Connector in [7]). At the same time, when called, the connector can generate an event and so it can "call" the requiring port (an Event Connector in [7]). The "Trigger" is "on access" (i.e., when the value is got). Of course, an "Input" can replace another "Input". In that case, the connector is used to adapt the provided data to what is expected (a Conversion Connector in [7]). Keeping an "Input" is a case where nk(i) could either be 0 (pi replacing pi) or be greater than 0 (pi replacing a pt).

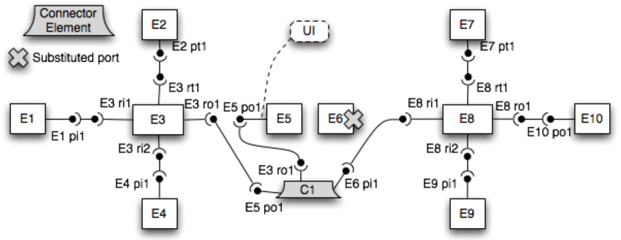


Figure 4 : (E5, po1) replacing (E6, pi1), connector C1 before (E5, po1.)

D. Keeping a Required port

In the "*subst*" function, if P_k is a required port, all substituted pairs must be made of ports with the same role as P_k. Even through a connector, the requirements could not be changed: a setter requirement (required output) could not become a getter requirement (required input). So, even if a connector could have two ports, one "po" and "ri" to be connected to a "pi", inside that connector, the setter used by E_k could not be functionally translated in a getter. The connector could not appropriately exploit the value coming from E_k. Such substitutions are cases where nk(i) > 0 $\forall(E_j, P_m)_i$. All connectors are not only conversion one [7] but they may: (i) merge all input or trigger if Role(P_k) is ri or rt or (ii) call of setter on output if Role(P_k) is ro.

E. Towards Automatic Substitution

From this substitution operator, we can define an operator at a higher level. The objective is to compose two Components from the new application app_r. Based on substitutions between ports of Components, we can define the substitution of two Components. Let E_1 be the removed Component and E_k be the kept Component. For each P belonging to UsedPorts(E_1, app_r), we define:

CompatiblePorts(P, E_k), the set of all possible port P' of E_k for a substitution subst({(E_1, P)}, (E_k, P'))
If isRequired(P) or P = po, CompatiblePorts(P, E_k) = { P' $\in$ Ports(E_k) / Role(P') = Role(P) }
If Role(P) = pi, CompatiblePorts(P, E_k) = {P' $\in$ Ports(E_k) / Role(P')$\in${po, pi} }
If Role(P) = pt, CompatiblePorts(P, E_k) = { P' $\in$ Ports(E_k) / isProvided(P') }

We denote card(CompatiblePorts(P, E_k)) the number of ports in CompatiblePorts(P, E_k). We apply the algorithm PairSelection(P, KeptElements):

Let KeptElements the set of Components used in the substitution. Initially KeptElements = {E_k}.
Let nb_potential_pairs = Σcard(CompatiblePorts(P, E)), E$\in$ KeptElements

If (nb_potential_pairs = 1), (E, P) could be substituted by only one pair is possible. Let E' $\in$ KeptElements / $\exists$P'$\in$ CompatiblePorts(P, E'). The following substitution is computed: subst({(E, P)}, (E', P')}.

If (nb_potential_pairs > 1), one of the ports in CompatiblePorts(P) must be selected. That selection may be by the developer operating the composition or by an external algorithm.

If (nb_potential_pairs = 0), (E, P) could not be substituted by a pair involving a Components from KeptElements.

If KeptElements = app_r, the algorithm finishes without substituting (E, P). Else, we extend the substitution by searching possible ports in Components linked with Components from KeptElements:

ExtendedSelection = { E_j $\in$ app_r / $\exists$E'$\in$ KeptElements / $\exists$ $P_m$$\in$ UsedPorts(E_j, app_r) and $\exists$ $P_n$$\in$ UsedPorts(E', app_r) / Link((E_j, P_m), (E', P_n), app_r) }.
Then we apply PairSelection(P, ExtendedSelection)

At the end of the process, if UsedPorts(E_1, app_r) is empty, E_1 is removed from app_r.

F. Enforcement Of Substitutions On Case Study

The corresponding operations to obtain the case study composition shown in Figure 3 are the substitution of (E7,pt1) - *the button of "Cinema Localization" app* - by (E1,pt1) - *the text field of "Movie Theater" app* - then the substitution of (E6,pi1) - *the text field of "Cinema Localization" app* - by (E1,pi1). There will be only one text entry left E1 and only one button left E2. As the composition finalizes, we can delete button E2 cause of its misspelled action label (see "Discussion" part). So, there is only E1 left to lunch the research because no port is used in E6 and E7. When typing the name of the movie theater, the research could be launched (at each key stroke or only after an "enter").

VII. DISCUSSION

The composition by substitution introduced in this paper needs to be integrated in a larger process as in [1]. This process can include another step to finalize the composition. This finalization can add several classic operators as a delete operator to suppress some links in the final components assembly, as we need to complete our case study described in Section VI. We also add a step to let developer rearrange the various pieces of UI in the new composed UI. We illustrate this in Figure 3 when we position component E10 on the right of component E5. The scalability of our approach and its enforcement on large-scale application rely on the scalability on the ontology engine we use to annotate components. Our composition approach is described with small applications. However, we expect our approach to be appreciated in large-scale applications. *The selection of the different pieces to compose of applications* [1] is improved thanks to the same annotations presented in this paper. With such help during the whole composition process (selection and substitution), the developer may be more efficient during the composition. We plan to test this idea in our future work.

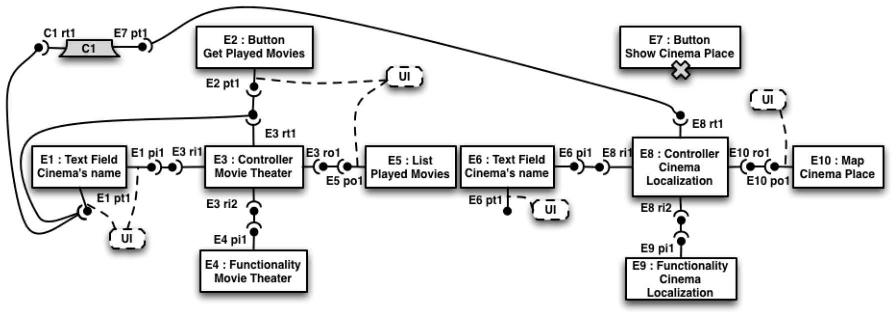

C1 rt1
C1
E7 pt1
E2 : Button Get Played Movies
E2 pt1
UI
E7 : Button Show Cinema Place
UI
E8 rt1
E3 rt1
E1 : Text Field Cinema's name
E1 pi1
E3 ri1
E3 : Controller Movie Theater
E3 ro1
E5 po1
E5 : List Played Movies
E6 : Text Field Cinema's name
E6 pi1
E8 ri1
E8 : Controller Cinema Localization
E10 ro1
E10 po1
E10 : Map Cinema Place
E1 pt1
UI
E3 ri2
E4 pi1
E4 : Functionality Movie Theater
E6 pt1
UI
E8 ri2
E9 pi1
E9 : Functionality Cinema Localization

Figure 5 : (E1, pt1) replacing (E7, pt1), connector C1 between (E1, pt1) and (E8,rt1).

VIII. Conclusion

In this paper we presented a new application composition approach. The challenge is to integrally compose applications, considering both business part and UI part. Our approach is based on description of components constituting the applications. Our model enables substitution of Components coming from former applications, according to their known ports roles. Thus, we can merge controls, inputs or outputs and keeping operational functional links.

Out next challenge is to propose rules for the representation of elements to compose. Indeed an application may have several representations such as its component assembly, its UI or its task model. Our intuition is that to quickly specify a composition, working on the UI is the most adapted. But to ensure consistency of the usability, we will explore the use of task models. And for making complex merge of application, we probably need to manually manipulate links between components. So we want to verify our intuitions and we will study the limits of each approach.

References

[1] Brel C., Pinna-Déry A.-M., Renevier P., and Riveill M. OntoCompo: A Tool To Enhance Application Composition. In Proceedings of the 13th IFIP TC13 Conference in Human-Computer Interaction, Lisboa, Portugal, 2011, pp. 588-591.

[2] Criado, J., Padilla, N., Iribarne, and L., and Asensio, J. User Interface Composition with COTS-UI and Trading Approaches: Application for Web-Based Environmental Information Systems. Communications in Computer and Information Science vol. 111, 2010, pp. 259-266.

[3] Gabillon, Y., Petit, M., Calvary, G., and Fiorino, H. Automated planning for userinterface composition. In Proceedings of the 2nd International Workshop. on Semantic Models for Adaptive Interactive Systems, Palo Alto, CA, USA, 2011, [retrieved: October, 2012].

[4] Ginzburg, J., Rossi, G., Urbieta, and M., Distante, D. Transparent Interface Composition in Web Applications. In Proceedings of the 7th International Conference on Web Engineering, Como, Italy, 2007, pp. 152-166.

[5] Lau K.-K. and Rana T. A Taxonomy of Software Composition Mechanisms. In Proceedings of 36th EUROMICRO Software Engineering and Advanced Application, Lille, France, 2010, pp. 102–110.

[6] Lepreux S., Vanderdonckt J., and Kolski C. User Interface Composition with UsiXML. In Proceedings of 1st Int. Workshop on User Interface Extensible Markup Language, Berlin, Germany, 2010, pp. 141-151.

[7] Mehta N. R., Medvidovic N., and Phadke S. Towards a taxonomy of software connectors. In Proceedings of International Conference on Software Engineering, Limerick, Ireland, 2000, pp. 178-187.

[8] Nestler T., Feldmann M., Preußner A., and Schill A. Service Composition at the Presentation Layer using Web Service Annotations. In Proceedings of the 1st Intl. Workshop on Lightweight Integration on the Web, San Sebastian, Spain, 2009, pp. 63-68.

[9] Pietschmann S., Voigt M., Rümpel A., and Meissner K. CRUISe: Composition of Rich User Interface Services. In Proceedings of International Conference on Web Engineering, San Sebastian, Spain, 2009, pp. 473-476.

[10] Pinna-Déry A.-M., Joffroy C., Renevier P., Riveill M., and Vergoni C. ALIAS: A Set of Abstract Languages for User Interface Assembly. In Proceedings of Software Engineering and Applications, Orlando, FL, USA, 2008, pp. 77-82.

[11] Stuerzlinger W., Chapuis O., Phillips D., and Roussel N. User Interface Façades: Towards Fully Adaptable User Interfaces. In Proceedings of the ACM Symposium on User Interface Software and Technology, Montreux, Switzerland, 2006, pp. 309-318.

[12] Tan D.S., Meyers B., and Czerwinski M. WinCuts: Manipulating Arbitrary Window Regions for more Effective Use of Screen Space. In Proceedings. of ACM Conference on Human Aspects in Computing Systems, Vienna, Austria, 2004, pp. 1525-1528.

[13] Zhao Q., Huang G., Huang J., Liu X., Mei H., Li Y., and Chen Y. A Web-Based Mashup Environment for On-the-Fly Service Composition. In Proceedings. of Symposium on Service-Oriented System Engineering, Jhongli, Taiwan, 2008, pp. 32-37.

Cognitive Engineering meets Requirements Engineering

Bridging the Traceability Gap

Alexandra Mazak
Junior Research Studio Cognitive Engineering (CoE)
Research Studios Austria Forschungsgesellschaft
Vienna, Austria
alexandra.mazak@researchstudios.at

Horst Kargl
SparxSystems Software GmbH
Vienna, Austria
horst.kargl@sparxsystems.eu

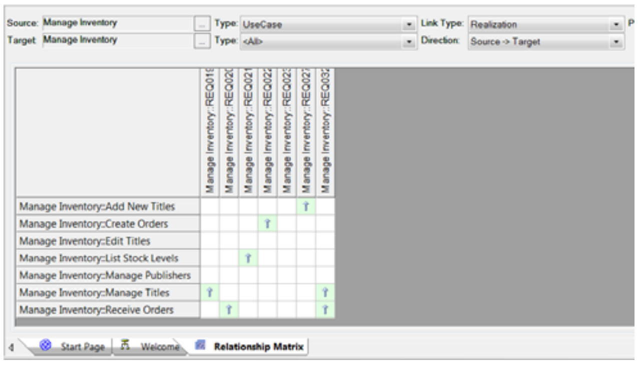
***Abstract*—Support for various stakeholders (customer, project manager, system architect, requirements engineer) involved in the design and management of large software systems is needed since frequently, misinterpretations occur already when specifying customer requirements into system requirements. This problem is mainly caused by the various perspectives and intentions of the involved parties that may lead to diverging interpretations during said process. Therefore, the focus of our work in progress is on the requirements engineers when transforming customer requirements into system requirements. There is still a gap to trace design decisions especially at this early stage of the system development life cycle. We introduce a heuristic-based approach in order to make a contribution to bridge this gap. We propose to consider the requirements engineer's "cognitive perspective" on traceability links by a heuristic-based weighting procedure that can be performed during the design process. We enhance the established relationship or traceability matrix to make it possible for requirements engineers to annotate their informal knowledge to the linkage (i.e., visualized realizations) in that matrix.**

Keywords-Requirements management; requirements traceability; cognitive engineering; traceability matrix; design decisions.

I. INTRODUCTION

There is a variety of stakeholders involved in large software projects, each having a different set of goals and priorities [9]. Generally, requirements are prioritized by stakeholders (e.g., based on the software projects' purpose, certain functionalities which should be targeted by the system). Various methods and supporting tools exist for guiding this process of requirements' prioritization. What we are missing is another prioritization when linking customer requirements to system requirements and system requirements to design artifacts (i.e., model components) in the specification task. Misinterpretations at this stage of a software project are resulting from misconceived or misvalued linking. Validation continues to be a "big pain" due to the lack of trusted documentation of traceability drives validation teams to leave no doubt (aka double or triple effort). There is a multitude of tools (e.g., Rational DOORS [12]) by which requirement lifecycle management is supported. Often, tools produce non- or less compelling documentation [13]. However, experience has shown that a "distributed traceability of requirements" guided by external tools or methods, which means that it is not integrated in the system's architecture design process itself, is often error-prone [13].

We propose focusing on the requirements engineers' informal knowledge when design decisions are made during the specification task of customer requirements. According to the used modeling language a certain traceability link can be used (*realization* in UML [2], *satisfy* in SysML [3]). Generally, a trace chain is constructed through linking by which an impact analysis can be made. Mainly, such information is presented in form of a matrix called *relationship matrix*. In Fig. 1, which represents an excerpt of a relationship matrix from the tool Enterprise Architect (EA) [13], the linkage is visualized in form of arrows in the cells. This type of linking merely express for instance that *Receive Orders* is realized by two system artifacts *REQ019* and *REQ032* (e.g., Fig. 1). It cannot be derived to what extent the realization is proceeded, or the system requirements' level of utility to fulfill this customer requirement. Generally, a certain customer requirement is covered by more than a single system component and not each component has the same relevance to realize that requirement within the system's architecture; just as customer requirements do not have equal prioritizations. Thus, each system component has a different level of utility (relevance).


Source: Manage Inventory
Target: Manage Inventory
Type: UseCase
Type: <All>
Link Type: Realization
Direction: Source -> Target

	Manage Inventory::REQ019	Manage Inventory::REQ020	Manage Inventory::REQ021	Manage Inventory::REQ022	Manage Inventory::REQ023	Manage Inventory::REQ027	Manage Inventory::REQ032
Manage Inventory::Add New Titles						↑	
Manage Inventory::Create Orders				↑			
Manage Inventory::Edit Titles							
Manage Inventory::List Stock Levels			↑				
Manage Inventory::Manage Publishers							
Manage Inventory::Manage Titles	↑						↑
Manage Inventory::Receive Orders		↑					↑

Figure 1. Relationship matrix in EA.

In the early past, methods have emerged by which informal knowledge of the original engineers can be annotated to the model itself, as for example presented in the field of *Ontology Alignment* in [7][8]. There the modelers' intention involved during the design process, that is *modeler's cognition* or *cognitive perspective*, is made visible to users.

Currently, there is no tool support to requirements engineers in order to aid them to make their informal knowledge visible. This open issue motivated us to work on an heuristic-based approach to bridge the traceability gap at the early stage of system design that is of commercial interest, too.

II. BACKGROUND

A. Requirements Traceability

Requirements traceability in all stages of the system development life cycle is an important field of requirements management. Gotel and Finkelstein [4] define requirements traceability as *"the ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases"*. The authors differentiate between *pre-requirements specification (pre-RS) traceability* and *post-requirements (post-RS) specification traceability*. The first refers to those aspects of a requirement's life prior to inclusion in the specification task (e.g., when stakeholders prioritize requirements depending on their expectations they place on the system); whereas the latter refers to those aspects that result from inclusion in the requirement's specification [4].

In our work in progress, we focus on post-RS traceability by which system components and their relations to certain customer requirements are considered. Today, requirements traceability is a key factor in the project management of large-scale systems and it plays an important role in the quality control of software engineering processes [9]. It acts as an indicator to define the system's maturity of development [8]. The aim is to support reliable, up to date, and high-quality traceability—right from the start.

B. Requirements Traceability Matrix

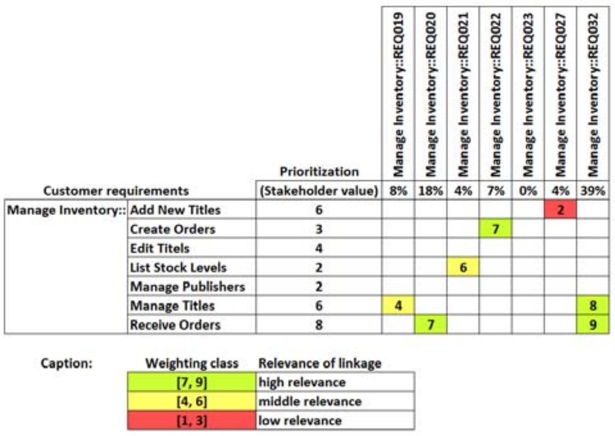
Validation pain can be reduced when using the *traceability matrix* [6]. This is a kind of "completeness indicator" by which the relationships among requirements and artifacts can be traced. It is an aid to determine the complexity of dependencies. There are tools by which a graphical as well as a textual traceability is supported to engineers. For example, in the tool Enterprise Architect the relationship matrix is a spreadsheet display of relationships between different sets of model elements (e.g., Fig. 1). A source package and a target package, the relationship type, and the direction can be defined. All relationships among source and target elements can be identified by highlighting a grid square and displaying an arrow indicating the relationship's direction. The matrix is a convenient method of visualizing relationships quickly and definitively. It also enables users to create, modify and delete relationships between elements with a single mouse click - another quick way to create complex sets of element relationships with a minimum of effort.

Nevertheless, the literature-based research has shown that the traceability matrix is a useful template to trace if a certain requirement is covered by a single or more components. We argue that a simple representation of linkage alone is not enough. For instance, it cannot be derived from the matrix how important a system component is for implementation (i.e., when transforming system components to design artifacts, or when coding) an aspect which is also relevant in agile software development. For instance, in Scrum [5] in order to provide a reference value for the product owner when prioritizing stories for the product backlog.

III. THEORETICAL APPROACH

Requirements engineers decide about how to transform customer requirements into system requirements, i.e., they decide on the realization of customer requirements in the system architecture. Generally, this design step is visualized in the traceability matrix, where the realizations can be traced (e.g., Fig. 1, the customer requirements form the source and the system requirements or components the target nodes). The better this task is guided and monitored, the fewer problems (e.g., over-engineering) will occur during implementation which leads to time and cost savings.

Therefore, we propose to use the engineers' informal knowledge of system design specification by introducing a *cognitive heuristic* in form of a relevance weighting procedure. This procedure is based on an intuitive mental judgement made by the engineers during the design process. We adapt the approach from concepts, which we have introduced in [8]. In our current approach, the requirements engineer determines the importance of a linkage (source $\rightarrow$ target) by annotating the linkage with a weighting in the range of [1, 9]. The weighting is mainly based on his/her intuition (or cognitive perspective [8]) of the system component's relevance in order to cover a certain customer requirement.



Customer requirements		Prioritization (Stakeholder value)	Manage Inventory::REQ019	Manage Inventory::REQ020	Manage Inventory::REQ021	Manage Inventory::REQ022	Manage Inventory::REQ023	Manage Inventory::REQ027	Manage Inventory::REQ032
			8%	18%	4%	7%	0%	4%	39%
Manage Inventory::	Add New Titles	6						🟥 2	
	Create Orders	3				🟩 7			
	Edit Titels	4							
	List Stock Levels	2			🟨 6				
	Manage Publishers	2							
	Manage Titles	6	🟨 4						🟩 8
	Receive Orders	8		🟩 7					🟩 9

Caption:

Weighting class	Relevance of linkage
🟩 [7, 9]	high relevance
🟨 [4, 6]	middle relevance
🟥 [1, 3]	low relevance

Figure 2. Design Decision Traceability Grid (DDT-Grid).

In the tool-guided procedure (i.e., the engineer is prompted by the system), he/she affixes a relevance weighting on each realization in the matrix. This quantified mental judgement is made visible in an enhanced form of the traceability matrix, which we denote as *design decision traceability grid (DDT-grid)* (e.g., Fig. 2). We introduce this DDT-grid as a controlling instrument for engineers in the form of a weighted decision matrix, where the engineer can reflect on the made decisions and its alternatives at any time during the design process. The requirements engineer can distinguish among three weighting classes (i) high relevance in the range of [7, 9], (ii) middle relevance in the range of [4, 6], and (iii) low relevance in the range of [1, 3] (e.g., Fig. 2).

The cognitive heuristic can be classified as post-requirements specification traceability method. The system components level of utility or (functional) degree of fulfillment is automatically computed by an algorithm by which additionally the customer requirements' prioritizations, made by the stakeholders, are taken into account. The computation is based on (i) the number of covered customer requirements, (ii) their prioritization, and (iii) the relevance weightings of linkage. For each system component a relative utility value is expressed as a percentage, which constitutes the component's importance in the entire system architecture and thereby, a pairwise comparison between components is facilitated. On the one hand, also alternative modeling solutions of equal value can be identified, and on the other hand system components with a low level of importance can be easily detected in order to minimize a possible over-engineering risk even before the next step in the system's development process will be started (i.e. when transforming system requirements to design artifacts, or when coding).

The introduced cognitive-based approach to formalize the requirements engineers' informal knowledge by an importance weighting metric and turning this knowledge into operating figures provides a new dimension of requirements traceability controlling at the early stage of requirements management.

IV. EXPECTED RESULTS

Making informal design knowledge explicit (i.e., visible to users by the DDT-grid) would favorably impact the interpretation of the system's architecture. This means that the DDT-grid forms an innovative communication base for system architects and customers. For example, by identifying what are the critical, highly relevant system components in the model in order to assess the developers' efforts when implementing them. Additionally, stakeholders can track, at any time, how the scoring of system artifacts is affected when customers vary the prioritization, delete, or add requirements. Thus, the cognitive-based approach makes a contribution to the fulfillment of an efficient *requirements negotiation* as proposed in [1]. The DDT-grid and the calculated metrics provide a kind of cognitive map for engineers to rethink about their design decisions (e.g., to check if they are on the "right way"). Each map represents a personal view of prioritized solutions of the system. By comparing these maps a hint to a different system understanding can be provided. Additionally, a quick overview can be given through the highlighted visualization in the DDT-grid by which different weightings of realizations are made visible. By describing and discussing its own view, a common understanding between stakeholders can be established.

ACKNOWLEDGMENT

The introduced approach will be realized in the course of the *TraCo* project. TraCo (Traceability Controlling) is supported and promoted by the Austrian Research Promotion Agency (FFG) by the funding program BRIDGE. Launch of the 18-month project is October 01 2012. The cooperation partner in this FFG - project is SparxSystems Software GmbH based in Vienna. SparxSystems specializes in high-performance and scalable visual modeling tools for planning, design and construction of software intensive systems. TraCo is to be conceptualized as plug-in for the core product of the company, the Enterprise Architect, and it is to be implemented as a component of the business logic of EnArWEB (Enterprise Architect in the WEB), where the Enterprise Architect is used as repository.

REFERENCES

[1] L. Cao and B. Ramesh, "Agile Requirements Engineering Practices: An Empirical Study," IEEE Software, vol. 25, IEEE Computer Society, January/February 2008, pp. 60–67, doi: 10.1109/MS.2008.1.

[2] M. Flower, UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3th edition, Addison-Wesley Object Technology, 2003.

[3] S. Friedenthal, A. C. Moore, and R. Steiner, A Practical Guide to SysML: The Systems Modeling Language, The Morgan Kaufmann Omp Press, Elsevier Inc., 2012.

[4] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," In Proceedings of the 1th International Conference on Requirements Engineering, Colorado Springs (CO US), April 1994, pp. 94–101, doi: 10.1.1.137.5052.

[5] H. Kniberg, Scrum and XP from the Trenches, How we do Scrum, Enterprise Software Development Series, C4Media Inc, 2007.

[6] J. Macmillan and J. R. Vosburgh, "Software Quality Indicators," Final Report, Scientific Systems Inc Cambridge (MA US), September 1986, Accession Number: ADA181505.

[7] A. Mazak, M. Lanzenberger, and B. Schandl, "iweightings: Enhancing Structure-based Ontology Alignment by Enriching Models with Importance Weightings," In Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 10), Krakow (PL), February 2010, IEEE Press, pp. 992–997, doi=10.1109/CISIS.2010.164.

[8] A. Mazak, "CoMetO: A Cognitive Design Methodology for Enhancing the Alignment Potential of Ontologies," doctoral thesis, Information & Software Engineering Group (ifs), Department of Software Technology and Interactive Systems, Vienna University of Technologie, Vienna, April 2012.

[9] K. Pohl, Requirements Engineering: Grundlagen, Prinzipien, Techniken, 2. Auflage, dpunkt.verlag, 2008.

[10] B. Ramesh, C. Stubbs, T. Powers, and M. Edwards, "Requirements traceability: Theory and practice," in Journal

Annals of Software Engineering, vol. 3, J. C. Baltzer AG, Science Publishers, January 1997, pp. 397–415.

[11] B. Ramesh and M. Jarke, “Toward Reference Models for Requirements Traceability,” in IEEE Transactions of Software Engineering, vol. 27, January 2001, pp. 58–93.

[12] IBM, Rational DOORS version 9.2, available at http://publib.boulder.ibm.com/infocenter/rsdp/vlr0m0/index.jsp?topic=/com.ibm.help.download.doors.doc/topics/doors_version9_2.html.

[13] D. Steinpichler and H. Kargl, “Enterprise Architect, project management with UML and EA,” Manual revised edition for Version 9.3, SparxSystems Software GmbH, Vienna, January 2011, available at http://www.sparxsystems.de/?gclid=CNap8rfwr7MCFYq7zAodxR4AXg.

Knowledge Management Practices in GSD: A Systematic Literature Review

Smeea Arshad
Department of CS and IT
Mirpur Univesity of Science & Technology
Mirpur, Pakistan
smeeaarshad@gmail.com

Muhammad Usman
Department of CS and S E
International Islamic University Islamabad
Islamabad, Pakistan
m.usman@iiu.edu.pk

Naveed Ikram
Department of Computer Science and Software Engineering
Ripah International University Islamabad
Islamabad, Pakistan
naveed.ikram @riu.edu.pk

Abstract*—Global software Development (GSD) is a popular software development setting that aims at developing software at low cost with geographically distributed teams. Knowledge Management (KM) is an important issue in GSD. Plethora of research is available to solve GSD issues with Knowledge management practices (KMPs).** ***Evidence about the effectiveness of these practices is scattered among different studies*. The need exists to collect, synthesize and review this research at one place. This study explores GSD issues due to lack of knowledge management (KM) and knowledge management practices (KMPs) used to solve these issues. Systematic literature review (SLR) is performed for the identification of KMPs used in GSD projects to handle GSD issues. The study has identified GSD issues due to lack of KM and KMPs used to address these issues.** ***Effectiveness of knowledge management practices is seen by associating a frequency count with each practice. **Knowledge transfer, shared understanding and communication are mostly reported problems. Collaborative technologies are widely used practice to solve GSD issues due to lack of KM.**

Keywords-Knowledge Management; Knowledge Management Practices; Global software Development; Systematic Literature Review

I. INTRODUCTION

Knowledge management (KM) is an asset for software development organizations. It addresses different issues in software development and at the same time also contributes to software process improvement [1][2].

Global software development (GSD) is a methodology to develop software with teams at multiple locations to get the edge of round the clock development and nearness to the market. However, geographical separation introduces many issues such as communication, coordination, control and knowledge management. Knowledge, in offshore development teams is scattered across continents and GSD barriers make its coordination and synthesis difficult [2]. At the same time, effective KM plays a paramount role in solving the issues innate in offshore software development. Knowledge management facilitates the organization operating globally to successfully integrate and coordinate knowledge resources [2]. Knowledge acquisition and sharing is helpful in achieving shared understanding in GRE [6].

The area of KM is explored to see its influence in global software engineering. Number of studies have highlighted the issues that arise in GSD due to lack of KM. Desouza et al. [2] empirically investigated different organizations and highlighted the importance of KM in their study. They identified access to skilled knowledge group as one force among the other compelling forces for global software development. They found seeking relevant knowledge, knowledge sharing, synthesis and transfer are some of the KM problems faced in GSD [1].

Damian et al. [14] focused on the impact of remote communication, knowledge management, time and culture differences on requirements engineering activities and found that ineffective knowledge management influences requirements negotiation, prioritization, specification and validation. Avram [7] focused on socio-cultural impact on knowledge exchange. During the empirical investigation they found maintaing awareness and knowledge transfer are the problems that mainly arise due to lack of informal communication.

Realizing the importance of KM, many practices are suggested to manage knowledge in GSD. Avram [3] identified the knowledge work practices that are used in the actual work setting. The focus of the study is on people, “their values and connections” to deal with issues in distributed development. The identified practices deal with the issues of knowledge transfer, mutual knowledge and knowledge sharing. Desouza [2] empirically found the strategies and models used to manage knowledge in software industry. Clerc [4] reviewed the architectural knowledge management approaches. He categorized the approaches in personalization and codification strategies and suggested to focus on hybrid approaches. Paiva

[8] narrates the experience of implementing community of practice (CoP) by Brazil Global Development Center. CoP helped in project management, information reuse, reducing time in trouble shooting, requirements specification and reverse engineering.

Despite the acceptance of knowledge management practices (KMPs) to solve GSD issues little evidence based research exists to GSD practionars to select appropriate knowledge management practice to deal with a particular issues. This paper intends to fill the gap by conducting the systematic literature review (SLR) about the state of practice of knowledge management in GSD. We followed the guidelines proposed by Barbara Kitchenham [5].The questions investigated are:

RQ1: What GSD issues occur due to lack of knowledge management (KM)?
RQ2: What KMPs are used in GSD projects?
RQ3: What GSD issues are addressed by existing knowledge management practices (KMPs)?

We have identified GSD issues due to lack of KM and KMPs to sort out these issues. We have classified the practices into predefined categories of codification and personalization. The paper is organized as follows: Section 2 describes the background. Section 3 reports the research methodology. Section 4 presents findings from the systematic literature review and analysis with some discussion. Section 5 describes the conclusion.

II. RESEARCH MEHODOLOGY

The research is conducted using systematic literature review (SLR). SLR is a well defined, thorough and fair "means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest" [7]. Barabra Kitchenham's guidelines [5] were followed to define the protocol and conduct SLR. The steps to perform review are Identification of research, primary studies selection, Study quality assessment, Data extraction & analysis.

A. Search Strategy

Search strategy consists of deriving major terms from questions, developing search strings from major terms and their synonyms using AND/OR operators. Major search terms were 'knowledge management' and 'GSD'. These terms and their synonyms are shown in table 1 below. These two terms (or their synonyms) were ANDed to form the generic search string. Initial pilot study helped in selecting the synonyms and major search terms.

TABLE I :MAJOR SEARCH TERMS AND THEIR SYNOYMS

GSD:"Global Software development" OR "distributed software development" OR "multi-site software development" OR "global software engineering" OR "global requirements engineering" OR "distributed software engineering" OR "distributed requirements engineering" OR "multisite software development" OR GSD OR GSE OR "offshore software development" OR GRE
KM: (**"**knowledge management**"** OR "knowledge sharing" OR "knowledge acquisition" OR "knowledge transfer" OR "knowledge creation" OR "knowledge capture" OR "tacit knowledge" OR "explicit knowledge" OR "knowledge retention" OR "knowledge valuation" OR "knowledge use" OR "knowledge application" OR "knowledge discovery" OR "knowledge integration" OR "knowledge theory" OR "organization knowledge" OR "knowledge engineering" OR "information management" OR "information sharing" OR "information transfer" OR "information reuse" OR "common understanding" OR "shared understanding"

A total of 525 papers were obtained from a range of databases. Databases searched and no. papers retrieved from each database were: Inspec, IET, IEEE Explore (38), ACM Digital Library(85), Science Direct (149) , Springerlink (215), EICompendex (107). Customized search strings were developed from generic string for each database. The selected databases encompass the major database in software engineering and most of the research work is published in these databases. Google Scholar was also searched for the publications at first, but later, during pilot study was excluded as it gave multiple results of the same query at different time.

B. Publication Selection

Inclusion and exclusion criteria applied to the studies are given below:

1) *Inclusion Criteria:* It is used to include the studies for data extraction. We included studies that

- Are about KM in GSD AND
- Are supported by some evidence in the form of case study or industrial/Experience report or experiment AND
- Are published in peer reviewed journal or conference.

2) *Exclusion Criteria:* It is used to screen out studies that are not included for data extraction. We excluded studies that

- Are not directly related to KM or KMPs in GSD context OR
- Lack evidence support OR
- Studies that describe GSD problems not relevant to KM

3) *Selecting Primary and Secondary Resources*: Primary studies selection was carried out at two levels. Level 1 screening was based on title, abstract and keywords. This excluded the papers that were not relevant to our research question. After level 1 screening of 525 papers, 51 studies were selected as candidate primary studies. In Level 2 screening, incusion/exclusion criteria was applied on full text of 51 candidate primary studies. After this step, a total of 27

studies were selected as primary studies. Data extraction and quality assessment was performed for these 27 primary studies.

Single study in multiple papers was considered only once. Most recent and comprehensive study was selected when focus of the study was same.

Secondary studies selection was based on references of primary studies; 6 secondary studies were selected for data extraction.

C. Publication Quality Assessment

Quality assessment was applied on the final selection of papers in parallel with data extraction. We consulted protocols*" Agile software development"* and *"EPIC Case Study 2 – Extension of a Tertiary Study"* and Kitchenham guidelines [5],[10],[11] to develop quality assessment criteria that consists of questions. Every quality question has three options; yes means 2 points, no means 0 point and partial means 1 point. Quality Assessment Criteria for Industrial/ Experience report are:

- Does the study clearly describe the context?
- Does the links between data, interpretation and conclusion are illustrated well?
- Does study adds value to research?

And for evidence based studies are:

- Does study clearly narrate objectives?
- Does study clearly describe context?
- Does the sampling method and its rational given?
- Does the data collection method and rational given?

D. Data Extraction

Single researcher (1st author) was responsible for the data extraction. Secondary reviewers (other authors) were consulted in case of problem or confusion.

E. Data Synthesis

We identified list of GSD issues and KMPs used to address these issues at the end of data extraction phase. The KMPs were reviewed to make their categories along the codification and personalization lines.

III. RESULTS

Results of RQ1, RQ2 and RQ3 are shown in the section below:

1) GSD issues identified through SLR (RQ1): Table II shows the list of GSD issues due to lack of knowledge management identified through SLR. Complete citation of SLR references is given in appendix A. The identified issues vary from frequently reported in different studies to the ones that are reported only once. The table shows only the most occurring issues. Issues that fall in frequency range of 3, 2 and 1 are not shown in table due to the shortage of space.

TABLE II GSD ISSUES DUE TO LACK OF KM

#	**GSD Issues due to lac of KM**
1	Shared understanding
2	Knowledge sharing
3	Communication
4	Knowledge transfer
5	Relationship building or team cohesion
6	Trust
7	Finding the right people

Issues that are reported rarely (frequency less than 5) are coordination, requirements engineering, awareness, culture, cost, quality, alignment of process and tools/objectives and faster ramp-up time, knowledge reuse, mutual knowledge, knowledge creation, Knowledge externalization, knowledge exchange, information gathering, ineffective decision making meetings, cycle time, time to market, fill knowledge gap, gap in knowledge flow, face-to-face meetings difficult, codified knowledge compatibility and exchangeability, knowledge retrieval, time zone difference, outsourcing success.

2) KMPs used in GSD projects identified through SLR (RQ2): Table III shows list of Knowledge Management Practices (KMPs) applied to address GSD issues. There are many practices to solve a GSD issue which indicates that an organization must follow some guidelines to select the appropriate practice prior conducting the project. Frequently reported practices are shown in table.

Practices that are reported rarely are: Division of work, Informal communication, Guidelines or training programs, Clear project/organization structure with clear roles and responsibilities, Cross continental mini teams, Adapt scrum, Learn by watching, Direct request, Information update, Knowledge centric product life cycle management, Reverse Presentation Method, Shared team and task knowledge, Surviving the Babel tower, Mutual adjustment, Process Knowledge Tracer, Shared infrastructure, Clear project structure with clear communication responsibilities, Discussion board, Knowledge reuse.

TABLE III KMPS USED IN GSD PROJECTS

#	**KMPs used in SD Projects**
1	Collaborative technology
2	Meetings or visits
3	Documentation
4	Asking the developers/boundary spanners/colleague
5	Transactive memory
6	Knowledge sharing
7	Standard tools and methods

3) KMPs used to address GSD issue (RQ3): Table IV shows the GSD issues due to lack of KM and KMPs used to address these issues along with the number of times the pair (of GSD issue and KMP used to address it) is reported together. This is different from table III, wherein we only showed the reported practices.

TABLE IV KMPS AND GSD ISSUES

GSD Issues due to lack of KM	**KMPs used to address GSD issues due to lack of KM**	**f**
Shared understanding	1) Collaborative technology 2) Meetings 3) Documentation 4) Standardized tools and methods 5) Transactive memory 6) Asking the colleague 7) Guidelines/training program 8) Reverse Presentation method (RPM)	3 3 3 2 2 1 1 1
Knowledge sharing	1) Collaborative technology 2) Meetings 3) Surviving the Babel tower 4) Process Knowledge Tracer 5) Cross continental mini teams 6) Direct communication 7) Division of work 8) Shared infrastructure 9) Discussion board 10) Transactive memory	4 2 1 1 1 1 1 1 1 1 1
Communication	1) Meetings/Visits 2) Asking the colleague 3) Collaborative technology 4) Clear project/organization structure with clear roles and responsibilities 5) Transactive memory 6) Information update 7) Adapt scrum process 8) Reverse Presentation method (RPM) 9) Knowledge centric product life cycle management 10) Documentation	3 3 2 2 1 1 1 1 1 1
Knowledge transfer	1) Collaborative technology 2) Meetings 3) Asking the colleague 4) Documentation 5) Division of work 6) Transactive memory 7) Standard tools and methods 8) Surviving the Babel tower	2 2 2 2 1 1 1 1
Team cohesion	1) Visits/meetings 2) Mutual adjustment	4 1
Trust	1) Meetings/Visits 2) Collaborative technology 3) Adapt scrum	3 2 1
Finding the right people	1) Transactive memory 2) Collaborative technology 3) Meetings or Visits 4) Asking the colleague 5) Standard tools and methods	3 2 1 1 1

IV. DISCUSSION

Discussion section is divided in two parts i.e., discussion on GSD problems due to lack of KM and discussion on KM practices used to address the issues.

1) Global Software Development Problems due to lack of Knowledge Management:

Lack of common understanding is the most occurring problem identified during this study. 13 papers reported lack of shared understanding as a problem in GSD. Difference in organizational culture has great impact on shared understanding and creates problems in gaining common understanding of different aspects of project because of the difference in terminologies used by organizations for the same concept, difference in standard of documentation. Research shows that successful projects implemented standard tools and methods to achieve shared understanding ([3][9]). Communication gap in GSD teams gives rise to misunderstandings and takes more time to correct misunderstandings [14]. Language difference is another reason of the lack of common understanding in distributed software development teams. Difference in time zones of teams introduces communication gap; thereby giving rise to misunderstandings among time members and also providing them less time to clear these misunderstandings ([15][16]). Cultural diversity, communication gap, difference in technical background, gap in knowledge flow also create difficulty in achieving shared understanding ([11][17][20][21][22][23][24]). Misunderstanding of requirements can introduce delay [18].

Knowledge sharing is another important problem identified. 40% of the studies (11 papers) reported knowledge sharing as a problem. This confirms our results with the previous studies that mentioned knowledge sharing as the critical success factor for outsourcing relationships success [25]. Only one paper reported that information was not appropriately shared among team members, whereas other studies mentioned that they found sharing knowledge to be problematic. Research suggests that tacit nature of knowledge and lack of trust among team members are the reasons behind the problem of knowledge sharing(Two papers suggested that factor that contributes to problem of knowledge sharing is tacit nature of knowledge, while one paper mentioned trust among cross site team members as a reason for lack of knowledge sharing). Tacit nature of knowledge creates problem only in sharing implicit knowledge whereas trust is the factor that is necessary for sharing both kinds of knowledge either implicit or explicit [12][14][26]. Building trust and sharing tacit knowledge both require face to face interaction that is difficult to achieve in GSD.

This SLR identifies lack of Communication; either formal or informal as another problem that arises mostly during global software development projects. 37% of the studies (10 papers) reported the problem of lack of communication. The identified studies narrate the inability of team to have appropriate communication but failed to describe the specific problem they faced. The main reasons of lack of communication are geographical, socio-cultural and temporal distance. Due to lack of communication several problems arise i.e. lack of trust, relationship building etc. which ultimately cause lack of knowledge sharing. Informal communication is also badly impacted and almost become impossible due to geographical distance.

Knowledge transfer, relationship building, and finding the right people are some other important problems that require social aspect to be considered. Various other problems are also identified but these are less important with low frequency.

2) Discussion on Knowledge Management Practices:

Collaborative technology is an important practice reported in 51% of studies, supports hybrid strategy (codification & personalization) and solves 14 problems. Use of collaborative technology reduces social distance, makes people aware of other's presence, produces the sense of being a team, synchronizes communication and reduces delay by providing in time feedback. Collaborative technology includes email, video conferencing, IM, online data bases, etc. Email, Instant Messaging (IM) and video conferencing are more frequently used among identified practices of collaborative technology. IM was mostly used when to get information about a certain problem or for interaction with experts whereas video conferencing became an interaction medium for a group most of the time, yet maintained open communication between two managers as well. Email is more frequently used for sharing artifacts, circulating logs and exchanging documents. The Pros and cons of using collaborative technology are:

- *It has the ability to deal with the problem of communication;* a major challenge in GSD projects. Geographical distance hinders *face to face communication in GSD.* The only way to communicate among team members across different locations is through the use of collaborative technology. Collaborative technology is rich media for communication. It supports formal, informal and synchronous and asynchronous communication [14]. Lack of Informal communication is found to be the reason behind many problems identified in this study such as knowledge sharing/transfer, building trust, finding the right people etc.;[11][28]. *We argue IM* and video conferencing can enable informal communication. Collaborative technology also facilitates formal communication; thereby impacting knowledge sharing/transfer and shared understanding. Exchanging documents via email i.e.; asynchronous communication tends to cope with time zone difference. Synchronous communication via collaborative technology impacts communication and shared understanding.
- Another advantage of using collaborative technology is it supports both personalization and codification strategy [14]. Other practices supporting hybrid strategies contrary to collaborative technology are either specific to one or two organizations or some specific problems.
- However, use of collaborative technology is not without problems. Temporal distance can't be overcome by collaborative technology. Another limitation of collaborative technology is when a time slot is dedicated for informal communication but till the end of meeting formal conversation goes on leaving no room for informal communication [14].

Meetings/visits are considered more useful to cope with problems created by geographical, temporal and socio-cultural distance. Their ability to develop trust and build sense of being a team is an important factor for the wide spread adoption of this practice. Arranging visits can be costly as compared to technical meetings but these have more benefits. Research has shown that GSD projects that lack visits were unsuccessful and those supported travelling were successful [27].

Documentation is third most frequently used practice and solves seven issues among the issues identified. Documentation is beneficial in keeping the group aware of what's happening in the project and is also necessary to keep aware if new person joins the team or some expert or relevant person leaves.

Asking the colleague and transactive memory also highlight the importance of social aspect in dealing with the GSD issues. Transactive memory supports both codification and personalization strategy and indicates that both type of practices must be used to be successful in GSD projects.

Knowledge management practices are broadly categorized in codification and personalization strategy in literature. We attempted to categorize Knowledge management practices used to address GSD problems due to lack of KM identified in this SLR along these two dimensions.

Codification: This category includes practices of Documentation, Standardize tools and methods, Shared infrastructure, Information update and Knowledge reuse.

Personalization: This category includes practices of Meetings or visits, Asking the colleague, Informal communication, Cross continental mini teams, Surviving the Babel tower, Direct request, Learn by watching, Mutual adjustment, Shared team and task knowledge, Adapt Scrum processes, Information update, Clear project/organization structure with clear roles and responsibilities and Discussion board.

Hybrid: Practices in this category are Collaborative technology, Reverse Presentation Method, Transactive memory, Division of work, Process Knowledge Tracer, Knowledge centric product life cycle management, Guidelines/training programs and Knowledge sharing.

This categorization can be helpful in appropriately selecting and devising a knowledge management strategy at the start of GSD project. Managers should include practices from all categories in overall KM strategy so that the complementary nature of practices helps in tackling different types of issues in GSD projects. Following guidelines can be helpful in dealing with issues due to lack of KM in GSD:

- Encourage informal communication among team members. It will produce a sense of being a team and develop trust.
- Standardize tools /methods and procedures as it will keep all the team members at the same level of understanding
- Focus on building social interaction that is badly impacted by geographic distance. It is identified as main reason behind many problems identified.
- Arrange travelling across sites. This may be costly but has long range benefits. Research has shown that successful projects have adopted this practice.
- Use both personalization and codification strategies in the project. Regularly plan for meetings and use collaborative technology and documentation in these meetings. Use collaborative technology for communication mainly informal communication between two meetings.
- Collaborative technology is one of the most used practices and its use can be beneficial for the project. However, it is effective when accompanied by other practices such as meetings and documentation
- Be proactive in dealing with the problems that arise in GSD projects due to lack of KM. Frequency of GSD problems identified and KMPs used to addressed these problems can help you in doing so.

We are currently working on purposing a model for the selection of the appropriate practice to deal with identified problems. A brief description of the model is provided in this paper. The proposed model deals with the most frequently occurring issues and most widely used practices to handle these issues. Collaborative technology, documentation and visits are at the core of the model indicating that these practices must be used to solve different kind of problems.

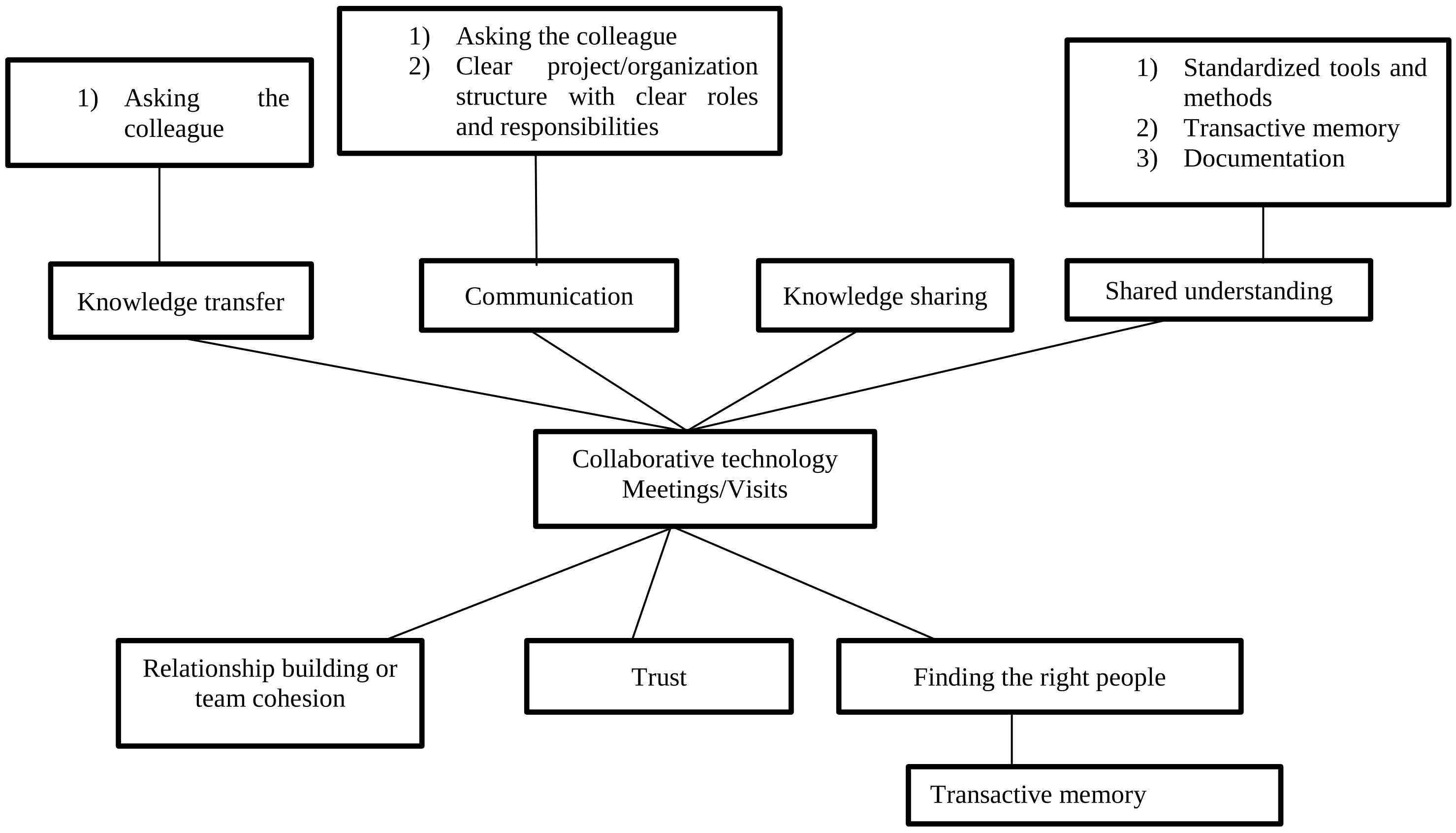


Figure 1. Model for selecting KMPs.

Team cohesion, lack of trust and relationship building are the problems that arise because of lack of social interaction or informal communication. Collaborative technology provides the supports social interaction among team members either in meetings or through direct contact. Meetings and visits also help in building sense of being a team that produces trust.

Problems at on the top require use of both codification and personalization practices. Knowledge sharing and transfer require social interaction to share tacit knowledge.. Collaborative technology and meetings/visits are the appropriate practices for this purpose. However, documentation is also required to share knowledge.

V. CONCLUSION AND FUTURE WORK

We have identified through SLR, GSD issues due to lack of KM and KMPs to solve these issues. We have identified that knowledge sharing, shared understanding and communication as most prevailing issues and use of collaborative technology and meetings or visits are mainly used practices to handle these issues. We have also found that most of the issues in GSD due to lack of knowledge management are due to lack of social interaction.

Our work facilitates the practioners in comparing which practice is better to adopt to deal with a certain issue. Frequency of each problem and practice narrates the severity of the problem in terms of its occurrence and effectiveness of a practice. This helps in identifying which problem will be confronted most and which practice is used widely to handle it. Making an aware choice by considering all the alternatives and consequences leads to better result. This piece of research helps in making an aware choice and thus leads to better results by applying this choice in offshore software development.

The following directions can be taken into account for future research:

- Relatively little empirical work has been done in software engineering. Similar situation is prevailing in the area of KM. Studies have been identified that provide only the theoretical base and lack the empirical evidence. This area requires consideration for future research.
- An important area in KM that lacks empirical evidence is KM tools. There are a lot of KM tools, but they lack the empirical ground. This area also has the potential for considering it for future research.
- This work can be extended by providing industrial perspective from Pakistan. Studies included in this SLR did not account any organization from Pakistan. A survey can be launched to identify GSD problems due to lack of KM and KMPs used to handle these issues. The results can be compared with the above mentioned results and a model can be proposed and validated to select the best practice to solve GSD issues due to lack of KM.

VI. LIMITATIONS

The SLR incorporates the data of last ten years.

APPENDIX A: RESEARCH PAPERS INCLUDED IN THE SLR AS PRIMARY STUDIES

s1 G. Avram, “”Of Deadlocks and Peopleware - Collaborative Work Practices in Global Software Development,” Global Software Engineering, 2007. ICGSE 2007.

s2 J. Bosch and P. Bosch-Sijtsema, “From integration to composition: On the impact of software product lines, global development and ecosystems,” Journal of Systems and Software, vol. 83, pp. 67-76, July 2009.

s3 J. Kotlarsky, P.C. Van Fenema, and L.P. Willcocks, “Developing a knowledge-based perspective on coordination: The case of global software projects,” Information & Management, vol. 45, pp. 96-108, Feburary 2008.

s4 G. Avram, “Knowledge Work Practices in Global Software Development,” The Electronic Journal of Knowledge Management, vol. 5.

s5 S. Komi-Sirvio, and M. Tihinen, “Lessons learned by participants of distributed software development, ”Knowledge and Process Management, vol. 12**,** pp. 108-122, 2005.

s6 M. Biro, and P. Feher, “ Forces affecting offshore software development,” 12th European Conference on Software Process Improvement, EuroSPI 2005, November 9, 2005 .

s7 J. Kotlarsky, and I. OSHRI, “Social ties, knowledge sharing and successful collaboration in globally distributed system development projects,” European Journal of Information Systems, vol. 14, pp. 37-48, 2005.

s8 S. LEE, and H.-S. Yong, “Distributed agile: project management in a global environment,” Empirical Software Engineering, vol. 15**,** pp. 204-217.

s9 K. Mohan, and B. Ramesh, “Traceability-based knowledge integration in group decision and negotiation activities,” Decision Support Systems, vol. 43, pp. 968-989, 2007.

s10 A. Mathrani, D. Parsons, and R. Stockdale, “Workgroup structures in offshore software development projects: A vendor case study,” 2009 13th Enterprise Distributed Object Computing Conference Workshops, EDOCW - IEEE EDOC 2009 Workshops and Short Papers, September 1, 2009 - September 4, 2009.

s11 A. Boden, and G. Avram, “Bridging knowledge distribution - The role of knowledge brokers in distributed software development teams,” Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09. ICSE Workshop on software engineering.

s12 E. D. DAMIAN, and D. ZOWGHI, “ RE challenges in multi-site software development organisations.” Requirements Engineering, vol. 8**,** pp. 149-160, 2003.

s13 C. Ebert, and & J.D MAN, “Effectively utilizing project, product and process knowledge,” Information and Software Technology, vol. 50, pp. 579-594, 2008.

s14 A. Taweel, B. Delaney, T.N. Arvanitis, and L. Zhao, “Communication, knowledge and co-ordination management in globally distributed software development: Informed by a scientific software engineering case study,” 2009 4th IEEE International Conference on Global Software Engineering, ICGSE 2009.

s15 M. Wiener, and R, Stephan, “Reverse Presentations,” Business & Information Systems Engineering.

s16 B.E. Munkvold, and I. Zigurs, “Process and technology challenges in swift-starting virtual teams,” Information & Management, vol. 44, pp. 287-299, 2007.

s17 V. Clerc, P. Lago, and H. Van Vliet, “The usefulness of architectural knowledge management practices in GSD,” 2009 4th IEEE International Conference on Global Software Engineering, ICGSE 2009.

s18 M.Jensen, S. Menon, L.E. Mangset, and V. Dalberg, “Managing offshore outsourcing of knowledge-intensive projects a people centric approach,” International Conference on Global Software Engineering, ICGSE 2007.

s19 L. Taxen, “An integration centric approach for the coordination of distributed software development projects,” Information and Software Technology, vol. 48, pp. 767-780,2006.

s20 G. Avram, “Developing Outsourcing Relationships: A Romanian Service Provider Perspective,” First Information Systems Workshop on Global Sourcing: Services, Knowledge and Innovation Val d'IsèreFrance 13-15 March 2007.

s21 .Oshri, P. V. Fenema, and J. Kotlarsky, “Knowledge transfer in globally distributed teams: the role of transactive memory,” Information Systems Journal, vol. 18, pp. 593-616,2008.

s22 J.Z. Gao, F. Itaru, et al., (2002) “Managing Problems for Global Software Production – Experience and Lessons,” Information Technology and Management vol. 3, pp. 85-112.

s23 A. Taweel, B. Delaney and Z. Lei, “”Knowledge Management in Distributed Scientific Software Development,” Global Software Engineering, 2009. ICGSE 2009.

s24 L. Pilatti, J. Audy and R. Prikladnicki, “Software configuration management over a global software development environment: lessons learned from a case study,” 2006.

s25 K. Desouza, T. Dingsoyr, and & Y. Awazu, “Experiences with conducting project postmortems: reports versus stories,” Software Process: Improvement and Practice, vol. 10, pp. 203-215, 2005

s26 J.A. Espinosa, S.A.Slaughter, R.E. Kraut, and J.D. Herbsleb, (2007) “Team knowledge and coordination in geographically distributed software development”, Journal of Management Information Systems, 24**,** 135-169, 2007.

s27 A. Gupta, E. Mattarelli, S. Seshasai, and J. Broschak, “Use of collaborative technologies and knowledge sharing in co-located and distributed teams: Towards the 24-h knowledge factory,” The Journal of Strategic Information Systems, vol.18, pp. 147-161, 2009.

REFERENCES:

[1]. J. Herbsleb, and D. Moitra, Global software development. *Software, IEEE,* vol. 18**,** pp. 16-20, Mar/April 2001.

[2]. K. Desouza, Y. Awazu, and P. Baloh, “Managing knowledge in global software development efforts: Issues and practices. IEEE software, vol. 23, pp. 30-37, October 2006.

[3]. G. Avram “Of Deadlocks and Peopleware - Collaborative Work Practices in Global Software Development,” Global Software Engineering, ICGSE 2007.

[4]. V. Clerc, “Towards architectural knowledge management practices for global software development. ACM, 2008.

[5]. B. Kitchenham, and S. Charters, “Guidelines for performing systematic literature reviews in software engineering. Engineering, Vol. 2,2007.

[6]. D. Damian, L. Izquierdo, J. Singer, and I. Kwan, “Awareness in the wild: Why communication breakdowns occur,” IEEE, 2007.

[7]. A.Boden, G. Avram, L. Bannon and V. Wulf, “Knowledge Management in Distributed Software Development Teams - Does Culture Matter?,” Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering. IEEE Computer Society, 2009.

[8]. E.A. Paiva, “The test community of practice experience in Brazil.,” IEEE International Conference on Global Software Engineering, ICGSE , October 2006.

[9]. J. Kotlarsky, P.C. Van Fenema, and L. P. Willcocks, “Developing a knowledge-based perspective on coordination: The case of global software projects.” Information and Management, vol. 45, pp. 96-108, Feburary 2008.

[10]. T. Byba, Review Protocol–Agile Software Development.

[11]. B. Kitchenham, O. Brereton, and M. Turner, “EPIC Case Study 2–Extension of a Tertiary Study,” EPIC Technical Report, EPIC-2009-007.

[12]. J. Bosch, and P. Bosch-Sijtsema, “From integration to composition: On the impact of software product lines global development and ecosystems,” Journal of Systems and Software, Vol. 83**,** pp. 67-76, July 2009 .

[13]. J.-N. LEE, M. Huynh, and R. Hirschheim, “An integrative model of trust on IT outsourcing: Examining a bilateral perspective,” Information Systems Frontiers, Vol. 10**,** pp. 145-163, 2008.

[14]. D. Damian, and D. Zowghi, “Requirements Engineering challenges in multi-site software development organizations,” Requirements Engineering, Vol. 8, pp. 149-160, 2003.

[15]. G. Avram, “”Of Deadlocks and Peopleware - Collaborative Work Practices in Global Software Development,” Global Software Engineering, 2007. ICGSE 2007.

[16]. S. Komi-Sirvio, and M. Tihinen, “Lessons learned by participants of distributed software development, ”Knowledge and Process Management, vol. 12**,** pp. 108-122, 2005.

[17]. M. Wiener, and R, Stephan, “Reverse Presentations,” Business & Information Systems Engineering.

[18]. L. Pilatti, J. Audy and R. Prikladnicki, “Software configuration management over a global software development environment: lessons learned from a case study,” 2006.

[19]. A. Boden, and G. Avram, “Bridging knowledge distribution - The role of knowledge brokers in distributed software development teams,” Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09. ICSE Workshop on software engineering.

[20]. V. Clerc, P. Lago, and H. Van Vliet, “The usefulness of architectural knowledge management practices in GSD,” 2009 4th IEEE International Conference on Global Software Engineering, ICGSE 2009

[21]. G. Avram, “Developing Outsourcing Relationships: A Romanian Service Provider Perspective,” First Information Systems Workshop on Global Sourcing: Services, Knowledge and Innovation Val d'Isère, France 13-15 March 2007.

[22]. I.Oshri, P. V. Fenema, and J. Kotlarsky, “Knowledge transfer in globally distributed teams: the role of transactive memory,” Information Systems Journal, vol. 18, pp. 593-616,2008.

[23]. L. Taxen, “An integration centric approach for the coordination of distributed software development projects,” Information and Software Technology, vol. 48, pp. 767-780,2006.

[24]. A. Mathrani, D. Parsons, and R. Stockdale, “Workgroup structures in offshore software development projects: A vendor case study,” 2009 13th Enterprise Distributed Object Computing Conference Workshops, EDOCW - IEEE EDOC 2009 Workshops and Short Papers, September 1, 2009 - September 4, 2009.

[25]. J.A. Espinosa, S.A.Slaughter, R.E. Kraut, and J.D. Herbsleb, (2007) “Team knowledge and coordination in geographically distributed software development”, Journal of Management Information Systems, 24**,** 135-169, 2007.

[26]. C. Ebert, and & J.D MAN, “Effectively utilizing project, product and process knowledge,” Information and Software Technology, vol. 50, pp. 579-594, 2008.

[27]. K. Mohan, and B. Ramesh, “Traceability-based knowledge integration in group decision and negotiation activities,” Decision Support Systems, vol. 43, pp. 968-989, 2007.

[28]. G. Avram, “Knowledge Work Practices in Global Software Development,” The Electronic Journal of Knowledge Management, vol. 5.

Modelling the Strategic Alignment of Software Requirements using Goal Graphs

Richard Ellis-Braithwaite[1] Russell Lock[1] Ray Dawson[1] Badr Haque[2]

[1]Loughborough University
Leicestershire, United Kingdom

[2]Rolls-Royce Plc.
Derby, United Kingdom

{r.d.j.ellis-braithwaite@lboro.ac.uk, r.lock@lboro.ac.uk, r.j.dawson@lboro.ac.uk, badr.haque@rolls-royce.com}

***Abstract*—This paper builds on existing Goal Oriented Requirements Engineering (GORE) research by presenting a methodology with a supporting tool for analysing and demonstrating the alignment between software requirements and business objectives. Current GORE methodologies can be used to relate business goals to software goals through goal abstraction in goal graphs. However, we argue that unless the extent of goal-goal contribution is quantified with verifiable metrics and confidence levels, goal graphs are not sufficient for demonstrating the strategic alignment of software requirements. We introduce our methodology using an example software project from Rolls-Royce. We conclude that our methodology can improve requirements by making the relationships to business problems explicit, thereby disambiguating a requirement's underlying purpose and value.**

Keywords*—*Requirements Engineering; Strategic Alignment; Quantified Goal Graphs; Requirements Traceability

I. INTRODUCTION

The stakeholders of a software project should share an understanding of the potential business benefit that a software requirement offers. If such an understanding can be achieved, the likelihood that a solution will satisfy a real business problem will be improved. Although such statements may sound obvious, it has been reported that 45% of software requirements are never deemed to be useful after implementation [1]. These unnecessary requirements cause costs and delays that perhaps could have been avoided by benefit analysis. The existence of a requirement should be questioned if it does not demonstrate potential to offer value to the business. Conversely, valuable requirements are at risk of being de-prioritised if they fail to demonstrate their potential benefit. In an organisational setting, business benefit can be gained from an alignment to business strategy.

Technically worded requirements or solution oriented requirements (i.e., specified for the machine rather than for the world [2]) hide the business problem to be solved and leave stakeholders with little understanding of the potential value. It is therefore important that the strategic alignment of such a requirement is explored in order to avoid wastage.

This paper explores the suitability of goal graphs for demonstrating a software requirement's strategic alignment. Current Goal Oriented Requirements Engineering (GORE) standards, such as GRL [3], do not quantify the contribution one goal makes to another using metrics from the application domain, opting instead to use scales such as high, medium and low, or numerical scales such as 0-9. As a result, any strategic alignment proposed by the use of goal graphs is not specific, measurable or testable. Proposed extensions by Van Lamsweerde [4] do not consider that a chain of linked goals may contain a variety of metrics that need to be translated in order to demonstrate strategic alignment. Additionally, the current methods do not consider how the contribution score is calculated and how that affects the credibility and accuracy of the proposed benefits. This paper attempts to demonstrate how the above problems can be addressed, thereby allowing goal graphs to be used to analyse the strategic alignment of software requirements. Our methodology complements frameworks that require business value analysis, such as value-based software engineering [5], by making assumptions about business value explicit.

We have developed and implemented our methodology in partnership with an industrial partner (Rolls-Royce) to ensure its utility in real world settings. We use examples in the context of a software project to be implemented in the Transmissions Structures & Drives (TS&D) Supply Chain Unit (SCU). The software will automate geometry design and analysis for aero engine components, as well as for their manufacturing tools such as casting molds. Simply put, engineers will input the desired design parameters and the software will output the component's geometry.

In Section II, we introduce the problem that this paper addresses, while in Section III, we present and evaluate the extent to which existing solutions address it. Section IV presents our methodology and tool as an extension of an existing GORE methodology in order to address the gaps outlined in Section III. We conclude in Section V with remarks on the paper's contributions and future work.

II. THE PROBLEM

Ross and Schoman stated that software requirements "must say why a system is needed, based on current or foreseen conditions" as well as "what system features will serve and satisfy this context" [6]. Popular Requirements Engineering meta models [7], [8] and templates [9], [10] tend not to focus on "why", typically addressing it by stipulating that rationale be attached to a requirement. However, rationale is not always an adequate description of why the requirement is valuable to the business. If only one "why" question is asked about the requirement then the rationale can still be distant from the true problem to be solved (i.e., the essence of the requirement), and it may be defined without consideration of its wider implications.

As an example of the problem that this paper examines, we introduce the following high-level requirement taken from our example software project: "While operating in an analysis solution domain and when demanded, the system shall run analysis models". The business value of this software requirement and the underlying problem to be solved is not immediately obvious, so to better understand the need for this requirement, we examine the attached rationale: "So that structural integrity analysis models can be solved as part of an automated process". The requirement's benefit to the business and its alignment with strategy are still not clear after one level of abstraction above the requirement. Additionally, the extent of the problem to be solved is not explained, i.e., the problems associated with solving structural integrity analysis models manually and the wider implications of doing so. Perhaps the manual process is costing the business in terms of human resource time or inaccuracy of the analysis due to error. If so, what is the business impact and how far can it be reduced? Additional broader implications may exist that are not immediately obvious - it might be the case that design innovation is constrained by the slower manual process. Clearly there is more to the rationale than is written, and arguably more than it would be sensible to express within a requirement, partly due to the duplication this would incur; several requirements may achieve the same business benefit but at varying levels of contribution and with potentially complex dynamics.

In summary, this paper argues that the strategic alignment of a requirement should be examined so that:

1. The root of the requirement can be understood so that the software can solve the right problem.
2. The extent of the problem can be understood to prove the requirement's value and validity.
3. The value of the requirement can be understood to better inform prioritisation and project funding.

III. Background

The following areas of research are related to the strategic alignment of software requirements: (A) Goal Oriented Requirements Engineering, (B) Strategic Alignment and (C) Software Metrics.

A. Goal Oriented Requirements Engineering

Goal Oriented Requirements Engineering (GORE) seeks to provide answers to the, so far, largely unanswered question of "why" software functionality should exist through the use of goal graphs. Van Lamsweerde defines the term goal in the context of GORE as an optative statement (i.e., desired future state) about an objective that the system hopes to achieve [11]. "Goal" in the context of GORE is therefore more concerned with the goals of the system than the goals of the business. Furthermore, this definition of goal does not differentiate it from an objective. In order to relate the goals of the system to the goals of the business, we need an integrated definition of the terms used in business strategy. The Object Management Group (OMG) defines these terms in the Business Motivation Model (BMM) [12]. The BMM defines a goal as an indication of "what must be satisfied on a continuing basis to effectively attain the vision of the business". An objective is then defined as a "statement of an attainable, time-targeted, and measurable target that the enterprise seeks to meet in order to achieve its goals". Objectives therefore contrast with goals in that "goals are allowed to be unrealistic and unachievable" [13]. Attempting to prove strategic alignment to non-specific goals such as "maximise profit" would be difficult since it would not be possible to prove the extent of its satisfaction. Therefore, requirements should be abstracted to objectives rather than goals for strategic alignment. Fortunately, business strategies are usually decomposed into objectives that follow SMART [14], which allows contribution between objectives to be specified, e.g., "objective x will satisfy half of objective y".

Since the only significant difference between an objective and a goal is in its hardness and specificity (i.e., whether its satisfaction can be determined), GORE methodologies can still be applied. The most well-known GORE methodologies include KAOS [15], i* [16] and GRL [17]. Such methodologies produce goal graphs whereby goals at a high level represent the end state that should be achieved and lower level goals represent the means to that end. The relationships between goals are typically expressed as means-ends links with AND/OR refinement. Additional elements such as agents, obstacles and dependencies are typically included. A goal graph is traversed upwards in order to understand why a goal should be satisfied and downwards to understand how that goal could be satisfied.

Three methods for applying weights to goal-goal contribution links in goals graphs were proposed by Van Lamsweerde [4] with the intention of extending KAOS, but the concepts could be applied to any GORE method:

1. Subjective qualitative scores e.g., --, -, +, ++.
2. Subjective quantified scores e.g., 0 to 100.
3. Objective gauge variables (i.e., a quantity prescribed by a leaf soft goal to be increased, reduced, etc.).

After evaluating the above options, Van Lamsweerde concludes that the specification of link weight scores with objective gauge variables is the most appropriate approach due to its verifiability. However, the method presented in the paper calculates the gauge variables additively from the leaf goals (goals that have no further decomposition); that is, a goal's satisfaction level is calculated by summing the gauge variables that feed into it, which are only defined by the leaf goals. The methodology instructs that when a gauge variable cannot be sensibly added, it should be ignored. This is a problem for proving strategic alignment because often a number of translations between metrics have to be made. For example, take the following goal related to the requirement we introduced in Section II: "reduce the time taken to perform a structural integrity check". This goal cannot contribute through additive gauge variable accumulation to its parent goal: "reduce the fabricated structure design costs" because the metrics differ, (one is time while the other is cost) and to convert them both to cost would omit a level of abstraction, causing ambiguity in how the costs are reduced.

Goal Requirements Language (GRL) was recently made an international standard through ITU specification Z.151 as part of the User Requirements Notation (URN) [3]. GRL integrates the core concepts of i* and NFR [17]. Link contributions in GRL are specified with subjective quantified contribution scores, much the same as outlined in Van Lamsweerde's paper [4]. For example, the time reduction goal might contribute to the cost saving goal with a contribution weight of 67 out of 100. This contribution score is untestable and meaningless; moreover it is not refutable, which, according to Jackson [2], means that the relationship is not described precisely enough because no one can dispute it. The only way such scales could be meaningful is if the goals were specified with fit criterions (e.g., a cost to be saved) and if the scales implied percentage satisfaction (which they do not). In which case, a 50/100 contribution might imply that 50% of a £20,000 annual cost saving will be achieved. However, this is only applicable for goals whose satisfaction upper bound is 100%, since the scale's upper bound is 100; which is not the case for goals involving increases, which may specify more than 100%. Recently, the jUCMNav tool allowed for the relation of Key Performance Indicators (KPIs) to goals in GRL [18]. KPIs specify business targets that measure the performance of a business activity. However, since KPIs do not affect the way in which contribution is measured (i.e., the contribution that a chain of goals makes to a KPI), subjectivity and ambiguity still exists.

One of the most popular tools to compare product qualities with customer requirements is the House of Quality (HoQ) diagram [19]. The fundamental failing of the HoQ is that the score values used to measure the strength of the contribution are subjective, much like those used in GRL. Additionally, since the HoQ is constructed using a 2D grid, only two dimensions can be compared in the same grid, i.e., requirements can be related to software goals, but if those software goals are to be related to customer or business goals, then additional grids will be required for each extra dimension. If these dimensions are not explored (e.g., if the software project goals are not abstracted to business goals), then the goals that the alternative solutions will be evaluated against may be incorrect (e.g., solution specific or aiming to solve the wrong problem). GORE methodologies which evaluate alternative solutions against their effect on goals, e.g., [20], also depend on the alignment of those goals to higher level goals for the resulting decision to be correct.

B. *Strategic Alignment*

One of the most suitable methodologies for relating software requirements to business strategy is B-SCP [21], due to its tight integration with the OMG's Business Motivation Model (BMM). B-SCP decomposes business strategy towards organisational IT requirements through the various levels of the BMM (e.g., the vision, mission, objective, etc.). However, B-SCP cannot accurately show that a requirement satisfies a strategy since no contribution strengths are assigned to links. Since strategic alignment depends on the extent to which the strategy is satisfied (e.g., for the goal "reduce costs", the extent is the amount of cost to be reduced), the extent to which a goal contributes towards another needs to be considered. Indeed, a large proportion of software requirements will only partially satisfy the strategic objectives. Moreover, B-SCP's methodology refines business strategy towards IT requirements, which means that completeness of the model is dependent on the completeness of the business strategy, i.e., there is no opportunity to refine software functionality upwards to propose new business strategy. Additionally, B-SCP does not consider goal conflicts, dependencies, actors or obstacles, as in the GRL and KAOS methodologies.

The Balanced Scorecard and Strategy Maps [22] approach offers guidance on formulating and relating business goals to each other under four perspectives: financial, customer, internal processes, learning and growth. The approach does not concern software requirements; but such an approach could be performed before software requirement to business strategy alignment analysis takes place, in order to ensure business strategy completeness.

C. *Metrics*

Fit criterions as specified by Volere [9] and Planguage [23] can be attached to requirements in order to make them measurably satisfiable. However, assumptions made about the benefits that may be reaped after satisfaction of a fit criterion are not addressed in either methodology. Additionally, Volere and Planguage propose textual representation of requirements; and as such, relationships between requirements are hard to maintain, understand, and visualise. GQM+Strategies™ [24] was developed to extend the Goal Question Metrics methodology by providing explicit support for the relation of software metrics measurement effort (e.g., measuring the impact that pair programming has on quality) to high-level business goals. However, the approach falls short in areas similar to the other methodologies reviewed; contribution links between goals are not quantified (i.e., assumed benefit), there is a fixed number of goal abstraction levels per diagram and there are no additional concepts that are typically included in GORE methodologies to place the goals into context (e.g., actors, conflicts, AND/OR refinement).

IV. METHODOLOGY

We propose that GRL goal graphs can be used to demonstrate strategic alignment by linking requirements as tasks (where the task is to implement the requirement) and business objectives as hard goals (where the hard goal brings about some business benefit) with contribution links (where the requirement is the means to the objective's end). The requirements should be abstracted (asking "why?") until they link to business objectives. We have used GRL's notation because it is part of the Z.151 international standard [3] and because its notation is well known (it originates from i*).

Soft goal elements (e.g., goals and visions from the Business Motivation Model) should not be defined in the goal graph for the purpose of demonstrating strategic alignment since their satisfaction is often immeasurable (if their satisfaction is possible at all); therefore, it is nonsensical to consider that a requirement may either partially or completely satisfy a goal or a vision. However, since objectives exist to quantify goals, and since goals exist in order to amplify the vision of the business [12], non-weighted traceability between an objective and its goals (and their related vision) should be maintained for posterity.

For our reference implementation, we have used the Volere requirements template to define the attributes of a requirement, primarily because it specifies a fit criterion field used for testing the requirement's satisfaction. An "estimated effort" field (specified in person-hours) could be added to the template so that cost-benefit analysis can be performed. Software implementation effort estimation methods such as COCOMO [25] could be useful in refining estimated values.

We define objectives using our modified GQM+Strategies formalisation template [26], as shown in Figure 1. Our modifications to the textual template attempt to improve integration with visual GRL diagrams through:

1. The addition of the scale concept from Planguage, which specifies the metric used for measurement. An objective's contribution to another is then given in terms of the second (parent) objective's scale.
2. The specification of the objective's activity attribute in the past tense, since objectives represent a desired outcome rather than an activity.
3. The removal of the constraints and relations fields since these can be expressed diagrammatically.
4. The addition of the author field so that newly proposed objectives can be identified and traced.

Activity	Reduced
Object	TS&D Fabricated Structure Manufacturing
Focus	Lead Time
Magnitude	3 months
Scale	Time in months required to have parts manufactured from the inception of a new engine
Timeframe	1 year after system deployment
Scope	Transmissions Structures & Drives (TS&D) SCU
Author	John Smith (Component Engineer, TS&D)

Figure 1:Example GQM+Strategies Formalisation

An objective is satisfied when the specified magnitude is achieved within the specified timeframe. The contribution links going toward an objective specify how that magnitude will be achieved (or exceeded). If the contributions of the child objectives additively amount to meet or exceed the objective's specified magnitude, then the satisfaction of the objective can be considered more likely than if not.

Figure 2 shows the GRL notation that is used to represent the requirements and objectives. Other notations could be used on the condition that they support the same concepts.

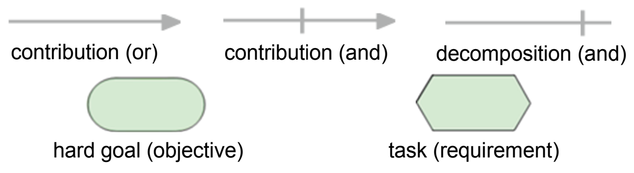


Figure 2: GRL Diagram Notation

In order to visualise the objectives specified with the GQM+Strategies template in a goal graph, we use GRL hard goal elements with the naming syntax: "Activity[Object Focus](magnitude)". We represent software requirements as tasks (i.e., the task of implementing the requirement) using the naming syntax: "{F/NF}[Requirement](Fit Criterion)", where "F/NF" is either Functional or Non-Functional, "Requirement" is a short headline version of the requirement description, and "Fit Criterion" is the short-hand version of the metric used to test the requirement's satisfaction.

A contribution link between a requirement and an objective specifies that the satisfaction of the requirement (tested by its fit criterion) will achieve some satisfaction of the objective, where the extent of the satisfaction is defined by the contribution specified by the link. A link between two objectives is similar, except for that the satisfaction of an objective is measured by its magnitude rather than by a fit criterion. An "OR" contribution specifies that if there are multiple "OR" links, a decision has to be made about which should be satisfied. An "AND" contribution specifies that all "AND" links are required for the objective to be satisfied. A decomposition link decomposes a requirement into a more specific requirement, much like SysML's "deriveReqt" link stereotype [8]. Figure 3 shows an example diagram produced by the methodology, which demonstrates the usage of the elements in Figure 2 to explore and visualise the strategic alignment of three high-level software requirements.

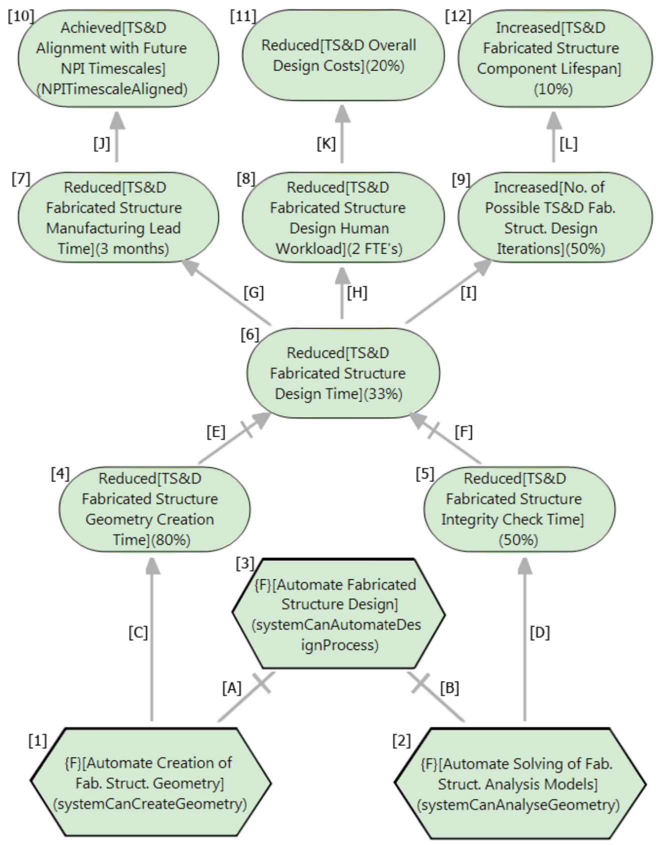

[10] Achieved[TS&D Alignment with Future NPI Timescales](NPITimescaleAligned)
[11] Reduced[TS&D Overall Design Costs](20%)
[12] Increased[TS&D Fabricated Structure Component Lifespan](10%)
[J]
[K]
[L]
[7] Reduced[TS&D Fabricated Structure Manufacturing Lead Time](3 months)
[8] Reduced[TS&D Fabricated Structure Design Human Workload](2 FTE's)
[9] Increased[No. of Possible TS&D Fab. Struct. Design Iterations](50%)
[G]
[H]
[I]
[6] Reduced[TS&D Fabricated Structure Design Time](33%)
[E]
[F]
[4] Reduced[TS&D Fabricated Structure Geometry Creation Time](80%)
[5] Reduced[TS&D Fabricated Structure Integrity Check Time](50%)
[3] {F}[Automate Fabricated Structure Design](systemCanAutomateDesignProcess)
[C]
[D]
[A]
[B]
[1] {F}[Automate Creation of Fab. Struct. Geometry](systemCanCreateGeometry)
[2] {F}[Automate Solving of Fab. Struct. Analysis Models](systemCanAnalyseGeometry)

Figure 3: Example Strategic Alignment Diagram

The high-level software requirement (3) in Figure 3 is decomposed to two lower level software requirements (1 & 2) to represent the hierarchy of requirement abstraction. Such refinements through decomposition links will continue until the lowest level of requirements are represented. The decomposed requirements (1 & 2) then link to objectives (4 & 5) with contribution links in order to represent what those requirements hope to achieve. The contribution links (E & F) are of the "AND" type, since both objectives (4 & 5) are required if objective (6) is to be satisfied.

Table 1 shows a sample of the quantifications that complement the diagram. They have been separated out of the goal graph due to space constraints, but ordinarily would be annotated on the edges (connecting links) of the graph.

TABLE 1: QUANTIFIED CONTRIBUTIONS

Link	[Contribution] [Activity] [Scale]	Confidence
C (1→4)	[80%] [Reduction] in [Geometry Creation Time]	1
D (2→5)	[50%] [Reduction] in [Integrity Check Time]	0.75
E (4→6)	[20%] [Reduction] in [Time Required to Design]	1
F (5→6)	[13%] [Reduction] in [Time Required to Design]	0.75
G (6→7)	[3 months] [Reduction] in [Manufacturing Lead Time]	0.75

The quantified contributions in Table 1 tell us that objective (4) will be satisfied if requirement (1) is satisfied, since objective (4)'s required magnitude of satisfaction (80%) will be contributed by link (C) (80%). It is important to note that where percentages are used as contribution weights on links, this does not infer that a certain percentage of the objective's magnitude will be achieved (in this case, 80% of 80%). Instead, the focus of the objective (e.g., geometry creation time) will be affected by that percentage in the context of the activity (e.g., a reduction by 80%). Objective (4) is then abstracted until the benefits are expressed in terms of high-level business objectives, which disambiguates estimated business value by placing the quantifications into context (i.e., a large saving from a small cost may be less than a small saving from a large cost).

Confidence levels allow users to represent how sure they are that achieving the first objective (or requirement) affects the second objective by at least the specified contribution.

TABLE 2: CONFIDENCE LEVEL ENUMERATIONS

Confidence	Description
0.25	Poor credibility, no supporting evidence or calculations, high doubt about capability
0.5	Average credibility, no evidence but reliable calculations, some doubt about capability
0.75	Great credibility, reliable secondary sources of evidence, small doubt about capability
1	Perfect credibility, multiple primary sources of evidence, no doubt about capability

The confidence level concept is similar to that used by Gilb for impact estimation [23], so we base our confidence levels on a similar scale in Table 2. Basic confidence adjustment can be performed by multiplying contributions by their associated confidence level so that users are reminded of the impact confidence has on estimations. For example, when confidence levels are taken into consideration in Table 1, the satisfaction of requirement (1) still leads to the full satisfaction of objective (4). However, when confidence levels are considered for links (E & F), the satisfaction of objective (6) is in doubt, since (20*1) + (13*0.75) is less than the 33% required by the objective's magnitude attribute. Additional confidence levels can be applied to the user's estimations to represent how qualified that user is at providing estimations. For example, someone who has implemented similar systems should be able to provide more accurate estimations than someone who has not. The accuracy of previous estimates made by that person could also be considered in order to improve the reliability of the estimations (i.e., calibration of the confidence levels).

By traversing the quantified GRL goal graphs, the business value of a requirement can be calculated by the contribution it makes to business objectives. This calculation can be automated by using a graph traversal algorithm (e.g., depth-first search) to calculate how much a given requirement contributes to a business objective. This calculation could then be used to improve the outcome of requirements prioritisation methods such as the Analytics Hierarchy Process [27], since such pairwise methods depend on the practitioners understanding of a requirement's value.

It is important to note that software engineers and business analysts may not know the objectives (or the goals and visions, for that matter) at different levels of the business (i.e., the project, the business unit, the department, the overall business, etc.). Therefore, managers should work with stakeholders to define the business objectives before the requirements can be abstracted toward them. Indeed, it is likely that some software requirements will be abstracted toward business objectives that were not previously elicited.

Where typical goal abstraction (asking "why?") would allow a non-specific goal such as "improve the engine", this method requires the user to be specific in how the engine is to be improved by asking for the metric that will be affected, e.g., "component lifespan" from objective (12) in Figure 3. Users may resist quantifying benefits of requirements, especially for non-functional requirements where the subject may be intangible, however, Gilb has found that it has always been possible to do so in his experience (e.g., by polling customers to quantify customer satisfaction) and has provided guidance on doing so in [23]. Even if the magnitude cannot be elicited at first, providing a scale by which the objective's success will be measured improves the definition of the objective by reducing ambiguity.

We suggest that this methodology should be performed after the high-level requirements have been elicited, so that resources are not wasted eliciting lower level requirements that do not align well to business strategy.

Tool support (GoalViz) has been developed (free to download at [28]) to support the methodology through:

- Input support for the requirement and objective templates with prompt question generation.
- Automatic diagram drawing to focus the user on the methodology and data rather than the graph layout.
- Automatic evaluation and summarisation of chains of links to enable efficient understanding.
- Project libraries to facilitate learning about the estimated contributions made in previous projects to improve future quantification confidence levels.

- What-if analysis allowing comparison of outcomes for different inputs where there is some uncertainty.

V. Conclusion and Future Work

This paper's unique contribution is twofold. First, we have shown how quantified goal graphs can be used to visualise the alignment of software requirements to business objectives. We have shown that in order to demonstrate strategic alignment, a chain of objectives may contain different measurement scales, and, since strategic alignment is based on estimated benefit, confidence in the estimations should be made explicit. Secondly, we have shown how goal link contribution scores can be made testable by specifying them in terms of the estimated effect they will have on the parent goal's measurement scale. Our methodology not only facilitates disambiguation of a requirement's business value, but more importantly, it requires that the needs of the business (i.e., business objectives) are related to requirements to ensure that the software can add value to the business. Since the requirements are abstracted to several levels of objectives, the problem to be solved will have been defined even if the requirement was originally solution oriented.

Future work will evaluate this approach against the related work detailed in Section III within different industrial settings to examine its benefit in a range of domains. We also intend to evaluate integration with SysML [8] to improve traceability to the design that will realise the requirements.

References

[1] "CHAOS Chronicles v3.0," Standish Group International, 2003.

[2] M. Jackson, *Software requirements & specifications: a lexicon of practice, principles, and prejudices*. ACM Press, 1995.

[3] International Telecommunication Union, "Z.151 : User requirements notation (URN) - Language definition." [Online]. Available: http://www.itu.int/rec/T-REC-Z.151/en. [Accessed: 02-Jul-2012].

[4] A. Van Lamsweerde, "Reasoning about alternative requirements options," *Conceptual Modeling: Foundations and Applications*, pp. 380–397, 2009.

[5] B. Boehm, "Value-based software engineering," *ACM SIGSOFT Software Engineering Notes*, vol. 2, no. 28, pp. 3–15, 2003.

[6] D. T. Ross and K. E. Schoman Jr, "Structured analysis for requirements definition," *IEEE Transactions on Software Engineering*, no. 1, pp. 6–15, 1977.

[7] A. Goknil, I. Kurtev, and K. van den Berg, "A metamodeling approach for reasoning about requirements," in *Model Driven Architecture–Foundations and Applications*, 2008, pp. 310–325.

[8] M. S. Soares and J. Vrancken, "Model-driven user requirements specification using SysML," *Journal of Software*, vol. 3, no. 6, pp. 57–68, 2008.

[9] S. Roberson and J. Robertson, "Volere: Requirements Specifcation Template." The Atlantic Systems Guild, 2012.

[10] The Institute of Electrical and Electronics Engineers, *IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications*. New York: IEEE-SA Standards Board, 1998.

[11] A. Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," *Fifth IEEE International Symposium on Requirements Engineering, 2001. Proceedings.*, pp. 249–262, 2001.

[12] Object Management Group, "BMM 1.1." [Online]. Available: http://www.omg.org/spec/BMM/1.1/. [Accessed: 16-Mar-2012].

[13] I. Alexander, "10 small steps to better requirements," *Software, IEEE*, vol. 23, no. 2, pp. 19 – 21, Apr. 2006.

[14] K. F. Cross and R. L. Lynch, "The 'SMART' way to define and sustain success," *National Productivity Review*, vol. 8, no. 1, pp. 23–33, 1988.

[15] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Sci. Comput. Program.*, vol. 20, no. 1–2, pp. 3–50, Apr. 1993.

[16] E. S. K. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, 1997, pp. 226–235.

[17] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu, "Evaluating goal models within the goal-oriented requirement language," *International Journal of Intelligent Systems*, vol. 25, no. 8, pp. 841–877, 2010.

[18] A. Pourshahid, D. Amyot, L. Peyton, S. Ghanavati, P. Chen, M. Weiss, and A. J. Forster, "Business process management with the user requirements notation," *Electronic Commerce Research*, vol. 9, no. 4, pp. 269–316, Aug. 2009.

[19] J. R. Hauser and D. Clausing, "The house of quality," *Harvard Business Review*, pp. 63–73, 1988.

[20] W. Heaven and E. Letier, "Simulating and optimising design decisions in quantitative goal models," in *Requirements Engineering Conference (RE), 2011 19th IEEE International*, 2011, pp. 79–88.

[21] S. J. Bleistein, K. Cox, J. Verner, and K. T. Phalp, "B-SCP: A requirements analysis framework for validating strategic alignment of organizational IT based on strategy, context, and process," *Information and Software Technology*, vol. 48, no. 9, pp. 846–868, Sep. 2006.

[22] R. S. Kaplan and D. P. Norton, "Linking the balanced scorecard to strategy," *California management review*, vol. 39, no. 1, 1996.

[23] T. Gilb, *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Butterworth-Heinemann Ltd, 2005.

[24] V. Basili, J. Heidrich, M. Lindvall, J. Münch, M. Regardie, D. Rombach, C. Seaman, and A. Trendowicz, "Bridging the gap between business strategy and software development," in *Twenty Eighth International Conference on Information Systems, Montreal, Canada*, 2007, pp. 1–16.

[25] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, "Cost models for future software life cycle processes: COCOMO 2.0," *Annals of Software Engineering*, vol. 1, no. 1, pp. 57–94, 1995.

[26] V. Mandić, V. Basili, L. Harjumaa, M. Oivo, and J. Markkula, "Utilizing GQM+Strategies for business value analysis: an approach for evaluating business goals," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, New York, NY, USA, 2010, pp. 20:1–20:10.

[27] K. Joachim, W. Claes, and R. Bjorn, "An evaluation of methods for prioritizing software requirements," *Information and Software Technology*, vol. 39, pp. 939–947, 1998.

[28] R. Ellis-Braithwaite, "GoalViz Tool," *GoalViz Tool*. [Online]. Available: http://www.goalviz.info/ICSEA2012/index.html. [Accessed: 15-Mar-2012].

A Constraint-based Method to Compute Semantics of Channel-based Coordination Models

Behnaz Changizi, Natallia Kokash
Leiden Institute of Advanced Computer Science (LIACS)
Leiden, The Netherlands
b.changizi@umail.leidenuniv.nl, nkokash@liacs.nl

Farhad Arbab
Centrum Wiskunde & Informatica (CWI)
Amsterdam, The Netherlands
farhad.arbab@cwi.nl

***Abstract*—Reo is an exogenous channel-based coordination language that acts as glue code to tie together software components and services. The building blocks of Reo models are connectors that impose constraints on the data-flow in component or service-based architectures in terms of data synchronization, buffering, mutual exclusion, etc. Several semantic models have been introduced to formalize the behavior of Reo. These models differ in terms of expressiveness, computation complexity and purposes that they serve. In this paper, we present a method and a tool for building formal automata-based semantics of Reo that unifies various aspects of existing semantics. We express the behavior of a Reo network as a mixed system of Boolean and numerical constraints constructed compositionally by conjuncting the assertions for its constituent parts. The solutions of this system are found with the help of off-the-shelf constraint solvers and are used to construct the constraint automaton with state memory that gives the sound and complete semantics of Reo with respect to existing models. Our approach is more efficient compared to the existing methods for generating formal semantics of Reo connectors.**

***Keywords*-formal semantics; Reo; constraint automata; coloring semantics; constraint solving.**

I. Introduction

Service-oriented architecture [1] (SOA) is an indispensable solution for many of todays' problems. The SOA implementation depends on a mesh of functionality units, called services. Services are loosely coupled and do not invoke or communicate with each other directly. Instead, they employ a pre-defined protocol, which specifies the way they can exchange messages amongst themselves. As a result, the correctness of a SOA implementation relies not only on the correctness of its involved services but also on the properness of its communication protocol.

Coordination languages and models provide dedicated frameworks to study the communication protocols as separate concerns. They define the "glue code" that ties together the services to enable the message passing among the involved services. Some recent coordination models include: i) a Calculus for Orchestration of Web Services (COWS) [2], which specifies the combination of service-oriented applications and models their dynamic behavior; ii) Orc [3], a process calculus for distributed and concurrent programming which provides uniform access to computational services, including distributed communication and data manipulation; and iii) Reo [4], an exogenous coordination language that realizes the coordination patterns in terms of its complex *connector*s, also called *network*s, that are built out of simple primitives called *channel*s. In the sequel, we focus on Reo.

Each channel in Reo defines a form of coordination in terms of synchronizing, buffering, retaining data, etc., along with constraining its input and output data items. Reo allows hierarchical modeling where arbitrarily complex connectors can be formed out of simpler networks. In our previous work [5] [6], we have presented the suitability of Reo to model behavioral patterns describable by business process models. We have also developed tools for automatic transformation of these models into Reo [6]. This enables the use of Reo analysis methods and tools on the coordination protocols that originally were not expressed in Reo.

To perform formal analysis on Reo networks, formal semantics of these models are necessary. Several operational semantics have been proposed for Reo [7] with various styles of I/O streams [4], automata, coloring [8] and constraints [9]. The most basic automata-based semantics of Reo is Constraint Automata (CA) [10]. An advantage of CA and its extensions is their support for data-constraints that are part of the coordination primitives in Reo. This is in contrast with the coloring semantics that abstracts data-flow and expresses the behavior of a connector only in terms of existence or lack of data-flow.

Constraint Automata with State Memory (CASM) [11] is an extension of CA that due to its state abstraction and data-awareness, is suitable as a more compact semantic model for Reo in model checking. In this paper, we present a constraint-based technique and a tool to generate CASMs from Reo networks in a compositional manner.

A shortcoming of earlier work stems from its lack of support for data-dependent behavior. We overcome this shortcoming in the work we present in this paper. Our tool is a necessary step for providing fully automated model checking for data-aware and context-dependent composition of services coordinated by Reo.

The rest of this paper is organized as follows. In Section 2, we explain the basics of Reo. In Section 3, we introduce

Table I: Graphical Representation of the Most Frequently Used Reo Channels

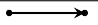
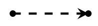
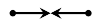
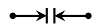
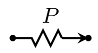
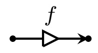
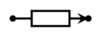
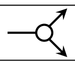
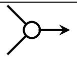
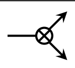
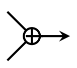
	A *Sync* channel accepts data from its source end iff it can dispense it simultaneously through its sink end.
	A *LossySync* reads a data item from its source end and writes it simultaneously to its sink end. If the sink end is not ready to accept the data item, the channel loses it.
	A *SyncDrain* reads data to discard through its two source ends iff both ends are ready to interact simultaneously.
	An *AsyncDrain* accepts and discards a data item from either of its source ends that offers one. If both ends offer data items simultaneously, then the channel chooses one non-deterministically.
	A *Filter* accepts a data item that does not match its predefined filter pattern P from its source and loses it. For a data item that matches its filter pattern P, a filter channel behaves as a *Sync* channel; it accepts the data item iff it can dispense it simultaneously through its sink end.
	A *Transformer* acquires data at its source end, applies a predefined transform function f on it and simultaneously writes the result to its sink end, iff the data item is in the domain of the function f. Otherwise, the channel loses the data item.
	If a *FIFO*$_1$ is empty, it accepts incoming data from its source end and buffers it. Being full, the channel is ready to dispense data through its sink end and become empty. Because this channel has a buffer capacity of one data item, it is either full or empty at any given time, and thus the ends of this channel cannot interact simultaneously.

Table II: Graphical Representation of Reo Nodes and Two Frequently Used Components

	A *Replicator* replicates the incoming data of its source to its sink ends simultaneously.
	A *Merger* non-deterministically chooses one of its source ends that is ready to communicate, take its incoming data item and writes it to its sink end.
	A *Router* accepts data from its source end and simultaneously writes it on one of its non-deterministically chosen sink ends that is ready to accept the data.
	A *Cross-product* reads one incoming data item from each of its incoming source ends, forms a tuple in which the data elements are set in the counter-clock-wise order with respect to its sink node, and simultaneously writes the resulting tuple on its sink end.

constraint automata with state memory. In Section 4, we formalize the mentioned semantic model of Reo in systems of constraints along with techniques to solve them. In Section 5, we show how we abstract from the internal ports in a Reo network in a minimized representation. In Section 6, we briefly discuss some properties of our presented constraint encoding of Reo networks. Finally, in Section 7, we conclude the paper and outline our future work.

II. Reo

Reo [4] is an exogenous channel-based coordination language, which can act as "glue code" to tie together software components and services. The building blocks of Reo models, *connector*s, impose constraints on the data-flow in terms of synchronization, buffering, mutual exclusion, etc. Every connector contains some *primitive*s. The set of Reo primitives is open-ended, meaning that a user can define new primitives and extend the expressiveness of Reo.

The simplest Reo connector is a *channel* that has two *end*s, also called *port*s. Channel ends are either of type *source* that reads data into the channel or *sink* that writes the channel's data out. Table I shows the most commonly used channels in Reo.

Reo nodes connect channels to each other to form Reo connectors, also called *circuit*s or *network*s. Depending on whether all channel ends that coincide on a node are source ends, sink ends or a combination of both, nodes become *source*, *sink* or *mixed* nodes. A source node behaves as a synchronous *Replicator* that replicates the incoming data of its source to its sink ends simultaneously, while a sink node acts as a non-deterministic *Merger* that combines the flows of its source ends to its sink end. A mixed node is an atomic combination of a replicator and a non-deterministic merger. Each read and write action needs all of its involved source and sink ends to be able to interact synchronously; otherwise, the action cannot take place. Reo also allows hierarchical modeling and abstraction from inner structures by means of components. A component can be written in any programming language. Moreover, a connector can be converted into a component, exhibiting (part of) its inner logic as an observable behavioral interface. Table II shows the graphical representation of the Reo nodes and two frequently used components, *Router* and *Cross-product*.

A. Formal Semantics of Reo

Coordination patterns that a Reo network imposes on data-flow define the network's *behavior*. Formal analysis of a Reo network requires formal modeling of its behavior. The behavior of a Reo network consists of various dimensions that involve:

- *State*: The states of elements forming a Reo network at any point in time define the state, also called *configuration*, of the network. The network state affects and is affected by the data-flow at each step.
- *Synchronization/Exclusion*: The term *synchronization* refers to atomic concurrent data-flow through ports and nodes. Depending on its constituents and their arrangement inside the network, in each configuration, a Reo network allows, requires, or forbids a group of ports to synchronize. We consider *mutual exclusion* as a special case of synchronization.
- *Data-dependent flow*: The value of data items that the ports of a Reo network exchange can affect the network behavior, particularly if the network contains elements, such as filter or transformer channels, that allow or forbid exchange of special data values.
- *Context dependency*: The choices some Reo elements can take change non-monotonically as the context changes [12]. A context-dependent semantics of Reo

defines context in terms of presence or absence of pending I/O requests on boundary primitive ends. As an example of such behavior consider the original description of a *LossySync* channel, which writes on its sink end the data item it reads from its source end. The channel can lose a data item only if its sink end is not write-enabled [4].

- *Timing*: The behavior of a Reo network can be subject to time constraints. For example, we can formalize a deadline for availability of some data using a FIFO_1 channel, which associates an expiration time with the data that it buffers. If the channel's sink end does not dispense a buffered data item before its expiration, the channel loses it [13].
- *Priority*: Presence of a prioritized element in a Reo network can influence some non-deterministic synchronization choices in the network by favoring data-flow through ports related to that prioritized element.

B. Related work and motivation

Several formal semantic models have been proposed to specify the behavior of Reo. An extensive overview and comparison of these models is presented in [7].

Connector coloring (CC) [8] is a formal semantics for Reo that describes the behavior of a connector by assigning different colors to its ports to designate presence or absence of data-flow. CC accounts for synchronization and context dependency. This model captures context dependency by propagating negative information about the absence of data-flow inside the network.

The majority of Reo semantic models are based on automata. These semantic models allow the semantics of a large network to be constructed from the given automata for its constituents using the *product* of automata. The earliest automata-based Reo semantics is Constraint Automata (CA) [10] that accommodate synchronization, states, and data-dependent flow. Transitions in CA contain the set of synchronized ports and constraints over data that the ports exchange. An extension of CA, Constraint Automata with State Memory (CASM) [11] elaborates on states by introducing state memory cells and extends data constraints to accommodate them.

Extending CA with clock assignments and timing constraints, Timed Constraint Automata (TCA) [13] express the time-aware aspect of Reo networks. A SAT-based approach for bounded model checking of TCA is presented in [14]. Another extension of CA, Constraint Automata with Priority (CAP) [15] supports the propagation of prioritized requests.

The most recent semantics of Reo [9] that is based on constraint solving, deals with synchronization, data-, and context-dependency. It uses different constraints for each of these notions, which are added conjunctively to capture their composition. This approach is the basis of the *DREAMS* [9] execution engine for Reo. Although in theory, this semantics accounts for data-dependency, the proposed implementation ignores data. This leads to incorrect results when dealing with data-sensitive elements such as *Filter* channel.

A translation of CA and CC to a process algebraic specification language called mCRL2 [16] has been done [17]. The mCRL2 toolset compiles the specifications into Labeled Transition Systems (LTS)s, which can be verified using the mCRL2 toolset. This approach represents certain aspects of existing semantic models for Reo in an automata-based form. For instance, it models the context dependency in the behavior of Reo networks using LTSs with labels corresponding to colors in CC.

Despite the abundance of the semantics that capture different aspects of Reo, there are expressiveness gaps among them. For instance, a network with priority dependent and time sensitive behavior cannot be readily expressed by any of the mentioned formal semantics. In this paper, we present a unified symbolic constraint-based framework, where these different semantics coexist. This framework encodes the behavior of Reo networks in terms of constraints whose solutions form an existing automata-based semantic model of Reo. Unlike [17], our framework treats data symbolically, so that the state space does not blow up. Our framework extends *DREAMS* with time and priority. It also provides tool support using existing constraint solving tools to incorporate data and time constraints.

For a given configuration of a Reo network, *DREAMS* uses constraint solving to compute only a single solution for synchronous data-flow in this configuration, which leads to the next configuration. In contrast, our approach considers all solutions and configurations, which in turn enable us to map the solutions to an automata-based semantics that is well suited for model checking.

Generating the CA corresponding to a Reo network from the CA of its constituents using the CA *product* operator [10], is computationally expensive and memory inefficient. The complexity of composing m CA each having n transitions is $\mathcal{O}(n^m)$. However, we can convert this problem to the NP-complete problem of SAT-solving for which many efficient solvers exist.

Contribution: Instead of generating the CA corresponding to a Reo network directly by composing the CAs of its constituents, we encode each constituent CA into constraints whose conjunction forms the system of constraints that captures the full behavior of the network. The solutions of this system of constraints describe the behavior of the network in terms of variables representing different aspects of the behavior of Reo, such as synchronization, data-dependent flow, etc. From these solutions, we generate the CA corresponding to the network. We show that our solution is more efficient compared to the previous attempts in generating the formal semantics of a Reo network.

We have developed a tool to automate our approach that is integrated in the Extensible Coordination Tools (ECT)

[18]. ECT is a framework to design, develop, model check, test, and execute component-based software modeled by the Reo coordination language. The tools in this framework are integrated as Eclipse plug-ins and operate based on the operational semantics of Reo, most notably, connector coloring and constraint automata. Our tool is the only tool that supports propagation of priorities in Reo networks. Furthermore, it unifies various behavioral aspects of Reo networks in a single encoding, which enables analysis of networks whose behavior spans over several existing semantic models.

III. Constraint Automata with State Memory

Constraint Automata with State Memory (CASM) [11] extends CA with variables that represent local memory cells of the states of the automata. Due to its elaboration on state information, we choose CASM as the main semantic model of Reo that our framework generates.

Definition 3.1 (Constraint Automaton with State Memory): A constraint automaton with state memory (CASM) is a tuple $A = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$ where

- Q is a finite set of states.
- $\mathcal{N}$ is a finite set of names.
- $\rightarrow$ is a finite subset of $Q \times 2^{\mathcal{N}} \times DC(\mathcal{N}, \mathcal{M}, \mathcal{D}) \times Q$ is the transition relation of A, where $DC(\mathcal{N}, \mathcal{M}, \mathcal{D})$ is the set of data constraints, defined below.
- $q_0 \in Q$ is an initial state.
- $\mathcal{M}$ is a set of memory cell names, where $\mathcal{N} \cap \mathcal{M} = \emptyset$.

Every $n \in \mathcal{N}$ represents a node in a Reo connector. The set $\mathcal{N}$ is partitioned into three mutually disjoint sets of source nodes $\mathcal{N}^{src}$, mixed nodes $\mathcal{N}^{mix}$, and sink nodes $\mathcal{N}^{snk}$. Because we make the replication and merge inherent in Reo nodes explicit as *replicator* and *merger* primitives (in Table II), at most two primitive ends coincide on every node $n \in \mathcal{N}$. Thus, it follows that a source or a sink node contains only a single (source or sink) primitive end, and a mixed node contains exactly one source and one sink primitive ends.

We write $q \xrightarrow{N,g} p$ instead of $(q, N, g, p) \in \rightarrow$. For every transition $q \xrightarrow{N,g} p$, we require that $g \in DC(N, \mathcal{M}, \mathcal{D})$, where $\mathcal{D}$ is the global set of numerical data values and $DC(N, \mathcal{M}, \mathcal{D})$ is the language defined by the following grammar:

$$g \ ::= \ \textit{true} \ \mid \ \neg\, g \ \mid \ g \wedge g \ \mid \ u = u \ \mid \ u < u,$$

$$u \ ::= \ d(n) \ \mid \ m' \ \mid \ m \ \mid \ v.$$

In this grammar, $=$ is the symmetric equality relation, $<$ is a total order relation, $n \in N \subseteq \mathcal{N}$ denotes a node name, $d(n)$ represents the data item exchanged through the node n, $m \in \mathcal{M}$ correspond to a memory cell in the current state, which is the source state of the transition, m' stands for the memory cell $m \in \mathcal{M}$ in the next state, which is the target state of the transition, and $v \in \mathcal{D}$. As usual, *false* stands for $\neg$*true*, $x > y$ stands for $y < x$, and other logical operators, such as $\vee$ and $\Rightarrow$ (the implication symbol) can be built from the given operators.

Transitions with data constraints that can be reduced to *false* using the Boolean laws are impossible and we omit them. A data constraint g that is always *true* can be left out. We use $\mathcal{M}_g$ to represent the set of all $m \in \mathcal{M}$ that syntactically appear as m in a data constraint g; and $\mathcal{M}'_g$ to refer to the set of all $m \in \mathcal{M}$ that syntactically appear as m' in g. The valuation function $\mathcal{V}_q : \mathcal{M} \rightarrow 2^{\mathcal{D}}$ designates the set of values $\mathcal{V}_q(m)$ of a memory cell $m \in \mathcal{M}$ in a state $q \in Q$, where $\mathcal{V}_{q_0}(m) = \emptyset$ for all $m \in \mathcal{M}$.

A transition $q \xrightarrow{N,g} p$ in a given constraint automaton with state memory is possible only if there exists a substitution for every syntactic element $d(n)$, m, and m' that appears in g to satisfy g. A substitution simultaneously replaces in g:

- every occurrence of $d(n)$ with the data value exchanged through the node $n \in \mathcal{N}$;
- every occurrence of m' of every $m \in \mathcal{M}$ with a value $v \in \mathcal{D}$;
- every occurrence $m \in \mathcal{M}$ with:
  - the special symbol $'\circ'$ if $\mathcal{V}_q(m) = \emptyset$
  - a value $v \in \mathcal{V}_q(m)$, otherwise.

The guard g is satisfied if proper replacement values can be found to make g *true*. Making this transition, the automaton defines the valuation function $\mathcal{V}_p$ for the target state p, as follows: for every $m \in \mathcal{M}'_g$, $\mathcal{V}_p(m)$ is the set of all $v \in \mathcal{D}$ whose replacements for m' satisfy g. For every other $m \in \mathcal{M}$, $\mathcal{V}_p(m) = \emptyset$.

A relational operator evaluates to *true* only if the values of its operands are in its respective relation. Thus, any operator with one or more $\circ$ as an operand always evaluates to *false*. We call a CASM, *normalized* iff a) it does not have two states with the same set of state memory variables, and b) every two transitions differ at least in their start states, their target states, or their sets of synchronizing ports. For any arbitrary CASM that is not normalized, we can normalize it by a) introducing auxiliary variables, to make the set of state memory variables unique for each state, and b) by merging the transitions that have the same start and target states and synchronize the same ports. In the sequel, we consider only *normalized* CASMs.

Following are the definitions for *product* and *hiding* operations on CASM. Both definitions are adapted from [10].

Definition 3.2 (Product-automaton): For the CASMs $\mathcal{A}_1 = (Q_1, \mathcal{N}_1, \rightarrow_1, q_{0,1}, \mathcal{M}_1)$ and $\mathcal{A}_2 = (Q_2, \mathcal{N}_2, \rightarrow_2, q_{0,2}, \mathcal{M}_2)$, their product is defined as:

$$\mathcal{A}_1 \bowtie \mathcal{A}_2 = (Q_1 \times Q_2, \mathcal{N}_1 \cup \mathcal{N}_2, \rightarrow, q_{0,1} \times q_{0,2}, \mathcal{M}_1 \cup \mathcal{M}_2)$$

where the following rules define the transition relation $\rightarrow$:

$$\frac{q_1 \xrightarrow{N_1, g_1}_1 p_1,\ q_2 \xrightarrow{N_2, g_2}_2 p_2,\ N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle p_1, p_2 \rangle}$$

$$\frac{q_1 \xrightarrow{N_1,g_1}{}_1 p_1, N_1 \cap \mathcal{N}_2 = \emptyset}{\langle q_1, q_2\rangle \xrightarrow{N,g} \langle p_1, q_2\rangle} \qquad \frac{q_2 \xrightarrow{N_2,g_2}{}_2 p_2, \mathcal{N}_1 \cap N_2 = \emptyset}{\langle q_1, q_2\rangle \xrightarrow{N,g} \langle q_1, p_2\rangle}$$

We can abstract from the data-flow on certain Reo nodes using the *hiding* operator defined as follows:

Definition 3.3 (Hiding): Let $\mathcal{A} = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$ be a constraint automaton and $C \in \mathcal{N}$. The constraint automaton that results from hiding the node C in automaton $\mathcal{A}$ is $\exists C\,[\mathcal{A}] = (Q, \mathcal{N}\backslash\{C\}, \rightarrow_C, q_0, \mathcal{M})$ and the transition relation $\longrightarrow_C$ is defined as follows:

$$\frac{p \xrightarrow{N,g} q, N' = N\backslash\{C\}, g' = \exists C\,[g]}{p \xrightarrow{N',g'}{}_C\ q}, \text{ where}$$

$$\exists C\,[g] = \bigvee_{d \in \mathcal{D}} g\,[d\,(C)\,/d]\,.$$

IV. Connector Coloring

The connector coloring semantics [8] denotes the existence or absence of data-flow through the primitive ends by marking them with different colors. Let $\mathcal{C}olors$ be the set of colors. A set of two colors, $\mathcal{C}olors = \{$—, - -$\}$, where — denotes an occurrence and - - represents an absence of data-flow is adequate to express the formal semantics of many Reo networks. However, this two-color set cannot express the semantics of some Reo networks.

A traditional example of such a network is when the sink end of a *LossySync* channel connects to an empty *FIFO*$_1$ channel; in this case, the semantics of this network according to the two-color set includes the case where the *LossySync* loses its incoming data item, while the *FIFO*$_1$ channel is empty. This is an unacceptable behavior for a so-called context sensitive *LossySync* channel: it must lose its incoming data only if its sink end cannot dispense it. In the sequel, when we refer to a *LossySync* we mean its context sensitive version.

The three coloring semantics, $\mathcal{C}olors = \{-, \triangleleft, \triangleright\}$, addresses this problem by propagating negative information regarding the absence of data-flow: it replaces - - with $\triangleleft$ and $\triangleright$ meaning that the associated primitive end, respectively, *provides* or *requires* a reason for no-flow. Considering that no-flow can occur only when at least one of the involved primitive ends *provides* a reason for it, and that an empty *FIFO*$_1$ cannot *provide* a reason for no-flow on its source end, the invalid behavior described above does not arise in the three coloring semantics.

Definition 4.1 (Coloring): A coloring $l : \mathcal{P} \rightarrow \mathcal{C}olors$ is a total function from the primitive ends to a set of colors. We refer to the global set of colorings as $\mathcal{L}$.

Definition 4.2 (Coloring Composition): The composition of colorings l_1 and l_2, denoted $l_1 \bullet l_2$, is defined as:

$l_1 \bullet l_2 = \{c_1 \cup c_2 | c_1 \in l_1, c_2 \in l_2, p_1 \in dom(c_1),$
$p_2 \in dom(c_2), p_1$ and p_2 are the source and sink ends of a node $n, \neg(c_1(p_1) = \triangleleft \wedge c_2(P_2) = \triangleright)\}$

Definition 4.3 (Next function): The next function $\eta : \mathcal{L} \rightarrow 2^{\mathcal{L}}$ maps a coloring to a set of colorings, which can succeed it.

Definition 4.4 (Coloring Semantics): A coloring semantics of a Reo network is a tuple $CC = \langle \mathcal{P}, 2^{\mathcal{L}}, l_0, \eta\rangle$, where:

- $\mathcal{P}$ is the set of primitive ends,
- $l_0 \in \mathcal{L}$ is the initial set of possible colorings,
- $2^{\mathcal{L}}$ is a set of colorings,
- η is a next function that maps a coloring to a set of colorings.

V. Reo Constraint Satisfaction Problem

In Section II-A, we presented an overview of the various behavioral dimensions of a Reo network. We extend the constraint-based framework in [9] to incorporate all behavioral dimensions addressed by various semantic models for Reo. In our framework, we denote each of these elements by variables over their proper domains. We relate these variables to each other and restrict possible values they can assume using constraints whose solutions give the underlying formal semantics of the network. In the sequel, we deal only with networks whose semantics can be expressed in CASM or CC. However, we are currently extending our framework to also support timing and priority.

Let $\mathcal{N} = \mathcal{N}^{src} \cup \mathcal{N}^{mix} \cup \mathcal{N}^{snk}$ be the global set of nodes, $\mathcal{M}$ the global set of state memory variables, and $\mathcal{D}$ the global set of numerical data values. The set of primitive ends $\mathcal{P}$ consists of all primitive ends p derived from $\mathcal{N}$ by marking its elements with superscripts c and k, according to the following grammar:

$$p ::= r^c \mid s^k$$

where $r \in \mathcal{N}^{src} \cup \mathcal{N}^{mix}$ and $s \in \mathcal{N}^{snk} \cup \mathcal{N}^{mix}$. Observe that the primitive ends n^c and n^k connect on the common node n.

Let $p \in \mathcal{P}$, $n \in \mathcal{N}$ and $m \in \mathcal{M}$ be a primitive end, a node, and a state memory variable, respectively. A free variable v that occurs in the constraints encoding the behavior of a Reo network has one of the following forms:

- $\tilde{n}$ ranges over $\{\top, \bot\}$ to show presence or absence of flow on the node n.
- $\hat{n}$ ranges over $\mathcal{D}$ to represent the data value passing through the node n.
- $\mathring{m}, \mathring{m}'$ range over $\{\top, \bot\}$ to denote whether or not the state memory variable m is defined in, respectively, the source and the target states of the transition to which the encoded guard belongs.
- $\hat{m}, \hat{m}'$ range over $\mathcal{D}$ to represent the values of the state memory variable m in, respectively, the source and the target states of the transition to which the encoded guard belongs.
- $\overrightarrow{p}$ ranges over $\{\top, \bot\}$ to state that the reason for lack of data-flow through the primitive end p originates

from, respectively, the primitive to which p belongs or the context (of this primitive).

Note that not all of the introduced variables are required for encoding the behavior of every Reo network. In presence of context dependent primitives like *LossySync* or in priority-sensitive networks, constraints include variables of the form $\overrightarrow{p}$. For the stateful elements such as *FIFO*$_1$, variables like $\mathring{m}, \mathring{m}', \hat{m}$, and $\hat{m}'$ appear in the constraints.

Observe that the interpretation of some of the mentioned variables depends on the values of other variables. Referring to the variable $\overrightarrow{p}$ makes sense only if $\tilde{n} = \bot$, where $p = n^c$ or $p = n^k$ (i.e., the primitive end p belongs to the node n); and $\hat{n}$, $\hat{m}$ and $\hat{m}'$ make sense only if $\tilde{n} = \top$, $\mathring{m} = \top$ and $\mathring{m}' = \top$, respectively.

The grammar for a constraint Ψ encoding the behavior of a Reo network is as follows:

$$\begin{array}{rcll} t & ::= & \hat{n} \mid \hat{m} \mid \hat{m}' \mid d \mid t \circledast\ d & \text{(terms)} \\ a & ::= & \tilde{n} \mid \overrightarrow{p} \mid \mathring{m} \mid \mathring{m}' \mid t = t \mid t < t & \text{(atoms)} \\ \Psi & ::= & a \mid \neg\Psi \mid \Psi \wedge\ \Psi & \text{(formulae)} \end{array}$$

where $d \in \mathcal{D}$ is a constant, $\circledast \in \{+, -, *, /, \%, \hat{}\,\}$, and p is either of the form n^c or n^k.

Definition 5.1 (Reo Constraint Satisfaction Problem): A Reo Constraint Satisfaction Problem (RCSP) is a tuple $\langle \mathcal{P}, \mathcal{M}, M_0, \mathcal{V}, C \rangle$, where:

- $\mathcal{P}$ is a finite set of primitive ends.
- $\mathcal{M}$ is a finite set of state memory variables.
- $M_0 \subseteq \mathcal{M}$ is a set of state memory variables that define the initial configuration of a Reo network.
- $\mathcal{V}$ is a set of variables v defined by the grammar

  $$v ::= \tilde{n} \mid \overrightarrow{p} \mid \mathring{m} \mid \mathring{m}' \mid \hat{n} \mid \hat{m} \mid \hat{m}'$$

  for $n \in \mathcal{N}, p \in \mathcal{P}$, and $m \in \mathcal{M}$. The values that the variables of the forms $\hat{n}, \hat{m}$, and $\hat{m}'$ can assume are subsets of $\mathcal{D}$, and the other variables are Boolean, with values in $\{\top, \bot\}$.
- $C = \{C_1, C_2, ..., C_m\}$ is a finite set of constraints, where each C_i is a constraint given by the grammar Ψ involving a subset of variables $V_i \subseteq \mathcal{V}$.

Example 5.1: The RCSP of a *Sync* channel with the source end a and the sink end b is $\langle \{a, b\}, \emptyset, \emptyset, \{\tilde{a}, \tilde{b}, \hat{a}, \hat{b}\}, \tilde{a} \Leftrightarrow \tilde{b} \wedge \tilde{a} \Rightarrow (\hat{a} = \hat{b}) \rangle$. The solutions for this constraint problem give the behavior of the *Sync* channel as the channel allows data-flow on its source end iff its sink end can dispense it simultaneously (which agrees with the semantics of this channel as defined in other formal models of Reo). In case of data-flow, the values of the data items passing through the ends of this channel are equal.

We obtain the constraints corresponding to a Reo network by composing the RCSPs of its constituents as defined below.

Definition 5.2 (Composition): The composition of two RCSPs $\rho_1 = \langle \mathcal{P}_1, \mathcal{M}_1, M_{0,1}, \mathcal{V}_1, C_1 \rangle$ and $\rho_2 = \langle \mathcal{P}_2, \mathcal{M}_2, M_{0,2}, \mathcal{V}_2, C_2 \rangle$ is defined as follows:

$$\rho_1 \odot \rho_2 = \langle \mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{M}_1 \cup \mathcal{M}_2, M_{0,1} \cup M_{0,2}, \mathcal{V}_1 \cup \mathcal{V}_2, C_1 \wedge C_1 \rangle$$

However, connecting two Reo networks must not introduce incorrect data-flow possibilities. This is done by enforcing a restriction on the possible solutions through the following axiom:

Axiom 1 (Mixed node axiom): When two Reo networks connect on the common node x, where x^c is a source end in one network and x^k is a sink end in the other, the following constraint must hold:

$$\neg\tilde{x} \Leftrightarrow (\overrightarrow{x^c} \vee \overrightarrow{x^k})$$

The *mixed node axiom*, which applies to all mixed nodes in a network, states that a node x cannot produce the reason for no-flow all by itself.

A. Encoding Reo Elements in RCSPs

Table III summarizes the constraint encodings associated with commonly used Reo elements. If a Reo network does not contain any context dependent channel, the variables encoding the context dependency can be ignored in its RCSP. Table IV shows the encoding of Reo elements from Table III where the context variables are removed. Note that in these tables, a and b denote the source and the sink ends of a primitive, respectively, and that dom refers to the domain of the given function or predicate. In the case of elements with more than one source or sink ends, we use indices.

The intuition behind these constraints is that their solutions reflect the semantic model of each element as given by CASM and CC.

Example 5.2: Figure 1 shows a Reo network that consists of a *transformer* channel with the function $3 * \hat{a}$, whose domain is the set of numbers $Number$ and a *filter* channel with the condition $\hat{b}\%2 = 0$ and domain $Number$.

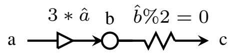


Figure 1: A Data-Aware Reo Network

Since none of the Reo primitives in Figure 1 is context dependent, we use the constraints corresponding to the primitives in this network as defined in Table IV.

Equation 1 states that flow occurs on the source end of the *transformer* channel iff it occurs on its sink end. In addition, flow can exist only if the data item that enters the source end of the channel is a number. In this case, the data item written on the sink end is three times the value of the source data item.

Equation 2 expresses that flow on the source end of the *filter* channel leads to flow on its sink end, iff the data item belongs to the channel's accepting pattern (which is

Table III: Context Dependent Encoding of Reo Primitives (Extending [9])

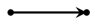
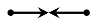
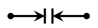
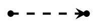
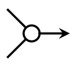
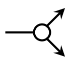
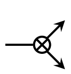
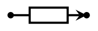
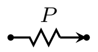
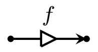
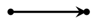
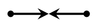
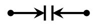
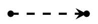
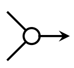
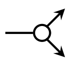
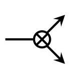
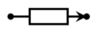
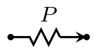
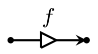
Channel	Constraints
	$\psi_{Sync}(a,b) : \tilde{a} \Leftrightarrow \tilde{b} \wedge \tilde{a} \Rightarrow (\hat{a} = \hat{b}) \wedge \neg(\overrightarrow{a^{c}} \wedge \overrightarrow{b^{k}})$
	$\psi_{SyncDrain}(a_1,a_2) : \tilde{a}_1 \Leftrightarrow \tilde{a}_2 \wedge \neg(\overrightarrow{a_1^{c}} \wedge \overrightarrow{a_2^{c}})$
	$\psi_{AsyncDrain}(a_1,a_2) : \tilde{a}_1 \Rightarrow (\neg\tilde{a}_2 \wedge \overrightarrow{a_2^{c}}) \wedge \tilde{a}_2 \Rightarrow (\neg\tilde{a}_1 \wedge \overrightarrow{a_1^{c}})$
	$\psi_{LossySync}(a,b) : \tilde{b} \Rightarrow \tilde{a} \wedge \tilde{b} \Rightarrow (\hat{a} = \hat{b}) \wedge \neg\overrightarrow{a^{c}} \wedge \neg\tilde{a} \Rightarrow \overrightarrow{b^{k}}$
	$\psi_{Merger}(a_{0..i},b) : \tilde{a}_i \Leftrightarrow \tilde{b} \wedge \tilde{a}_i \Rightarrow (\hat{a_i} = \hat{b}) \wedge \neg\tilde{b} \Rightarrow ((\neg\overrightarrow{b^{k}} \bigwedge_i \overrightarrow{a_i^{c}}) \vee (\overrightarrow{b^{k}} \wedge \neg\overrightarrow{a_i^{c}} \bigwedge_{j,j!=i} \overrightarrow{a_j^{k}}))$
	$\psi_{Replicator}(a,b_{0..i}) : \tilde{a} \Leftrightarrow \bigwedge_i \tilde{b}_i \wedge (\tilde{a} \Rightarrow \bigwedge_i(\hat{b}_i = \hat{a})) \wedge \neg\tilde{a} \Rightarrow ((\neg\overrightarrow{a^{c}} \bigwedge_i \overrightarrow{b_i^{k}}) \vee (\neg\overrightarrow{b_i^{k}} \bigwedge_{j,j\neq i} \overrightarrow{b_j^{k}} \wedge \overrightarrow{a^{c}}))$
	$\psi_{Router}(a,b_{0..i}) : \tilde{a} \Leftrightarrow (\bigvee_i \tilde{b}_i) \bigwedge_{j,j\neq i} \neg(\tilde{b}_i \wedge \tilde{b}_j) \wedge \tilde{b}_i \Rightarrow (\hat{b_i} = \hat{a}) \wedge \tilde{a} \Leftrightarrow (\neg\overrightarrow{a^{c}} \vee \neg(\bigvee_i \overrightarrow{b_i^{k}}))$
	$\psi_{FIFO_1}(a,b,m) : \tilde{a} \Rightarrow (\neg\mathring{m} \wedge \mathring{m}' \wedge (\hat{m}' = \hat{a})) \wedge \tilde{b} \Rightarrow (\mathring{m} \wedge \neg\mathring{m}' \wedge (\hat{m} = \hat{b})) \wedge (\neg\tilde{a} \wedge \neg\tilde{b}) \Rightarrow (\mathring{m} \Leftrightarrow \mathring{m}' \wedge \mathring{m} \Rightarrow (\hat{m} = \hat{m}')) \wedge \neg\mathring{m} \Rightarrow \overrightarrow{b^{k}} \wedge \mathring{m} \Rightarrow \overrightarrow{a^{c}}$
	$\psi_{Filter}(a,b,P) = \tilde{b} \Rightarrow (\tilde{a} \wedge \hat{a} \in dom(P) \wedge P(\hat{a})) \wedge \tilde{b} \Rightarrow (\hat{a} = \hat{b}) \wedge (\neg\tilde{a} \Rightarrow (\neg\overrightarrow{a^{c}} \Leftrightarrow \overrightarrow{b^{k}})) \wedge (\tilde{a} \wedge \neg\tilde{b} \Rightarrow \overrightarrow{b^{k}})$
	$\psi_{Transformer}(a,b,f) = \tilde{b} \Rightarrow (\tilde{a} \wedge \hat{a} \in dom(f)) \wedge \tilde{b} \Rightarrow (\hat{b} = f(\hat{a})) \wedge \neg(\overrightarrow{a^{c}} \wedge \overrightarrow{b^{k}})$

Table IV: Encoding Reo Primitives (Extending [9])

Channel	Constraints
	$\psi_{Sync}(a,b) : \tilde{a} \Leftrightarrow \tilde{b} \wedge \tilde{a} \Rightarrow (\hat{a} = \hat{b})$
	$\psi_{SyncDrain}(a_1,a_2) : \tilde{a}_1 \Leftrightarrow \tilde{a}_2$
	$\psi_{AsyncDrain}(a_1,a_2) : \neg(\tilde{a}_1 \wedge \tilde{a}_2)$
	$\psi_{LossySync} : \tilde{b} \Rightarrow \tilde{a} \wedge \tilde{b} \Rightarrow (\hat{a} = \hat{b})$
	$\psi_{Merger}(a_{0..i},b) : \tilde{b} \Leftrightarrow (\bigvee_i \tilde{a}_i) \bigwedge_{j,j\neq i} \neg(\tilde{a}_i \wedge \tilde{a}_j) \wedge \tilde{a}_i \Rightarrow (\hat{a}_i = \hat{b})$
	$\psi_{Replicator}(a,b_{0..i}) : \tilde{a} \Leftrightarrow (\bigwedge_i \tilde{b}_i) \wedge \tilde{a} \Rightarrow (\bigwedge_i(\hat{b}_i = \hat{a}))$
	$\psi_{Router}(a,b_{0..i}) : \tilde{a} \Leftrightarrow (\bigvee_i \tilde{b}_i) \bigwedge_{j,j\neq i} \neg(\tilde{b}_i \wedge \tilde{b}_j) \wedge \tilde{b}_i \Rightarrow (\hat{b}_i = \hat{a})$
	$\psi_{FIFO_1}(a,b,m) : \tilde{a} \Rightarrow (\neg\mathring{m} \wedge \mathring{m}' \wedge (\hat{m}' = \tilde{a})) \wedge \tilde{b} \Rightarrow (\mathring{m} \wedge \neg\mathring{m}' \wedge (\hat{m} = \tilde{b})) \wedge (\neg\tilde{a} \wedge \neg\tilde{b}) \Rightarrow (\mathring{m} \Leftrightarrow \mathring{m}' \wedge \mathring{m} \Rightarrow (\hat{m} = \hat{m}))$
	$\psi_{Filter}(a,b,P) = \tilde{b} \Rightarrow (\tilde{a} \wedge \hat{b} \in dom(P) \wedge P(\hat{a}) \wedge (\hat{a} = \hat{b}))$
	$\psi_{Transformer}(a,b,f) = \tilde{b} \Rightarrow (\tilde{a} \wedge \hat{b} \in dom(f)) \wedge \tilde{b} \Rightarrow (\hat{b} = f(\hat{a}))$

$$\psi_{Transformer}(a,b,3*\hat{a}) = \tilde{a} \Leftrightarrow \tilde{b} \wedge \tilde{a} \Rightarrow (\hat{a} \in Number \wedge \hat{b} = 3*\hat{a}) \quad (1)$$

$$\psi_{Filter}(b,c,\hat{b}\%2 = 0) = \tilde{c} \Rightarrow (\tilde{b} \wedge \hat{b} \in Number \wedge (\hat{b}\%2 = 0)) \quad (2)$$

$\hat{b}\%2 = 0$). In this case, the value of data items passing through the ends are equal. No flow through the sink end c is either due to no flow on b or that the incoming data item does not satisfy the accepting pattern. As mentioned, the conjunction of these constraints (subject to Axiom 1, which trivially holds in this case) encodes the behavior of the given Reo network.

B. Solving RCSPs

In this section, we formalize the solutions of RCSPs and show how to obtain them.

Definition 5.3 (Solution): A solution $\mathcal{S}$ for a constraint C is a function $\mathcal{S} : \mathcal{V} \rightarrow 2^{\mathcal{D}} \cup \{\top, \bot\}$ such that for all distinct $v_i \in \mathcal{V}, 1 \leq i \leq n = |\mathcal{V}|$, we have $z_i \in \mathcal{S}(v_i)$ implies $C[v_1, v_2, \ldots, v_n \setminus z_1, z_2, \ldots, z_n]$ is true.
Since Reo Constraint Satisfaction Problems (RCSPs) have predicates with free variables of types Boolean ($\{\top, \bot\}$) and data ($\mathcal{D}$), a SAT-solver or a numeric constraint solver cannot solve them alone. Satisfiability Modulo Theories (SMT) [19] solvers find solutions for propositional satisfiability problems where propositions are either Boolean or constraints in a specific theory. However, SMT-solvers are not applicable in our case either, because unlike SAT-solvers they find only an instance of a solution as opposed to the complete set of answers. Another drawback of most SAT- and SMT-solvers is that they work only on quantifier-free formulae, while we use existential quantifies to implement the *hiding* operator of Constraint Automata (see Section VI).

To generate the CASM corresponding to a given Reo network, we need all solutions and thus resort to a hybrid approach that uses both SAT-solvers and Computer Algebra Systems (CASs), namely, REDUCE [20], which is a system for general algebraic computations. First, we form a pure Boolean constraint system by substituting data dependent constraints with new Boolean variables. We find all solutions for the new constraints using a SAT-solver. Then, by substituting each such solution into the original constraints, we obtain a data dependent constraint satisfaction problem that a CAS can solve symbolically. From these solutions, we extract a CASM corresponding to the Reo network encoded by the original set of constraints. Our approach avoids state explosion by treating data constraints symbolically. In the following, we elaborate on our approach.

In an RCSP $\langle \mathcal{P}, \mathcal{M}, M_0, \mathcal{V}, C \rangle$, let $\mathcal{V}_B$ and $\mathcal{V}_D$ be the sets of free Boolean and free data variables of C, respectively, where $\mathcal{V} = \mathcal{V}_B \cup \mathcal{V}_D$, and let A_D be the set of atomic predicates of C containing data variables. The following is our procedure for solving C.

1) We obtain C_B from C by replacing every occurrence of $x \in A_D$ with a unique new Boolean variable $y \notin \mathcal{V}$. For example, for $C = (\tilde{c} \Rightarrow \tilde{b}) \wedge (\tilde{c} \Rightarrow (\hat{b} \in Number \Rightarrow \hat{b}\%2 = 0))$ in Example 5.2, we obtain C_B as $(\tilde{c} \Rightarrow \tilde{b}) \wedge (\tilde{c} \Rightarrow (y_1 \Rightarrow y_2))$ where y_1 and y_2 replace $\hat{b} \in Number$ and $\hat{b}\%2 = 0$, respectively.
2) An off-the-shelf SAT-solver can find the set of solutions S_B for C_B. We define the finite set of constraints $C[S_B] = \{C[v_1, v_2, \ldots, v_n \setminus z_1, z_2, \ldots z_n] \mid$ for all distinct $v_i \in \mathcal{V}_B, 1 \leq i \leq n = |\mathcal{V}_B|, z_i \in S(v_i), S \in S_B\}$.
3) Every $C_D \in C[S_B]$ is a numerical constraint satisfaction problem, which we (symbolically) solve using

a Computer Algebra System. Every solution to each C_D along with the SAT solution $S \in \mathcal{S}_B$ that produced $C_D \in C\left[S_B\right]$ in the previous step, constitute a solution to the RCSP.

Using the presented technique, we obtain the solutions for the RCSP corresponding to Examples 5.2 as follows:

1) $\langle \{\tilde{a} = \bot, \tilde{b} = \bot, \tilde{c} = \bot\}, \top \rangle$,
2) $\langle \{\tilde{a} = \top, \tilde{b} = \bot, \tilde{c} = \bot\}, \hat{a} \notin Number \rangle$,
3) $\langle \{\tilde{a} = \top, \tilde{b} = \top, \tilde{c} = \bot\}, \hat{a} \in Number \wedge \hat{b} = 3 * \hat{a} \ \wedge \ \hat{b}\%2 \neq 0 \rangle$,
4) $\langle \{\tilde{a} = \top, \tilde{b} = \top, \tilde{c} = \top\}, \hat{a} \in Number \wedge \ \hat{b} = 3 * \hat{a} \ \wedge \ \hat{b}\%2 = 0 \ \wedge \ \hat{b} = \hat{c} \rangle$.

Example 5.3: Figure 2 depicts a Reo network that consists of a *LossySync* channel and a *FIFO*$_1$ channel connecting on the node b.

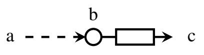


Figure 2: A Context Sensitive Reo Network

Since the Reo network in Figure 2 contains a *LossySync* that is a context dependent channel, we use the context-aware RCSP encoding from Table III:

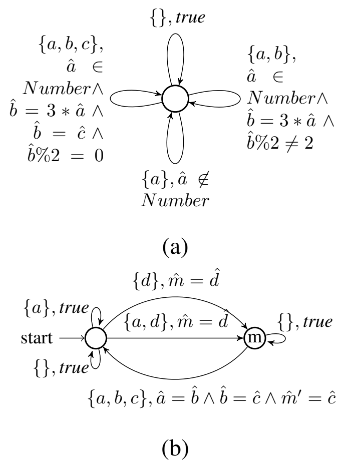
$$\psi_{LossySync}(a, b) = \tilde{b} \Rightarrow (\tilde{a} \wedge (\hat{a} = \hat{b})) \wedge \neg \overrightarrow{a^{c}} \wedge \neg \tilde{a} \Rightarrow \overrightarrow{b^{k}}. \tag{3}$$

$$\psi_{FIFO_1}(b, c, m) = \tilde{b} \Rightarrow (\neg \mathring{m} \wedge \mathring{m}' \wedge (\hat{m}' = \hat{b})) \wedge \tilde{c} \Rightarrow (\mathring{m} \wedge \neg \mathring{m}' \wedge (\hat{m} = \hat{c})) \wedge (\neg \tilde{b} \wedge \neg \tilde{c}) \Rightarrow ((\mathring{m} \Leftrightarrow \mathring{m}') \wedge \mathring{m} \Rightarrow (\hat{m} = \hat{m}')) \wedge \neg \mathring{m} \Rightarrow \overrightarrow{c^{c}} \wedge \mathring{m} \Rightarrow \overrightarrow{b^{k}}. \tag{4}$$

Equation 3 states that flow on the sink end of the *LossySync* is due to flow on its source end. If there is flow on the sink end of the *LossySync*, the data items exchanged at the source and the sink ends are the same. However, it is possible that the source end has flow, but the sink end does not. In this case, the reason for no flow comes from the environment with which the sink end communicates. The third possible behavior of the channel is that there is no flow on the source end due to the environment, in which case the channel provides a reason for no flow on its sink end.

Equation 4 expresses the behavior of the *FIFO*$_1$ channel as follows: The flow on the source end of the channel states that the value of the variable representing the state memory (of the current state) is undefined. The flow on the source end defines the state memory variable for the next state to contain the value of the incoming data item. On the other hand, flow on the sink end means that the value of the state memory variable is defined. The data item leaving the sink end is equivalent to the buffer's data item. In addition, the value of the state memory variable becomes undefined in the next state. If there is no flow on the ends, the variables related to the states stay the same. Being empty, the *FIFO*$_1$ channel provides a reason for no flow on its sink end, while being full does so on the source end of the channel.

The solutions for the RCSP of Example 5.3, (where for brevity, we omit the values of the variables representing the context, such as $\overrightarrow{b^{c}}$) are as follows:



$\{\}, true$
$\{a, b, c\}, \hat{a} \in Number \wedge \hat{b} = 3 * \hat{a} \wedge \hat{b} = \hat{c} \wedge \hat{b}\%2 = 0$
$\{a, b\}, \hat{a} \in Number \wedge \hat{b} = 3 * \hat{a} \wedge \hat{b}\%2 \neq 2$
$\{a\}, \hat{a} \notin Number$
(a)
$\{d\}, \hat{m} = \hat{d}$
$\{a\}, true$
$\{a, d\}, \hat{m} = \hat{d}$
$\{\}, true$
start
m
$\{\}, true$
$\{a, b, c\}, \hat{a} = \hat{b} \wedge \hat{b} = \hat{c} \wedge \hat{m}' = \hat{c}$
(b)

Figure 3: CASMs Generated for Examples 5.2 (a) and 5.3 (b)

1) $\langle \{\tilde{a} = \bot, \tilde{b} = \bot, \tilde{c} = \bot, \mathring{m} = \bot, \mathring{m}' = \bot\}, \top \rangle$,
2) $\langle \{\tilde{a} = \top, \tilde{b} = \top, \tilde{c} = \bot, \mathring{m} = \bot, \mathring{m}' = \top\}, \hat{a} = \hat{b} \wedge \hat{m}' = \hat{b} \rangle$,
3) $\langle \{\tilde{a} = \top, \tilde{b} = \bot, \tilde{c} = \bot, \mathring{m} = \top, \mathring{m}' = \top\}, \hat{m} = \hat{m}' \rangle$,
4) $\langle \{\tilde{a} = \bot, \tilde{b} = \bot, \tilde{c} = \bot, \mathring{m} = \top, \mathring{m}' = \top\}, \hat{m} = \hat{m}' \rangle$,
5) $\langle \{\tilde{a} = \top, \tilde{b} = \bot, \tilde{c} = \bot, \mathring{m} = \top, \mathring{m}' = \bot\}, \hat{m} = \hat{c} \rangle$,
6) $\langle \{\tilde{a} = \bot, \tilde{b} = \bot, \tilde{c} = \top, \mathring{m} = \top, \mathring{m}' = \bot\}, \hat{m} = \hat{c} \rangle$.

C. *CASM Construction*

In order to construct the CASM from the set of solutions S for an RCSP $\langle \mathcal{P}, \mathcal{M}, M_0, \mathcal{V}, C \rangle$, we first define

- $\mathcal{N} = \{n \mid n^c \in \mathcal{P} \vee n^k \in \mathcal{P}\}$

and then map each solution $\langle s, s_d \rangle \in S$ into a transition $t : q \xrightarrow{N,g} p$ as follows:

- $q = \langle \{m \mid m \in \mathcal{M}, s\left(\mathring{m}\right) = \top\} \rangle$,
- $p = \langle \{m \mid m \in \mathcal{M}, s\left(\mathring{m}'\right) = \top\} \rangle$,
- $N = \{n \mid n \in \mathcal{N}, s\left(\tilde{n}\right) = \top\}$,
- The data constraint g is (a syntactic variant of) s_d.

We obtain the CASM $A = (Q, \mathcal{N}, \rightarrow, q_0, \mathcal{M})$ from the set $\longrightarrow$ of all transitions generated above, where:

- $Q = \{q \mid q \xrightarrow{N,g} p \vee p \xrightarrow{N,g} q\}$,
- $q_0 = \langle \{m \mid m \in M_0, s(\mathring{m}) = \top\} \rangle$,
- $\mathcal{M}$ is the same $\mathcal{M}$ as in the RCSP.

Applying the above procedure to the solutions of RCSPs constraints generates their corresponding CASMs. For instance, the first solution for the constraints in Example 5.2 generates the transition $q \xrightarrow{\emptyset, true} q$, where q is the only state of the CASM, which has no state memory variable. This is so because the set of variables of the form $\mathring{m}$ is empty. Also, the transition has no synchronizing port, because the value of every one of variables $\tilde{a}, \tilde{b}$ and $\tilde{c}$ is $\bot$. Figures 3a and 3b show the CASMs derived from the RCSPs in Examples 5.2 and 5.3.

Our approach deals with data in a symbolic fashion, where we partition the global set of data values to equivalence classes toward which a Reo network behaves differently.

This is in contrast with the traditional way of dealing with data in the formal semantics of Reo (and other models), where they consider a different state for each possible value that can be stored in buffers and a distinct transition for each data value passing through the ports. Our symbolic approach allows working with an infinite data domain. In addition, rather than implementing the highly time- and memory-demanding custom-made algorithms to generate Reo formal semantics, we use the efficient SAT-solvers and computer algebra systems to solve constraints whose solutions are equivalent to these models. An experimental study on the efficiency of using SAT-solvers to generate Reo formal semantics is reported in [9].

VI. Hiding

We use *hiding* to abstract from internal transitions. The author in [9] proposes applying the existential quantifier to the constraints encoding of the behavior of a network to abstract from internal ports and their corresponding data variables. Similarly, we use existential quantifiers such as $\exists \tilde{e}, \hat{e}, \overrightarrow{e} : C$, where C is the RCSP of a Reo network and e is an internal node to hide.

Although several algorithms exist for the problem of quantifier elimination in Boolean algebra and first order logic [21], [22] and [23], we are not aware of any working tool that does quantifier elimination on Boolean algebraic formulae. Therefore, our tool implements the *hiding* operator as defined for CASM.

Hiding the internal nodes on some transitions can make the set of their synchronized nodes empty. Here, we refer to such a transition as an *empty* transition, if the free variables of its guard are merely state memory variables. Under some circumstances, we can merge the source and the target states of empty transitions. Let q and p be two states in a CASM such that $q \xrightarrow{\emptyset,g} p$. The following are the conditions under which the state p can merge into the sate q:

1) The states q and p have the same number of state memory variables.
2) The guard g consists of the conjunction of the predicates of the form of $x = y'$, for $x, y \in \mathcal{M}$. This way, g defines a correspondence relation between the state memory variables of the state q and those of the state p.
3) For each transition $q \xrightarrow{N,g'} r$ where $r \notin \{p, q\}$, there is a transition $p \xrightarrow{N,g''} r$ such that $g' \Leftrightarrow g''_g$, where g''_g is obtained from g by replacing all occurrences of the next state memory variable y' with the next state memory variable x', if g contains $x = y'$ for state memory variables $x, y \in \mathcal{M}$.
4) For each transition $r \xrightarrow{N,g'} p$ where $r \notin \{p, q\}$, there is a transition $r \xrightarrow{N,g''} q$ such that $g'' \Leftrightarrow g'_g$, where g'_g is derived from g by substituting all occurrences of the state memory variable x in g with the state memory variable x, if g contains $x = y'$ for state memory variables $x, y \in \mathcal{M}$.

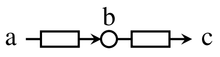
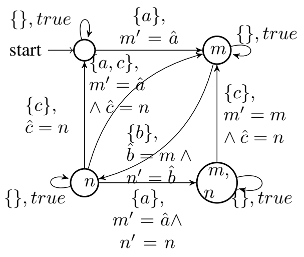
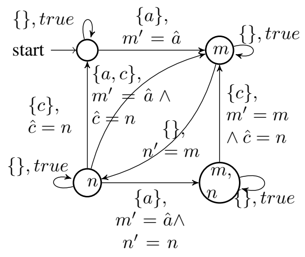


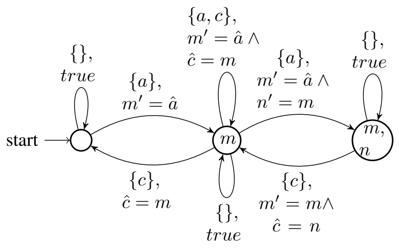
Figure 4: Two FIFO$_1$s Forming FIFO$_2$



(a) CASM of Example 6.1



(b) Hiding Internal Ports



(c) Merging the States

Figure 5: Hiding the Empty Transition and Merging Its Source and Target States for the CASM of FIFO$_2$ in Figure 4

Provided that the above conditions hold, the state p merges into the state q as follows:

1) We eliminate the transition $q \xrightarrow{\emptyset,g} p$.
2) We remove the state p after substituting y, y', and p with x, x', and q in all transitions. Observe that such substitutions convert the non-eliminated transitions between the states q and p into loops over the state q.

Example 6.1: Figure 4 shows a *FIFO*$_2$ derived from composing two *FIFO*$_1$s. The CASM corresponding to the *FIFO*$_2$ is in Figure 5a. Figure 5b depicts the CASM resulting from hiding the mixed node b. Figure 5c presents the result of eliminating the empty transitions.

VII. Conclusions and Future Work

In this paper, we have presented a constraint-based framework that encodes the semantics of Reo networks as constraint satisfaction problems whose predicates are either Boolean propositions or numerical constraints. We presented a hybrid approach to find the solutions for these problems. An advantage of our approach is that it treats data constraints symbolically to mitigate the state explosion problem. From this solution, we construct the semantic model corresponding to a Reo network in the form of constraint automata with state memory. Our framework supports *product* and *hiding* operations on constraint automata. We have implemented and integrated our approach as a tool in the ECT. As part of our ongoing work, we are using this framework to encode other aspects of the semantics of Reo, namely, priorities and timed behavior. In this way, our work will be the most expressive framework that exists to analyze Reo networks. Furthermore, we will prove soundness and completeness of the RCSP encoding of Reo networks along with its compositionality.

REFERENCES

[1] M. Bell, "Introduction to Service-Oriented Modeling," *Service-Oriented Modeling: Service Analysis, Design, and Architecture*, p. 3, 2008.

[2] R. Pugliese and F. Tiezzi, "A Calculus for Orchestration of Web Services," *J. Applied Logic*, vol. 10, no. 1, pp. 2–31, 2012.

[3] D. Kitchin, W. R. Cook, and J. Misra, "A Language for Task Orchestration and Its Semantic Properties," in *CONCUR*, vol. 4137 of *Lecture Notes in Computer Science*, pp. 477–491, Springer, 2006.

[4] F. Arbab, "Reo: a Channel-Based Coordination Model for Component Composition," *Mathematical Structures in Computer Science*, vol. 14, pp. 329–366, 2004.

[5] F. Arbab, N. Kokash, and S. Meng, "Towards Using Reo for Compliance-Aware Business Process Modeling," in *ISoLA*, pp. 108–123, 2008.

[6] B. Changizi, N. Kokash, and F. Arbab, "A Unified Toolset for Business Process Model Formalization," in *7th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA 2010)*, pp. 147–156, ENTCS, 2010.

[7] S.-S. T. Jongmans and F. Arbab, "Overview of Thirty Semantic Formalisms for Reo," *Scientific Annals of Computer Science*, vol. 22, pp. 201–251, 2012.

[8] D. Clarke, D. Costa, and F. Arbab, "Connector Colouring I: Synchronisation and Context Dependency," *Science of Computer Programming*, vol. 66, no. 3, pp. 205–225, 2007.

[9] J. Proença, *Synchronous Coordination of Distributed Components*. PhD thesis, Institue for Prgramming research and Algorithms, 2011.

[10] C. Baier, M. Sirjani, F. Arbab, and J. J. M. M. Rutten, "Modeling Component Connectors in Reo by Constraint Automata," *Science of Computer Programming*, vol. 61, no. 2, pp. 75–113, 2006.

[11] B. Pourvatan, M. Sirjani, H. Hojjat, and F. Arbab, "Symbolic Execution of Reo Circuits using Constraint Automata," *Sci. Comput. Program.*, vol. 77, no. 7-8, pp. 848–869, 2012.

[12] M. M. Bonsangue, D. Clarke, and A. Silva, "Automata for Context-Dependent Connectors," in *COORDINATION* (J. Field and V. T. Vasconcelos, eds.), vol. 5521 of *Lecture Notes in Computer Science*, pp. 184–203, Springer, 2009.

[13] F. Arbab, C. Baier, F. D. Boer, and J. Rutten, "Models and Temporal Logics for Timed Component Connectors," in *2nd International Conference on Software Engineering and Formal Methods (SEFM 2004)*, pp. 198–207, IEEE Computer Society, 2004.

[14] S. Kemper, "SAT-based Verification for Timed Component Connectors," *Electr. Notes Theor. Comput. Sci.*, vol. 255, pp. 103–118, 2009.

[15] F. Arbab and C. Baier, "Priority in Reo and Constraint Automata," tech. rep., Centrum voor Wiskunde en Informatica. In preparation.

[16] J. F. Groote, A. Mathijssen, M. Reniers, Y. Usenko, and M. V. Weerdenburg, "The Formal Specification Language mCRL2," in *Methods for Modelling Software Systems (MMOSS 2006)*, vol. 06351 of *Dagstuhl Seminar Proceedings*, IBFI, 2006.

[17] N. Kokash, C. Krause, and E. de Vink, "Reo + mCRL2: A Framework for Model-checking Dataflow in Service Compositions," *Formal Aspects of Computing*, 2011.

[18] F. Arbab, C. Koehler, Z. Maraikar, Y.-J. Moon, and J. Proenca, "Modeling, Testing and Executing Reo Connectors with the Eclipse Coordination Tools," in *5th International Workshop on Formal Aspects of Component Software (FACS 2008)*, vol. 8, ENTCS, 2008.

[19] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability Modulo Theories," *Handbook of Satisfiability*, vol. 4, 2009.

[20] G. Rayna, *REDUCE: Software for Algebraic Computation*. New York, NY, USA: Springer-Verlag New York, Inc., 1987.

[21] L. Bordeaux and L. Zhang, "A Solver for Quantified Boolean and Linear Constraints," in *Proceedings of the 2007 ACM symposium on Applied computing*, SAC '07, (New York, NY, USA), pp. 321–325, ACM, 2007.

[22] A. Ayari and D. Basin, "QUBOS: Deciding Quantified Boolean Logic using Propositional Satisfiability Solvers," in *Formal Methods in Computer-Aided Design*, pp. 187–201, Springer, 2002.

[23] J. H. Davenport, "Computer Algebra Applied to Itself," *J. Symb. Comput.*, vol. 6, pp. 127–132, August 1988.

Predicting Quality Requirements Necessary for a Functional Requirement Based on Machine Learning

Ken Tanaka
Department of Information Sciences
Kanagawa University
Kanagawa, 259-1293, Japan
Email: ktanaka@info.kanagawa-u.ac.jp

Haruhiko Kaiya
Department of Computer Science
Shinshu University
Nagano, 380-8553, Japan
Email: kaiya@shinshu-u.ac.jp

Atsushi Ohnishi
Department of Computer Science
Ritsumeikan University
Shiga, 525-8577, Japan
Email: ohnishi@cs.ritsumei.ac.jp

***Abstract*—In the early stage of the software development, quality requirements should be explicitly specified as well as functional requirements. Software architecture and/or design decision should be largely reconsidered if some quality requirement is overlooked in the early stage. We thus propose a technique for predicting quality requirements necessary for each functional requirement. A functional requirement is represented with a semi-formal language called eXtended Japanese Requirements Description Language (X-JRDL), which is based on the case grammar. In our previous work, the results of the prediction largely depended on human such as domain experts and requirements analysts because prediction rules were manually written by them. We thus introduce machine learning to avoid this problem. To predict quality requirements necessary for any kinds of functional requirements, training data should be appropriately chosen. We choose the training data so that we can predict necessary quality requirements for all types of functional requirements. Since semantically impartial data are suitable for such training data and one of the cases called concept is semantically dominant in an X-JRDL sentence, we choose the training data set in which any of the concepts evenly occurs. Through the experiments, we confirm our technique works well for predicting necessary quality requirements.**

Keywords*-requirements analysis; quality requirements; machine learning; case grammar.*

I. Introduction

In order to write a requirements specification of high quality, we must take the following characteristics into account; correctness, consistency, unambiguity, completeness, rank of importance, stability, verifiability and traceability [1]. For functional requirements, these characteristics are taken into account well, but taking them into account is still a research challenge in non-functional or quality requirements. Quality requirements specify how well functions are accomplished [2], and they are very important. Some systems such as the online computer aided instruction (CAI) systems or the online shopping sites should be highly reliable, while others such as web browsers and desktop publishing systems should be usable. If quality requirements are not correctly specified in a requirements specification, software system may not be correctly developed. Because there are more problems in quality requirements than in functional requirements, the special issue was published in IEEE Software [2]. This introductory article of the issue focuses on the following three problems; implicit understanding of stakeholder, trade-offs among quality requirements and difficulty to measure and to track quality requirements.

To resolve these problems, detecting quality requirements necessary for each functional requirement is crucial because such detection is a basis of discussing stakeholders' understanding, their trade-offs and tacking. We have already proposed a technique for detecting such quality requirements [3]. The technique uses a semi-formal notation for a functional requirement called eXtended Japanese Requirements Description Language (X-JRDL) [4] because the notation explicitly represents the semantic structure of a functional requirement and the structure directly gives influences on quality requirements necessary for the functional requirement. The rules for the detection thus can be written in the if-then rules. Although the results of applying the technique were useful for defining quality requirements [3], it took a lot of effort for preparing the rules for detection. In addition, the quality of rules largely depended on the expertise of people (normally domain experts and requirements analysts) who wrote the rules.

The main contribution of the new technique proposed in this paper is to avoid these problems. In our new technique, a machine learning technology is used to detect the quality requirements necessary for each functional requirement. By using a machine learning technology, rules for detection are automatically generated based on the existing results of detection (we called such results training data). Necessary quality requirements are thus automatically detected based on the rules. The detection results become more accurate than ever when the appropriate training data increase. We still use the semi-formal notation for a functional requirement X-JRDL because the semantic information is explicitly represented in an X-JRDL sentence, and such information is convenient for machine learning. In addition, converting a natural language sentence to a sentence in X-JRDL is already studied [5]. If a sentence is characterized in more than hundreds components, it is important to choose limited number of components for the efficient machine learning. Because an X-JRDL sentence (called a requirements frame) consists of less than ten components (cases), we do not have to do it. It is rather important to choose appropriate training data because the appropriate choice of training data enables us to detect the quality requirements necessary for all types of functional requirements. In general, we choose the training data so that the data is evenly chosen with respect to the cases in a requirements frame. Because a requirements frame is regarded as a vector of cases, we can use the cosine similarity to choose such training data. However, we have to focus on specific cases to choose such training data if such cases are more dominant than others. Through the experiments, we found one of the cases called concept is more dominant than other cases. In addition, the training data in which any of the concepts evenly occurs enabled us to predict necessary quality requirements successfully.

The rest of this paper is organized as follows. In the next section, we briefly introduce our previous rule-based technique for predicting quality requirements necessary for each functional requirement. Because the technique requires rules written by experts, it is not easy to use it in practice. In Section III, we introduce our new technique for such prediction using machine learning. In our technique, training data are carefully chosen for better prediction. We made an experiment to evaluate our proposed technique. The experiment, the results and its discussion are also reported. We then review existing researches about both the quality requirements analysis and the application of machine learning to the software engineering field. We, finally, summarize our current results and show future issues.

II. Rule-based Technique for predicting Quality Requirements

In this section, we explain our previous rule-based technique [6] [3] for predicting quality requirements necessary for each functional requirement because the inputs and the outputs of the technique are the same as those of our new technique presented in the next section.

A. Overview of the rule-based technique

The goal of our rule-based technique is to predict quality requirements types such as usability, reliability and accuracy for each functional requirement. The steps of the technique for predicting quality requirements are as follows.

1) We have to prepare the rules for each problem domain such as web based information systems, drawing software and so on. Each rule decides whether specific quality requirements types are necessary for a functional requirement. An example of a rule is shown in Figure 2.

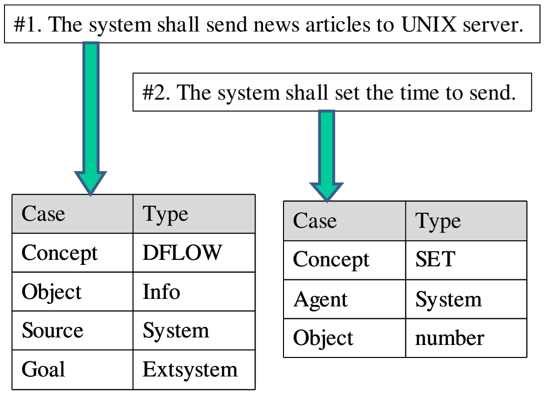

#1. The system shall send news articles to UNIX server.
#2. The system shall set the time to send.
Case
Type
Concept
DFLOW
Object
Info
Source
System
Goal
Extsystem
Case
Type
Concept
SET
Agent
System
Object
number

Figure 1. Examples of converting a requirement sentence to a requirements frame.

2) Requirements specifications are usually written in natural language such as English. We have to convert each sentence in a specification into a requirements frame, which is a sentence represented in a semi-formal language called X-JRDL [4] based on the case grammar [7].
3) For each requirements frame, all rules are applied and candidates of necessary quality requirements types are detected. Examples of quality requirements types are “usability”, “accuracy”, “time behavior” and so on. We can use the quality sub-characteristics in ISO9126 [8] and/or NFR framework [9] as the catalog of quality requirements types.
4) Based on the results of the detection, requirements analysts adds quality requirements descriptions (usually represented as adverbs) each functional requirement. Examples of quality requirements descriptions are “without specific training for its operation”, “more than 80% correct results for its query”, “within 3 seconds for its response” and so on.

B. Requirements Frames for Functional Requirements

Semantics of a functional requirement largely gives influences on the decision which types of quality requirements are necessary for the functional requirement. We thus convert a functional requirements sentence (usually written in natural language) into a requirements frame. A requirements frame is based on the case grammar [7], and consists of several cases and its types. We thus represent a requirements frame as a tabular form. The examples of the requirements frames represented in a tabular form are shown at the bottom of the Figure 1.

The mandatory case is called “Concept” in a requirements frame, and it corresponds to the verb in an original requirements sentence. We have prepared a list of typical types of a concept as shown in Table I. For each concept type, complementary cases may be specified for each functional

Table I
TYPICAL TYPES OF A CONCEPT

Concept	Meaning
DFLOW	Data flow
CFLOW	Control flow
ANDSUB	And-tree structure
ORSUB	Or-tree structure
GEN	Data creation
SET	Set the value to data
RET	Retrieve a record in a file
UPDATE	Update a record in a file
DEL	Delete a record in a file
INS	Insert a record in a file
MANIP	File manipulation
EQ, NE, LT, GT, LE, GE	Logical operators

Table II
TYPICAL TYPES OF COMPLEMENTARY CASES

Noun Type	Meaning
Human	active and external object
Function	active and internal object
File	passive object of information set
Data	passive object of a single information
Info	information in the real world
Control	passive object for control transition
Device	passive object of an instrument
System	the system to be defined
Extsystem	external systems related to the system to be defined
Number	numerical data or information

requirements sentence. Such complementary cases correspond to a subject, objects and a complement in a sentence. Each of such complementary cases also takes a type as shown in Table II. How to convert each sentence into a requirements frame is out of scope of this paper because we have already studied the issue in our previous works [4] [5].

We show examples of requirements frames and their corresponding original sentences in Figure 1. At the top of the figure, two typical functional requirements sentences are represented. Because the sentence #1 is about data flow from the system to be developed to an external system (UNIX server), DFLOW is chosen as a type of its concept. According to the definition of our requirements frames, DFLOW requires complementary cases such as Object, Source and Goal. Object corresponds to the data to flow. Source and goal correspond to be the source and the destination of the data flow. In the sentence #1, the information flows from the system to the external system. We thus assign Info, System, Extsystem types to each case as shown in the figure. A requirement #2 is also converted in the same way.

C. Rules

Because a requirements frame explicitly represents the semantic information about the original requirements sentence, we may simply check the types of cases to detect necessary quality requirements for each functional requirement. We thus simply construct if-then rules for deciding whether a specific quality requirement is necessary for a requirement frame. At the top in Figure 2, we show an example of such a rule (Rule A). The rule decides whether Interoperability is necessary for a requirement frame. The if-then part of the rule focuses on the types of cases about Concept, Source and Goal. Because a requirement frame #1 in the figure satisfies this condition, the rule decides Interoperability is necessary for the requirement frame #1. On the other hand, the rule does not decide Interoperability is necessary for a requirements frame #2 because the condition is not satisfied in it.

D. Discussion about the rule-based technique

We have written 14 rules for web-based information system [3], and the rules are applied in a case study [10]. One of the big problems of this technique is the effort for writing such rules. It takes about a few weeks for two experts of requirements engineering for writing such 14 rules. Even if the rules can be reused in the same domain and they can be improved during their usage, such expectations largely depend on the expertise of requirements analysts. Another problem is about their application. According to the result of the case study, an expert simply referred the results of rule application and he basically subjectively updated the original requirements. One of the reasons was that the expert considered the rules to be still immature and the rules should be manually improved during more applications. If such rules can be improved automatically along the progress of their usage, the effort for the rule users (normally, requirements analysts) largely decreases.

III. QUALITY REQUIREMENTS PREDICTION USING MACHINE LEARNING

The problem of obtaining quality requirements from a requirements frame can be formalized as a classification problem of obtaining classification rules from a finite data set. Let the set of input vectors be denoted by $I = B^n$ and the set of labels by $L = \{l_1, l_2, \cdots, l_m\}$, where $B = \{0, 1\}$. Training data D denotes a finite set $D \subseteq I \times L$.

Definition 3.1:

$$D = \{(d^{(1)}, l^{(1)}), (d^{(2)}, l^{(2)}), \cdots (d^{(n)}, l^{(n)})\}$$

Here, $d^{(1)}, d^{(2)}, \cdots, d^n$ are called instances, and $l^{(1)}, l^{(2), \cdots, l^{|D|}}$ are called classes to which the individual instances belong. $n = |D|$ denotes the number of training data samples, and m denotes the number of labels. Classification rules are represented by the function

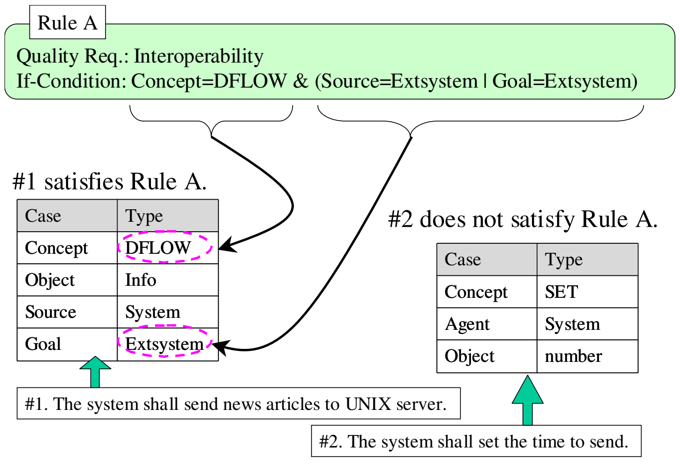

Rule A
Quality Req.: Interoperability
If-Condition: Concept=DFLOW & (Source=Extsystem | Goal=Extsystem)
#1 satisfies Rule A.
Case
Type
Concept
DFLOW
Object
Info
Source
System
Goal
Extsystem
#2 does not satisfy Rule A.
Case
Type
Concept
SET
Agent
System
Object
number
#1. The system shall send news articles to UNIX server.
#2. The system shall set the time to send.

Figure 2. Examples of applying a rule to requirements frames.

$\delta: I \Rightarrow L$. A classification problem is the problem of obtaining optimal δ for given D based on certain criteria.

Various learning algorithms are used in classification problems, such as support vector machines, decision trees, maximum entropy models, and naive Bayes classifiers. In choosing the learning algorithm, it is necessary to choose an algorithm that can suitably capture the structures included in the data set of the target classification problem [11].

In the case where a support vector machine is used, it is necessary to determine a kernel function that is suitable for the problem. A kernel function is a function that gives the similarity between instances. However, it is not easy to obtain a function that appropriately represents the semantic similarity between instances represented by requirements frames. As for a decision tree, although it is advantageous in that classification rules obtained can be readily understood, in the case of the type of problem being considered in this research, which requires manual preparation of training data, since the number of instances is limited, excessive segmentation might occur, resulting in degraded generalization ability. Furthermore, since the training data samples investigated in this research are sparse binary vectors, and it is assumed that the same instance might be classified into different classes, deterministic learning algorithms are not suitable. Therefore, in order to obtain appropriate results, it will be a better approach here to acquire corresponding relationships between inputs and outputs by directly using word co-occurrences rather than capturing complex contextual structures. Accordingly, a naive Bayes classifier is adopted here as a learning machine in view of the simplicity of the model and the ease of computation.

Representations of requirements frames of the specifications prepared by experts are converted into binary vectors. For example, the concepts of the case frame are associated with a 17-dimensional vector since the number of types is 17. This vector changes in accordance with the number of concepts defined in accordance with the relevant domain. Structures included in the concepts, such as objects, sources, and goals, are also represented as binary vectors of predetermined orders. Similarly, quality characteristics required for individual requirements frames are also converted into binary vectors.

A. *Instance selection using cosine values*

Binarized specifications and quality characteristics are considered as instances and classes of the instances, and the set of these will be denoted by R. A portion of the set R is used as the training data D. Here, half of the instances prepared are used as the training data D. The set of instances correctly classified by the learned function δ will be distinguished by attaching the subscript $_c$. The correct answer rate representing the ratio of correctly classified instances among the other half instances $D' = R - D$ will be referred to as the successful learning rate E_{suc}, and the correct classification rate for all instances will be referred to as the learning accuracy E_{acc}.

$$E_{suc} = \frac{|D'_c|}{|D'|} \tag{1}$$

$$E_{acc} = \frac{|D_c \cup D'_c|}{|R|} \tag{2}$$

In selecting instances, it is necessary to select training data samples uniformly in some sense from the set of instances. As a criterion of the similarity between instances, here, the sum of cosine values between learning data samples will first be used. The sum S_i of cosine values of an instance i will be defined as follows.

Definition 3.2:

$$S_i = \sum_{j(\neq i)}^{n} \frac{d^{(i)} d^{(j)}}{|d^i||d^j|} \tag{3}$$

By choosing instances with small values of S_i, orthogonal instances will be preferentially used for learning from the data set, so that unbiased instance selection can be expected.

In this research, four requirements specifications were picked up from the specifications of procurement examples at the Information Systems and Welfare Division of the Ministry of Economy, Trade and Industry of Japan and were converted into requirements frames, which were used as training data. The numbers of instances of the training data used in the experiment are given in the table below. The instances corresponding to the four requirements specifications were sorted in ascending order according to equation (3), and $\lfloor \frac{|R|}{2} \rfloor$ instances were picked up as the training data D.

Table III
NUMBERS OF INSTANCES OF REQUIREMENTS SPECIFICATIONS USED IN LEARNING EXPERIMENT.

Spec. 1	Spec. 2	Spec. 3	Spec. 4
37	41	52	58

In the learning experiment, the naive Bayes classifier scheme of WEKA [12], which is a machine learning platform, was used. In order to confirm the effect of cosine-based selection, a learning experiment in which instances were chosen at random was also conducted. Learning was performed for each quality characteristic of the individual specifications, and E_{suc} and E_{acc} values for each quality characteristic, as well as average E_{suc} and average E_{acc} for all quality characteristics, were obtained. The average E_{suc} values are shown in Figure 3, and the average E_{acc} values are shown in Figure 4, in which the horizontal axis represents the number of instances and the vertical axis represents the average E.

The cosine-based instance selection was effective when $|D|$ was small; however, there was a tendency for both E_{suc} and E_{acc} to decrease as $|D|$ increased. One reason for this tendency was that the cosine values of requirements frames with the same representation were added up into the sum S_i of cosine values for each instance, used in equation (3), so that frequently used requirements frames were excluded. In instance selection, some measure is needed to avoid adding up the cosine values of requirements frames with the same representation. Therefore, the sum defined by equation (4) below will be used hereafter.

Definition 3.3:

$$S_i = \sum_{j(d^j \neq d^i)}^{n} \frac{d^{(i)} d^{(j)}}{|d^i||d^j|} \tag{4}$$

B. Requirements frames representations and features

In order to adapt to the learning scheme, here, it is assumed that training data samples are represented by simple binary vectors. However, individual cases included in

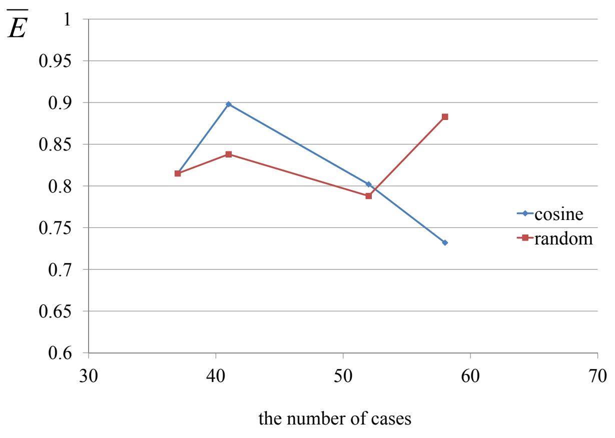


Figure 3. Relationship between the number of instances and E_{suc}.

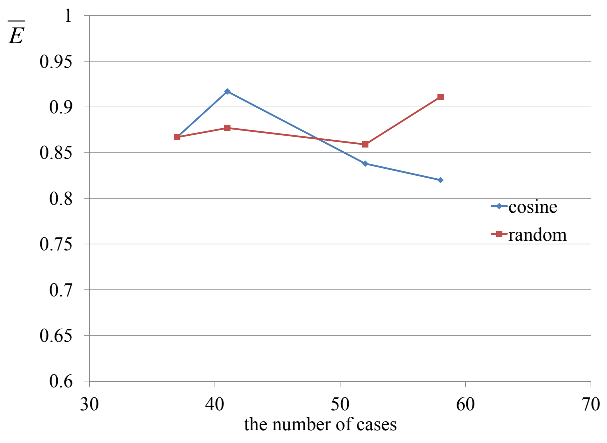


Figure 4. Relationship between the number of instances and E_{acc}.

requirements frames representations include elements that are necessary for quality characteristics identification and those that are not necessary. Here, of the elements of the case structure, such as concept, function, and reliability, elements that are necessary for quality characteristics identification were revealed experimentally. The necessary elements correspond to features used in learning, which can be utilized for appropriate selection of example problems. Note that, in this paper, we place the term "feature" in the context of machine learning; so, the meaning of it is different from that in software engineering.

Features necessary for quality characteristics identification were estimated for Specification 4, which was found to have the lowest level of E in the results described in the preceding section. Table IV shows the individual requirements frames representations included in the instances of Specification 4.

Table V shows the results of learning in which half of the instances were selected as training data based on cosine values in view of five frames representations.

Table IV
REQUIREMENTS FRAMES REPRESENTATIONS OF SPECIFICATION 4.

Concept	DFLOW, CFLOW, SET, RET, UPDATE, DEL, INS, MANIP
Agent	System, Human
Goal	System, Human, Extsystem
Object	Data, Info, Function
Source	System, Human, Info

Table V
RESULTS OF LEARNING IN WHICH INSTANCES WERE SELECTED FOR EACH REQUIREMENTS FRAME REPRESENTATION.

	Concept	Agent	Goal	Object	Source
E_{acc}	0.90	0.83	0.70	0.86	0.70
E_{ttl}	0.93	0.87	0.81	0.89	0.81

The highest E_{suc} and E_{acc} were obtained when instances were selected based on concept among the requirements frames representations. Concept has the highest order among the requirements frames representations, effectively serving for instance separation, so that it is suitable feature that can be used for learning.

C. Instance selection based on concept

In order to confirm the estimation in Section III-B, a learning experiment in which instances were selected based on concept was conducted. Regarding the four requirements specifications used in Section III-A, instances were selected based on concept among the requirements frames representations of the individual specifications. When the number of instances was less than half, orthogonality of the remaining instances was evaluated based on equation (4), and instances were selected accordingly until the number reached half. E_{suc} and E_{acc} for each quality characteristic, as well as the average E_{suc} and E_{acc}, were obtained. The E_{suc} average values are shown in Figure 5, and the E_{acc} average is shown in Figure 6. For the purpose of comparison, the results of learning in the case where instances were chosen at random and the results of learning in the case where instances were chosen at random from concept and are also plotted. The experimental results demonstrate that the success rate in the case where instances were selected from concept monotonically increased as the number of instances increased, suggesting its effectiveness in learning. In the case where instances were selected from concept, a maximum success rate of about 0.91 was achieved for unknown data.

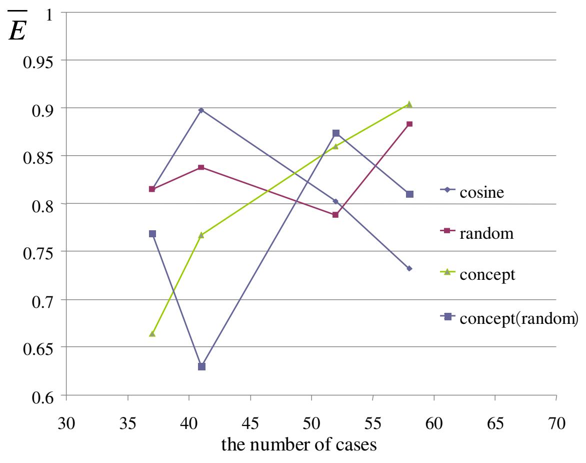


Figure 5. E_{suc}in the case where instances were selected from concept.

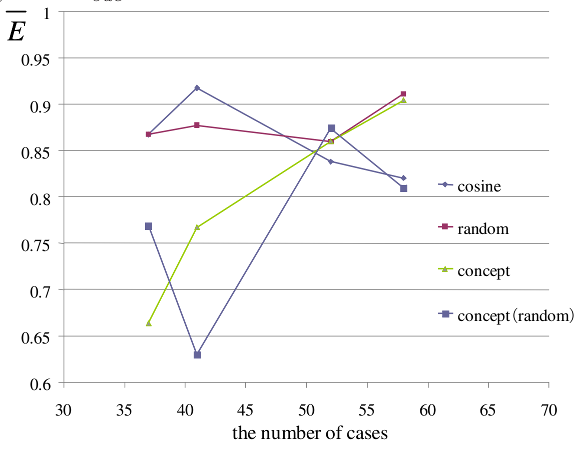


Figure 6. E_{acc} in the case where instances were selected from concept.

In the case where instances were chosen at random, the success rate varied considerably depending on the number of instances. From what has been described above, it is estimated that, when obtaining the quality characteristics of a large-scale specification, it will be effective to select instances in such a manner that overall quality characteristics are determined based on concept in requirements frames representations and that further classification is performed based on cases as needed.

IV. RELATED WORK

There are several studies how to define each quality requirement, but most of them requires the huge amount of human effort. In ISO 25021 [13], concrete examples how to measure quality requirements are shown, and these examples help analysts to make quality requirements measurable. Donald Firesmith gives some format to specify quality requirements rigorously [14]. In Architecture Tradeoff Analysis Method (ATAM) [15] [16], a template for quality requirements called “quality attribute scenario” is provided to evaluate the validity of architectural decision. Such template may be used to support stakeholders writing quality requirements. In an article by Ozkayad et al. [17], an empirical data of the most common quality attributes was shown based on the ATAM. This kind of empirical data help requirements analysts to specify quality requirements. For similar systems, similar kinds of quality requirements are normally required. For example, most functions in typical Web browsers require security and usability, but do not so require accuracy and fault tolerance. This kind of analysis helps analysts to validate their quality requirements definition in a specification by comparing to specifications of other similar systems [18]. However, such analysis does not directly point out missing quality requirements in each functional requirement. The analysis just suggests that the specification is unbalanced with respect to quality requirements definition. UML is the most popular semi-formal notation for software development now, and there are some challenges to introduce quality requirements into it [19] [20]. However, how to specify such introduced information is normally out of scope in each research. Using ontology, dictionary and/or thesaurus [21] is one of the useful ways to improve the quality of requirements with respect to semantic aspect. However, it is a little bit weak because simple words/terms matching cannot completely represent the semantic information in a requirement.

There are a lot of software engineering researches using machine-learning techniques, such as cost estimation [22], defect prediction [23] and design pattern mining [24]. Most researches focus on variable selection rather than training data selection because plenty of variables exist in such application area. For requirements engineering researches, machine-learning techniques are rarely used. One of the exceptions is a method for classifying non-functional requirements (NFR) automatically using a machine learning technique [25]. In this method, usual natural language documents are used for the classification, but semi-formal notation is used in our research. Training data sets are chosen empirically so as to effectively classifying NFRs in this research, but training data sets are systematically chosen based on the theory of machine learning in our research.

V. Conclusion and Future WORK

In this paper, we proposed and evaluated a new technique for predicting quality requirements necessary for a functional requirement based on the machine learning. The main contribution of this research is to avoid human effort for preparation of detection, i.e., writing and improving detection rules manually. In our technique, each functional requirement is represented in X-JRDL, which is a semi-formal language based on case grammar. Because X-JRDL explicitly represents the semantic structure of a functional requirement, we can easily decide which kinds of quality requirements are necessary for a functional requirement.

In our previous work [3] [6], we proposed a rule-based technique for predicting necessary quality requirements. The results of the previous work were not bad, but it took a lot of effort to write rules for prediction. In addition, the quality of prediction largely depended on the expertise of the people who wrote the rules. Our new technique avoids such problems because detection rules are automatically generated by machine learning, and the rules can be also automatically improved.

In this research, requirements frames representations that were suitable as features were revealed experimentally through selection of training data based on cosine values. In the case of a large-scale requirements specification, uniform selection of instances from concept among requirements frames representations gave the most appropriate quality characteristics. Although duplicates increased as the data volume increased in the case of instance selection based on simple similarity of cosine values, a high success rate was achieved by selectively choosing instances from suitable features.

The results shown here are average values of E_{suc} and E_{acc}. In view of the individual quality characteristics, the results for reliability were lower than those for the other quality characteristics. Although independence of individual attributes is assumed when classes are given by naive Bayes classifiers, presumably, some dependencies exist in reality.

In future research, it is necessary to examine instance representations and selection methods that are free of such dependencies. We would like to also develop a supporting tool (a CASE tool) to support a requirements analyst to write a requirements specification based on our proposed technique.

References

[1] "IEEE Recommended Practice for Software Requirements Specifications," 1998, IEEE Std. 830-1998.

[2] J. D. Blaine and J. Cleland-Huang, "Software Quality Requirements: How to Balance Competing Priorities," *IEEE Software*, vol. 25, no. 2, pp. 22–24, Mar./Apr. 2008.

[3] H. Kaiya and A. Ohnishi, "Finding incorrect and missing quality requirements definitions using requirements frame," *IEICE Transactions*, vol. 95-D, no. 4, pp. 1031–1043, 2012.

[4] A. Ohnishi, "Software requirements specification database based on requirements frame model," in *ICRE*, 1996, pp. 221–228.

[5] Y. Matsuo, K. Ogasawara, and A. Ohnishi, "Automatic transformation of organization of software requirements specifications," in *RCIS*, 2010, pp. 269–278.

[6] H. Kaiya and A. Ohnishi, "Quality requirements analysis using requirements frames," in *QSIC*, 2011, pp. 198–207.

[7] R. Shank, "Representation and Understanding of Text," *Machine Intelligence*, vol. 8, pp. 575–607, 1977.

[8] International Standard ISO/IEC 9126-1, “Software engineering - Product quality - Part 1: Quality model,” 2001.

[9] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Academic Publishers, 1999.

[10] H. Kaiya and A. Ohnishi, “Improving software quality requirements specifications using spectrum analysis,” in *COMPSAC Workshops*, 2012, pp. 379–384.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning*, new ed. Springer-Verlag, 2008.

[12] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: an update,” *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[13] International Standard ISO/IEC 25021, “Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Quality measure elements,” Oct. 2007.

[14] D. Firesmith, “Quality Requirements Checklist,” *Journal of Object Technology*, vol. 4, no. 9, pp. 31–38, Nov.-Dec. 2005.

[15] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, “The Architecture Tradeoff Analysis Method,” in *IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 1998, pp. 68–.

[16] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley, 2003.

[17] I. Ozkayad, L. Bass, R. S. Sangwan, and R. L. Nord, “Making Practical Use of Quality Attribute Information,” *IEEE Software*, vol. 25, no. 2, pp. 25–33, Mar./Apr. 2008.

[18] H. Kaiya, M. Tanigawa, S. Suzuki, T. Sato, and K. Kaijiri, “Spectrum analysis for quality requirements by using a term-characteristics map,” in *CAiSE*, 2009, pp. 546–560.

[19] Y. Zhang, Y. Liu, L. Zhang, Z. Ma, and H. Mei, “Modeling and Checking for Non-Functional Attributes in Extended UML Class Diagram,” in *Annual IEEE International Computer Software and Applications Conference (COMPSAC2008)*, 2008, pp. 100–107.

[20] Z. M. Yi Liu and W. Shao, “Integrating Non-Functional Requirement Modeling into Model Driven Development Method,” in *17th Asia-Pacific Software Engineering Conference (APSEC 2010)*, Dec. 2010, pp. 98–107.

[21] D. V. Dzung and A. Ohnishi, “Improvement of quality of software requirements with requirements ontology,” in *QSIC*, 2009, pp. 284–289.

[22] D. G. e Silva, M. Jino, and B. T. de Abreu, “Machine learning methods and asymmetric cost function to estimate execution effort of software testing,” *Software Testing, Verification, and Validation, 2008 International Conference on*, vol. 0, pp. 275–284, 2010.

[23] E. Ceylan, F. O. Kutlubay, and A. B. Bener, “Software defect identification using machine learning techniques,” *EUROMICRO Conference*, vol. 0, pp. 240–247, 2006.

[24] R. Ferenc, Á. Beszédes, L. J. Fülöp, and J. Lele, “Design pattern mining enhanced by machine learning,” in *ICSM*, 2005, pp. 295–304.

[25] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, “Automated classification of non-functional requirements,” *Requir. Eng.*, vol. 12, no. 2, pp. 103–120, 2007.

Abductive Logic Programming with Tabled Abduction

Luís Moniz Pereira*, Ari Saptawijaya *†
*Centro de Inteligência Artificial (CENTRIA)
DI/FCT Universidade Nova de Lisboa
2829-516 Caparica, Portugal
Email: lmp@fct.unl.pt, ar.saptawijaya@campus.fct.unl.pt
†Fakultas Ilmu Komputer Universitas Indonesia
Kampus UI Depok 16424, Indonesia

***Abstract*—In abductive logic programming, abductive solutions are typically computed without attending to the abductive context. These abductive solutions can actually be reused in a different abductive context. In this paper, we employ a tabling mechanism and propose a tabled abduction mechanism, that consists of a transformation from abductive normal logic programs into tabled dual programs, by tabling abductive solution entries and without requiring any meta-interpreter. Recomputation of abductive solutions for a different context, but consistent with them, can then be avoided, by reusing the tabled abductive solution entries. Though our implementation is in XSB-Prolog, its concepts may be imported to other systems, not necessarily Logic Programming ones.**

***Keywords-tabled abduction*; *abduction transformation*; *well-founded semantics*; *XSB-Prolog*.**

I. Introduction

Abductive logic programming offers a formalism to declaratively express and solve problems in a variety of areas, e.g., decision-making, diagnosis, planning, belief revision and hypothetical reasoning (cf. [1]–[4]). On the other hand, the tabling mechanism, now supported by a variety of Prolog systems, ensures termination and optimal complexity for query evaluation to a large class of logic programs, viz. under the Well-Founded Semantics [5].

We explore the idea of how to benefit from the tabling mechanism in order to reuse priorly obtained abductive solutions in a given new abductive context. For so doing, we propose a tabled abduction system, dubbed TABDUAL, which includes a transformation (itself a logic program) from abductive normal logic programs into tabled dual programs, that allows query evaluation without resorting to a meta-interpreter. The transformation makes use of the formal dual transformation defined in [6], and implements an innovative and pragmatic re-uptake of prior abductive solution entries in tabled predicates. More precisely, we contribute on the following aspects:

- We cater for various critical issues in the program transformation, illustrated in successive prototypical examples. They incrementally show how the transformation evolves from the basic core idea into its current actual state, by gradually dealing with the unavoidable issues arising, viz. introducing tabled predicates, dealing with direct positive loops as well as positive and negative loops over negation, handling programs with variables and also non-ground queries.
- We have developed a prototype for TABDUAL based on its specification [7]. It has been tested with benchmark core examples pertaining to each of the above mentioned issues, under the latest version of XSB (version 3.3.6).
- We discuss some ideas on how to migrate core features of TABDUAL into the underlying engine-level of Prolog systems wanting to encompass tabled abduction.

The paper is structured as follows. Section 2 reviews basic logic programming and abductive logic programming notions. The motivation and the key concept of tabled abduction, along with its implementation, and related work are then discussed, in Section 3. The ideas on how to migrate key ingredients of TABDUAL into an engine-level of Prolog systems are discussed in Section 4, and we conclude in Section 5.

II. Abductive Logic Programming

We briefly review in this section the abductive logic programming formalism, which has been well studied for a few decades by now [8]–[13]. We start with basic logic programming background. A *logic rule* has the form

$$H \leftarrow B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_n$$

where $n \geq m \geq 0$ and H, B_i with $1 \leq i \leq n$ are atoms. H and $B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_n$ are the head and the body of the rule, respectively. Any variables occurring in a rule are universally quantified. We use 'not' to denote default negation. The atom B_i and its default negation $not\ B_i$ are named positive and negative *literals*, respectively. When $n = 0$, we say the rule is a *fact* and write it simply as H. The atoms *true* and *false* are, by definition, respectively true and false in every interpretation. A rule in the form of a denial with the empty head, or equivalently with *false* in the head, is called an *integrity constraint* (IC) or *denial*. A *normal logic program* is a set of such logic rules, where non-ground rules (i.e., rules containing variables) stand for all their ground instances. A program may represent an infinite set of rules, when it contains at least one function symbol. For example, consider the following program with two rules:

$$nat(0). \qquad nat(s(X)) \leftarrow nat(X).$$

The second rule stands for an infinite number of its ground instances (as we have one function symbol s and a constant 0), namely:

$$\begin{array}{lcl} nat(s(0)) & \leftarrow & nat(0). \\ nat(s(s(0))) & \leftarrow & nat(s(0)). \\ nat(s(s(s(0)))) & \leftarrow & nat(s(s(0))). \\ \ldots \text{ etc.} & & \end{array}$$

Logic program theoretical semantics are usually defined by reference to all the ground instances of a program's rules. This set of instances may be infinite even when the set of rules (with variables) is finite, as exemplified above. Rule bodies however must, by definition, be finite. In programming practice, necessarily finite programs only, albeit with variables, are employed. Consequently, note that in the construction of the dual program, discussed in Section III, we ensure that its rules' bodies are still finite, because our transformation applies to possibly non-ground but nevertheless finite programs.

Abduction, or inference to the best explanation (a common designation in the philosophy of science), is a reasoning method whereby one chooses those hypotheses that would, if true, best explain observed evidence.

An *abductive logic program* is a normal logic program that allows for abducibles (i.e., positive literals with no rules in the program), or their default negations, to appear in the body of rules. Abducibles stand for hypotheses, whose truth value is not assumed initially. Queries to the program represent the evidence we would like to explain, by resorting to a given set of permitted abducibles. Abducibles in rules' bodies may have arguments, but must be a ground instance on the occasion of their abduction. Note that the negation '*not* A' of an abducible A does not refer to its default negation, but instead to the explicitly assumed negation of A. The truth value of abucibles may be independently assumed *true* or *false*, either in their positive or negated form, as the case may be, in order to produce an abductive solution to a query, by means of a consistent set of assumed hypotheses.

An *abductive solution* to a query is thus a consistent set of abducible instances or their negations that, when substituted by their assigned value *true* everywhere in the program P, affords us with a model of P (for the specific semantics used on P), which satisfies both the query and the ICs – a so-called *abductive model*. Because explicit negation entails default negation, the substitution above is warranted. Often, an abductive solution S for goal G in program P is defined as a set of abducibles consistent with P such that $P \cup S \models G$ in the semantics employed. The above definition of abductive solutions is consistent with this one, but avoids the complication of introducing the positive and the negative abducibles of a solution as facts. Instead, they are replace by true wherein the program. Note that though the semantics may be other than two-valued, the abductive solutions themselves are enforced two-valued by definition.

When performing abductive reasoning, we typically wish to find, by need only, via top-down computation, the abductive solutions to a query. This top-down computation, dubbed backward chaining, is possible only when the underlying semantics is relevant, a property enjoyed by the Well-Founded Semantics (WFS) [5]. That is, it guarantees that it is enough to use only the rules relevant to the query (those in its procedural call-graph) to find its truth value, thus it avoids computing a whole model in order to find an answer to a query. In order to satisfy the ICs too, we conjoin each query with '*not false*'. Mark that when a semantics enjoys relevancy, the values of abducibles not mentioned in the abductive solution are indifferent to the solution. We use WFS in the implementation of our prototype tabled abduction system, TABDUAL, described in Section III-E.

III. Tabled Abduction

We begin with issues that motivate the need for tabled abduction, and subsequently propose a novel abduction system, termed TABDUAL, that involves a transformation of an abductive logic program into another program with tabled abductive solutions.

A. Motivation and Related Work

An abductive solution to a problem is typically formed from the abductive solutions of its subproblems. Consider the following abductive logic program.

$$q \leftarrow a. \qquad r \leftarrow b, q. \qquad p \leftarrow r, q.$$

where a and b are abducibles. Suppose three queries: q, r and p, are launched, in that order. Query q simply gives $[a]$ as the abductive solution. The next query, r, is typically solved by abducing b and followed by invoking q. But since q has previously been invoked, query r can in fact be solved by reusing the previously obtained abductive solution $[a]$ of q and extending it with the newly abduced abducible b; yielding $[b, a]$ as the abductive solution to r. One may also view that solving query r amounts to extending the ongoing current abductive context $[b]$ when q is invoked, with the priorly discovered and registered in a table, abductive solution $[a]$ of q, resulting in $[b, a]$. Using similar reasoning, the final query p can be solved by reusing the abductive solutions of r and q. More precisely, we can view that the abductive solution $[b, a]$ of r in the body of p becomes the current abductive context of q and since it subsumes the previously obtained abductive solution $[a]$ of q we can safely have $[b, a]$ as the abductive solution to query p.

The above example illustrates how abductive solutions can actually be reused in a new abductive context, avoiding unnecessary recomputation of abductive solutions to subproblems, and thereby gaining in efficiency (imagine if the definition of the first rule q is scaled up by another huge program). One may observe that the table size would be proportional to the number of distinct (positive) goals in the procedural call-graph, i.e., each first call of the goals in a given query will table, as the solution entry, the abductive solutions of the called goal.

Reusing solutions in logic programming, without abduction, is commonly performed using the tabling mechanism. It therefore allows dealing with loops in the program, ensuring termination of looping queries. Surprisingly, to the best of our knowledge, no work has addressed how reusing abductive solutions, as we hinted above, can be realized using the tabling mechanism. Tabling has only been employed limitedly, i.e., to table a meta-intepreter, which in turn allows abduction to be performed in the presence of loops in a program, but with no analysis of abductive solution subsumption at all (cf. ABDUAL [6], [14]).

Our current work pushes the benefit of the tabling mechanism to abduction, by employing it to table abductive solution entries (effectively achieved by tabling language-level predicates carrying these entries) for their later reuse in new abductive contexts, by abductive subsumption.

Our approach differs from that of [15]. Therein, abducibles are coded as odd loops, it is compatible with and uses constructive negation, and involves manipulating the residual program. It suffers from a number of problems, which it identifies, in its Sections 5 and 6, and its approach was not pursued further.

Like ABDUAL [6], we use the dual transformation and rely on the same theoretic underpinnings, but the ABDUAL code caters only for ground programs and queries. It also requires a meta-interpreter which makes tabled abduction awkward because it is enacted only at the level of tabled meta-interpreter predicates, and does not cater for abductive subsumption. In short, it affords no particular treatment for the tabling of abduction. Moreover, by design of omission, it does not address at all the issues raised by the desirable reuse of tabled solutions. We employ no meta-interpreter, but generate a self-sufficient program transform. Hence, it avoids meta-interpretation and, moreover, explicitly addresses the concerns of making better use of tabling for abduction, so that known abductive solutions may be appropriated by subsequent abductive goals. We do so not just in core conception, but also adumbrating optimizations and quick-kill trickery.

Our tabled abduction implementation, termed TABDUAL, also allows dealing with programs containing loops, and thus extending the usual tabling mechanisms. We have conducted an experiment to compare TABDUAL with the meta-interpreter ABDUAL [14] using our test suite of programs

```
abds([a/0,b/0,c/0]).

p0 <- q0.      p3 <- q3.          p4 <- q4.
p0 <- a.       q3 <- not r3.      q4 <- p4.
q0 <- p0.      r3 <- p3.          q4 <- not a,not b.
q0 <- b.

p8 <-  not q8, a.      p11 <-  not q11, a.
q8 <-  not p8.         q11 <-  p11,not a.
q8 <-  b.
```

Figure 1. Some programs with loops from the test suite. Predicate $abds$ lists the abducibles with their corresponding arity.

Table I
COMPARISON OF RESULTS: TABDUAL VS. ABDUAL FOR FIGURE 1

Queries	TABDUAL	ABDUAL
not p0	[not a, not b]	[not a, not b], [not a]
p3	[] undefined	[]
not p3	[] undefined	[]
not p4	[a], [b]	[a], [b], [a,b]
q8	[] undefined, [not a], [b]	[not a], [b]
not q11	[a], [not a]	[], [a], [not a]

with various kinds of loops. Some distinguishing results, with respect to the programs given in Figure 1, are shown in Table I; the complete result is available in [7]. TABDUAL provides more correct and complete results within the test suite, given that our scope of programs and queries is more general than ABDUAL's:

- For query $not\ p0$, $[not\ a, not\ b]$ should be the only solution, because $not\ p0$ succeeds by abducing $not\ a$ **and** failing $q0$. To fail $q0$, $not\ b$ has to be abduced **and** $p0$ has to fail. Here, there is a positive loop on negation between $not\ p0$ and $not\ q0$, so the query succeeds and gives the solution $[not\ a, not\ b]$ as the only solution.
- For queries $p3$ and $not\ p3$, unlike ABDUAL, TABDUAL returns *undefined* (and abduces nothing) as expected, due to the negative loops over negation.
- Query $not\ p4$ shows that TABDUAL does less abduction than ABDUAL, by abducing a or b only; not both.
- For query $q8$, TABDUAL has an additional answer: [] *undefined* (i.e., undefined by abducing nothing), due to the negative loop over negation between $q8$ and $p8$. Similar reasoning equally applies to query $not\ p8$. This additional answer is missing by ABDUAL.
- For query $not\ q11$, the first solution is obtained by abducing a to fail $q11$. Another way to fail $q11$ is to fail $p11$, which gives another solution, by abducing $not\ a$. These are the only two abductive solutions which are returned by TABDUAL and follows correctly the definition of abductive solutions. There is no direct positive loop involving $q11$ in the program, hence $not\ q11$ will never succeed with [] abductive solution, as returned by ABDUAL.

We have not yet compared efficiency, but it seems apparent that TABDUAL is an improvement over ABDUAL.

TABDUAL does not concern itself with constructive negation, like the NegABDUAL system [16] and its follow-up [17]. NegABDUAL uses abduction to provide constructive negation plus abduction, by making the disunification predicate '\=/2' an abducible. Again, it does not concern itself with the issues of tabled solutions reuse, which is the main purpose of TABDUAL.

NegABDUAL transforms programs and uses minimal meta-interpretation. Its transformations are intricate, especially on account of its constructive negation design goal, which is no concern of TABDUAL. In the past, we have used it extensively; but, for now, we have not yet compared it to TABDUAL. It will surely be heavier and not benefit from our abductive solutions tabling. However, because of its constructive negation ability, NegABDUAL can deal with problems that TABDUAL does not, and intends not, precisely because it aims at being lighter and more adaptable. Consider program P, with no abducibles, just to illustrate the point of constructive negation induced by dualization:

$$p(X) \leftarrow q(Y). \qquad q(1).$$

In NegABDUAL, the query $not\ p(X)$ will return a qualified 'yes', because it is always possible to solve the constraint Y\=1, as long as one assumes there are at least two constants in the Herbrand Universe. Indeed, the local variable Y in the dualizing transmutation of $p(X)$ produces a default negation. In TABDUAL implementation, there is no floundering. However, distinct from NegABDUAL, our TABDUAL answers 'no' to $not\ p(X)$, which is correct, even in the absence of a conditional answer afforded only by having constructive negation in place.

To the best of our knowledge, no other work on using tabling and dual programs for abduction exists, nor has the problem of tabled abduction that we currently addressed even been formulated by others.

We next introduce TABDUAL through a sequence of prototypical examples, which illustrates how, for easier understanding, it incrementally evolves to cope with the major issues of concern arising in tabled abduction. We begin with the key transformation employing the very idea of tabling and of reusing abductive solutions. We further successively argue why and how more constructs should be added to the transformation, to gradually deal with the issues involved, as illustrated in ever more complex and demanding examples.

In TABDUAL, we introduce a so-called *abductive context* (referred to "context" hereafter), as illustrated in the example at the beginning of this section, for every predicate defined by the transformation. The context indicates the ongoing tentative abductive solution for any given tabled goal. It allows to relay the abductive solution from the head to the body of a rule and back, and from one subgoal to subsequent subgoals. The context is effectively catered for and realized by adding two extra arguments to every predicate, as defined by the transformation: one for the input context and the other for the output context of any ongoing abductive solution for the predicate. Abducibles are 'parsed' as terminals in an ongoing derivation and tracked in these additional arguments.

```
1.  :- table p_ab/1, q_ab/1.
2.  p_ab(E) :- q([a],E).
3.  q_ab([b]).          q_ab([c]).
4.  p(I,O) :- p_ab(E),produce(O,I,E).
5.  q(I,O) :- q_ab(E),produce(O,I,E).
6.  p_st(I,O) :- p_st_1(I,O).
7.  q_st(I,O) :- q_st_1(I,T),q_st_2(T,O).
8.  p_st_1(I,O) :- not_a(I,O).
9.  p_st_1(I,O) :- not_q(I,O).
10. q_st_1(I,O) :- not_b(I,O).
11. q_st_2(I,O) :- not_c(I,O).
12. not_p(I,O) :- tnot(p_ab([])),p_st(I,O).
13. not_q(I,O) :- tnot(q_ab([])),q_st(I,O).
```

Figure 2. Main rules obtained from the transformation of P_1

We fix some notation. We use capital letters to denote variables appearing in a program and write P/N to denote that predicate P has arity N. In the examples, the set of abducible atoms are declared in the predicate $abds/1$. In the result of the transformation, we use $:-$ to separate the head and the body of a rule, instead of $\leftarrow$. As the implementation is done in XSB Prolog [18], [19], we borrow from it its syntax, e.g., for the tabled negation $tnot/1$ and for the compiler directive to declare tabled predicates.

B. Basic Idea

We start with the basic idea by means of the example below. Consider ground program P_1 below.

$$abds([a/0, b/0, c/0]). \quad p \leftarrow a, q. \quad q \leftarrow b. \quad q \leftarrow c.$$

The transformation produces, for every defined predicate in P_1 ($p/0$ and $q/0$), several sets of rules. The transformation of P_1 contains the main rules as shown in Figure 2.

The first set of rules defines the tabled predicates $p\_ab/1$ and $q\_ab/1$, declared in line 1. The tabled predicate essentially tables the abductive solution entry, always assuming, for facilitating reuse, empty input context. Since the input context calls for tabled predicates are always empty, only one extra argument (i.e., for solution entry context) is needed. The p_ab rule (line 2) is derived from the p rule of the original program, where p is defined by the subgoals q and by abducing a. The tabled entry E of p_ab is passed from the output context of q, i.e., the second argument of the subgoal q. Since q rules in the original program are defined solely by abducibles, they are transformed into facts q_ab with those abducibles as their abductive solution entries (line 3).

The second set of rules provides the definition of $p/2$ and $q/2$, now with the input and output contexts I and O (lines 4-5). Predicate $p/2$, for example, reuses the abductive solution entry E from tabled predicate p_ab and then, using it together with the input context I to produce its output context O. Similar reasoning applies equally to predicate $q/2$.

The context updating is performed externally by predicate $produce/3$. It concerns itself with: whether E is already contained in I and whether there are any abducibles from E, consistent with I, that can be added to produce O. If E is inconsistent with I then the specific entry E cannot be reused with I, produce fails and another entry E is sought. In other words, $produce/3$ should guarantee that it produces a consistent output context O from I and E, eliminating any redundant abduction.

The third set of rules (lines 6-11) contains the *dual* rules, $p\_st/2$ and $q\_st/2$, where p_st is true iff p of the original program is false [6]; similar reasoning also applies to $q\_st/2$. Note that the st stands for '$*$', as in p^*, a notation often used in abduction to denote the negation of p. The set of dual rules is defined in two layers. The first layer (lines 6-7) captures the idea that, e.g., to make q_st (line 7) true, we need to (non-deterministically) fail each q rule: q_st is defined by q_st_1 and q_st_2 that correspond to both failing the first and the second q rules, respectively. Note that the abductive solution from subgoal q_st_1 is relayed to the subsequent subgoal q_st_2 via the intermediate context T. The second layer of q's dual rules (lines 10-11) defines how to fail each q rule, by alternatively and non-deterministically failing one subgoal in q's body at a time, i.e., by negating just one literal in q's body at a time, in order to avoid excessive abduction. The negated literal is renamed into its corresponding positive one, e.g., $not\ a$ into not_a, and also equipped with the input and output contexts. The two-layer dual definition for p (lines 6 and 8-9) follows similar reasoning. The first layer p_st (line 6) is simpler, because in the original program p has only one rule.

The fourth set of rules consists of the *negated* rules $not\_p/2$ and $not\_q/2$ (lines 12-13). These will be the ones that generate abductive solutions for the such 'positive' literal, say not_p. Indeed, every default literal is replaced with its corresponding positive. In line 12, predicate $not\_p(I, O)$ is defined by two subgoals: $tnot(p\_ab([]))$ and $p\_st(I, O)$. The first subgoal serves as an optimization, to immediately fail not_p without the need to launch the call to the more elaborate p_st rule that follows. The idea behind the 'quick-kill' $tnot(p\_ab([]))$ is to permit to see whether goal '$not\ p$' has no hitting set at all, pertaining to the set of abductive solutions of p_ab. This is done by inspecting whether p_ab has an empty abductive solution entry, i.e., p can be satisfied without abducing anything, in which case not_p can immediately fail. Indeed, an abductive solution of the negation of positive atom A is construable as a set that negates the members of a hitting set for the abductive solutions of A. If one of these latter sets is empty then no hitting set exists. Our approach consists in generating such hitting sets incrementally, by means of finding abductive solutions to the dual rules of A, without thus having to wait for the explicitly availability of all abductive solutions for A. Nevertheless, a quick-kill option is readily available, just in case there exists an empty abductive solution for A. An optimization consists in simply detecting if such an entry is already in the table for A, rather than generating solutions for A trying to produce the empty one.

In addition to the rules shown in Fig 2, for each predicate having no rule in the original program, a fact about its negation is added. For example, since there is no IC defined in P_1, fact $not\_false(I, I)$ is added. Note that the two contexts are the same. Having no body, the output context does not depend on the context of any other goals, but depends only on its corresponding input context. When ICs exist, they are transformed exactly like the dened predicates.

Finally, for each abducible, a pair of rules is created: a rule for the positive abducible, e.g., $a(I, O) \colon\!-insert(a, I, O)$, and another rule for its negation, e.g., $not\_a(I, O) \colon\!-insert(not\ \ a, I, O)$. Similar pairs of rules are also added for abducibles b and c, and their negations. Note that the rules are defined by the external predicate $insert/3$, which inserts the corresponding abducible with respect to the input context I and results in the output context O. It maintains a consistent context during the insertion, and avoids redundant abduction.

A query to a program, consequently, should be transformed too, in order to conform to the transformation: positive goals are augmented with the two extra arguments for the abductive context, whereas negative goals are made 'positive' in addition to the two extra context arguments. Moreover, a query should always be conjoined with $not\_false/2$ to ensure that all ICs are satisfied. For example, query $not\ p$ is transformed into $not\_p(I, O)$. Its complete call, as a top goal, becomes $not\_p([], T)$, $not\_false(T, O)$, where O is an abductive solution to the query, given initially an empty input context. Note, how the abductive solution for not_p is further constrained by passing it to the subsequent subgoal not_false for confirmation, via the intermediate context T.

Note that at this point we are not concerned with incremental IC checking, as this is not a specific tabled abduction problem, but a general tabling problem that others are addressing and that tabled abduction does not preclude any reuse of.

C. Dealing with Loops

The next examples concern programs involving loops between predicates. Consider the ground program P_2 below.

$$abds([a/0]). \qquad p \leftarrow q, a. \qquad q \leftarrow p.$$

XSB with its tabling mechanism supports Well-Founded Semantics [5], which would detect direct positive loops and fail predicates involved in such loops. For P_2, query p fails, due to the direct positive loop between tabled predicates $p\_ab/1$ and $q\_ab/1$. On the other hand, query $not\ p$ should succeed with two abductive solutions $[]$ and $[not\ \ a]$. The call to the latter query, after transformation, becomes $not\_p([], T), not\_false(T, O)$. Instead of succeeding, the first subgoal $not\_p([], T)$ will loop indefinitely! This loop occurs because of the mutual dependency between $not\_p/2$ and $not\_q/2$ through $p\_st\_1/2$ and $q\_st\_1/2$. The dependency creates a positive loop on negative non-tabled predicates, and such loops should succeed, precisely because the corresponding source program's loop is a direct one on positive literals, which hence must fail. Indeed, since any source program's direct positive loops must fail, the loops between their corresponding transformed negations must succeed [6]. For example, whereas $r \leftarrow r$ fails query r, perforce $not\_r \leftarrow not\_r$ succeeds query not_r. The problem can be remedied by detecting such loops in a program. Since XSB's tabling mechanism already supports dealing with direct positive loops, we need only concern ourselves with positive and negative loops on negation in the transform.

*1) **Positive Loops on Negation (PLoN):*** We detect PLoN by tracking the ancestors of negative subgoals, whenever they are called from other negative subgoals. In the transformation, a list of ancestors, dubbed the *close-world-assumption (CWA) list* is maintained and serves as another extra argument for the dual and negated rules. The new transformation, with PLoN detection, of P_2 is shown below (without showing the usual transformation for not_false and the abducibles).

```
1.  :- table p_ab/2, q_ab/2.
2.  p_ab(E) :- q([a],E).     q_ab(E) :- p([],E).
3.  p(I,O) :- p_ab(E),produce(O,I,E).
4.  q(I,O) :- q_ab(E),produce(O,I,E).
5.  p_st(I,O,CWA) :- p_st_1(I,O,CWA).
6.  p_st_1(I,O,CWA) :- not_q(I,O,[not p|CWA]).
7.  p_st_1(I,O,_) :- not_a(I,O).
8.  q_st(I,O,CWA) :- q_st_1(I,O,CWA).
9.  q_st_1(I,O,CWA) :- not_p(I,O,[not q|CWA]).
10. not_p(I,I,CWA) :- member(not p,CWA), !.
11. not_p(I,O,CWA) :- tnot(p_ab([])),p_st(I,O,CWA).
12. not_q(I,I,CWA) :- member(not q,CWA), !.
13. not_q(I,O,CWA) :- tnot(q_ab([])),q_st(I,O,CWA).
```

The list is only updated in the second layer of dual rules, which are essentially the rules for negative goals (cf. lines 6 and 9). The update is done by adding the negative goal (without any context) into the CWA list of the negative subgoal in the body. For example, in case of p_st_1 (line 6), not_p is added into the CWA list of the subgoal not_q. Note that in line 10, another rule of not_p is added (similarly in line 12, for not_q) to detect PLoN, by membership testing, i.e., whether we are returning to the same call of not_p. In that case, the output context is equal to the input context. By placing this additional rule before the other not_p rule (line 11), we anticipate the loop by immediately succeeding it and, using cut, to prevent the call to the next not_p (which would lead to looping).

*2) **Negative Loops over Negation (NLoN):*** XSB with its Well-Founded Semantics and tabling mechanism is aware of negative loops over negation (NLoN) and makes predicates involved in such loops undefined. Consider the ground program P_3 below:

$$p \leftarrow q. \qquad\qquad q \leftarrow not\ p.$$

where p and q are tabled predicates and, written in XSB, the tabled negation $tnot/1$ is used instead of $not/1$ to so indicate. In this example, p and q (also their default negations) are undefined. The CWA lists previously introduced are able to detect PLoN, but not NLoN. Query p, for example, with respect to the transformation (in the presence of CWA lists) will fail, instead of being undefined. It fails, because the tabled predicate p_ab is involved in a direct positive loop through the call of not_p and not_q. More precisely, whereas in the source program q is defined by the negative subgoal *not* p, in the resulting transformation q_ab is defined by the positive subgoal not_p. Hence, one way to resolve the problem is to wrap the positive subgoal not_p in the body of the rule q_ab with the tabled negation predicate ($tnot/1$ in XSB) twice so as to keep its truth value; thereby creating NLoN (instead of direct positive loops), but also preserving the semantics of the rule. Apart from other usual predicates produced by the transformation, the new definition of q_ab is as follows:

```
1. :- table q_ab/1, over/1, not_p/1, p_st/3.
2. q_ab(E) :- tnot p_ab([]),not_p_ab([],E).
3. not_p_ab(I,O) :- call_tv(tnot over(not_p(I)),V),
         (V = undefined, O = I, undefined;
          inspect(p_st(I,O,[]))).
4. not_p(I) :- p_st(I,O,[]).
```

Here, $tnot\ over(not\_p(I))$ is the double-wrapping of not_p with $tnot$. It is realized via the intermediate tabled predicate $over/1$, defined as `over(G) :- tnot(G)`. The double-wrapping is called through an auxiliary predicate $not\_p\_ab/2$. The XSB system predicate $call\_tv/2$ calls the double-wrapping and unifies V with its truth value (*true* or *undefined*). The value of the output context O then depends on V's value: it is equal to the input context I when NLoN exists (i.e., V is undefined) or O's value is inspected from the tabled predicate p_st by means of predicate $inspect/1$, in case NLoN does not exist (i.e., V is true). The predicate $inspect/1$ can be defined, in XSB, using the combination of its table inspection predicates $get\_calls/3$ and $get\_returns/2$.

It is tempting to use the existing *not_p(I,O,CWA)* in the double-wrapping. Unfortunately, it would cause the call to $over/1$ to flounder, because the output context O is still uninstantiated; hence $not\_p/1$ is introduced instead, free from the output context. Note that in $not\_p/1$ we also omit the CWA context, because the call is made from q_ab, which actually has an empty CWA context (recall that the CWA context is only relevant for dual rules). Indeed, in the definition of $not\_p/1$ (line 4), where it is defined by the dual rule $p\_st/3$, the CWA context of p_st is empty. Its definition is similar to the $not\_p/3$ definition, except that the quick-kill $tnot\ p\_ab([])$ is moved into the q_ab definition (line 2). That is, it prevents the quick-kill to be wrapped in the $tnot$. Moreover, p_st is tabled, so that its output context O, which is computed when $not\_p/1$ is evaluated in the double-wrapping, can later be reused, via $inspect/1$.

D. Programs with Variables

The problem gets more interesting when we have variables in the program and we consider non-ground queries. Consider program P_4.

$$abds([a/1]). \quad \leftarrow q(X), r(X). \quad q(1). \quad r(X) \leftarrow a(X).$$

We shall discuss how to adapt the construction of dual rules involving predicates with variables. Recall that the IC in P_4 is transformed like any other rule. Instead of only placing a negated literal in the body of a (second layer) dual rule, we are also going to keep all positive non-abducible literals of the original rule that appear before this negated literal. For example, the second dual rule $false\_st\_1$ of the IC in P_4 is now defined by the negated literal $not\_r/4$ and also by all positive non-abducible literals that appear before this negated literal in the original rule, in this case $q/3$, as shown below.

```
false_st(I,O,CWA) :- false_st_1(I,O,CWA).
false_st_1(I,O,CWA) :- not_q(X,I,O,CWA).
false_st_1(I,O,CWA) :- q(X,I,T),
       not_r(X,T,O,[not false|CWA])).
```

The idea to keep these literals before the negated literal in the body of dual rules is to provide an opportunity for the negated literal to be ground when it is called. For example, $not\_r/4$ in the transformation of P_4 can be made ground because X is instantiated when $q/3$ is called, i.e., $X = 1$, like the case in the original rule. This avoids floundering when $tnot(r\_ab(X,[]))$ is called through $not\_r/4$ due to the uninstantiated X. Query $q(1)$, for example, will now correctly return the abductive solution $[not\ a(1)]$.

There are some points to remark on regarding this refinement. First, the newly introduced positive literals should be tabulated to avoid duplication of their derivations. Second, one could also introduce all other positive literals in the rule originating the 'not'. This may help produce additional grounding, though, in going against the left-to-right pragmatics coded by the programmer it may create inefficiencies of its own. Third, the semantics doesn't change because the conditions for failure of the positive rules are that one literal must fail even if the others succeed. The cases where the others do not succeed are handled in the other dual cases. Finally, knowledge of argument types (+, −, ?), and of shared variables in the body and whether they are local or not, would help refine the transformation to avoid introducing positive literals not contributing to further grounding.

In yet another grounding refinement, TABDUAL also allows us also to deal, in the quick-kill rules, with non-ground positive goals. For example, query $q(X)$ gives the abductive solution $[(not\ a(1)]$ for $X = 1$. But, non-ground negative goals, like $not\ q(X)$ flounders due to the uninstantiated X in the quick-kill $tnot(q\_ab(X,[]))$. To resolve the problem, we may restrict the applicability of the quick-kill to ground negative goals only; non-ground negative goals may immediately call the $q\_st/4$ rule without calling the quick-kill. This leads to an improved transformation that allows us to have non-ground negative goals without a $tnot/1$ floundering error. The further refinement of the transformation with respect to the negated rules $not\_q/4$ of P_4 is shown below ($not\_r/4$ is also treated similarly):

```
not_q(X,I,O,CWA) :-
      (ground(not_q(X)),tnot(q_ab(X,[]));
       \+ground(not_q(X))), q_st(X,I,O,CWA).
```

The following example will take us to the actual final transformation. Consider program P_5.

$$\begin{array}{lll} abds([a/1]). & p(X) \leftarrow q(X). & p(1) \leftarrow a(1). \\ & q(X) \leftarrow p(X). & q(2) \leftarrow a(2). \end{array}$$

Query $p(X)$ to program P_5 succeeds under TABDUAL, giving two abductive solutions: $[a(1)]$ and $[a(2)]$ for $X = 1$ and $X = 2$, respectively. But query $not\ p(X)$ does not deliver the expected solution: the only solution returned is $[not\ a(1)]$ for a particular $X = 1$, instead of the expected $[not\ a(1), not\ a(2)]$ for any instantiation of X. The culprit is in the p_st definition used to answer the query:

```
p_st(Y,I,O,CWA) :- [Y] = [X], [Y] = [1],
                   p_st_1(Y,I,T,CWA),
                   p_st_2(Y,T,O,CWA).
```

Note that the p_st rule above is obtained by unifying the argument Y (from the call) with the argument of each p rule, i.e., X and 1 from the first and second p rule respectively, and then failing both rules by the subgoals $p\_st\_1/4$ and $p\_st\_2/4$. In general, four p_st rules from P_5 can be obtained to fail both p rules, by considering the unification of the call argument Y with the arguments X and 1 of the two p rule heads. But in practice it can be minimized, by removing unnecessary unifications, as explained in Section III-E.

When the goal $not\_p(X,[],O,[])$ is launched, the variable Y in both p_st_1 and p_st_2 is instantiated with 1. But recall that p_st_1 and p_st_2 are derived from two different p rules, hence failing p should be achieved by calling p_st_1 and p_st_2 independently. In other words, different variants of the calling argument Y should be used in the call of p_st_1 and p_st_2, as shown below:

```
p_st(Y,I,O,CWA) :-  variant([Y],[Y1]), [Y1] = [X],
                    variant([Y],[Y2]), [Y2] = [1],
                    p_st_1(Y1,I,T,CWA),
                    p_st_2(Y2,T,O,CWA).
```

Now the call of p_st_1 and p_st_2 are independent through the use of different variants $Y1$ and $Y2$, respectively. Note we are just looking for abductive solutions for failing a calling goal, not for constraints on its free calling variables.

E. TABDUAL Implementation

We have made an implementation [20] on the basis of the TABDUAL specification [7] and have tested it under the current XSB Prolog version 3.3.6. Apart from the '$tnot$ quick-kill' optimization and the relying of the loop detection as much as possible on the tabling mechanism of XSB, we pushed some optimizations further in the implementation:

- We reorder literals in the body of the rules by prioritizing abducibles to come first in the body. This gives an advantage that the second layer dual rules are first defined by abducibles, if they exist. In this way, abductive solutions for queries with negative goals can be obtained faster, for it is easy to incorporate negated abducibles.
- We simplify the second layer dual rules by removing unnecessary unifications, i.e., those that succeed or fail independently of the instantiation of arguments from the calls. For example, in the refined definition of p_st, from the last paragraph of Section III-D, the unification $[Y1] = [X]$ can be removed because unifying with variables always succeeds. Moreover, other p_st alternative rules with the body containing $[Y] \neq [X]$ can also be removed (these rules always fail).
- We split the list of abductive solutions into positive and negative parts. Thus, checking consistency of abductive solutions, in predicates $insert/3$ and $produce/3$, can be done faster.

In the next section we discuss some ideas on how to migrate key ingredients of tabled abduction into an engine-level of logic programming systems, but they can also be appropriated by other systems.

IV. Discussion

Our high level specification design and its implementation, by means of a transformation in XSB-Prolog, produces a transformed program that aims at being near the potential uptake of certain operations by the underlying engine. We sketch some ideas on how to migrate key constructs of TABDUAL into an engine-level, say like XSB.

1) ***Hiding Data Structures****:* The CWA list (and attending operations), which is being deployed at the object language level, should migrate to the engine level, even disappearing from the generated code. New operations are needed concerning loop detection, in particular making positive loops on negation succeed rather than fail, as it happens with direct positive loops. Similarly, the abductive context can be hidden from the object language and the operations on them moved into the engine level, but with the proviso that these could be inspected for debugging purposes. These signify that, avoiding the data structures being kept, and the operations on them carried out currently at object language level will help improve efficiency.

2) ***Tabling Abduction Entries****:* This is the core feature of our tabled abduction, which needs migration to the (tabling) engine to be more fruitful. At the object language level, we table only the output abductions entries and not the input abductive context to allow for improved reuse, because the input abduction table entries are not there. Reuse and consistency are done at the language level, above the tabling level one.

A new tabling mechanism could instead cater for the two extra table entries, concerning the input and output abducible sets, and provide the special lookup and update mechanisms pertaining to these special sets-arguments. Moreover, the sets would require an efficient data structure representation consistent with the operations on them and the backtracking mechanism.

V. Conclusion and Future Work

We have addressed the issue of tabling abductive solutions, in a way that they can be reused in abductive contexts different from those in which they were produced. We do so by resorting to a program transformation approach, resulting in a tabled abduction implemented system, TABDUAL. It makes use of the dual program technique, whereby abducibles are treated much like terminals in grammars, with an extra argument for input and another for output abductive context accumulation. A few other original innovative techniques are employed to make the approach correct and more efficient, and to bring it closer to the engine level. Thence, the XSB System (and other Logic Programming systems affording tabled negation) can migrate to their inwards what is best done therein.

We hope this will lead, in particular, to an XSB System that can provide its users with specifically tailored tabled abduction. Indeed, abduction is by now a staple feature of hypothetical reasoning and non-monotonic knowledge representation. It is already mature enough in its deployment, applications, and proof-of-principle, to warrant becoming a run-of-the-mill ingredient in a Logic Programming environment.

Future work will consist in perfecting this implementation approach to abduction, and liaising it seamlessly with the underlying engine. That in turn will permit the continued exploration of our applications of abduction and provide feedback for system improvement. For more applications of LP abduction consult our home pages, and references therein.

And a future conceptual application area in the context of abduction in logic, pertains to the issue that, whenever discovering abductive solutions, i.e., explanations, for some given primary observation, one may wish to check too whether some other given additional secondary observations are true, being a logical consequence of the abductive explanations found for the primary observation. In other words, whether the secondary observations are plausible in the abductive context of the primary one, a quite common scientific reasoning task.

Acknowledgment

Ari Saptawijaya acknowledges the support from FCT-Portugal, grant SFRH/BD/72795/2010. We thank David Warren for his comments on prior drafts and corresponding implementation.

References

[1] A. C. Kakas and A. Michael, “An abductive-based scheduler for air-crew assignment,” *J. of Applied Artificial Intelligence*, vol. 15, no. 1-3, pp. 333–360, 2001.

[2] J. F. Castro and L. M. Pereira, “Abductive validation of a power-grid expert system diagnoser,” in *Procs. 17th Intl. Conf. on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA-AIE’04)*, 2004.

[3] J. Gartner, T. Swift, A. Tien, C. V. Damásio, and L. M. Pereira, “Psychiatric diagnosis from the viewpoint of computational logic,” in *7th Intl.Conf. on Principles of Knowledge Representation and Reasoning, NMR ws on Abductive Reasoning*, 2000.

[4] R. Kowalski and F. Sadri, “Abductive logic programming agents with destructive databases,” *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 1, pp. 129–158, 2011.

[5] A. V. Gelder, K. A. Ross, and J. S. Schlipf, “The well-founded semantics for general logic programs.” *J. of ACM*, vol. 38, no. 3, pp. 620–650, 1991.

[6] J. J. Alferes, L. M. Pereira, and T. Swift, “Abduction in well-founded semantics and generalized stable models via tabled dual programs,” *Theory and Practice of Logic Programming*, vol. 4, no. 4, pp. 383–428, 2004.

[7] L. M. Pereira and A. Saptawijaya, “Appendix: Abductive logic programming with tabled abduction,” https://dl.dropbox.com/u/47496395/appendix_icsea12.pdf, retrieved: September, 2012.

[8] A. Kakas, R. Kowalski, and F. Toni, “The role of abduction in logic programming,” in *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. Gabbay, C. Hogger, and J. Robinson, Eds. Oxford U. P., 1998, vol. 5.

[9] M. Denecker and A. C. Kakas, “Abduction in logic programming,” in *Computational Logic: Logic Programming and Beyond*. Springer Verlag, 2002.

[10] M. Denecker and D. de Schreye, “SLDNFA: An abductive procedure for normal abductive programs,” in *Procs. of the Joint Intl. Conf. and Symp. on Logic Programming*. The MIT Press, 1992.

[11] T. Eiter, G. Gottlob, and N. Leone, “Abduction from logic programs: semantics and complexity,” *Theoretical Computer Science*, vol. 189, no. 1-2, pp. 129–177, 1997.

[12] K. Inoue and C. Sakama, “A fixpoint characterization of abductive logic programs,” *J. of Logic Programming*, vol. 27, no. 2, pp. 107–136, 1996.

[13] R. Kowalski, *Computational Logic and Human Thinking: How to be Artificially Intelligent*. Cambridge U. P., 2011.

[14] “ABDUAL System,” http://www.cs.sunysb.edu/~tswift/interpreters.html, retrieved: September, 2012.

[15] J. J. Alferes and L. M. Pereira, “Tabling abduction,” 1st Intl. Ws. Tabulation in Parsing and Deduction (TAPD’98), http://centria.di.fct.unl.pt/~lmp/publications/online-papers/tapd98abd.ps.gz, retrieved: September, 2012.

[16] “NegABDUAL System,” http://centria.di.fct.unl.pt/~lmp/software/contrNeg.rar, retrieved: September, 2012.

[17] V. P. Ceruelo, “Negative non-ground queries in well founded semantics,” Master’s thesis, Universidade Nova de Lisboa, 2009.

[18] “XSB Prolog,” http://xsb.sourceforge.net/, retrieved: September, 2012.

[19] T. Swift and D. S. Warren, “XSB: Extending Prolog with tabled logic programming,” *Theory and Practice of Logic Programming*, vol. 12, no. 1-2, pp. 157–187, 2012.

[20] “TABDUAL System,” https://dl.dropbox.com/u/47496395/tabdual_icsea12.zip, retrieved: September, 2012.

Towards a Methodology for Hardware and Software Design Separation in Embedded Systems

Gaetana Sapienza, Tiberiu Seceleanu
ABB Corporate Research
and Mälardalen University
School of Innovation, Design and Engineering
Västerås, Sweden
{gaetana.sapienza,tiberiu.seceleanu}@se.abb.com

Ivica Crnkovic
Mälardalen University
School of Innovation, Design and Engineering
Västerås, Sweden
ivica.crnkovic@mdh.se

***Abstract*—Development of embedded systems in automation industry often includes development of both software and hardware, which requires both software and hardware expertise. In the current practice these expertise are not often completely combined in synergic ways. Traditionally, design gets separated into hardware design and software design at very early stage which negatively impacts the overall application development process due to design flow interruption and redesign. In order to overcome to the aforementioned problems, this paper presents a new design methodology that provides platform independent design first, and pushes hardware- and software-dependent design to a later stage. This enables "software-independent" hardware and "hardware-independent" software development after the separation stage, which collectively improve the overall development process.**

Keywords: Development Process; Design Methodology; Partitioning; Multi Criteria Decision Analisys (MCDA).

I. INTRODUCTION

The continuous increase in complexity of embedded industrial applications constantly demands improvements of the overall development process. Ideally, the development process has to be able to simultaneously satisfy two main driving requests imposed by the today's market trends: (i) significantly decreasing time-to-market, and (ii) significantly decreasing development and product costs, while preserving quality and launching high-competitive products. In addition to the above, the technology advancements in semiconductor and electronics fields in a combination with the growing demands of providing more sophisticated software functionalities constantly challenge the design methodologies in order to improve the overall development process [1]. Due to the intrinsic nature of embedded systems i.e., the tight coupling between hardware and software, the development process is extremely affected by the efficiency of the design phase which has to rely on methodologies that are able to integrate key paradigms of hardware design and software design in an effective manner.

Traditionally, the system design starts with a separation of software and hardware design [2] at an early stage of the development process. The common practice of the separation into hardware and software is an iterative process, approached in a manually controlled "trial and error" mode, which is not supported by suitable and effective tools or systematic decision process. The hardware-software separation is typically done by invoking individual back-end tools several times in order to later decide which architectural solution appears to be the most suitable one. This approach, unfortunately, is prone to negatively affect the overall application development process due to e.g., issues such as flow interruptions and redesigns.

In this paper, we present a new systematic design methodology which enables hardware and software design separation as late as possible after the overall specification and design activities and a well-structured decision process. The approach is inspired by Model-Driven Architecture with Platform-Independent Model (PIM) and Platform-Specific Model (PSM) stages [3]. PIM identifies software functions independent of the underlying technology, while PSM defines technology-specific solutions. Our approach focuses on the specification and design part of the system valid for both software and hardware (the PIM part), and the design specifically for software and hardware (the PSM part). By doing this, when designing software and hardware specific parts, it is possible to minimize the dependencies between hardware and software after the design separation. Specifically, the proposed methodology will be applied in embedded applications targeting the automation domain. The concepts highlighted in this paper, are supported by years of experience in industry with design methodologies and embedded systems development. The remainder of the paper is organized as follows: the next section discusses the current state of practice for embedded application design in the automation industry domain. Section 3 describes the new proposed design methodology. Section 4 describes a case study. Section 5 concludes the paper and future work is outlined.

II. THE CURRENT STATE OF PRACTICE

A typical software-hardware industrial development process can be described as a number of sequential phases [2][4]: requirements management and system specification, design, implementation, verification and validation, as shown by the diagram A in Figure 1. The development process starts with the specification phase in which requirements are supposed to be identified and analysed. After the specification phase, the design phase usually branches into two separated design flows, for hardware and software, respectively. These flows evolve separately and get into their own implementation. When both hardware implementation and software implementation are completed, the integration takes place. Subsequently the verification and validation phase get in progress. The diagram A depicted by Figure 1 represents a rather simplified development process flow. In reality, it is more complex: phases get interleaved and each of them might require be iterated and/or optimized several times over the entire development process. Consequently, this process meets several serious problems and drawbacks. We describe them briefly for each phase.

A. Early start of the design phase

During the specification phase, and before starting with the design phase, the requirements are expected to be fully finalized in order to efficiently support the design phase. However, in practice due to time and resources constrains the design phase is enforced to start before the requirements have reached a reasonable mature and stable stage. The incompleteness of the specification negatively affects the quality and fluidity of the design phase, and also contributes to originate the issues subsequently described.

B. Early separation into hardware and software

Despite the fact that hardware and software for embedded applications are tightly connected, typically the design phase splits very early into the two design flows. After the separation, hardware and software are considered as two separated activities which are seldom integrated until the integration and verification phases. In principle, (i) hardware does not take into account the computational power required by the software and the capability that the software might offer for enabling hardware optimization and (ii) software does not impact the hardware design specifications, and does not fully exploit the available hardware resources. The too early design start corresponding to the too early flow (hardware and software) separation, does not allow to properly focus on the most important and core part of the design phase which is referred in this paper as *partitioning decision process.* This process is supposed to determine which parts of the application will be designed in hardware and which parts of the application will be designed in software. Problem statement on the partitioning problem can be found in [5].

The impact of the initial decisions is critical since it will condition the remaining development process and the entire application's lifecycle; as a consequence, any decision change afterwards is arduous and costly. Starting a design phase relying on an apparently appropriate set of partitioning decisions potentially poses higher risks for the successful accomplishment of the application development process.

Although the modern design tools (e.g., The MathWorks Simulink®, IBM® Rational® Rhapsody® (UML (Unified Modelling Language)-based tool)) support well the "trial-and-error" approach, in practice the problems remain since a systematic decision process with the appropriate support is missing. Due to the aforementioned aspects related to the early start of the design, it can be highlighted that the development process (as represented by the diagram A in Figure 1) is negatively impacted in terms of quality, costs and time by the following emerging problems: (i) hardware or software flow interruptions and (ii) hardware or software redesigns.

C. Hardware or software design and implementation interruptions

Hardware or software design flow interruptions are observed as a break in the continuity of the design flow, due to the (partial) lack of specifications that have impact on the partitioning. The diagram A in Figure 1 shows a representation of the flow interruptions for both hardware and software. They are undesired since causing an increase in the complexity in the design flow, while affecting the overall quality. The first interruption occurs in the hardware design flow the second interruption occurs during the software implementation.

D. Hardware or software redesign

The need of performing redesign (either hardware or software) is usually dictated by reasons of different nature, e.g., new requirement/s, requirement/s changing, non-feasibility of requirement/s, lack of application-specific knowledge, etc. In literature, research work discussing redesign issues for embedded systems can be found in [1][6]. The hardware and software redesign process is illustrated on the diagram A in Figure 1. It may happen during the initial design, or it can be required after the implementation. It represents one of the most typical scenarios of redesigns encountered in practice: "redesign after implementation" caused by a very late integration of hardware and software. In diagram A in Figure 1, the hardware redesign is caused by the non-feasibility of the requirement A (Req_A) which leads to the necessity of the software redesign due to the non-fulfilment of the requirement (Req_B).

III. The New Approach Proposal

Given the current state of practices in automation industry, we present a new systematic design methodology able of minimizing or even overcoming the issues described above. Our proposal is a process which is mainly characterized by the following key features: (i) providing support/feedback to the specification phase, and (ii) starting with a model-based design common for both software and hardware and continuing with its separation to software-specific and hardware-specific design process when collecting all artefacts that enable software-independent hardware design and hardware-independent software design separation, as depicted in Figure 1 by the diagram B. The explanation of the key features is subsequently done through the description of the proposed approach and the overall overview presented in Figure 2.

The approach is divided into three essential stages: *Identification*, *Decomposition* and *Partitioning*. The Identification stage provides inputs to the Decomposition, while the Decomposition provides inputs to the Partitioning stage.

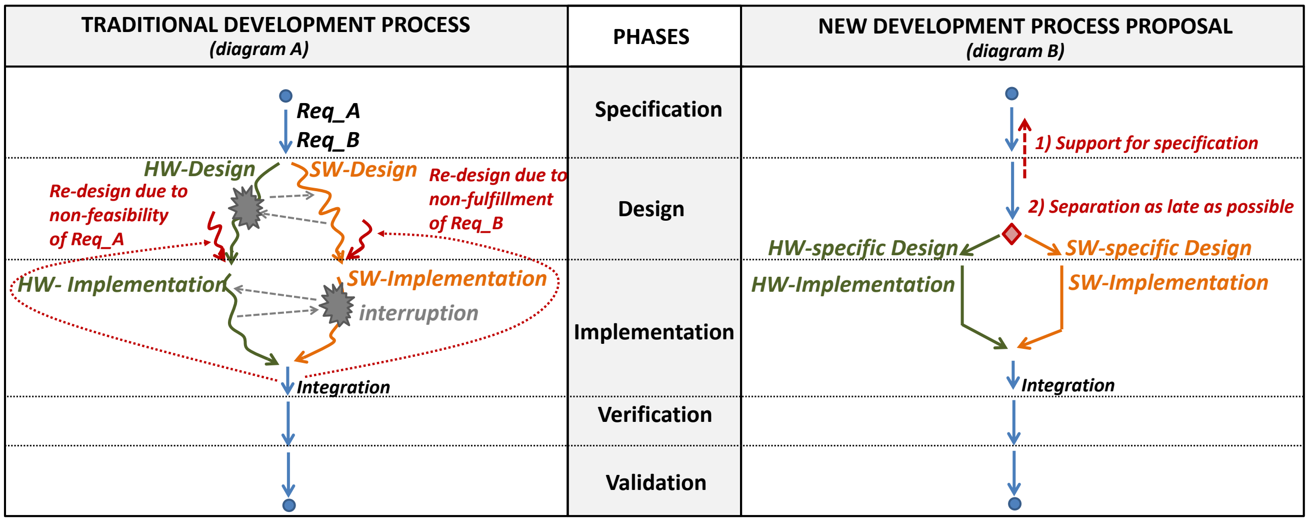


Figure 1. Traditional application development process (A) and the development process proposal

A. *Identification of key design criteria*

In our experience, a design methodology tailoring industrial automation applications has to able of meeting and efficiently trading-off a number of design boundary conditions deriving from: stakeholder concerns, technology and feasibility studies, functional and non-functional requirements, human factors (e.g., expertise, knowledge, etc.), constraints (e.g. legacy, reuse of existing platform, tool chains, manufacturing platforms and cost, etc.), technology advances in semiconductors and software, domain-specific features. As a consequence, it is crucial that all of these boundary-conditions are identified and carefully evaluated before starting the separation into hardware or software. By our approach they get identified and mapped into a set of *key design criteria*. Later they serve as inputs for supporting the subsequent stage of application decomposing and allowing the application to go through a decision process, as shown in Figure 2.

In order to define the set of key criteria, an accurate analysis of several design processes related to the application domain from different perspectives (performance, timing, overall quality, costs, etc.) will be performed. In details, it will be performed by the following steps:

1) Extrapolation of the mentioned boundary-conditions highlighting the relation with the design decisions in order to identify patterns like:
   a. the most high- impact decision choices,
   b. the most frequently adopted decision choices.
2) Classification of the above extracted design boundary conditions in relation to their hardware or software features. It is important to highlight what the cause-effect relations are in the entire design process.
3) Study to assess if and how well the design matches the required specifications, referred as design-specification matching for brevity. Interest will be also focus on biased decisions, to get a systematic interpretation of their impact on the overall design.
4) Identification of the criteria driving the strategic choices in the design.

In addition to establishing the motivation for decisions in the application development process, the key identified criteria will further provide guidelines for the refinement of the specifications. The analysis targets to gather a number of information related to the entire application life-cycle process (modelled by the extended V-Model in Figure 2) which in combination with the key identified criteria serve to complement and provide a systematic feedback to the specification phase.

B. *Application Decomposition*

Assuming that from a high abstraction level the application is modelled as a number of components, we propose an application decomposition process that extracts the elementary functionalities of the application and further refine the selection to the point in which the hardware or software implementation features of each component will be fully defined.

The proposed approach is based on both the analysis of the application specifications as well as the key design criteria identified in the previous stage. We propose a 2-step analysis:

1) Identification of the functionalities that directly matches the application specifications;
2) A decomposition of the identified functionalities in components strictly characterized by the key design criteria, and ready for the partitioning phase.

The above discussed decomposition strategy is supported by the diagram depicted in Figure 3. In practice, the two identified steps will be implemented using the following methodologies: (i) analysis of the application requirements and generation of specific functional components constituting different hypotheses of coarse-grained components suitable to be represented through well-known existing model-driven based tools like: The MathWorks

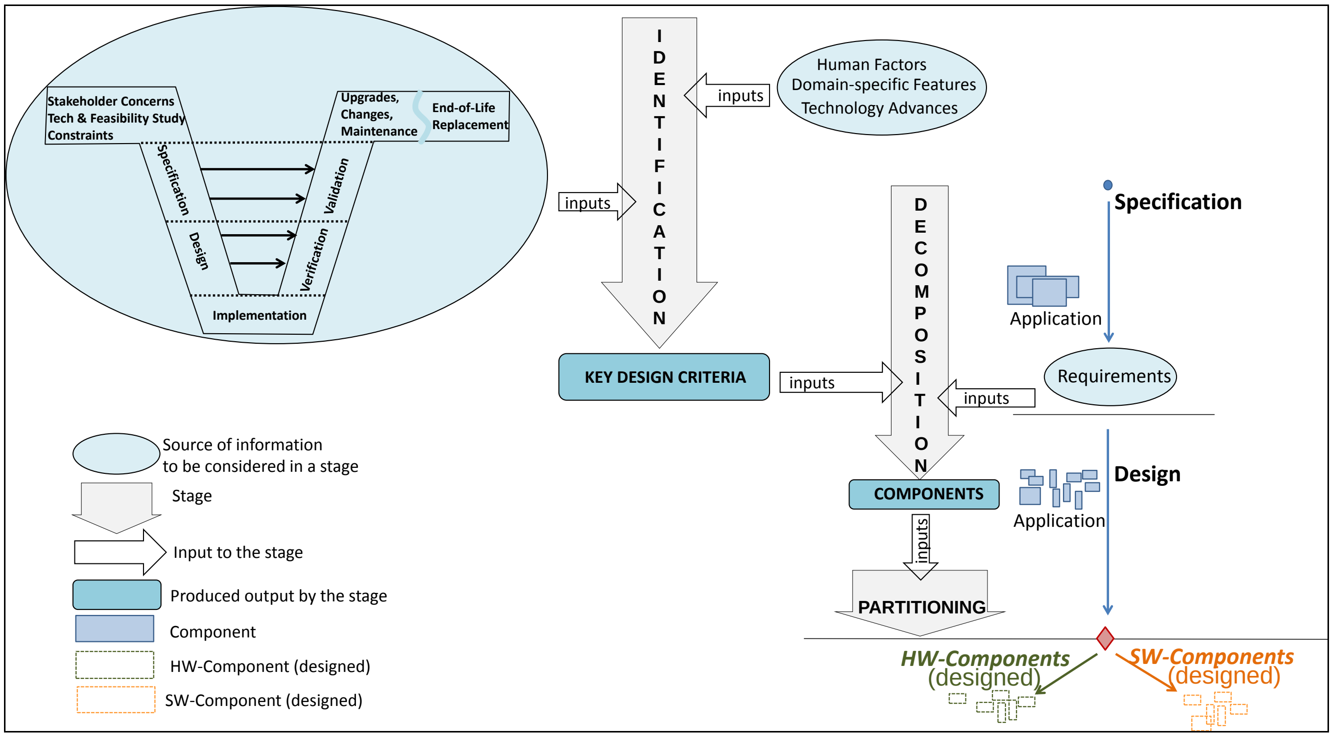


Figure 1. Relation between the proposed approach and V-Model and the Identification stage. Identification, Decomposition, Partitioning Flow

Simulink®, IBM® Rational® Rhapsody® (UML-based tool) etc., and (ii) the initial selection of coarse-grained components will be further decomposed through the key design criteria characterization to bias the generated components towards implementation issues, to create a well-posed problem as inputs to the subsequent decision process. Hence, all generated subcomponents will be strongly characterized by the key design criteria involved.

C. *Partitioning Decision Process*

Despite classical partitioning schemes that have been proposed in the past [7][8] which treat the problem as a nondeterministic polynomial problem to be optimized, we propose to face it as Multi-Criteria Decision Analysis (MCDA) problem. Unlike the approach proposed by [9] we do not intend to use MCDA for ranking the choice, but for targeting the design partitioning decisions in an efficient way. The choice of using such approach is driven by the variety and quantity of design decision criteria that require to be taken into account and their strong inter-dependencies. In addition to the above it is also motivated by the need of having a full traceability of the decision process. An intuitive and transparent procedure for generating the decisions is of crucial importance for studying the sensitivity of the design criteria in the overall decision process.

Additionally, in case of issues such as redesign or interruptions, caused by incompleteness or misleading of the specifications, it is possible to back-propagate the error and identify the major source of the unexpected behaviour in order to effective adapt the design strategy to further re-iterations. Further, the design feedback provided to the specification, enables of the hardware-specific and software-specific design separation as late as possible through the combined effects produced by the Identification as well as the Decomposition stage. Using the key design criteria for guiding the stepwise component discretization, implicitly allows the possibility of accumulating the required energy (i.e. in form of key components information) to start, after separation, design hardware and design flow where the dependencies are minimized. Furthermore, by performing the partitioning, after that the decomposition stage is completed, the set of components have been fully analysed and characterized, which consequently decreases the probability of assigning components to hardware or software based on wrong poses assumptions.

IV. Towards To An Industrial Application Case Study

The status of this case study is referred to the context of the two first phases of the extended V-Model (i.e., Specification and Design) as well as the Decomposition stage discussed in Section III.B.

In order to verify and validate the proposed methodology, we started working on the specification and design of a wind turbine application that is supposed to be deployed in an industrial prototype within the integration framework specified and developed by the Artemisia iFEST (industrial Framework for Embedded Systems Tools) project [10]. The main purpose of the application is to convert the rotational mechanical energy of the rotor blades caused by the wind into electrical energy to be redistributed via a power network.

A core component of the application is represented by the wind turbine controller, which has to be able of providing the dynamic regulation of the rotor blades at different wind profiles while maximizing the production of electrical energy. In parallel it has to be able of supervising the entire transformation process such as to guarantee the proper overall functioning of the wind turbine and minimizing any risk of damage to the physical wind turbine system.

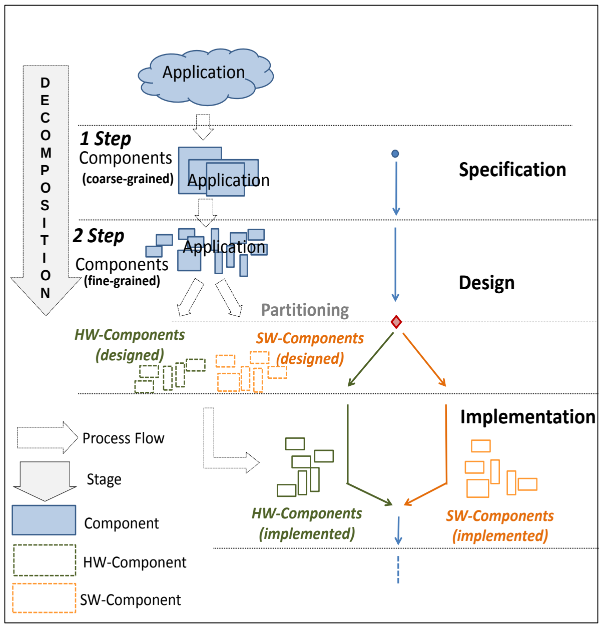


Figure 2. The Application Decomposition Process, 2-step analysis.

We intend to implement the design on several platforms, providing both for software (single and dual-core processors) and hardware (FPGA - Field-Programmable Gate Array) solutions. As tools used in the process we have chosen:

- HP ALM (Hewlett-Packard Application Lifecycle Management): for the specification and analysis phase.
- The MathWorks Simulink®: mostly for the design phase but also for Verification and Validation (simulation), and for the implementation (translation of design into C and VHDL).

According to the development process flow as well as the part of application decomposition process described above, we started with the specification phase. In order to go through the first step of the application decomposition process described in Section III.B, we took into account all of the info depicted in Figure 2, for instance the domain-specific features, the constraints, the stakeholder concerns, etc. Few examples follow:

- Domain-specific features: the application has to provide control functions allowing pitch regulation; the application has to be standard-compliant (i.e., IEC-61400); power network disturbances, etc.
- Constraints: the application has to be implemented into hardware and software; the implementation has to integrate legacy C-language code parts; the application has to allow the firmware to be field-upgradeable
- Requirements: time constraints for operations; reaction time at system failure, ambient temperature and relative humidity values; normal and extreme electrical conditions; safety procedures, etc.
- Stakeholder concerns: the project has to be able of delivering a high quality product with short time-to-market to pay-back the development cost and have a large margin profit.

After this first step, the identified key functionalities were mapped into components: (i) the pitch regulation, and (ii) the supervision.

In addition to this, we also identified the need for diagnostic and filtering functionalities. The components were modelled by using Simulink. The outcome of the mapping of the specification into the design is presented in Figure 4. It shows a two-level decomposition of the wind turbine application into components, which is achieved by the analysis of the application requirements. Level A models the Wind Turbine Plant and the Wind Turbine Controller. Level B shows a further decomposition of the Wind Turbine Controller component into four components: the Pitch Regulator, the Supervision, the Filtering and the Diagnostic.

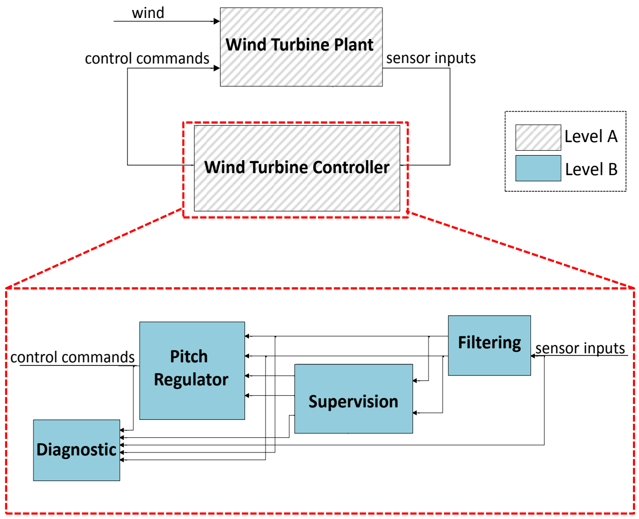


Figure 1. Wind Turbine Model (Plant and Controller). Decomposition of the Wind Turbine Controller (2-level).

What we have presented above is the first step of the application decomposition process. The next step will be to achieve a more detailed design decomposition of the application, as described in Section III.B. After that, the application will be applied for a multi-criteria decision process in order to decide about which components will be implemented in hardware and in software. Subsequently the application will be deployed into several platforms in order to evaluate the proposed new approach.

V. CONCLUSION

The paper presented a proposal of a new systematic design methodology that is able to improve the overall process from the design perspective as well as from the application lifecycle perspective. It consists of three main stages: Identification, Decomposition and Partitioning, that collectively drive through the definition of the main methodology characteristics such as the support towards the specification phase and the enabling of software-independent hardware design and hardware-independent software design separation.

The next step of our research work is defining which requirements the MCDA approach has to fulfil in order to support the aforementioned partitioning process. Subsequently, we will analyse if any already available MCDA method (or a combination of more MCDA methods) is able of meeting the identified requirements and can be applied for the application partitioning. After that, we will focus on the identification and formalization of the key design criteria to use as the set of inputs (i) for guiding the fine-grained application decomposition and (ii) for supporting the partitioning process into designed hardware and software components as described by Figure 2. As final step, the proposed methodology will be evaluated on the above presented industrial application case study.

ACKNOWLEDGMENT

This research is supported by the Knowledge Foundation (KKS) through ITS-EASY, an industrial research school in Embedded Software and Systems, affiliated with the School of Innovation, Design and Engineering (IDT) at Mälardalen University (MDH), Sweden.

REFERENCES

[1] P. Koopman, “Embedded System Design Issues (the rest of the story)”, Proceedings of IEEE International Conference on VLSI in Computers and Processors, Oct. 1996.

[2] A.S. Berger, "Embedded Systems Design: An Introduction to Processes, Tools and Techniques", CMP Books; 1 edition, Dec. 15, 2001.

[3] A. G.Kleppe, Jos Warmer, Wim Bast, “MDA Explained: The Model Driven Architecture: Practice and Promise”, Addison-Wesley Professional, 1 edition , May 1 2003.

[4] H.Van Vliet,"Software Engineering: Principles and Practice", Wiley, 3 edition, Jun 27, 2008.

[5] G.De Micheli, R.Gupta, “Hardware/Software Co-Design,” Proc. of the IEEE, vol. 85, No.3, 1997, pp.349-365.

[6] C.Coelho, C.Yang,V. Mooney, G.De Micheli, “Redesigning hardware–software systems,” in Proc. 3rd Int. Workshop on H/S Codesign, Grenoble, France, Sep. 1994.

[7] Y.Fan, T.Lee, "Grey Relational Hardware-Software Partitioning for Embedded Multiprocessor FPGA Systems", AISS: Advances in Information Sciences and Service Sciences,vol. 3, No. 3, 2011, pp. 32 - 39.

[8] M.L.Vallejo, J.C.Lopez, ”On the hardware-software partitioning problem: System Modeling and partitioning techniques”, ACM Transactions on Design Automation of Electronic Systems (TODAES) vol. 8, Issue 3, July 2003, pp. 269 – 297.

[9] P.Garg, A. Gupta, J.W.Rozenblit, "Performance analysis of embedded systems in the virtual component co-design environment", Proceeding of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, May 2004, pp. 61-68.

[10] iFEST. iFEST - industrial Framework for Embedded Systems Tools. ARTEMIS JU project #100203. Retrieved September 26, 2012, from http://www.artemis-ifest.eu/

Automatic Synthesis of Hardware-Specific Code in Component-Based Embedded Systems

Luka Lednicki, Ivica Crnković
Mälardalen Real-Time Research Centre
Mälardalen University
Västerås, Sweden
Email: {luka.lednicki,ivica.crnkovic}@mdh.se

Mario Žagar
Faculty of Electrical Engineering and Computing
University of Zagreb
Zagreb, Croatia
Email: mario.zagar@fer.hr

***Abstract*—In recent years, there has been a clear trend in research and practice to bring benefits of component based development into the embedded systems domain. However, one often neglected aspect in component models is support for integration of hardware devices like sensors and actuators. In most component models, communication with such devices is either left out completely or considered as an integral part of the software component code. In the latter case, the software components are highly device-specific, and can hardly be reused on different platform configurations. This paper introduces an approach for automatic synthesis of device-specific code in component models for embedded systems. We divide a system in reusable elements: device-specific code, platform-specific code and device-dependant software component code. Based on a software and hardware model of the system, we then automatically generate glue-code that creates connections between these reusable elements. The result of our synthesis is a system-specific deployable code. The approach is illustrated by a demonstrator and an implementation example using the ProCom component model.**

***Keywords*-Component-based; code synthesis; sensors; actuators; embedded systems.**

I. Introduction

Component-based development (CBD) is one of the approaches suggested to alleviate the constant rise in the complexity of embedded systems (ES) [1], [2]. Component-based systems are developed by composing preexisting components – reusable units that contain not only code, but also various models, and conform in syntax and semantics to a component model. The functionality of a system is defined by a system model, and implemented by the process of code synthesis i.e. automatic generation of glue-code that connects the reusable code defined by the components. Because of restricted processing and memory resources in (many) ES, most often an efficient code synthesis is very important, and solutions with no or very small middleware are preferable over large general-purpose frameworks.

One aspect that is crucial for use of CBD in ES is communication with hardware devices such as sensors and actuators. However, inclusion of hardware device-specific elements in software components decreases the components' reusability [3]; if a component includes device-specific code, or code that is specific to a platform, the component cannot be efficiently reused in case of changes in the underlying hardware. Therefore, by making software functionality independent from a specific hardware configuration, and by providing means to automatically generate the hardware-specific code, we can make code reuse in embedded systems more efficient.

In this paper, we present a novel way to provide code synthesis for component software in the ES domain, which allows a transparent use of hardware devices in software models. We do this by first separating software component code, device-specific code and platform-specific code, while strictly defining their content and interfaces they can use to communicate with each other. By this we get system-independent, reusable units of code. We then use a model that describes software components, hardware devices and the deployment platform, to automatically generate glue-code that connects the mentioned code parts into a deployable system. As a result, we are able to synthesize code using system models that completely separate high-level software functionality from hardware specifics. Our approach is based on the framework for handling interaction of software components with hardware devices that we proposed in [4], and implemented in the context of the ProCom component model [5]. The approach described in this paper is an extension of principles shown in [6].

The rest of the paper is organized as follows: Section II gives a brief overview of some of the approaches to treat hardware-specific code in component models for embedded systems. In Section III we present the framework we use for specification of hardware devices. Section IV describes our approach to hardware-specific code synthesis and gives an example of the synthesis implementation. A short description of how our work is implemented in the context of the ProCom component model is given in Section V. Section VI shows how our approach is used on a real hardware device example. Finally, Section VII concludes the paper.

II. Related Work: Hardware-Specific Code in Component-Based Embedded Systems

Synthesis of hardware-specific code has been explored in model-driven engineering [7], and in specific languages and models such as AADL [8] or MARTE [9]. However, automatic generation of glue code for connections to device-specific code has not been established in component models for ES.

While looking at component models used in research and practice, we can see a difference in the level of support for hardware devices and how they treat hardware-specific code [10]. In most component models used in research, hardware-specific code is externalized – not present in software components and put outside the scope of the component model. This is not surprising when we take into account that most of these models are used just for research purposes and do not have to provide working, deployable systems as a result. On the other hand, component models targeting industry must provide support for hardware devices to be able to generate functioning systems. However, most of these component models only provide implicit support for hardware devices, which means that the hardware-specific code is hard-coded in the software component code.

One example of externalized devices is SaveCCM [11], where the component code is not allowed to communicate with hardware. Instead, this communication is supposed to take place outside the component model and by some means the values provided as inputs or outputs of the component framework. Although it enables the reuse of all components, this approach does not provide support for the whole CBD life cycle of systems and limits its use in practice.

Rubus [12] (developed by Articus Systems and used for example by Volvo Construction Equipment) is an example of a component model used in practice, where all interactions with hardware devices are included (i.e. hard-coded) in software components. This approach allows for full code synthesis, but severely limits the ability to reuse components with different devices.

The AUTOSAR [13] framework, an initiative of different industrial partners, defines a component model that provides a standardized interface for software components in vehicular embedded systems. In AUTOSAR, interaction with hardware devices is implemented by specialized sensor/actuator components. These components are still dependent on specific hardware devices and are not fully appropriate for reuse. Also, AUTOSAR relies on deployment of components to an already existing hardware abstraction layer, and not on full code synthesis.

ProCom [5] is a component model aimed for the embedded systems domain. It relies on code synthesis for generating efficient code, both regarding memory and processor usage. Some of the principles used in code synthesis for ProCom are described in [14], [15]. A basic support for modeling of sensors and actuators in the component model is given by the Framework for Supporting Hardware Devices [4] (described in details in Section III). However, current synthesis for ProCom does not take this framework into account.

As opposed to all methods of treating hardware devices shown above, we propose a method that will enable explicit support for these devices and enable automatic synthesis of system-specific code. Such a synthesis method is currently not available in component models for ES. We aim to increase the reusability of components in the ES domain and provide more flexibility in system development. As a demonstrator of our approach we have implemented our hardware-specific code synthesis in the context of the ProCom component model.

III. Overview of the Hardware Device Specification Framework

To enable an effective code synthesis and efficient code reuse for hardware devices, we have developed a Hardware Device Specification Framework. The purpose of the framework is to allow explicit inclusion of hardware devices, such as sensors and actuators, into component models for embedded systems. In the framework, hardware devices are presented as software components, while leaving the components free of device- and platform-specific information. It then enables specification of device- and platform-specific information, and provides a way to associate it with software components. The part of the framework metamodel relevant for this work is shown in Figure 1. The framework includes three layers: *software layer*, *hardware layer* and *mapping layer*. The software layer includes device components, which represent hardware devices as software components, excluding any device- or platform-specific information. The lowest, hardware layer, contains information about hardware devices, the platform and how the two are connected. The mapping layer enables creating connections between hardware specific and hardware-independent layers. In the subsections below we provide a detailed description of each layer.

A. Software layer

In the software layer, the interaction of software components with hardware devices is represented by *device components* and their instances (context-specific representatives). Device components provide the same component interface and abide the same execution semantics as all other software components. Both types of components are treated the same during design – they can be used equivalently. But opposed to the ”pure” software components, which implement their functionality by code, device components do not implement any functionality, but serve as connection points to which we can associate device-specific functionality. Their functionality is defined once the component is mapped to a hardware device (as described in Section III-C). However, they are in no way dependent on a specific hardware device or how the device is connected to the platform.

B. Hardware layer

The actual interaction with hardware devices is specified in the hardware layer. It is encapsulated in entities called *IOs* and *hardware devices*, and also defined by *IO allocation*.

Input and output elements (e.g. pins or ports) of the platform are represented by *IOs*. An IO provides all information needed to communicate through the input or output it represents. IOs are reusable across different systems.

Each IO references an *IO type*. IO types are abstract entities which define functionality that an IO of that type must provide, along with the data types and structures used.

A *hardware device* model element represents a physical sensor or actuator and includes all information that is specific to that sensor or actuator. However, it does not cover the information describing how the device is connected to the underlying platform. Each hardware device entity refers to one software component from the software layer, indicating which functionality it can provide. It also defines the type of IO it requires from the platform. As with IOs, we can also reuse hardware device entities. Similar to device components in the software layer, hardware devices also have their context-specific instances.

Hardware devices contain one or more *required IO* element. Required IOs represent platform IOs that the actual physical sensor or actuator needs for communication. The types of these IOs are specified by referring to IO type elements.

To create an actual system, we need to create *IO allocations*, i.e. to describe an IO allocation to reflect the current system configuration.

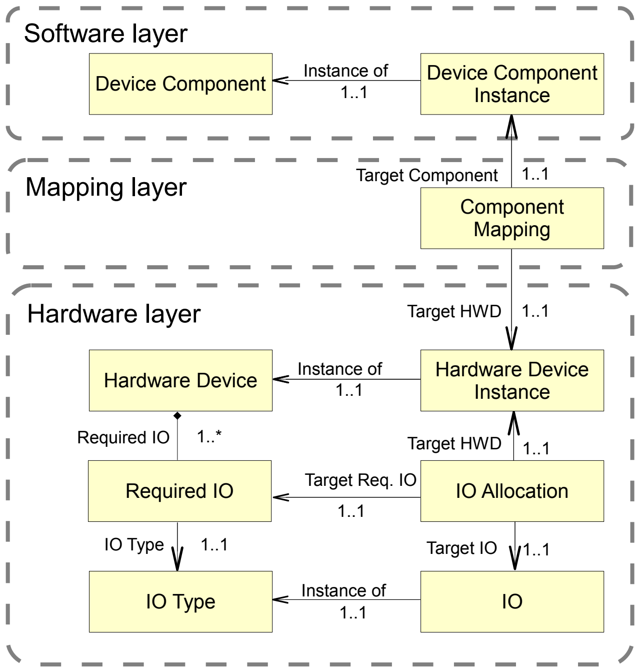


Figure 1. Metamodel of the Hardware Devices Specification Framework.

C. Mapping layer

The mapping layer allows us to create connections (*mappings*) between elements of the software and hardware layers. When we map a device component from the software layer to a hardware device from the hardware layer, we denote that the hardware device will be used as the realization for the device. Once such a mapping is defined we can utilize any information defined for the hardware device entity (and IO, if IO allocation is present) in the software layer.

IV. Code Synthesis

The code synthesis is based on two principles – (a) separation of reusable code from the device- and platform-specific code and (b) automatic generation of device- and platform-specific code based on a system model that includes software and hardware components. We achieve this by first defining a way to specify system-independent and reusable code elements for device- and platform-specific functionality. Besides just functionality, reusable code elements also define interfaces for communication. Using a system model we then generate code that utilizes these interfaces to combine the software component code with device- and platform specific functionality resulting in a system-specific deployable solution. An overview of the synthesis process is given in Figure 2.

Our approach consists of two categories of code: (a) *input code elements* which will be used as input to the synthesis, and (b) *generated code* that connects the input elements. An overview of all the code elements used in synthesis and the relations between them is given in Figure 3. These code elements are described in detail below.

A. Synthesis Input Code Definition

In the implementation of the solution we used the C programming language, as C is the language still mostly used to develop embedded systems. However, the principles used in this solution are not limited to C. They can easily be implemented in other programming languages.

The input code is separated into elements that are as independent as possible from each other, making them fit for reuse. These four elements are:

- *device component code* – platform, device and system independent code,
- *IO type code* – code that describes capabilities for different IOs,
- *IO code* – platform-specific code that implements IO functionality,

- *hardware device code* – device specific code that implements device functionality.

Besides defining parts of functionality, these elements also define structures and function signatures that will be used as communication interfaces. Coupling between them is loose: all function calls are performed using function pointers, which are assigned during the mapping and allocation phase. All these elements are system independent and can be reused in different systems or platform configurations.

1) Device component code: As we want to place device- and platform-specific code outside software components, device component code does not provide any functionality. Instead, it only provides a way to make calls to device-specific functions once the system model is defined. Device component code consists of a structure that includes: (a) definitions of zero or more variables that will be used to communicate data to and from a hardware device and (b) pointers that will be used to map and allocate appropriate device- and platform-specific functionality. Allocation data (described in Section IV-A2 and Section IV-A4) will be assigned to a void pointer. For mapping we use a function pointer to an entry function that will implement device functionality. This function receives the device component structure, and with it means to assign or read data and use appropriate platform IO functions. We describe how these pointers are assigned in Section IV-B.

2) IO type code: In the IO type code we define an *IO interface structure* that will be used to communicate through an IO. This interface structure contains pointers (signatures) for one or more functions that will be used for the communication. The number of functions can differ for different kinds of IOs. Some of the functions can also be used for configuration of the communication channel, rather than for the communication itself. The code can also contain definitions of data structures that will be used as arguments to functions in case the arguments are not basic C types. An instance of the interface structure will be used to allocate Hardware Devices to IOs (described in Section IV-B1).

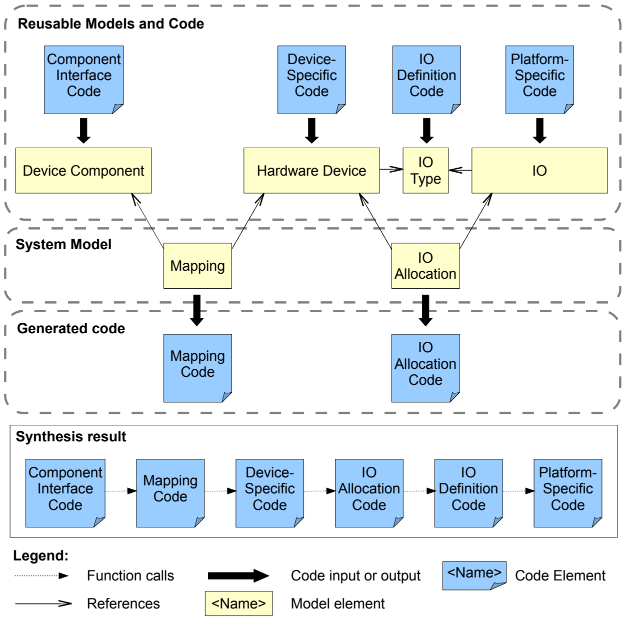


Figure 2. Overview of the synthesis process and results.

3) IO code: Code defined for IOs provides the platform specific implementation for functions defined in the interface structure of the IO type code. The IO code of each physical platform can be comprised of many C source files, all of which implement communication just for one specific IO. Separation of IO code into different files allows us to use the minimum amount of platform-specific code once we synthesize the code for the whole system. For an IO source file to be valid it must implement all the functions defined in the interface structure of the IO type. Also, each IO code definition must define an IO interface assignment function that receives an IO interface structure and assigns function implementations to the function pointers of the structure. How these functions are used for the actual allocation is described in Section IV-B1).

4) Hardware device code: The main purpose of hardware device code is to provide an implementation of communication for a specific sensor or actuator. This includes the protocol used to communicate with the device, possible adaptation of data and calls to IO functions. To provide device-specific implementation for a device component, the hardware device source code must implement a function which has a signature matching the signature of the component entry function. Code for each hardware device also includes an IO allocation structure that contains instances of IO interface structures for each IO type that the device requires. An instance of this structure will be referenced by a pointer in the device component structure, and can be used for making platform-specific IO function calls.

B. Generated code

Using a system model, which is based on the previously described framework, we are able to generate code that implements the functionality of the system. The code we generate creates connections between the various input code elements we defined, using their interfaces.

Code generation is divided into two phases: generation of IO allocation code and generation of device mapping code. Listing 5 shows mapping and allocation code generated for our example.

1) IO allocation code: The IO allocation code provides connections between instances of device components and platform IOs. It enables devices to make function calls to platform IO functions, abstracting away platform specifics.

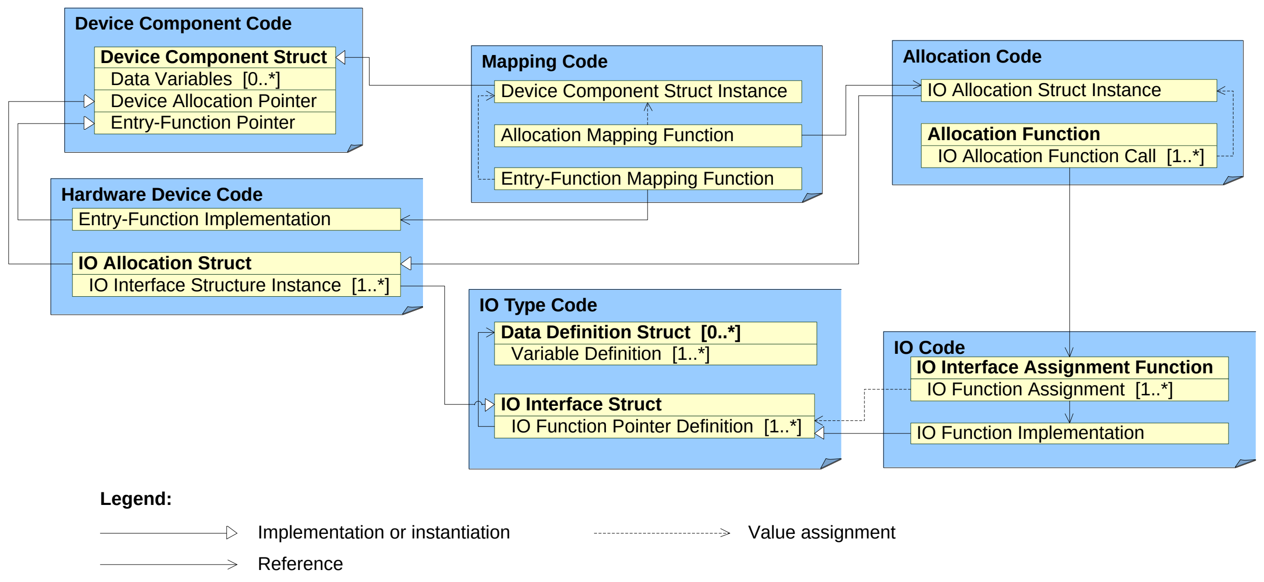

Device Component Code
Device Component Struct
Data Variables [0..*]
Device Allocation Pointer
Entry-Function Pointer
Mapping Code
Device Component Struct Instance
Allocation Mapping Function
Entry-Function Mapping Function
Allocation Code
IO Allocation Struct Instance
Allocation Function
IO Allocation Function Call [1..*]
Hardware Device Code
Entry-Function Implementation
IO Allocation Struct
IO Interface Structure Instance [1..*]
IO Type Code
Data Definition Struct [0..*]
Variable Definition [1..*]
IO Interface Struct
IO Function Pointer Definition [1..*]
IO Code
IO Interface Assignment Function
IO Function Assignment [1..*]
IO Function Implementation
Legend:
Implementation or instantiation
Value assignment
Reference

Figure 3. Code elements in our synthesis approach and relations between them. For the elements that can occur multiple times, multiplicity is shown in square brackets.

To generate IO allocation code we use hardware device elements, IO elements and IO allocation elements from the system model, and their respective input code elements.

First, we traverse the model and for each device referenced by IO allocation elements we define an instance of the *IO allocation structure* defined by the input code of the hardware device, giving each instance of the structure a unique name. Even if a device requires more than one IO for its functionality, only one such structure is created, as the structure contains variables for all required IOs.

After that we are able to generate an *allocation function* that will assign appropriate IO functions to function pointers defined in the IO allocation structures. In this function, for every IO allocation element we create a function call to the *IO interface assignment function* defined by the input code of the referenced IO element. As an argument, we provide the IO interface structure instance defined in the previous step of IO allocation code generation. The function call then assigns the correct platform function calls defined by the referenced IO to the IO allocation structure generated for the referenced hardware device. One IO interface assignment function call is created for each required IO.

2) Mapping code: Device- and platform-specific functionality is provided to software components by generation of mapping code. For this we use the device component elements, hardware device elements and device component mapping elements of the system model.

As a first phase of mapping code generation we create instances of device component data structures which will hold mapping data. The instance name is based on the unique ID of the component instance.

In the next phase we generate *allocation mapping code*. This code binds IO allocations generated during IO allocation code generation to device component instances. For each device component mapping we create a line of code that assigns an IO allocation structure which represents the instance of device component referenced by mapping (generated as part of IO allocation code) to the IO allocation pointer of device component data structure.

The final part of the mapping code is the *entry function mapping*. This code connects device software components with device entry functions which implement device-specific functionality. We achieve this by generating code that assigns an entry function of the device referenced by the mapping to the entry function pointer of the data structure of the device component the mapping is targeting.

V. Implementation

For the purpose of evaluation we have implemented our approach in the context of the ProCom component model. The implementation was done using Java. Our automated synthesis is included in PRIDE [16] – an Eclipse based IDE for ProCom. It leverages Eclipse Modeling Framework for model traversal and Java for code generation.

Standard ProCom components are implemented by a single C entry function that executes when the component is triggered to perform its functionality. This function receives a data structure that contains instance-specific component data. We adapted these elements to align with our device component code.

As a way to make function calls to the device-specific entry function, we have defined a standard entry function that all device components must implement. This entry function has only one line of code, which diverts the function call to the hardware specific entry function assigned to the entry function pointer of the component instance data structure.

VI. EXAMPLE

We will demonstrate our approach on a real example using the GDM2004 LCD display module. In code listings shown for the example we have removed some parts of the code (e.g. ”include” instructions and use of unique IDs in naming) that are not relevant to show the principles of our solution. The example consists of a device component instance (*DisplayComponent display*) which maps to *GDM2004* instance of the *GDM2004Device* hardware device. The *GDM2004Device* requires three IOs of type *OneBitIO* (named *registerSelect*, *RW* and *enable*), and one IO of type *IO8BitPort* (named *data*). These requirements are allocated to platforms IOs *PA0*, *PA1*, *PA2* and *PE*. A model of the example is shown in Figure 4.

A. Device component code

In our example, code that defines the *DisplayComponent* device component can be seen in Listing 1. The device component structure begins at line 1. Lines 2 to 3 contain data variables that will be used for communication. The pointer that will reference allocation data is defined on line 6, and line 7 defines the pointer that will reference the device entry function.

```
1  typedef struct DisplayComponent {
2    int row;
3    int column;
4    char* text;
5
6    void * ioAllocation; // Pointer to IO
         allocation
7    void (*entryFunction) (struct
         DisplayComponent*); // Pointer to
         device specific entry function
8  } DisplayComponent;
```

Listing 1. Display component code.

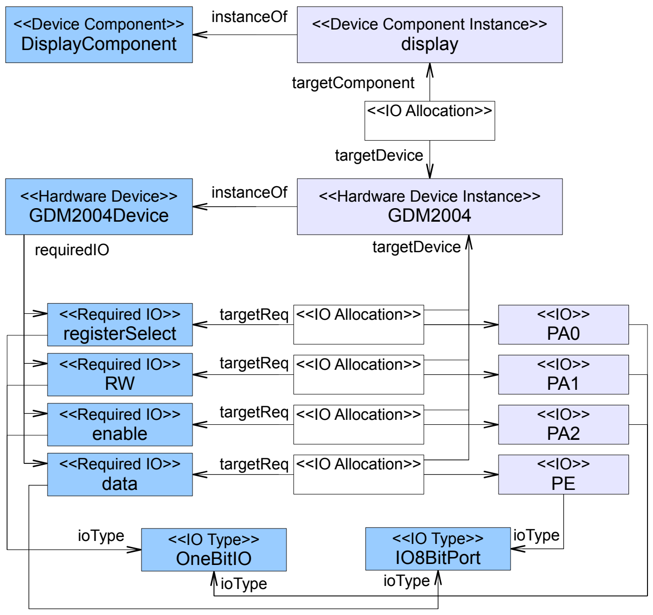


Figure 4. Model of the GDM2004 example.

B. IO type code

The IO type code for our example is shown in Listing 2. On line 2 we can see a definition of the data structure that will be used to communicate configuration data for the port. The IO interface structure that defines function pointers for function used for communication is located at lines 8 to 12.

```
1  // Data definition
2  typedef struct  {
3    int isOutput;
4    int isOpenDrain;
5  } OneBitIO_configData;
6
7  // IO interface structure
8  typedef struct  {
9    void (*writeData) (int);
10   void (*readData) (int*);
11   void (*configure) (OneBitIO_configData*);
12 } OneBitIO;
```

Listing 2. Example of IO type code.

C. IO code

Listing 3 shows the IO code for port *PA0* of our example. From lines 2 to 12 we can see functions that implement port communication, which adhere to function signatures defined in the IO type interface structure. The IO interface assignment function, which assigns functions defined in lines 2 to 12 to an instance of IO type interface structure, is on lines 16 to 20.

```
1  // START functions implementing IO
2  void PA0_writeData(int value) {
3    WrPortI(PADR, &PADRShadow, value, 0);
4  }
5
6  void PA0_readData(int* value) {
7    RdPortI(PADR, &PADRShadow, value, 0);
8  }
9
10 void PA0_configure(OneBitIO_configData*
       data) {
11   WrPortI(PADDR, &PADDRShadow, data->
         isOutput, 0);
12 }
13 // END functions implementing IO
14
15 // Assigning functions for allocation
16 void allocate_PA0(OneBitIO* allocation) {
    allocation->writeData = &PA0_writeData;
    allocation->readData = &PA0_readData;
    allocation->configure = &PA0_configure;
  }
```

Listing 3. Example of IO code.

1) Hardware device code: Hardware device code that implements functionality for the *GDM2004Device* defined in our example is given in Listing 4. The IO allocation structure for *GDM2004Device* begins on line 2. It contains one IO interface structure for each required IO, on lines 3 to 6. The device entry function definition can be seen on lines 10 to 18. How the allocated IO interface structures are used can be seen in the GDM2004PrintChar function which starts on line 26.

```
1   // Structure for IO allocation
2   typedef struct GDM2004_allocation{
3     OneBitIO registerSelect;
4     OneBitIO rw;
5     OneBitIO enable;
6     IO8bitPort data;
7   } GDM2004_allocation;
8
9   // Implementation of the entry function
10  void entry_GDM2004(DisplayComponent*
        instanceData) {
11    int i;
12    GDM2004_allocation* alloc = (
          GDM2004_allocation*) instanceData->
          ioAllocation;
13
14    GDM2004SetPosition(instanceData->column,
          instanceData->row, alloc);
15    for (i=0; i < strlen(instanceData->text);
          ++i) {
16      GDM2004PrintChar(instanceData->text[i],
            alloc);
17    }
18  }
19
20  void GDM2004SetPosition(int column, int row
        , GDM2004_allocation* alloc) {
21    /*
22    Implementation skipped
23    */
24  }
25
26  void GDM2004PrintChar(char c,
        GDM2004_allocation* alloc) {
27    alloc->registerSelect.writeData(1);
28    alloc->rw.writeData(0);
29    alloc->enable.writeData(1);
30    Delay_60usec();
31    alloc->data.writeData(c);
32    alloc->enable.writeData(0);
33    Delay_60usec();
34  }
```

Listing 4. GDM2004 HardwareDevice code.

D. IO allocation and mapping code

Listing 5 shows the IO allocation structure created for the *GDM2004* instance of *GDM2004Device* on line 2. The allocation function shown on lines 3 to 8 contains function calls to IO allocation functions defined by IO for *registerSelect*, *RW*, *enable* and *data* required IOs of the *GDM2004Device*.

Line 12 of Listing 5 contains an instance of the device component data structure for *DisplayComponent*. On line 14 we can find the allocation mapping function, in which the IO allocation structure instance from line 2 is assigned to the IO allocation pointer of the *DisplayComponent* data structure. The entry function mapping is shown on lines 18 to 20, where we assign the entry function defined by *GDM2004Device* to the entry function pointer of the *DisplayComponent* data structure.

```
1   // ***** START ALLOCATION
2   GDM2004_allocation display;
3   void doIOAllocation() {
4     allocate_PA0(&(display.registerSelect));
5     allocate_PA1(&(display.rw));
6     allocate_PA2(&(display.enable));
7     allocate_PE(&(display.data));
8   }
9   // ***** END ALLOCATION
10
11  // ***** START MAPPING
12  DisplayComponent displayComponent;
13
14  void doAllocationMapping() {
15    displayComponent.ioAllocation = &display;
16  }
17
18  void doEntryMapping() {
19    displayComponent.entryFunction = &
          entry_GDM2004;
20  }
21  // ***** END MAPPING
```

Listing 5. Example of generated mapping and allocation code.

VII. Conclusion

In this paper, we have presented an approach to code synthesis for embedded systems which utilizes a system model to generate hardware-specific code. The system model we use describes software components, hardware devices (i.e. sensors and actuators) and inputs and outputs of the platform, and connections between all these elements. We provide strict definitions of how to specify interface and implementation code for the model elements in order to get reusable units of code that we use as inputs for our synthesis. During the synthesis process we generate glue-code that connects these reusable code elements to provide the functionality defined by the system model.

Compared to having hardware-specific code hard-coded inside software components, in our approach software components are not directly dependant on hardware devices or how the devices are connected to the platform. The result of this is more efficient code reuse.

The approach presented in this paper provides a level of abstraction over hardware devices and the platform. It also separates the concerns of hardware-independent software development, hardware-specific software development and system integration. On a higher level, software systems can be developed without knowing the specifics of the underlying platform. On the other hand, low level hardware-specific functionality can be implemented independently of the rest of the system. Both levels can in the end be used as black-boxes and integrated into systems with no knowledge of their internals.

ACKNOWLEDGMENT

This work was supported by the Swedish Foundation for Strategic Research project RALF3 and the Swedish Research Council project CONTESSE (2010-4276). We would like to thank Jan Carlson for providing valuable comments on the paper.

REFERENCES

[1] I. Crnkovic and M. Larsson, *Building Reliable Component-Based Software Systems*. Norwood, MA, USA: Artech House, Inc., 2002.

[2] C. Atkinson, C. Bunse, C. Peper, and H.-G. Gross, "Component-based software development for embedded systems an introduction," in *Component-Based Software Development for Embedded Systems*, ser. Lecture Notes in Computer Science, C. Atkinson, C. Bunse, H.-G. Gross, and C. Peper, Eds. Springer Berlin / Heidelberg, vol. 3778, pp. 1–7.

[3] L. Lednicki, "Support for hardware devices in component models for embedded systems," in *International Doctoral Symposium on Software Engineering and Advanced Applications*, August 2011.

[4] L. Lednicki, J. Feljan, J. Carlson, and M. Žagar, "Adding support for hardware devices to component models for embedded systems," in *ICSEA 2011, The Sixth International Conference on Software Engineering Advances*, 2011, pp. 149–154.

[5] S. Sentilles, A. Vulgarakis, T. Bureš, J. Carlson, and I. Crnković, "A component model for control-intensive distributed embedded systems," in *Component-Based Software Engineering*, ser. Lecture Notes in Computer Science, M. Chaudron, C. Szyperski, and R. Reussner, Eds. Springer Berlin / Heidelberg, vol. 5282, pp. 310–317.

[6] L. Lednicki, I. Crnković, and M. Žagar, "Towards automatic synthesis of hardware-specific code in component-based embedded systems," in *SEAA 2012,38th Euromicro Conference on Software Engineering and Advanced Applications*, September 2012.

[7] S. Burmester, H. Giese, and W. Schaefer, "Model-driven architecture for hard real-time systems - from platform independent models to code," in *Model Driven Architecture - Foundations and Applications*, ser. Lecture Notes in Computer Science, A. Hartman and D. Kreische, Eds. Springer Berlin, Heidelberg, vol. 3748, pp. 25–40.

[8] J. Hugues, B. Zalila, L. Pautet, and F. Kordon, "From the prototype to the final embedded system using the ocarina aadl tool suite," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 4, pp. 42:1–42:25, Aug. 2008.

[9] A. Rodrigues, G. Frédéric, and J. Dekeyser, "An mde approach for automatic code generation from marte to opencl," *INRIA Lille-RR-7525 [Online]. Available: http://hal. inria. fr/inria-00563411/PDF/RR-7525. pdf/, Tech. Rep.*

[10] J. Feljan, L. Lednicki, J. Maras, A. Petričić, and I. Crnković, "Classification and survey of component models," Mälardalen University, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-242/2009-1-SE, December 2009.

[11] "The SAVE approach to component-based development of vehicular systems," *Journal of Systems and Software*, vol. 80, no. 5, pp. 655 – 667, 2007.

[12] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindberg, J. Lundback, and K.-L. Lundback, "The Rubus component model for resource constrained real-time systems," in *Industrial Embedded Systems, 2008. SIES 2008. International Symposium on*, june 2008, pp. 177 –183.

[13] H. Heinecke, W. Damm, B. Josko, A. Metzner, H. Kopetz, A. Sangiovanni-Vincentelli, and M. Di Natale, "Software components for reliable automotive systems," in *Proceedings of the conference on Design, automation and test in Europe*, ser. DATE '08, 2008, pp. 549–554.

[14] E. Borde and J. Carlson, "Automatic Synthesis and Adaption of Gray-Box Components for Embedded Systems – Reuse vs. Optimization," in *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*, july 2011, pp. 224–229.

[15] ——, "Towards verified synthesis of ProCom, a component model for real-time embedded systems," in *Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering*, ser. CBSE '11, 2011, pp. 129–138.

[16] E. Borde, J. Carlson, J. Feljan, L. Lednicki, T. Lévêque, J. Maras, A. Petricic, and S. Sentilles, "Pride - an environment for component-based development of distributed real-time embedded systems," in *Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, ser. WICSA '11, 2011, pp. 351–354.

Modeling Crosscutting Concerns with Roles

Fernando Sérgio Barbosa
Higher School of Technology
Polytechnic Institute of Castelo Branco
Castelo Branco, Portugal
fsergio@ipcb.pt

Ademar Aguiar
Department of Informatics Engineering
Faculty of Engineering of University of Porto
Porto, Portugal
ademar.aguiar@fe.up.pt

***Abstract*—Modularization allows the development of independent modules and their reuse. However a single decomposition strategy cannot neatly capture all the systems concerns. Thus some concerns are spread over several modules – the crosscutting concerns. To cope with this we need to have other class composition techniques than those available in traditional Object Oriented programming languages. One of such compositions is roles. If roles are used to compose classes and if a role models a crosscutting concern, then the concern is limited to the role and not spread over several classes. To validate this approach we conducted a case study. In the case study crosscutting concerns were identified in a system using a clone detection tool and roles were developed to model those crosscutting concerns. Results show that this approach reduces significantly the spreading of crosscutting concerns code.**

Keywords-Roles; Crosscuting concerns; Code clones.

I. INTRODUCTION

Modularization [1] is one of the most important concepts in software development. Decomposing a system into modules allows the independent development of each module. This shortens development time and allows the modification of a module without changing other modules.

But a single decomposition strategy cannot capture all possible views of a module.[2]. We could use multiple inheritance, but it has so many practical problems that it has been left out of recent programming languages. Even if we could use multiple inheritance, there are always concerns that cannot be adequately decomposed using a single decomposition strategy [3], and end up scattered among the various modules. These are called the crosscutting concerns.

A consequence of crosscutting concerns is replicated code. When classes must implement a crosscutting concern developers tend to copy-paste the code that deals with it [4]. Thus the presence of code clones in a system is an indicator that there are crosscutting concerns in that system [5].

An obvious problem of code clones is the increased system size. But code clone also impairs system's maintenance and evolution [6]. A particular problem is the inconsistence in updating, where a bug in a code block is propagated to all its clones, and is fixed in most but not all occurrences. Code clones also have negative effects in program comprehensibility, evolution, cost and may be an indicator of design flaws [7].

To prevent such consequences we need to use other decomposition techniques. Several proposals are available, like inheritance, mixins [8], traits [9], features [10], aspects [11] and roles [2][12]. We believe that if we explore the way roles can be used to compose classes we will find that roles are capable of modeling crosscutting concerns.

There are many definitions of the role concept in the literature [2][12], but we are interested in using roles as components of classes. For that purpose, we use the role definition used by Riehle [12], where roles are an observable behavioral aspect of an object. We can use roles to compose classes, meaning that an object's behavior is defined by the composition of all roles it plays.

For modeling crosscutting concerns with roles, we place each crosscutting concern in a role and classes that deal with one concern just play the respective role. This way the role encapsulates the concern code and prevents code clones.

To validate these ideas, we conducted a case study with the JHotDraw framework. In this case study, we used a clone detection tool to identify code clones in the framework. We identified crosscutting concerns by aggregating clones that deal with the same concern. Each crosscutting concern was analyzed and, whenever possible, a role that deals with that concern was developed using JavaStage [13], an extension to the Java language that supports roles. The results of the case study indicate that roles can in fact be used to model crosscutting concerns and reduce code clones from a system.

This paper is organized as follows. The next Section presents role modeling and how it can address crosscutting concerns. Section III presents the JHotDraw case study and its results. Related work is presented in Section IV, and Section V concludes the paper.

II. MODELING WITH ROLES

Role modeling using static roles was used as an integral part of the OORam method [14] and by Riehle in [12]. We took these modeling approaches into the programming level using roles as blocks for composing classes. To support roles, we developed the JavaStage language. We will not discuss JavaStage but refer the reader to [13]. JavaStage extends java but our approach may apply to other single-inheritance languages and to multiple inheritance languages.

In JavaStage, a role is a first class entity, so it can be described using an appropriate type specification. A class that plays a role type acts according to the role type specification. Classes may act according to several different role types. Thus, different clients may have different views on a class instance.

A class represents a domain abstraction, its properties and behavior. But, in JavaStage, a class also defines which roles it plays and how they are composed. The union of the operations defined in the class and the operations defined in the roles constitutes the class interface, and the composition of all role types constitutes the type of the class. This is to say that the class interface is the union of the role interfaces [15]. Because a class may be viewed as a class that plays only one role then this model is a canonical extension of the object model [16]. It means that existing software can be integrated into the role model without changes.

We could achieve the same effect by using multiple inheritance, defining each role in a separate class. The composing class would inherit from all classes. The use of multiple inheritance, however, has many problems. These come mostly by name collisions when a class inherits from two or more superclasses that have equally named methods or fields and duplicated code when a class inherits twice from the same superclass – the classic diamond problem.

In JavaStage, roles have features like a powerful renaming mechanism that allows classes to tailor methods' names for their specific situation; the possibility to play the same role more than once and the possibility to define multiple versions of a method [13]. Roles can inherit from roles and can play other roles thus giving developers a big range of modeling options.

To exemplify role modeling we present in Figure 1 the class diagram of a simplified graphical user interface (GUI) framework based on Java AWT/Swing frameworks. Framework classes represent the widgets (or components) that usually appear in a GUI, like windows, buttons, menus, toolbars, etc. Some components may own other components: a window may own several toolbars, and a toolbar may own several buttons. Some clients may be interested on knowing when the mouse is hovering a component so the Observer pattern is used. Other instances of this pattern are used as clients may be interested in other user's actions or if a component has lost focus, etc. A component has a collection of properties that specifies the way it should de drawn. Properties are represented by name-object pairs, where name is the property and object represents the property's value.

Clients that are interested in knowing if a component has lost focus are not interested in drawing a component or if the user clicked it with the mouse. For those clients, components assume the role of FocusSubject and only those operations related to that role are of interest. For the clients who want to know about mouse handling actions, components play the role of MouseSubject. Clients may set or read properties of the component so, for these, the component plays the role of PropertyProvider. The CompositeParent role is responsible for managing a collection of children.

The mentioned roles are depicted on the upper right side of Figure 1, where we also show the role associations and the revised class diagram, now using the roles.

A. *Role modeling advantages*

Role modeling comes with several advantages in terms of reuse, comprehension, development and documentation [12]. When a class is described as a set of roles it helps separating the various ways in which a class is used. This means that

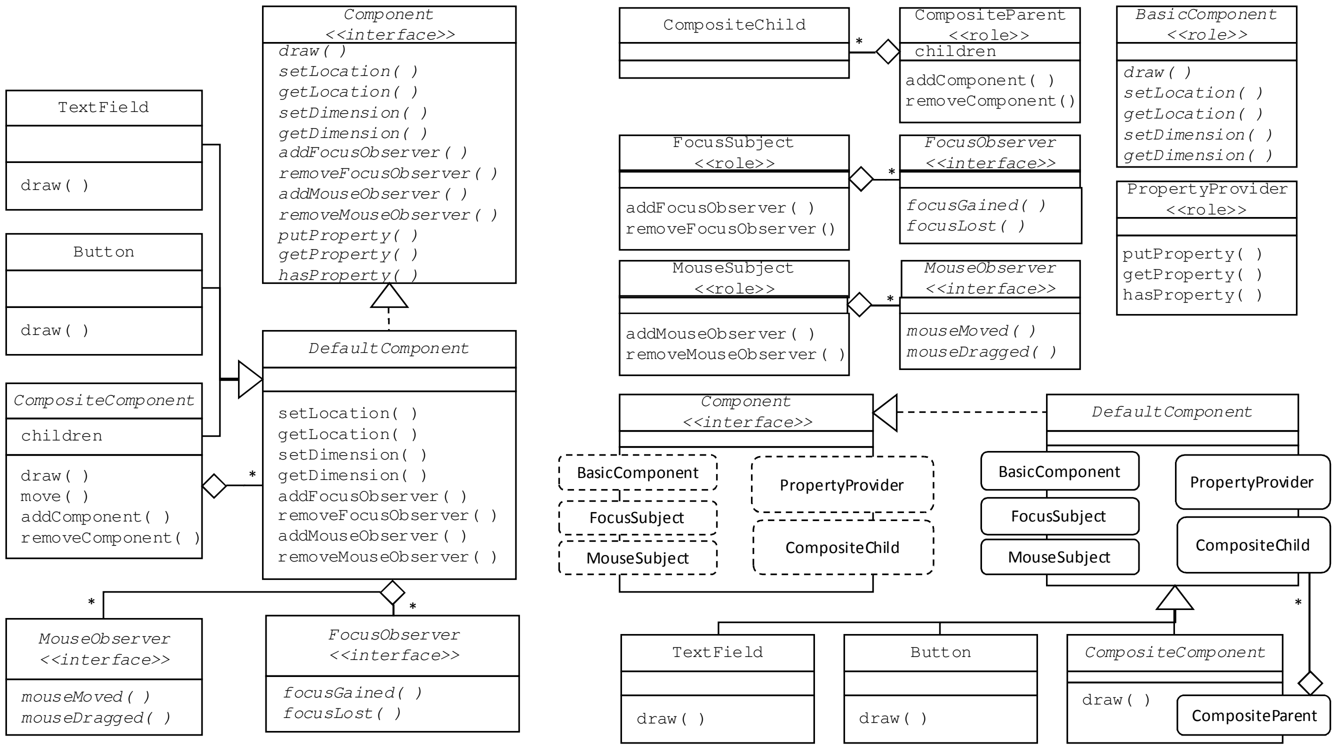


Figure 1. Example of modelling with roles. On the left: class diagram of the Component Framework. On the right: Roles and their relations in the Component Framework, and the revised class diagram of the Component Framework, now with roles. Rounded rectangles identify roles played by the class. Dashed round rectangles represent the interface provided by the role

the documentation can be done in these terms. That helps clients to better understand and use the class and focus on whichever aspect they are interested in. Designing the class can also be done in role terms, thus developers are able to focus only in one aspect of the class. This enables independent development of a class with all its benefits in terms of reduced development time and complexity.

Class relationships are reduced to role relationships. Because roles focus in a particular view of a class, we need not to understand the target class in its whole. This facilitates the understanding and development of these relationships. Whenever needed the broader perspective can also be used. Role modeling allows for a transition between the role level and the class level, without losing any information.

Role modeling also allows for better understanding using previous experiences. When a developer knows how to use roles that have a relationship in a system, then when he encounters different roles with similar relationships the past experience will allow a better understanding. One such example is the use of the Observer pattern. When experienced with a FocusSubject and how it works with a FocusObserver to use the MouseSubject and a MouseObserver is much simpler and straightforward.

B. *Modeling Crosscutting Concerns*

Crosscutting concerns are those concerns that appear when several modules must deal with the same problem because one cannot find a single module responsible for it in the light of a decomposition strategy. This leads to scattered, replicated code. Its consequences are the opposite of the benefits of modularizations. Since a module deals with a part of a problem that is spread over other modules, changes to that code may affect those modules. This affects independent development. Maintenance is impaired too because changes in the code needs to be done in all modules transversely.

Because a role is smaller composition unit than a class we can put the crosscutting concern in a role, or a set of roles, and the classes that have the crosscutting concern play those roles. Any changes to the crosscutting are limited to the roles thus greatly improving maintenance and reducing change propagation, or in other words, the crosscutting concerns become more modular.

Even the simplified GUI framework shows several examples. The component's main concept is not to act like a Focus Subject or a Mouse Subject but it has those roles superimposed on it. With roles we were able to extract those concerns from the class, thus reducing the scattering of code. Furthermore those roles are reusable whenever we need a class to address any of those concerns, even if it is not part of the Component hierarchy. We can also argue that being a PropertyProvider is not the component's main concern. It assumes that a property is identified by a name and that name is a String. It would be more reusable if it used generics for the property type. We can also use generics to specify the value type instead of type Object. After a closer look, the property provider is in fact a map that maps keys to values. We could reuse a map implementation if we inherited from a Map class, but that would be conceptually wrong. Our class is not a map, it plays the role of a property map.

Figure 2 shows the code for that mapper role and the code for a Component class playing the PropertyProvider role and also of a Figure class that also plays the same role, but for figure properties like line color, line width, etc. In both cases the map uses string as keys and objects as values and in both cases the methods are getProperty, putProperty and hasProperty, as defined by the configuration, but they could use different key/values types and methods' names.

```
public role Mapper<KeyType, ValueType> {
  private Map<KeyType,ValueType> map;

  public ValueType get#Thing#( KeyType name ){
    return map.get( name );
  }
  void put#Thing#( KeyType name, ValueType value){
    map.put( name, value );
  }
  public boolean has#Thing#( KeyType name ){
    return map.containsKey( name );
  }
}

class DefaultComponent implements Component {
  plays Mapper<String,Object>(
            Thing = Property ) mapper;
}
class DefaultFigure implements Figure {
  plays Mapper<String, Object>(
            Thing = Property ) mapper;
}
```

Figure 2. Definition of the Mapper role with configurable methods and two classes playing the role

III. CASE STUDY

A. *Case Study Subject*

To asses how roles are capable of modeling crosscutting concerns we applied them to the JHotDraw framework. JHotDraw is a Java GUI framework for technical and structured Graphics. JHotDraw is structured around four main inheritance hierarchies. These hierarchies reflect the main classes used in the framework. These are the Figures, Views, Tools and Handles.

JHotDraw has been used in works for the detection of crosscutting concerns for aspect mining [17] so it is a suitable candidate for this study, where we want to assess how roles handle those crosscutting concerns.

B. *Case Study Setup*

We searched for replicated code using CCFinderX [18] an established clone detection tool used in the aspect mining works [17]. We used the standard options of CCFinder.

We are interested in crosscutting concerns, so we are interested only in clones that are not solvable with traditional refactorings [19]. One of such refactorings is the Extract Method that usually deals with code inside a unique class. So to filter out such clones we only considered clones that appeared in, at least, two files. This also filter clones that do not deal with crosscutting concerns as a concern must be present in at least two classes to be considered a crosscutting concern. For simplicity and space reasons we will refer crosscutting concerns simply as concerns.

The first output included 271 clone sets. After filtering we ended up with 146 clones. After a manual inspection 41 false clones were removed leaving a final 105 sets. Some clones are not really identical, but as they focused on the same concern so we did not remove them. This will account for some of the unresolved concerns.

We grouped clones according to the concern they dealt with. We identified a total of 43 concerns. From those 43 concerns we removed 5 because 2 could be resolved by refactoring alone, 1 was deprecated code and 2 were classes pending substitution.

After this selection we again analyzed the clones and, if possible, we've built a role encapsulating the concern. We've decided not to change any class interface so the overall framework is unchanged. We've also set a rule not to change the concern implementation to retain the author's intent. Only minor changes were allowed, as they wouldn't compromise it. We only developed roles that respect the role concept. We detected some clones that could be removed using a different inheritance hierarchy. We did not use roles to reduce that replicated code, because changing the inheritance hierarchy was a better solution

Roles were developed with JavaStage. The JavaStage compiler and the developed JHotDraw framework, can be found at http://www.est.ipcb.pt/pessoais/fsergio/javastage.

C. *Case Study Results*

Results are shown on Table 1. For each concern it shows how many clones were associated and how many classes were affected. It also shows the number of lines of code (LOC) that the clone had, the lines of code that were used by Roles and the ratio between them. For the concerns where roles failed it states the reason why they failed.

We can see that from the 38 concerns only 8 (21%) were not resolved with roles. This seems to indicate that roles are suited to model crosscutting concerns. The final outcome is better than these numbers indicate as we will discuss.

LOC are a good measure on the effort that each approach requires but it is not a good measure on how the modularity issues are handled. One can write more lines of code but if the resulting system is more modular it is a better system.

We counted as LOC the requirements statements that roles must declare. We also counted as LOC the roles' plays directive. Assume one concern that presents 8 lines of replicated code in each class which could be resolved with a simple role. We would expect this role to have the same 8 LOC. That is not so because we do not count the class declaration as a clone LOC but count the role declaration as a solution LOC. Roles may also require methods, so these requirements are counted as LOC. Thus for the 8 LOC clone the role would have 1 more fixed, 1 more for each player and 1 more for each requirement. If the role requires 3 methods and the clone appears in two classes then the clone has 16 LOC and the role solution would count 14 LOC. That may not seem a great improvement but LOC do not account for the modularity and maintenance issues. Removing the clone gives the system a great advantage in modularity terms.

TABLE I. IDENTIFIED CONCERNS WITH THE NUMBER OF ASSOCIATED CLONES AND AFFECTED CLASSES. IT ALSO SHOWS THE LOC FOR EACH APPROACH AND RESPECTIVE RATIOS.

Concern	clone #	class #	Original LOC	Roles LOC	Roles / Original
Drawing Handles	8	15	64	40	63%
Setting up the undo activity before executing a Command	2	8	56	44	79%
BringToFront/SendToBack Commands	1	2	20	12	60%
Handle creation	11	20	70	87	124%
Drawing polygons	1	2	12	11	92%
Palette Listener	1	2	20	17	85%
DisplayBox persistence	2	5	35	12	34%
DisplayBox handling	6	8	58	29	50%
DesktopListener Subject	2	3	63	45	71%
Changing connections	3	3	98	53	54%
Finding connectable figure	1	3	98	53	54%
Testing command executability	5	7	14	14	100%
Floating text holder	2	2	47	36	77%
DrawingViewListener Subject	2	4	63	26*	41%
Setting text in a text Figure	2	2	36	22	61%
Enumerator	1	3	33	11*	33%
Figure Listener that resends notifications	2	3	35	23*	66%
Menu enabling	1	2	20	14	70%
Version control	1	2	12	9	75%
Selected button manager	1	2	18	12	67%
Text attributes management	2	2	206	120	58%
Updating DrawingView Strategy	1	2	29	26	90%
Connection insets computing	1	3	10	7	70%
Undo/Redo Commands	1	2	32	31	97%
Changing connection handles	1	2	20	19	95%
Polygon and PolyLine Handles	3	2	32	28	88%
Tools and Commands Dispatchers	6	4	89	32*	36%
Figure/Handle and Enumerator	1	2	33	2*	6%
Polygon locator	1	2	13	20	154%
Drawing editor	1	3	54	28*	52%
			Reason		
Desktop initial configurations	1	2	Too much configuration		
Persistence (read/write)	3	6	Similar but not identical code		
UndoActivity	13	24	UndoActivity inner classes declaration and constructor		
Creating UndoActivity	14	18	After other roles was just a line of code		
Handle manipulation starting action	3	5	Too much configuration		
Point is inside Figure	3	6	code too small		
DrawingView Listener	1	2	perfomance issues		
Mouse motion handling	1	2	code too small		

* = reused from library

1) *Modelled Concerns*

Roles succeeded in 30 (79%) of the 38 concerns. This indicates that roles are capable of reducing replicated code and modeling crosscutting concerns. Comparing the LOC ratio, one finds that, in average, roles only have 68% of the original code, so the effort of developing the role system is smaller. Taking the absolute LOC value, the original system has 1390 LOC and roles have only 883 LOC. This means that roles reduced the replicated code in 36,5%.

In 6 concerns we were able to reuse/place roles from a role library [20] we developed to capture the basic behavior of the Gang of Four design patterns [21]. This explains the great difference in LOC in these concerns. From all the concerns roles resolved, two exhibit a higher number of LOC than the original implementation.

The “Handle creation” concern deals with the creation of handles for each figure. We moved the handle creation to a handle creator class and the role class methods on that class. Since some clones only have similar code we had to reproduce every method in this creator class. The class code, plus the code original classes use to play the role and the definition of the role leads to more lines of code than the original implementation. But the role has one advantage: it can dynamically change the handle creator.

The “Polygon Locator” is responsible for returning a point inside a polygon given a point index. It is used in two classes but one of them uses an anonymous class. Currently JavaStage’s roles cannot be applied to anonymous classes so we had to develop an inner class. This code and the role configuration lead to a higher LOC, because the original code size was not enough to compensate for this overhead.

2) *Unresolved concerns*

A surprising result is that the two concerns with the most clone sets and class involved are unresolved with roles. This is due to the nature of the clones. They are clones only in the structure and not on the code itself. The ”Creating undo activity” concern creates an undo activity object for each of the various tools and commands supported by the framework. Each tool class has an UndoActivity inner class hence the undo activity creation is just a line of code instantiating an object of the respective inner class. Because each inner class constructor has different parameters in number and types, roles could not resolve this concern. UndoActivity concern clones are due to the inner classes, because they all have the same name and constructors with the same structure, even if not equal. Another example of such a concern is the “Handle manipulation starting action”: code is similar but not quite identical and most code would disappear with refactoring.

Another example is “Persistence”: because figures must be streamed they have a write and read method with similar structure, but not quite identical code. We reduced this duplicated code with our DisplayBoxed role, though.

Another unresolved concern is the “DrawingView listener”. An overriding method is redefining the original, allegedly for performance issues we failed to understand.

One unresolved clone, “Desktop Initial configuration”, dealt with a Desktop’s panel initialization, which initializes panel titles and adjusts a scrollPane. Each possible initialization is similar so we could configure a role for every way a scroll pane is configured and then reuse them. But knowing each possible role would require more effort than to know how to configure the scroll pane.

The other unresolved concerns were a single line in the form of `return getSomeObject().doSomething()`. Since the first method returns different objects that in turn call different methods, role configuration would be harder than writing the code itself.

Had we not considered some of these concerns as crosscutting concerns, we would count only 4 as unresolved.

3) *Threats to Validity*

One threat to this study results is that we only considered a single system. For results to be more decisive we might need to do the same test with more systems. Nevertheless the nature of roles allows us to say, with some confidence, that results for other systems would not be that different.

The clone detecting settings can also affect the detected clones that would lead to different concerns. That and the removal of clones from the same file could have removed important clones. However, we would need to reduce the amount of clone sets to a manageable number. We even go under the limit of the minimum 30 tokens recommended in [18] for limiting false clones. So while different settings would result in some different clones we believe that our settings provided a good result in detecting meaningful concerns.

IV. Related Work

Feature Oriented Programming (FOP) decomposes the system into features [10], which are the main abstractions in FOP during design and implementation. Features reflect user requirements and incrementally refine each other. FOP relies on a step-wise refinement of applications by adding new features or refining existing ones. FOP is mainly used in Software Product Lines and program generators. In FOP, Mixins are used to implement features [8]. Each mixin layer contains the code each class needs for a given feature and are composed into a static component. Roles can be used instead of mixins, as they offer more ways of configurations and don’t have mixins limitations like a linear composition order.

Aspect-Oriented Programming is another approach that tries to modularize crosscutting concerns [11]. But AOP is not close to OO and requires learning many new concepts. And while the modularization of crosscutting concerns is the flagship of AOP several authors disagree [22][23]. Concepts like pointcuts and advices are not easy to understand. The effects of these constructs are also more unpredictable than any OO concept. A particular one is the fragile pointcut. This problem arises when simple changes made to a method code make a pointcut either miss or incorrectly capture a joint point thus incorrectly introducing or failing to introduce the necessary advice. Thus simple changes in the class code can have unsought effects [24].

The obliviousness feature of AOP means that a class is aspect unaware so aspects can be plugged or unplugged as needed. But it also introduces problems in comprehensibility [25]. To fully understand the system we must not only know the classes but also have to know the aspects that affect each class. This is a major drawback when maintaining a system, since the dependencies aren’t explicit and there isn’t an explicit interface between both parts. With our approach all dependencies are explicit and the system comprehensibility is increased when compared to the OO version [26]. We do not have the obliviousness of AOP as the class knows and is aware of the roles it plays. But any changes to the class code are innocuous to the role, as long as their contract is fixed.

We do not believe our approach can replace AOP. They are different and approach different problems. We believe that for static concerns our approach is more suitable while AOP is better suited for (un)pluggable concerns.

Traits [6] offer a way of composing software that is somewhat similar to Mixins [6]. A trait is the primitive unit of code reuse, like our roles, which means that only traits can be used to compose classes. Traits can also be used to compose other traits. But traits only provide methods and not state and access levels. A class composed with traits can be seen either as a flat collection of methods or as being composed by traits. This flat property means that the code inside the trait can be seen as the code inside the class, for example, a super reference inside the trait code refers to the superclass of the class that uses the trait. In our approach we can also see a class as simply a set of methods, forgetting that it plays a role, but we have not this flat property, as a super reference in a role refers to the superrole.

V. Conclusion and Future work

We have presented a new way of modeling crosscutting concerns. Using roles we have a finer grain composition technique that allows the crosscutting concerns to be composed into the classes without its code being placed in the class itself.

We modeled crosscutting concerns by developing a role that addressed it. The crosscutting concern's code is therefore limited to the role. To better model those concepts roles support state and visibility control. Classes play the role and acquire the role behavior. Changes to the concern implementation are limited to the role.

We validated our approach developing roles for the JHotDraw framework and eliminated nearly all of the existing crosscutting concerns that exhibited duplicated code. We even reused some roles from our role library showing that they are really reusable.

For future work we are developing a role version of the Sun's java compiler and the Spring framework, using JavaStage. Results so far are promising as we already reused some of our library roles, like an Observer and Visitor. The use of these roles in those case studies can eliminate a great amount of duplicated code.

References

[1] Parnas, D. L., (1972): On the criteria to be used in decomposing systems into modules. Commun. ACM 15, 12, Dec. 1972, 1053-1058

[2] Kristensen, B. B., (1995): Object-oriented modeling with roles, in Proceedings of the 2nd International Conference on Object-Oriented Information Systems, Springer-Verlag.

[3] Tarr, P., Ossher, H., Harrison, W. and Sutton Jr., S. M. (1999), N degrees of separation: multi-dimensional separation of concerns, Proceedings of the 21st international conference on Software engineering, New York, NY, USA

[4] Miryung Kim, Lawrence Bergman, Tessa Lau, and David Notkin, An ethnographic study of copy and paste programming practices in oopl, Proceedings of the 2004 International Symposium on Empirical Software Engineering (Washington, DC, USA), ISESE '04, 2004, pp. 83–92.

[5] Bruntink, M. van Deursen, A. van Engelen, R. Tourwé, T., On the use of clone detection for identifying crosscutting concern code, IEEE Transactions on Software Engineering, Vol. 31, No. 10, (2005)

[6] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do Code Clones Matter? In Proc. Int. Conf. on Software Engineering, pages 485–495. IEEE Computer Society, 2009.

[7] C. Roy and J. Cordy. A Survey on Software Clone Detection Research. Technical Report 2007-451, School of Computing, Queen's University at Kingston, 2007.

[8] G. Bracha, and W. Cook. Mixin-Based Inheritance. In Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications / Proceedings of the European Conference on Object-Oriented Program-ming, pages 303–311, 1990. Ottawa, Canada.

[9] S. Ducasse, N. Schaerli, O. Nierstrasz, R. Wuyts and A. Black: Traits: A mechanism for fine-grained reuse. In Transactions on Programming Languages and Systems. 2004.

[10] S. Apel and C. Kästner. An Overview of Feature-Oriented Software Development, in Journal of Object Technology, vol. 8, no. 5, July–August 2009,pages 49–84

[11] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W.G. Griswold. An overview of AspectJ. In proceedings of ECOOP 2001, Budapest, Hungary, (LNCS, vol. 2072), Springer; 327–335, 2001

[12] D. Riehle Framework Design: A Role Modeling Approach, Ph. D. Thesis, Swiss Federal Institute of technology, Zurich. 2000

[13] Barbosa, F. and Aguiar, A. (2012). Modeling and Programming with Roles: Introducing JavaStage, In the 11th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT_12), Genoa, Italy, to appear.

[14] T. Reenskaug, P. Wold, and O. A. Lehne. Working with objects - the OOram software engineering method. Manning, 1996.

[15] Steimann, F., (2001): Role = interface: a merger of concepts, Journal of Object-Oriented Programming 14(4): 23–32.

[16] Chernuchin, D., and Dittrich, G. (2005). Role Types and their Dependencies as Components of Natural Types. In AAAI Fall Symposium: Roles, an interdisciplinary perspective.

[17] Ceccato, M., Marin, M., Mens, K., Moonen, L, Tonella, P. and Tourwe, T. A qualitative comparison of three aspect mining techniques, Proceedings of the 13th InternationalWorkshop on Program Comprehension (Washington, DC, USA), IWPC '05, 2005, pp. 13–22

[18] Kamiya, T., Kusumoto, S. and Inoue, K. (2002), Ccfinder: a multilinguistic tokenbased code clone detection system for large scale source code, IEEE Trans. Softw. Eng. 28, no. 7.

[19] Fowler, M., (1999), Refactoring: Improving the design of existing code, Addison-Wesley, Boston, MA, USA.

[20] Barbosa, F. and Aguiar, A. (2011). Generic roles, a test with patterns In 18th Conference on Pattern Languages of Programs, PloP 2011 Oct 21-23, Portland, OR, USA

[21] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., (1995): Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.

[22] Steimann, F., The paradoxical success of aspect-oriented programming", in OOPSLA '06, Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Languages, Systems, and Applications (2006)

[23] Przybyłek, A.(2001). Systems Evolution and Software Reuse in Object-Oriented Programming and Aspect-Oriented Programming , J. Bishop and A. Vallecillo (Eds.): TOOLS 2011, LNCS 6705, pp. 163–178.

[24] Kästner, C., Apel, S., Batory, D., 2007: A Case Study Implementing Features using AspectJ. In:11th International Conference of Software Product Line Conference (SPLC 2007), Kyoto, Japan

[25] Griswold, W.G., Sullivan, K., Song, Y., Shonle, M., Tewari, N., Cai, Y., Rajan, H., 2006: Modular Software Design with Crosscutting Interfaces. IEEE Software 23(1), 51–60 (2006)

[26] Riehle, D. and Gross, T. 1998. Role Model Based Framework Design and Integration." In Proceedings of the 1998 Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '98). ACM Press

Towards a Glue-Code Specification Framework for Component-Based Systems

Sajjad Mahmood* and Mohammed A. Al-Qadhi†
Information and Computer Science Department
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
Email: smahmood@kfupm.edu.sa*, moh6666_qadi@hotmail.com†

***Abstract*—Component integration is a key process activity during the development of Component-Based Software (CBS). CBS integrators use software components developed by either in-house or purchased off-the-shelf to develop an application. CBS integration process often requires adaption of selected components to meet the CBS-to-be requirements as individual components are usually developed for general purposes. Latest industrial studies for CBS development have indicated that system integrators reply on individual experiences to write the glue-code of a CBS. Hence, there is a need for an integration framework to handle the potential mismatches between individual components and identify the functionalities not provided by the available components for a CBS. In this paper, we present an initial glue-code framework to specify the glue-code required in integrating potentially mismatched components and the missing functionalities required to meet the requirements of a CBS.**

***Index Terms*—Component Based Systems; Component Integration; Glue-Code Specification; UML; Use Case Model.**

I. Introduction

Software component is a fundamental building block for an application. CBS development focuses on integrating pre-existing software components to build a software application [1], [2], [3], [4]. A system integrator puts together software components developed by different vendors who are usually unaware of each other [5]. Hence, the integration phase of a CBS development life cycle is a challenging activity as individual components are usually designed for general purposes and they might not completely satisfy requirements of a CBS-to-be. Furthermore, detailed documentation is rarely available for the majority of components [5], [6] and system integrators have to rely on component interface documentation during the integration phase of a CBS.

The glue-code written during the integration phase plays an important role in the overall success of a CBS. The glue-code provides a platform to integrate potentially mismatching components and implement the missing functionalities required to meet requirements of a CBS. CBS research [2], [6] has shown that system integrators rely on their experience to write the glue-code and there is a lack of glue-code development framework to support the important integration phase of the CBS development life cycle.

In this paper, we present an initial glue-code specification framework for writing the glue-code of a CBS. The glue-code specification framework will help system integrators in early identification of potential mismatches between component interfaces and missing functionalities required to satisfy stakeholder requirements of a CBS. We introduce the notation of use case conceptual mapping and component-based sequence mapping to specify interactions between components of a CBS. We also use a hotel reservation system [7] as a running example to explain the glue-code specification framework.

The rest of this paper is organized as follows: Section II reviews the related literature. In Section III, we present the glue-code specification framework. We conclude the paper and discuss future work in Section IV.

II. Related Work

Vigder and Dean [8] presented the concept of wrappers to glue software components by considering the elements of architecture during the integration process. Rine et al. [9] used adapters to integrate components. Each component has an associated adapter and components request services from each other through their associated adapters.

Dietrich et al. [10] used active rules to design wrappers to adapt components. The wrappers are automatically generated as components and they act as proxy objects. These proxy objects intercept method calls and provide the functionality required by the overall component-based system. Similarly, Canal et al. [11] presented a model based approach for component adaptation using synchronous vectors based notation to automatically generates adapter protocols during development of a CBS.

Kim et al. [12] proposed a process for CBS development where component integration occurs at the release phase. Zitouni et al [13] presented a contract-based approach to analyze and model the properties of components and their composition in order to detect and correct composition errors. The approach allows to characterize the structural, interface and behavioral aspects of the components. Chi [14] defined the signature view and behavior view of software components and used pi calculus expressions to model behavior of components.

The current CBS development approaches lack a systematic process to bridge the gap between requirements analysis and integration specification of a CBS. This results in an integration phase that heavily relies on system integrator's skills which increase the challenges associated with the glue-code specification of a CBS.

III. Glue-Code Specification Framework

The glue-code specification framework presents a systematic structure to identify the required interfaces and specify interactions between components of a CBS. The glue-code specification framework also helps reduce CBS development risks by identifying the interface mismatches between components and the missing functionalities required to implement a CBS.

The glue-code specification framework consists of two phases, namely, use case conceptual mapping and component based sequence mapping. The first phase - use case conceptual mapping - starts with a process of modeling a use case as a required interface and subsequently specifies all the required and provided interfaces for a CBS. The second phase - component based sequence diagram - uses the extended UML sequence diagram to model different scenarios associated with a use case to identify mismatches between required and provided interfaces of a CBS. Figure 1 shows the glue-code specification framework.

A. Use Case Conceptual Mapping

The UCCM phase takes the Unified Modeling Language (UML) use case diagram [15] and component interface documentation as an input to develop a realization mapping between software components and requirements of a CBS. First, we adopt UML component specification

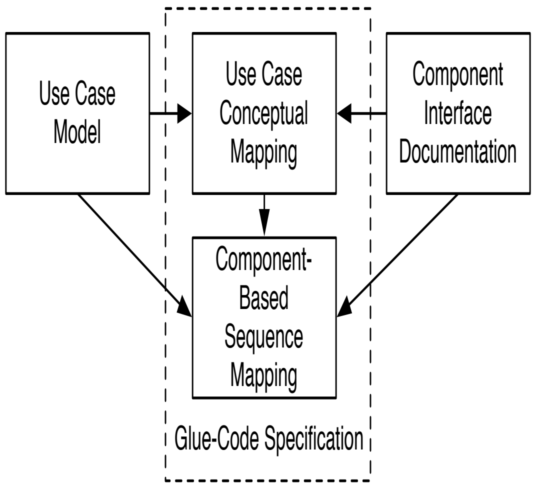

Use Case Model
Use Case Conceptual Mapping
Component Interface Documentation
Component-Based Sequence Mapping
Glue-Code Specification

Fig. 1. Glue-Code Specification Framework

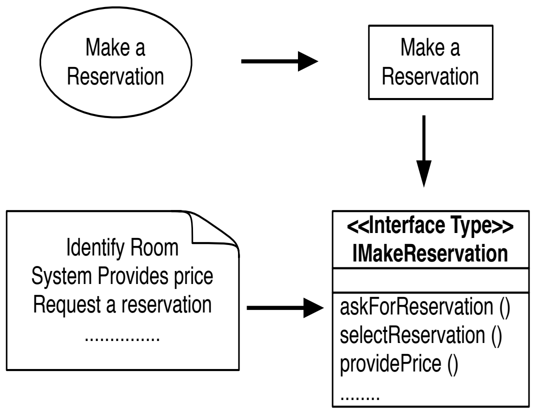


Fig. 2. 'IMakeReservation' System Interface

technique [7] to model each use case of a CBS as a 'conceptual interface' which consists of a set of operations corresponding to individual interactions of the use case. Figure 2 shows 'IMakeReservation' system interface for the hotel reservation system.

Second, each interface of a component is represented as a 'concrete interface' which consists of a number of concrete operations which are uniquely identified by a 'concrete operation code'. A 'concrete operation code' consists of components' name, interface number and corresponding operation number. For example, the concrete operation code (B-I2-O2) will represent the second operation of the second interface of component B.

Finally, a UCCM realization table is generated for each 'conceptual interface'. A UCCM realization table shows all conceptual and concrete interfaces that help realize a

TABLE I
'IMAKERESERVATION' REALIZATION TABLE

Conceptual Operations	Realization
$askForReservation()$	$Missing$
$selectReservation()$	$Missing$
$providePrice()$	$Partially(D - I2 - O6)$
$getCustomerInfo()$	$Missing$
$reservation()$	$(A - I1 - O1)AND$
	$(D - I2 - O8)AND$
	$(B - I2 - O5)OR(E - I2 - O4)$
$refineReserveDetails()$	$Missing$
$provideAlternative()$	$refineReservDetails()AND$
	$(D - I2 - O6)OR(E - I5 - O3)$
$acceptOrReject()$	$Missing$
$failure()$	$Missing$
$notifyBillingSys()$	$Missing$

'conceptual interface'. Table I shows UCCM realization table for the 'IMakeReservation' interface.

B. Component Based Sequence Mapping

The CompBSM phase uses UML sequence diagram to present scenarios associated with each 'conceptual interface' of a CBS. The sequence diagram developed during the CompBSM phase has a glue-code component that helps a system integrator in identifying the execution precedence, missing functionalities and potential mismatches between a set of participating interfaces. In this paper, we extend UML by introducing new stereotypes to help system integrators in specifying the glue-code required to integrate candidate components. The message interaction stereotypes are as follows:

1) <<Initiation>>: To specify first/initial messages.
2) <<Missing>>: To specify missing functionalities.
3) <<LibraryImporting>>: To specify importing header files and built-in libraries from components.
4) <<Adapter>>: To specify messages that involve mismatched data types involved in an interaction.
5) <<TemporaryStorage>>: To specify interactions that provides temporary variables to store values to be used through a scenario.

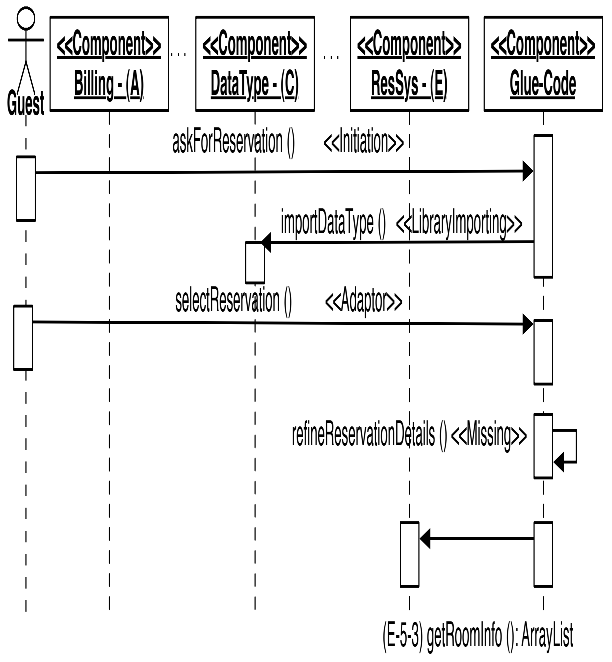

Guest
<<Component>> Billing - (A)
<<Component>> DataType - (C)
<<Component>> ResSys - (E)
<<Component>> Glue-Code
askForReservation () <<Initiation>>
importDataType () <<LibraryImporting>>
selectReservation () <<Adaptor>>
refineReservationDetails () <<Missing>>
(E-5-3) getRoomInfo (): ArrayList

Fig. 3. 'IMakeReservation' Sequence Diagram

6) <<ExceptionHandling>>: To specify interactions that provides exception handling functionality for the system.

Figure 3 shows a partial sequence diagram for the IMakeReservation use case. A guest actor starts the scenario by invoking method 'askForReservation ( )' from the glue-code component of a CBS. The interaction is labeled with <<Initiation>> stereotype to indicate the first interaction in the scenario. Next, glue-code component makes a message call to 'importDataTypes ()' method of the 'datatype - (C)' component. The interaction is labeled with <<LibraryImporting>> stereotype to specify that built-in datatype library is called form the 'datatype - (C)' component. Next, 'selectReservation ()' method is executed which is tagged with the <<Adaptor>> stereotype. This indicates that some sort of adaptation code needs to be written in the glue-code component to collect the required information from the customer.

Furthermore, 'refineReservationDetail' () method is tagged with <<Missing>> stereotype to indicate that the required functionality is not provided by any available component and it needs to be implemented in the glue-code component of a CBS. Finally, '(E-5-3) getRoomInfo ()'method is invoked that shows that 'getRoomInfo ()' method associated with 'ResSys' component is called to realize the desired functionality required for the IMakeReservation use case.

IV. Conclusion and Future Work

We have presented an initial glue-code specification framework to help system integrators in understanding potential mismatches between component interfaces and missing functionalities required to meet stakeholder requirements of a CBS. The glue-code specification framework consists of two parts; namely, use case conceptual mapping and component-based sequence mapping phases. The use case conceptual mapping is used to specify the component interfaces involved in realizing individual use cases a CBS. Furthermore, component-based sequence mapping phases uses UML sequence diagram to specify the component interface mismatches and missing functionalities required to successfully developing a CBS.

In this paper, we have presented a preliminary work on the glue-code specification framework. For future work, we plan to complete the framework by considering both integration and composition issues related to the glue-code of a CBS. There is a need to apply the framework on real world case studies to better understand its potential benefits for the system integrators of a CBS. We plan to develop an automated tool for supporting glue-code specification framework. Furthermore, there is also a need to investigate the potential benefits of the complete glue-code framework on integration testing, maintenance and evolution phases of a CBS.

Acknowledgement

The authors would like to acknowledge the research support provided at King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia.

References

[1] S. Mahmood, R. Lai, and Y. S, Kim, "Survey of component based software development," *IET Software*, vol. 1, no. 2, pp. 57–66, 2007.

[2] J. Li, R. Conradi, O. P. N. Siyngstad, C. Bunse, M. Torchiano, and M. Morisio, "Development with off-the-shelf components: 10 facts," *IEEE Software*, vol. 26, no. 2, pp. 80 – 87, 2009.

[3] M. A. Khan and S. Mahmood, "Optimal component selection for component-based systems," in *2009 International Conference on Systems, Computing Sciences and Software Engineering*, pp. 467 – 472, 2009.

[4] M. A. Khan and S. Mahmood, "A graph based requirements clustering approach for component selection," *Advances in Engineering Software*, vol. 54, pp. 1–16, 2012.

[5] A. Cechich and M. Piattini, "Early detection of cots component functional suitability," *Information and Software Technology*, vol. 49, no. 2, pp. 108 – 121, 2007.

[6] S. Mahmood and A. Khan, "An industrial study on the importance of software component documentation: A system integrators perspective," *Information Processing Letters*, vol. 12, pp. 583–590, 2011.

[7] J. Cheesman and J. Daniels, *UML Components A Simple Process for Specifying Component Based Software*. Addison-Wesley, 2001.

[8] M. R. Vigder and J. Dean, "An architectural approach to building systems from cots software components," in *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, p. 22, IBM Press, 1997. Toronto, Ontario, Canada.

[9] D. Rine, N. Nada, and K. Jaber, "Using adapters to reduce interaction complexity in reusable component based software development," in *Proceedings of the 1999 symposium on Software reusability*, pp. 37–43, ACM Press, 1999. Los Angeles, California, United States.

[10] S. W. Dietrich, R. Patil, A. Sundermier, and S. D. Urban, "Component adaptation for event-based application integration using active rules," *Journal of Systems and Software*, vol. 79, no. 12, pp. 1725 – 1734, 2006.

[11] C. Canal, P. Poizat, and G. Salaun, "Model-based adaptation of behavioral mismatching components," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 546 – 563, 2008.

[12] S. Kim, S. Park, J. Yun, and L. Y, "Automated continous integration of component-based software: An industrail experience," in *Proceedings of 23rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 423 – 426, 2008.

[13] A. Zitouni, L. Seinturier, and M. Boufaida, "Contract-based approach to analyse software components," in *13th IEEE International Conference on Engineering of Complex Computer Systems*, pp. 237 – 242, 2008.

[14] Z. Chi, "Software components composition compatability checking based on behaviour description," in *IEEE International Conference on Granular Computing*, pp. 757 – 760, 2009.

[15] K. Bittner and I. Spence, *Use Case Modeling*. Addison-Wesley, 2002.

AO-WAD: A Generalized Approach for Accessible Design within the Development of Web-based Systems

Adriana Martín[1,2]
[1]Department of Exact Sciences, Caleta Olivia
University of Patagonia Austral (UNPA-UACO)
Santa Cruz, Argentina
[2]GIISCo, Computer Science Department,
University of Comahue (UNCo)
Neuquén, Argentina
e-mail: adrianaelba.martin@gmail.com

Viviana Saldaño[1], Gabriela Miranda[1], Gabriela Gaetán[1]
[1]Department of Exact Sciences, Caleta Olivia
University of Patagonia Austral (UNPA-UACO)
Santa Cruz, Argentina
e-mails: {vivianas / gmiranda / ggaetan}@uaco.unpa.edu.ar //

***Abstract*— Web Engineering (WE) methods have evolved to support different concerns during the development process of current Web-based systems, as context-awareness, Business-to-Business (B2B) process modeling, Rich Internet Applications (RIAs) and live-regions or quality factors to improve users' experience. Therefore, developers have conceptual tools to focus on these concerns in advance, but unfortunately, the situation is not the same to early accessibility design. In this paper we provide a briefly overview of our proposal, called Aspect-Oriented Web Accessibility Design (AO-WAD), and generalize its use within some of the best known WE approaches to provide accessibility support through Aspect-Orientation techniques. We embed AO-WAD into OOHDM, UWE and OOWS methods and propitiate an ease understanding through a motivating example.**

Keywords- Web Accessibility; WE Approaches; UI Design; Aspect-Orientation.

I. INTRODUCTION

Nowadays, the advance of the Internet and the emerging technologies associated to the Web are universalizing information systems, allowing access to any connected potential user. The term "Web application" [1] refers to a new family of software applications specially designed due to the high growth of commercial activities on the Internet. These systems are being implemented in very short periods of time, without support of appropriate tools. For this reason, Web applications have low quality and very difficult maintenance. In most cases, development of Web application has been "ad hoc", lacking systematic approach, quality control and assurance procedures. Therefore, there is now great concern about how Web-based systems are developed to promote integrity and quality.

Web Engineering (WE), which is still an emerging discipline, encourages a process driven by systematic approaches to develop high quality Web-based systems. In the last decade, many WE methodological approaches, as Object-Oriented Hypermedia Design Method (OOHDM) [14], UML-based Web Engineering (UWE) [4], Object-Oriented Web Solution (OOWS) [3] and Web Site Design Method (WSDM) [16], have been proposed and evolved to provide support by means of abstract mechanisms that make easier the conceptualization and development of this kind of Web applications.

In contrast, the state-of-the-art shows that there are not many proposals for the early design with accessibility principles in mind and besides, even fewer proposals, provide conceptual tools to fully support accessibility nature to migrate to other WE approaches.

In general, a proposal for including accessibility design within systematic and unified Web development works only in association with a host WE approach. Therefore, there is a high dependence between host's process and deliverables and the proposed conceptual tools to support Web accessibility. The consequences are clear, since failing the design principle "low coupling" hinders embedding and easy connection with other WE approach. For example, Plessers et al. [13] is a well-known proposal that generates annotations for visually impaired users automatically from explicit conceptual knowledge existing during the WSDM [16] design process. The proposal prioritizes accessibility support using a rule-based mapping model to drive accessibility annotations, but by means of WSDM's modeling concepts to which these annotations are tightly bound. On the other hand, Moreno et al. [11] defines several constructs in UML meta-model to support the abstraction of Web accessibility concepts following the standard WCAG [18][19]. Thus, the proposal can be easily implanted into approaches following the MDA paradigm, but at expense of not fully addressing the non-functional, generic and "crosscutting" features of accessibility.

Our proposal for accessible design, called Aspect-Oriented Web Accessibility Design (AO-WAD) [6][7][9], recommends including accessibility concerns systematically within methods for Web application development. AO-WAD is born to join OOHDM [14] prioritizing accessibility at the very beginning of the Web design process. While OOHDM provides the main development framework, Aspect-Orientation ensures handling naturally the non-functional, generic and crosscutting characteristics of the accessibility concern within the framework.

At this point, let us define what is (and what is not) Web accessibility, and why it is a good idea to model its requirements as softgoals to be "satisficed" [2]. In short, Web accessibility is the ability to access the Web. However, you will never be perfectly accessible to everybody. From this point of view and since there is not a simple binary opposition between accessible and inaccessible [15], accessibility requires more loosely defined criteria, as the one proposed in [2] for non-functional requirements.

In this paper, we introduce AO-WAD as an example of having complete commitment to accessibility through Aspect-Orientation techniques without losing generality when developing within WE approaches. Supporting this statement, we develop a motivating example within OOHDM [14], UWE [4] and OOWS [3] as host methods, which are some of the most widespread and mature WE approaches.

This document is organized into eight sections as follows: in Section II we briefly introduce AO-WAD, while in Section III we explain the way our proposal provides accessibility support to OOHDM, UWE and OOWS Web development processes. Then, in Sections IV, V and VI we apply AO-WAD to a motivating example using as hosts these three WE approaches. In Section VII we achieve some insights about including accessibility design within Web development processes applying Aspect-Orientation techniques. Finally, in Section VIII we present the conclusions and future work.

II. AO-WAD in a Nutshell

The model we envisage to deal with accessibility concerns within a WE approach is illustrated in Figure 1 [6]. Step 1 (Figure 1 (1)) manages Web application requirements looking for those that involve accessibility needs. This is because it is at the user's interface level where accessibility barriers finally show, so we are particularly interested in discovering accessibility requirements at the user interface (UI) design. Then, Step 2 (Figure 1 (2)) proposes an early capture of accessibility concrete concerns by developing two kinds of diagrams: the User Interaction Diagram (UID) with accessibility *integration points* [6] and the Softgoal Interdependency Graph (SIG) *template* [6] for Web Content Accessibility Guidelines (WCAG) 1.0. Step 3 (Figure 1 (3)) aids designers making decisions through the abstract UI model (Figure 1 (3.1)), and then, at Step 4 (Figure 1 (4)) toward its implementation through the concrete UI model (Figure 1 (4.1)). Thus, given a user's task, the SIG diagram provides the WCAG 1.0 accessibility checkpoints that "crosscut" the UI widgets (both, abstract and concrete ones; Figure 1 (3.1) and (4.1) respectively), to help to an accessible user experience.

Figure 1 (3) shows that at Step 3, our approach provides a supporting tool to assist developers in the implementation of cases, and on the creation of their corresponding models by using reusable components (for a detailed description of AO-WAD and its contribution to the area of accessible design see [6]).

In the following section, we show how AO-WAD can be implanted to work not only with OOHDM [14], but also with UML-based Web Engineering (UWE) [4] as one of the most popular and recognized Object-Oriented WE approaches.

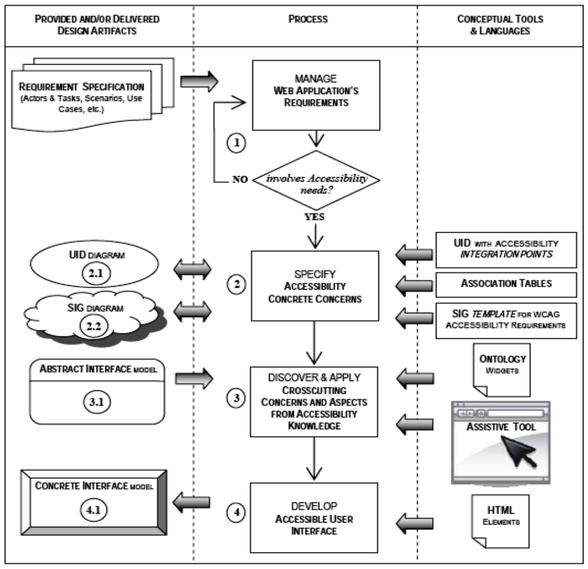

Provided and/or Delivered Design Artifacts
Process
Conceptual Tools & Languages
Requirement Specification (Actors & Tasks, Scenarios, Use Cases, etc.)
Manage Web Application's Requirements
1
NO
involves Accessibility needs?
YES
UID diagram
2.1
SIG diagram
2.2
2
Specify Accessibility Concrete Concerns
UID with accessibility integration points
Association Tables
SIG template for WCAG Accessibility Requirements
Abstract Interface model
3.1
3
Discover & Apply Crosscutting Concerns and Aspects from Accessibility Knowledge
Ontology Widgets
Assistive Tool
Concrete Interface model
4.1
4
Develop Accessible User Interface
HTML Elements

Figure 1. An overview of AO-WAD

III. Systematic Web Development and Accessibility Design

AO-WAD was developed in the spirit of model-driven paradigm to provide accessibility support within WE approaches. WE approaches are generally approaches as model-driven, because they address the different concerns involved in the development of a Web application using the following primary artifacts: (i) separate models (such as content, navigation and presentation), and (ii) model compilers to produce (semi) automated generation of most of the Web application's implementation from the original models [10]. AO-WAD focuses on preserving model-driven principles to enrich these artifacts (UI models and model transformations) with accessibility concerns. Thus, the integration of AO-WAD at the design level is immersed in a Web application development process.

In Section 2, we describe AO-WAD main process and interaction with OOHDM deliverables to model accessibility concerns in an Aspect-Oriented manner during Web developments. Figure 2 summarizes the embedding of AO-WAD within OOHDM Model-Driven Development process. The UID [17] is the conceptual tool used by OOHDM [14] to state transformations between Web application requirements (Use Case model) and the Conceptual, Navigation and UI models. AO-WAD propitiates the same principle between Web applications requirements and accessible UI models. The interaction between OOHDM models links and reinforces accessibility needs by applying two conceptual tools: the UID with *integration points* and SIG template for accessibility. The SIG diagram conveys the accessibility knowledge through WCAG 1.0 operationalizing softgoals [6] required to be applied at UI model. Due to accessibility nature, these accessibility softgoals "crosscut" the UI model more than once causing “crosscutting symptoms”. At this point, AO-WAD proposes to address these symptoms by modularizing softgoals into accessibility aspects. As Figure 2 shows, the deliverable of the process is an accessible and clean design, which means an OOHDM UI model enriched with accessibility concerns but free of “crosscutting symptoms”.

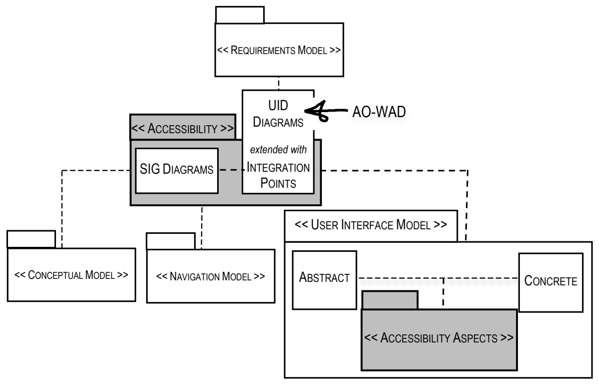


Figure 2. AO-WAD embedded into OOHDM Model-Driven Development process

As another good example of an established WE approach, UWE is based on OMG (modeling and metadata specifications) and uses UML for the analysis and design of Web applications. Figure 3 summarizes the embedding of AO-WAD within UWE Model-Driven Development process. In UWE [4], the Requirements model consists of two parts: (i) use cases of the Web application and their relationships and, (ii) activities describing use cases in detail. In particular, the Activity diagram is the conceptual tool used by UWE to describe more accurately each use case.

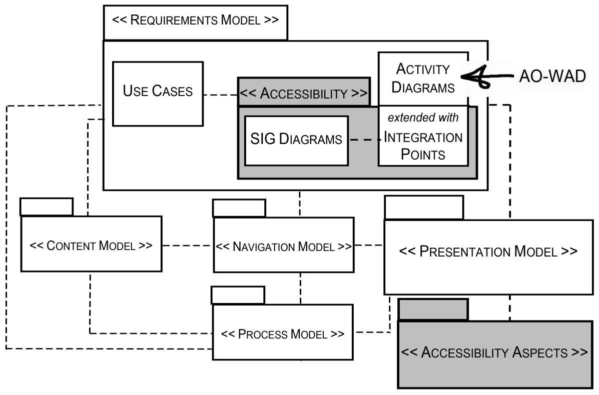


Figure 3. AO-WAD embedded into UWE Model-Driven Development process

UWE uses the Activity diagram to state transformations between Web application requirements and the Content, Navigation and Presentation models. Thus, as Figure 3 shows, AO-WAD embeds into UWE extending the Activity diagrams with integration points and through the SIG diagrams convey accessibility concerns as WCAG 1.0 operationalizing softgoals, which “crosscut” the Presentation model causing “crosscutting symptoms”. At this point and as we explained before, AO-WAD proposes to address these symptoms by modularizing softgoals into accessibility aspects. Figure 3 shows the deliverable of the process is an accessible and clean design, which means a UWE Presentation model with accessibility concerns but free of “crosscutting symptoms”.

OOWS extends an Object-Oriented software production method (called OO-Method [12]), for providing methodological support for Web application development. Figure 4 summarizes the embedding of AO-WAD within OOWS Model-Driven Development process. In OOWS [3], the Requirement model extends the OO-Method Task model to capture not only the structural and behavioral requirements (as happens in non-Web applications) but also navigational requirements using two extra diagrams: (i) a Task taxonomy, which hierarchically specifies the tasks that the users should achieve when interacting with the Web application, and (ii) a Task definition, which describes the interactions that users require from the Web application and the information that is exchanged in each interaction, using UML Activity diagrams and CRC cards [20], respectively. In particular, the OOWS Task definition model identifies and describes interaction points between the user and the Web application, which are very useful for our purpose.

As Figure 4 shows, AO-WAD can focus on this methodological support to embed into OOWS extending in first place, the Task definition model (Activity diagrams and CRC cards) with integration points and then, conveying accessibility concerns through the SIG diagram as WCAG 1.0 operationalizing softgoals. Again, AO-WAD proposes softgoals modularization into accessibility aspects to be injected into the Navigational and Presentation model.

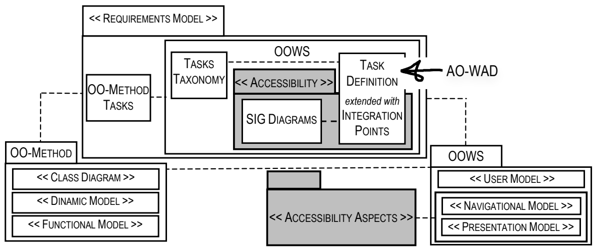


Figure 4. AO-WAD embedded into OOWS Model-Driven Development process

In order to ease understanding of AO-WAD within systematic Web development processes, we develop a motivating example in the following section, working with OOHDM, UWE and OOWS as host WE approaches.

IV. An Accessible UI for the Student’s Login

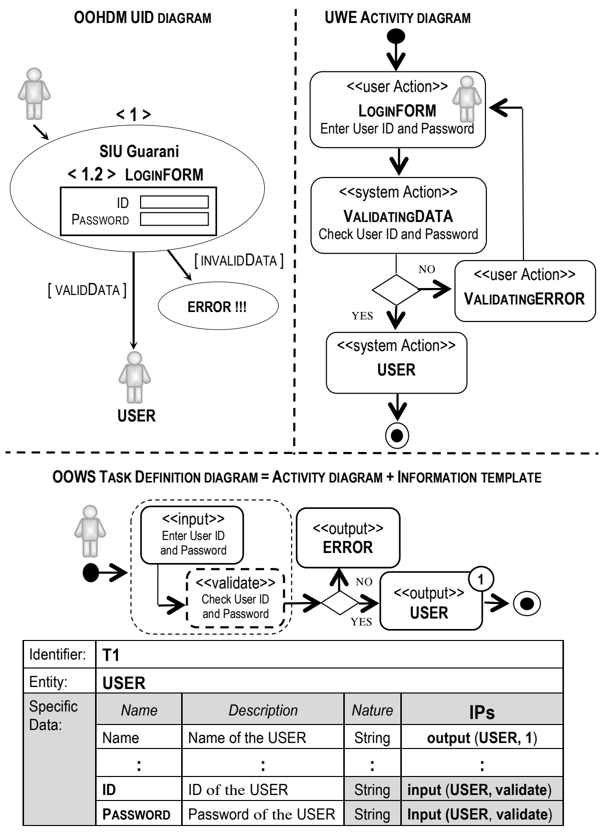
We describe the embedding of AO-WAD within OOHDM, UWE and OOWS approaches using the following use case specification “Login a User given the User’s ID and Password”:

Use Case: Login a User given the Users ID and Password
Brief Description**:** This use case describes the User login
Primary Actor: User

Description **Main Success Scenario**:	
Step	Action
1.	The system requests that the User enters his/her ID and Password.
2.	The User enters his/her ID and Password.
3.	The system validates the ID and Password and logs the User
Extensions:	
Step	Branching Action
3.a	The User enters an invalid ID and/or Password; the application displays an error message; the use case ends.

This example is simple but extremely representative mainly because of two reasons: (i) increasingly, business and government agencies are adopting a Web presence for sales and services to their customers, clients and citizens and, (ii) it clearly explains all of the issues concerning to accessible content that come into play when we think about how people with different capabilities interact with a Web page to input information [15]. The use case above describes the Web application's requirements for the user login and functionality that comprises user-application interaction; as we can see at the first step of the main success scenario, the user is requested by the application to enter his/her ID and Password. Since very often a specification based only on use cases is not enough [17], different kinds of refinement techniques are used to obtain a more detailed specification of functional requirements. OOHDM applies UID technique [17] to model user-system interactions and to specify the information that requires input from the user and choices that allow changes between interactions. UWE follows the principle of using UML whenever possible for specification and refines requirements with Activity diagrams for the main stream of the task to be performed. While OOWS, proposes the Task Taxonomy and Definition models [3] to capture Web application requirements, and in particular, the last one is the key model for specifying the interaction between the user and the Web application. Figure 5, illustrates the UID, the Activity diagram and also the Task Definition model), which provide a more detailed specification to the login use case within OOHDM, UWE and OOWS approaches, respectively. As we can see in Figure 5, an Activity diagram and an Information template implemented with the data technique CRC card, compound the OOWS Task Definition model.

As we already see in Section 2, looking at Step 1, AO-WAD proposes to examine the Web application requirements for the use case above, to identify accessibility concerns during the user-system interaction. It is clear in this specification that the FORM element is the key UI element to help achieve an accessible student's login. Following, in Sections V and VI we focus on modeling issues at Steps 2 (Figure 1 (2)) and 3 (Figure 1 (3)) respectively, as the main steps when implanting AO-WAD within WE approaches.


OOHDM UID DIAGRAM
UWE ACTIVITY DIAGRAM
< 1 >
SIU Guarani
< 1.2 > LOGINFORM
ID
PASSWORD
[INVALIDDATA]
[VALIDDATA]
ERROR !!!
USER
<<user Action>> LOGINFORM Enter User ID and Password
<<system Action>> VALIDATINGDATA Check User ID and Password
NO
YES
<<user Action>> VALIDATINGERROR
<<system Action>> USER
OOWS TASK DEFINITION DIAGRAM = ACTIVITY DIAGRAM + INFORMATION TEMPLATE
<<input>> Enter User ID and Password
<<validate>> Check User ID and Password
<<output>> ERROR
<<output>> USER
1

Identifier:	T1			
Entity:	USER			
Specific Data:	*Name*	*Description*	*Nature*	IPs
	Name	Name of the USER	String	**output** (**USER, 1**)
	:	:	:	:
	ID	ID of the USER	String	**input** (**USER, validate**)
	PASSWORD	Password of the USER	String	**Input (USER**, **validate)**

Figure 5. Requirements Specification with UID (left), Activity Diagram (right) and Task Definition Model (bottom)

V. SPECIFYING ACCESSIBILITY CONCRETE CONCERNS

When developing with OOHDM, AO-WAD proposes at Step 2.1 extending the UID diagram with integration points to support an early registration of accessibility concerns. This conceptual tool attaches an accessibility integration point to each one of those UI elements with impact on the dialog required by the use case functionality and modeled by the UID. Looking for the same modeling purpose, AO-WAD Step 2.1 can be also satisfied when developing with UWE, and OOWS extending the Requirements model of these WE approaches. When developing with UWE, the Activity diagram is enriched with accessibility *integration points*. Likewise in OOWS, the Task Definition model provides user-application interaction points (IPs) enabling AO-WAD accessibility *integration point* to be easily attached. Figure 6 illustrates the UID (left), the Activity diagram (right) and the Task Definition model (bottom), extended with an *integration point* that allows an early record of accessibility concerns for the FORM UI element -- i.e. HTML related controls. Also, Figure 6 shows two possible ways of attaching the accessibility *integration points* to these diagrams and model: (i) including a UML Note modeling construct or (ii) defining an Object Constraint Language (OCL) expression. As Figure 6 (bottom) shows, OOWS aids the Activity diagram with an

Information template, whose CRC card can be also extended to reinforce the specification of the accessibility *integration point* for the FORM UI element. As we see, the integration of Step 2.1 proposed by AO-WAD into the Requirement model of WE approaches is straightforward.

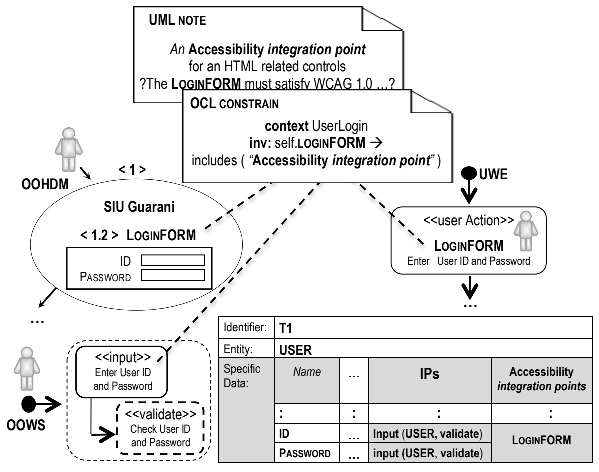


Figure 6. UID (left), Activity Diagram (right) and Task Definition Model (bottom) with Accessibility *integration points*

Then, AO-WAD proposes at Step 2.2, the specification of accessibility softgoals through a SIG tree. When developing with OOHDM, the SIG diagram is a consequence of instantiating the SIG template taking the UID with integration points as input --i.e the early registration of accessibility concerns for the FORM UI element, shown by Figure 6 (left), which is core to the required functionality. The SIG diagram specifies accessibility operationalizing softgoals to be satisfied for reaching the required WCAG 1.0 level of compliance. Applying the same modeling purpose, the Activity diagram extended with *integration points*, shown by Figure 6 (right and bottom, respectively), provides the required input for developing the SIG diagram within UWE and OOWS. Although the SIG *template* is not a UML specification tool, it can be easily transformed into an XML tree structure and work with other UML diagrams within the philosophy of the model-driven paradigm. Therefore, there are no major problems for including Step 2.2 proposed by AO-WAD during the development process of WE approaches under consideration.

VI. SOLVING ACCESSIBILITY CROSSCUTTING SYMPTOMS

AO-WAD proposes at Step 3, the specification of accessibility aspects to avoid "crosscutting symptoms" resulting from applying accessibility operationalizing softgoals to elements comprising the UI model. At the UI modeling stage, OOHDM delivers an Abstract UI model [14] whose vocabulary is established by the Abstract Widget Ontology extended by AO-WAD [6] to support new elements required by current UI, which are dynamic and with a high degree of complexity. Similarly, UWE delivers a Presentation model [4] from a Meta-model for modeling UI elements. Presentation requirements are specified in OOWS using a Presentation model that is strongly based on the Navigational model and uses the navigational contexts as basic entities to define the presentation properties; working together, these models capture the essential requirements for the construction of Web UI [3].

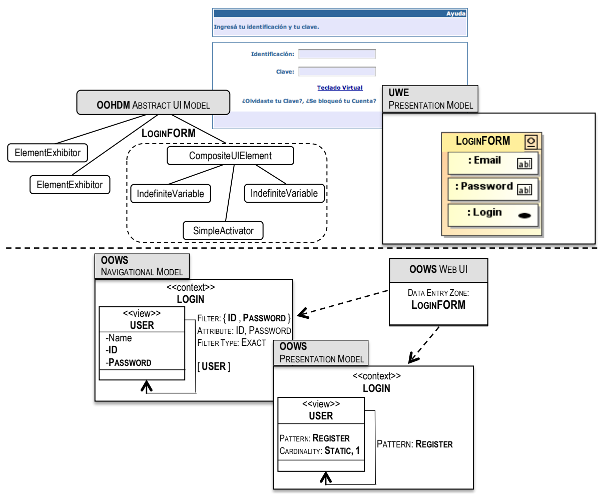


Figure 7. Abstract Interface Model in OOHDM (left), Presentation mModel in UWE (right) and Navigation-Presentation Models and Web UI in OOWS

Figure 7 shows the Abstract UI model delivered by OOHDM (left), the Presentation model provided by UWE (right), and the Navigation-Presentation models and Web UI provided by OOWS (bottom), for the screenshot (top) corresponding to the login example. In first place, AO-WAD recommends discovering "crosscutting symptoms" that manifest when applying accessibility operationalizing softgoals to the UI model --i.e. OOHDM Abstract Interface model, UWE Presentation model and OOWS Navigational-Presentation models and Web UI. These operationalizing softgoals are spread out and intermixed through the components of the login FORM UI element, causing "scattering" and "tangling" symptoms. Then, AO-WAD prescribes eliminating these symptoms through a modularization process that applies aspects to provide accessibility support at the user's technology and layout. Thus, aspects modularize operationalizing softgoals to be satisfied for properly convey the accessibility concerns required by UI elements. As Figure 8 depicts through a pseudo code, Aspect-Orientation provides a mechanism called "weaving", which requires that each aspect must specify "where or how" should be invoked and "what" should be injected into the core --i.e. a concrete UI model.

VII. DISCUSSING WE APPROACHES FROM THE ACCESSIBILITY PERSPECTIVE

We have been working for a while on accessibility [5] and particularly on accessibility design at early stages of Web applications development [6][7][8][9]. Particularly, we have been applying concepts from Aspect-Orientation in association with the WCAG 1.0 document to deal with accessibility concerns within WE approaches.

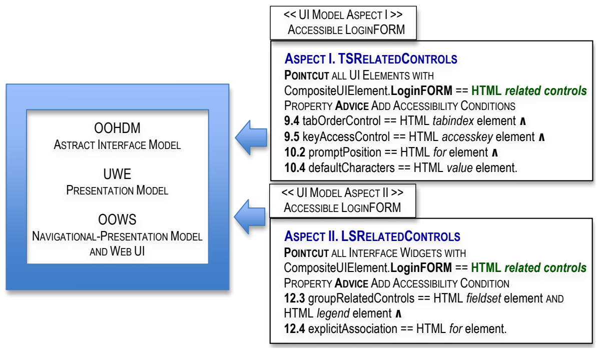

OOHDM
ASTRACT INTERFACE MODEL
UWE
PRESENTATION MODEL
OOWS
NAVIGATIONAL-PRESENTATION MODEL AND WEB UI
<< UI MODEL ASPECT I >>
ACCESSIBLE LOGINFORM
ASPECT I. TSRELATEDCONTROLS
POINTCUT ALL UI ELEMENTS WITH
CompositeUIElement.LoginFORM == HTML related controls
PROPERTY ADVICE ADD ACCESSIBILITY CONDITIONS
9.4 tabOrderControl == HTML tabindex element ∧
9.5 keyAccessControl == HTML accesskey element ∧
10.2 promptPosition == HTML for element ∧
10.4 defaultCharacters == HTML value element.
<< UI MODEL ASPECT II >>
ACCESSIBLE LOGINFORM
ASPECT II. LSRELATEDCONTROLS
POINTCUT ALL INTERFACE WIDGETS WITH
CompositeUIElement.LoginFORM == HTML related controls
PROPERTY ADVICE ADD ACCESSIBILITY CONDITION
12.3 groupRelatedControls == HTML fieldset element AND HTML legend element ∧
12.4 explicitAssociation == HTML for element.

Figure 8. Specification of Accessibility Aspects conveying Accessibility Concerns

Since the model-driven paradigm provides a good framework to develop for the Web 2.0, we believe that a proposal to somehow improve the users experience should be able to work within any WE approaches. Although AO-WAD is conceived within OOHDM to fully address accessibility features, its application can be generalized to work with other approaches, such as UWE and OOWS methods. The process proposed by AO-WAD (Figure 1) can be normalized to handle accessibility concerns through two conceptual tools: the accessibility *integration points* and SIG *template* techniques. These tools are core to AO-WAD generalization since they provide the required support to manage accessibility concerns within any WE development processes. The accessibility *integration points* technique provides early registration of accessibility concerns, while the SIG *template* technique allows instantiation for specifying concrete WCAG operationalizing softgoals to be applied [6]. These diagrams can be easily implanted into WE Requirements models, such as UIDs in OOHDM, Activity diagrams in UWE and Task Definition model in OOWS (Figures 5 and 6). Then, crosscutting symptoms are solved by the modularization of WCAG operationalizing softgoals into accessibility aspects to enrich WE Navigational and Presentation/UI models, such as Abstract UI in OOHDM, Presentation model in UWE and, Navigational-Presentation models and Web UI in OOWS (Figures 7 and 8). So, a first step in the normalization of AO-WAD for its generalization can be synthetized as follow: (1) extending requirements with accessibility *integration points*, (2) specifying the SIG diagram and, (3) modularizing WCAG operationalizing softgoals into accessibility aspects to be injected. Finally, since AO-WAD is developed to work with the model-driven paradigm, we would like to highlight advantages/disadvantages of this paradigm and how benefits/affects AO-WAD. On one hand, applying systematic and unified model-driven approaches brings the benefit of having full documentation and automatic application generation at the expense of introducing some bureaucracy into the development process. Since our proposal suggests the early treatment of the accessibility concerns through models, we may still be influenced by this reality and its disadvantages --i.e., time and cost consuming, complexity, learning effort, etc. On the other hand, using models and taking advantages of an iterative and incremental development process to deal with accessibility concerns, allows: (i) going back from UI models to Navigation models to look for alternatives in the navigation path, (ii) assessing the need and relevance of these alternatives to the functionality under develop, and (iii) going forward from Navigation models to UI models to check the accessibility of the UI related to these alternatives. Thus, the accessibility of all the alternative navigation paths that may compromise the desired functionality can be evaluated within AO-WAD.

AO-WAD supports accessible Web applications design by embedding Aspect-Orientated techniques into WE development approaches to proper address the non-functional, generic and "crosscutting" features of the accessibility nature.

VIII. CONCLUSIONS AND FUTURE WORK

The application of the model-driven paradigm to the domain of Web development has resulted in well-known WE approaches, which can be particularly useful because of the continuous evolution of Web 2.0 applications, technologies and platforms. The new generation of Web applications must offer user interfaces that enhance the experience and access to all Web users. In this context, we believe that WE approaches provide suitable models to carry with the improvements required by the application under development. In this paper we briefly introduce AO-WAD, which provides complete support to accessibility concerns by enriching WE models. Following OOHDM, UWE and OOWS processes, in this work we focus our efforts on the generalization of AO-WAD. We show that AO-WAD is flexible enough to be embedded within any WE approach, and therefore this can be a starting point that propitiates industry adoption.

As future work, we will continue working to complete the normalization of AO-WAD and validate its generalized use to systematic developing of accessible Web applications. Since, UWE and OOWS approaches provide tools for partial/full automating their design and/or implementation stages, we will analyze also the interaction of AO-WAD with these tools.

REFERENCES

[1] Baresi L., Garzotto F., and Paolini P. From Web Sites to Web Applications: New Issues for Conceptual Modeling. ER'2000 Workshop on Conceptual Modeling and the Web, LNCS 1921.Springer-Verlag, 2000.

[2] Chung, L. and Supakkul, S.: Representing FRs and NFRs: A Goal-Oriented and Use Case Driven Approach. SERA (2004) doi:10.1007/11668855_3

[3] Fons, J., Pelechena, V., Pastor, O., Valderas, P., and Torres, V. Applying the OOWS Model-Driven Approach for Developing Web Applications. The Internet Movie Database Case Study. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) WE, pp. 65-108. Springer (2008)

[4] Koch, N., Knapp, A., Zhang, G., and Baumeister, H.: UML-based Web Engineering: An Approach based on Standards. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) WE, pp. 157–191. Springer (2008)

[5] Martín, A., Cechich, A., and Rossi, G.: Comparing Approaches to Web Accessibility Assessment. In: Calero, C., Moraga, M.Á., Piattini, M. (eds.) Handbook of research on Web information systems quality, pp. 181–205. Information Science Reference, Hershey (2008)

[6] Martín, A., Rossi, G., Cechich, A., and Gordillo, S. Engineering Accessible Web Applications. An Aspect-Oriented Approach. World Wide Web Journal, 13(4), 2010, 419-440 doi:10.1007/s11280-010-0091-3

[7] Martín, A., Mazalú, R., and Cechich, A. Supporting an Aspect-Oriented Approach to Web Accessibility Design. ICSEA (2010), Francia, doi:10.1109/ICSEA.2010.10

[8] Martín A., Cechich, A., and Rossi, G. Accessibility at Early Stages: Insights from the Designer Perspective. W4A (2011), India, doi: 10.1145/1969289.1969302

[9] Mazalú, R., Huenuman, F., Martín, A., and Cechich, A. AO - WAD: A Supporting Tool to Aspect-Oriented Web Accessibility Design. ASSE (2011), Argentina.

[10] Moreno, N., Romero, J., and Vallecillo, A. An Overview of Model-Driven Web Engineering and the MDA. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modelling and Implementing Web Applications. pp. 109-155. Springer-Verlag, London (2008)

[11] Moreno, L., Martinez, P., and Ruiz, B. A MDD Approach for Modeling Web Accessibility. WOST (2008), USA, doi:10.1.1.163.9478

[12] Pastor, O., Gómez, Insfrán, E., and Pelechano, V.: The OO-Method Approach for Information Systems Modeling: From Object-Oriented Conceptual Modeling to Automated Programming. Inf. Syst. 26(7): 507-534 (2001)

[13] Plessers, P., Casteleyn, S., Yesilada, Y., De Troyer, O., Stevens, R., Harper, S., and Goble C.: Accessibility: A Web Engineering Approach. WWW (2005) doi:10.1145/1060745.1060799

[14] Rossi, G. and Schwabe, D.: Modeling and Implementing Web Applicactions with OOHDM. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) WE, pp. 109–155. Springer (2008)

[15] Thatcher, J., Burks, M., Heilmann, Ch., Henry, S., Kirpatrick, A., Lauke, P., Lawson, B., Regan, B., Rutter, R., Urban, M., and Waddell, C.: Web Accessibility - Web Standards and Regulatory Compliance. Friendsof ED, USA (2006)

[16] De Troyer, O., Casteleyn, S., and Plessers, P.: WSDM: Web Semantics Design Method. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) WE, pp. 303–351. Springer (2008)

[17] Vilain, P., Schwabe, D., and Sieckenius de Souza, C.: A Diagrammatic Tool for Representing User Interaction in UML. UML (2000) doi:10.1007/3-540-40011-7_10

[18] W3C: Web Content Accessibility Guidelines 1.0. (WCAG 1.0). http://accessibility.w3.org/TR/WAI-WEBCONTENT/ (1999). Accessed 15 April 2009

[19] W3C: Web Content Accessibility Guidelines 2.0 (WCAG 2.0). http://accessibility.w3.org/TR/WCAG20/ (2008). Accessed 25 January 2010.

[20] Wirfs-Brock, R., Wilkerson, B., and Wiener, L. Designing Object–Oriented Software. Prentice–Hall, 1990.

An Evaluation Framework for Requirements Elicitation in Agile Methods

Waleed Helmy
Faculty of Computers & Information
Cairo University
Cairo, Egypt

w.helmy@fci-cu.edu.eg

Amr Kamel
Faculty of Computers & Information
Cairo University
Cairo, Egypt

a.kamel@fci-cu.edu.eg

Osman Hegazy
Faculty of Computers & Information
Cairo University
Cairo, Egypt

o.hegazy@fci-cu.edu.eg

***Abstract*—Gathering, understanding and managing requirements is a key factor to the success of a software development effort. There are several requirement techniques available for requirement gathering which can be used with agile development methods. These techniques concentrate on a continuous interaction with the customer to address the evolution of requirements, changing requirements, prioritizing requirements and delivering the most important functionalities first. However, problems have been reported with the use of the agile methods in the area of requirements elicitation particularly with an over reliance on a customer and lack of elicitation guidelines. This paper describe how requirements elicitation is usually done in more conventional software development processes and makes an evaluation framework for the way requirements elicitation can be done in the agile methods and this could result in improvements to agile approaches.**

Keywords-Agile Methods; Requirements Elicitation; Agile Requirements Elicitation.

I. INTRODUCTION

Agile software development approaches have become more popular during the last few years. Several methods have been developed with the aim to be able to deliver software faster and to ensure that the software meets customer changing needs. All these approaches share some common principles: Improved customer satisfaction, adapting to changing requirements, frequently delivering working software, and close collaboration of business people and developers [4, 9, 13].

Requirements engineering (RE), on the other hand, is a software engineering process with the goal to identify, analyze, document and validate requirements for the system to be developed [14]. Often, requirements engineering and agile approaches are seen being incompatible: RE is often heavily relying on documentation for knowledge sharing while agile methods are focusing on face-to-face collaboration between customers and developers to reach similar goals [1].

This paper aims to discuss how requirements elicitation techniques can be used within agile development context. Several studies addressed the requirements elicitation in agile methods. In [15], a new method for automatically retrieving functional requirements from the stakeholders using agile processes is presented. The presented method is a machine learning system for the automation of some aspects of the software requirements phase in the software engineering process. This learning system encompasses knowledge acquisition and belief revision in a knowledge base. The aim of the algorithm is to collect information from the various stakeholders and integrate a variety of learning methods in the knowledge acquisition process, while involving certain and plausible reasoning.

The goal oriented requirements engineering method proposed in [11] identifies the requirements in terms of goals which are well understood by the stakeholders and the goals are generally extracted from the stakeholders. While extracting the goals, the high level goals are decomposed/refined/broken to get the lower level goals/sub-goals involving active participation of stakeholders through the process of goal decomposition/refinement/splitting involving Agents.

Since micro-businesses have restrictions with their budget, manpower, and technical exposure to software, some trade-offs must be addressed. A novel approach in [16] demonstrated how several models and techniques such as goals, business process models, patterns, and non-functional requirements, have helped in defining the software requirements of the micro-business.

However, problems have been reported with the use of the agile methods in the area of requirements elicitation particularly with an over reliance on a customer and lack of elicitation guidelines. This paper describe how requirements elicitation is usually done in more conventional software development processes and makes an evaluation framework for the way requirements elicitation can be done in the agile methods and this could result in improvements to agile approaches.

The next section gives an overview on what the requirements elicitation is and what its techniques are. Section III states the agile manifesto. Section IV discusses agile methods from the requirements elicitation perspective. Section V summarizes the agile requirements engineering. In Section VI, we provide an evaluation framework for requirements elicitation in agile methods. Section VII summarizes the requirements elicitation issues in agile methods. The last section presents the conclusion.

II. REQUIREMENTS ELICITATION

Requirements engineering is concerned with identifying, modeling, communicating and documenting the requirements for a system, and the contexts in which the system will be used. Requirements describe what is to be done but not how they are implemented [6]. There are many techniques available for use during the RE process to ensure that the requirements are complete, consistent and relevant. The fundamental principle underlying requirements engineering is that a system should be clearly specified before its design and implementation [11]. So, the aim of RE is to help to know what to build *before* system development starts in order to prevent costly rework. This goal is based on two major assumptions:

- The later mistakes are discovered the more expensive it will be to correct them [3].
- It is possible to determine a stable set of requirements before system design and implementation starts.

The RE process consists of five main activities [2]: Elicitation, Analysis and Negotiation, Documentation, Validation, and Management.

Requirements elicitation tries to discover requirements and identify system boundaries by consulting stakeholders (e.g., clients, developers, users). System boundaries define the context of the system. Understanding the application domain, business needs, system constraints, stakeholders and the problem itself is essential to gain an understanding of the system to be developed.

The most important techniques for requirements elicitation are described in the remainder of this section.

Interviews: Interviewing is a method for discovering facts and opinions held by potential users and other stakeholders of the system under development. Mistakes and misunderstandings can be identified and cleared up. There are two different kinds of interviews:

- The closed interview, where the requirements engineer has a pre-defined set of questions and is looking for answers
- The open interview, without any pre-defined questions the requirements engineer and stakeholders discuss in an open-ended way what they expect from a system.

In fact, there is no distinct boundary between both kinds of interviews. You start with some questions which are discussed and lead to new questions [8]. The advantage of interviews is that they help the developer to get a rich collection of information. Their disadvantage is that this amount of qualitative data can be hard to analyze and different stakeholders may provide conflicting information.

Observation and Social Analysis: Observational methods involve an investigator viewing users as they work and taking notes on the activity that takes place. Observation may be either direct with the investigator being present during the task, or indirect, where the task is viewed by some other means (e.g. recorded video). It is useful for studying currently executed tasks and processes. Observation allows the observer to view what users actually do in context. This overcomes issues with stakeholders describing idealized or oversimplified work processes.

Focus Groups: Focus groups are an informal technique where a small group of users from different backgrounds and with different skills discuss in a free form issues and concerns about features of a system prototype. Focus groups help to identify user needs and perceptions, what things are important to them and what they want from the system. They often bring out spontaneous reactions and ideas. Since there is often a major difference between what people says and what they do, observations should complement focus groups.

Focus groups can support the articulation of visions, design proposals and a product concept. Additionally, they help users in analyzing things that should be changed, and support the development of a ’shared meaning’ of the system [7].

Brainstorming: Brainstorming helps to develop creative solutions for specific problems. Brainstorming contains two phases - the generation phase, where ideas are collected, and the evaluation phase, where the collected ideas are discussed. In the generation phase, ideas should not be criticized or evaluated. The ideas should be developed fast and be broad. Brainstorming leads to a better problem understanding and a feeling of common ownership of the result.

Prototyping: A prototype of a system is an initial version of the system which is available early in the development process. Prototypes of software systems are often used to help elicit and validate system requirements. There are two different types of prototypes. Throw-away prototypes help to understand difficult requirements. Evolutionary prototypes deliver a workable system to the customer and often become a part of the final system. Prototypes can be paper based (where a mock-up of the system is developed on paper), ”Wizard of Oz” prototypes (where a person simulates the responses of the system in response to some user inputs) or automated prototypes (where a rapid development environment is used to develop an executable prototype).

III. AGILE MANIFESTO

The Agile manifesto as the Agile Manifesto official site states is as follows "*We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

- ***Individuals and interactions*** *over processes and tools.*
- ***Working software*** *over comprehensive documentation.*
- ***Customer collaboration*** *over contract negotiation.*
- ***Responding to change*** *over following a plan.*

That is while there is value in the items on the right; we value the items in the left more." [9].

IV. AGILE METHODS

A. Extreme Programming

XP uses story cards for elicitation [1]. A user story is a description of a feature that provides business value to the customer. Use cases, on the other hand, are a description of interactions of the system and its users and do not mandatory have to provide business value.

Before story cards can be written, customers have to think about what they expect the system to do. This process can be seen as brainstorming. Thinking about a specific functionality leads to more ideas and to more user stories. Every story is discussed in an open-ended way before implementation. Initially, developers ask for enough details to be able to estimate the effort for implementing the story. Based on these estimates and the time available, customers prioritize stories to be addressed in the next iteration. XP emphasizes writing tests before coding. Acceptance tests are defined by the customer and are used to validate the completion of a story card. XP is based on frequent small releases. This can be compared with requirements review and with evolutionary prototyping.

B. Scrum

The main Scrum techniques are the product backlog, sprints, and daily scrums [1]. With regard to Requirements Engineering the product backlog plays a special role in Scrum. All requirements regarded as necessary or useful for the product are listed in the product backlog. It contains a prioritized list of all features, functions, enhancements, and bugs. The product backlog can be compared with an incomplete and changing (a kind of “living”) requirements document containing information needed for development. For each sprint (= 30 day development iteration), the highest priority tasks from the backlog are moved to the sprint backlog. No changes are allowed to the sprint backlog during the sprint. I.e. there is no flexibility in the requirements to be fulfilled during a sprint but there is absolute flexibility for the customer reprioritizing the requirements for the next sprint. At the end of a Sprint a sprint review meeting is held that demonstrates the new functionality to the customer and solicits feedback.

C. Feature Driven Development

Feature Driven Development (FDD) is a short iteration process for software development focusing on the design and building phase instead of covering the entire software development process [4]. In the first phase, the overall domain model is developed by domain experts and developers. The overall model consists of class diagrams with classes, relationships, methods, and attributes. The methods express functionality and are the base for building a feature list. A feature in FDD is a client-valued function. The items of the feature list are prioritized by the team. The feature list is reviewed by domain members [5]. FDD proposes a weekly 30-minute meeting in which the status of the features is discussed and a report about the meeting is written. Reporting can roughly be compared with requirements tracking.

D. Agile Modeling

The basic idea of AM [2] is to give developers a guideline of how to build models that help to resolve design problems but not ’over-build’ these models. Like XP, AM points out that changes are normal in software development. AM does not explicitly refer to any RE techniques but some of the practices support several RE techniques (e.g. tests and brainstorming). AM highlights the difference between informal models whose sole purpose is to support face-to-face communication and models that are preserved and maintained as part of the system documentation. The later are what is often found in RE approaches.

V. AGILE REQUIREMENTS ENGINEERING

The agile principles applied to software engineering include *iterative and incremental development, frequent releases of software, direct customer involvement, minimal documentation and welcome changing requirements even late in the development cycle [6]*.

Conventional RE processes focus on gathering all the requirements and preparing the requirements specification document up front before proceeding to the design phase. These up front requirements gathering and specification efforts leave no room to accommodate changing requirements late in the development cycle. On the other hand,

On the other hand, agile requirement engineering [12] aims at applying agile thoughts to traditional requirement engineering. It is the optimization and improvement of traditional requirement engineering, getting it fit to the continuous changes of requirements.
Agile RE welcomes changing requirements even late in the development cycle [3]. This is achieved by using the agile practice of *Evolutionary Requirements* which suggests that requirements evolve over the course of many iterations rather than being gathered and specified up front. Hence, changes to requirements even late in the development cycle can be accommodated easily.

Initially, the high level features for the system are defined where features indicate the expected functionality. All the features have to be identified upfront in order to determine the scope of the system. These features describe the expected functionality of business value to the customers. The development period spans multiple release cycles. Only one feature or a subset of the identified features is considered for development during a release cycle. Then, the requirements for each feature are gathered just-in-time (JIT) from the customers before the development of that feature. As only a subset of the identified features is implemented during a release cycle, only details of this subset of features are gathered from the customers. Customer are actively involved in the Agile RE process. Usually, a customer is available onsite to provide details of the features to the development team. Direct customer involvement facilitates the adoption of the JIT philosophy.

Agile RE accommodates rapidly changing requirements. Changes to requirements identified are logged and are implemented in the following iterations. As only a subset of features is implemented during a release cycle, changes to these features do not affect the other features that are yet to be built.

Agile RE focuses on minimal documentation. No formal Requirements Specification is produced. The features and the requirements are recorded on story boards and index cards. The artifacts produced depend on the project. Some Agile RE artifacts are paper prototypes, use case diagrams and data flow diagrams. However, if the client requires formal documentation to be produced, the development team strives to produce the same.

Verification and Validation (V&V) of requirements in agile RE is more of a validation process. The agreed standards used for verification are usually stated in the form of user stories and hence, V&V is more of a validation process. Validation is not explicit and is carried out just-in-time. There is no specification of explicit validation activities in agile RE. As the customer is usually available onsite, the features/ requirements can be validated as and when required. Some Agile RE practices are listed below:

Evolutionary Requirements – The requirements are allowed to evolve over time. All the requirements are not identified upfront. This practice is called No Big Requirements Up Front (BRUF).

Incremental and iterative implementation of requirements – Agile RE suggests incremental development of software. The development period is divided into release cycles and each release cycle spans multiple iterations. Hence, the requirements are implemented in an iterative and incremental fashion.

Accommodate change late in the development life cycle – The main objective of Agile RE is to accommodate changing requirements even late in the development cycle. Usually changes identified to features are logged and incorporated during the future iterations.

Minimal requirements documentation – Documentation is usually in the form of features or stories recorded on index cards. No formal requirements specification documents are produced. However, when employing third party organizations for performing maintenance activities, minimal documentation is a disadvantage.

Gather details just-in-time– The development team defers gathering details till the latest responsible moment. Only the details of the features to be implemented during a release cycle are gathered. Adopting JIT philosophy helps accommodate changing requirements.

Implicit Verification and Validation (V&V) – As mentioned earlier, V&V is more of a validation process. Validation is not carried out explicitly.

Treat requirements like prioritized stack – Agile methods specify that the requirements should be considered similar to a prioritized stack. The features are prioritized by the customers based on their business value. These prioritized features are stored in a stack and ordered by their priorities.

Adopt user terminology – The features and requirements are recorded in the domain language of the user. This is done in order to help users understand the captured needs and requirements.

Direct customer involvement – Agile RE mandates the involvement of customers at every stage of the development process. As customers are involved throughout, the developers can gather details about the features just-in-time.

VI. AN EVALUATION FRAMEWORK

The table below is an evaluation framework for the requirements elicitation techniques in agile methods. The framework compares four agile methods with respect to requirements elicitation.

In the previous sections, we gave an overview on requirements elicitation techniques as well as on agile methods. Here, we now analyze potential synergies between these approaches.

TABLE 1: AN EVALUATION FRAMEWORK FOR REQUIREMENTS ELICITATION IN AGILE METHODS

Method / **RE Practice**	**XP**	**Scrum**	**Agile Modeling**	**FDD**
Req. Elicitation	*User Stories* Interview, Brainstorming	Product Backlog Interview	X Brainstorming	Feature List Interview

Customer involvement: The CHAOS report [7] showed the critical importance of customer involvement. Customer involvement was found to be the number one reason for project success, while the lack of user involvement was the main reason given for projects that ran into difficulties. A key point in all agile approaches is to have the customer 'accessible' or 'on-site'. Thus, traditional RE and agile methods agree on the importance of stakeholder involvement

Agile methods often assume an "ideal" customer representative: the representative can answer all developer questions correctly, she is empowered to make binding decisions and able to make the right decisions. Even if the requirements are elicited in group sessions (Scrum) it is not guaranteed that users or customers with all necessary backgrounds are present. On the other hand, RE has a less idealized picture of stakeholder involvement. The different elicitation techniques aim to get as much knowledge as possible from all stakeholders and resolve inconsistencies. In addition, RE uses externalization and reviews to ensure that all requirements are known and conflicting requirements are in the open

Another difference between traditional approaches and agile methods is that in traditional approaches the customer is mainly involved during the early phase of the project while agile methods involve the customer throughout the whole development process.

Feature: In agile development, a feature is a chunk of functionality that delivers business value. Features can include additions or changes to existing functionality. A feature should adhere to the following criteria:

- It should provide business value
- It should be estimable - it must have enough definition for the development team to provide an estimate of the work involved in implementing it
- It should be small enough to fit within an iteration - therefore, if it is too big, it should be broken down further
- It should be testable - you should understand what automated or manual test a feature should pass in order to be acceptable to the customer

The different agile methods use different terminology to refer to features. It is up to the team to decide which methodology or terminology to use. Extreme Programming uses the terms User Stories or Stories to represent features; Scrum uses Product Backlog to describe a feature list; Feature-Driven Development uses Feature. Ultimately, the goal is the same - to deliver business value regularly in small increments, and sooner rather than later.

Product Backlog, Features, User Stories: The product backlog is an ordered list of "requirements" that is maintained for a product. It contains product backlog Items that are ordered based on considerations like risk, business value, dependencies, date needed, etc. The features added to the backlog are commonly written in story format. The product backlog is the “What” that will be built, sorted in the relative order it should be built in. The product backlog contains rough estimates of business value and development effort, these values are often stated in story points.

Interviews As customer involvement is a primary goal of agile software development, the most common RE-related technique are interviews. Interviews provide direct and “unfiltered” access to the needed knowledge. It is known that chains of knowledge transfer lead to misunderstandings. All agile approaches emphasize that talking to the customer is the best way to get information needed for development and to avoid misunderstandings. If anything is not clear or only vaguely defined, team members should talk to the responsible person and avoid chains of knowledge transfer. Direct interaction also helps establishing trust relationships between customers and developers.

Brainstorming: This technique is not explicitly mentioned in any agile software development method but can be used with any approach.

VII. REQUIREMENTS ELICITATION ISSUES IN AGILE METHODS

The agile requirements elicitation approach toward requirements usually results in several architecture-related issues that can potentially have negative impact on architectural practices, artifacts or design decisions [10]. Following paragraphs describe the most commonly observed requirements elicitation issues when using agile approaches are:

Lack of focus on Non Functional Requirements: In agile approaches handling of non-functional requirements is ill defined [1]. Customers or users talking about what they want the system to do normally do not think about maintainability, portability, safety or performance. Some requirements concerning user interface or safety can be elicited during the development process and still be integrated. But most non-functional requirements should be known in development because they can affect the choice of database, programming language or operating system. Agile methods need to include more explicitly the handling of non-functional requirements in a way they can be analyzed before implementation.

Incomplete Requirements Elicitation: The “user stories” or the like are just the beginning points of both the requirements gathering and development processes in agile methods. Early requirements are simply a place to start. It is expected to add more requirements as more is known about the product. This attitude toward requirements makes software architecture development more difficult. The architecture that chosen by the team during the early cycles may become wrong, as later requirements becomes known [8].

VIII. CONCLUSION

This paper presented an evaluation framework of how the requirements are elicited in four common agile methods: XP, scrum, agile modeling, and FDD.

XP uses story cards for requirements elicitation through interviews and brainstorming techniques. In Scrum, all requirements regarded as necessary or useful for the product are listed in the product backlog. It contains a prioritized list of all features, functions, enhancements, and bugs. The requirements are elicited from users through interviews. In FDD, the overall domain model is developed that consists of class diagrams with classes, relationships, methods, and attributes. The methods express functionality and are the base for building a feature list. The feature list is elicited through interviews. AM does not explicitly refer to any RE techniques but some of the practices support several RE techniques (e.g. tests and brainstorming).

The agile requirements elicitation approach toward requirements usually results in several architecture-related issues that can potentially have negative impact on architectural practices, artifacts or design decisions [10]. The “user stories” or the like are just the beginning points of both the requirements gathering and development processes in agile methods. Early requirements are simply a place to start. It is expected to add more requirements as more is known about the product. Agile methods, however, have a lack of focus on certain parts of what is considered as important in requirements engineering. The customers don't usually cover non-functional requirements when they define requirements.

REFERENCES

[1] Eberlein, A., Maurer, F., and Paetsch, F., "Requirements Engineering and Agile Software Development", Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE CS Press, pp. 308-313, 2003.

[2] Scott, W., "Agile Modeling", John Wiley & Sons, 2001.

[3] Scott, W., "Agile Requirements Modeling", http://www.agilemodeling.com/essays/agileRequirements.htm , retrieved: October, 2012.

[4] Pekka, A., Outi, S., Jussi, R., and Juhani, W., "Agile software development methods - Review and analysis", VTT Publications, No. 478, 2002.

[5] Peter, C., Eric, L., and Jeff, L., "Java Modeling in Color with UML", Prentice Hall PTR, Chapter 6, 1999.

[6] Soundararajan, S." Agile Requirements Generation Model: A Soft-structured Approach to Agile Requirements Engineering". Master Thesis. Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, VA. 2008.

[7] Standish Group: Chaos Report, http:// www.standishgroup.com, retrieved: October, 2012.

[8] Tomayko, J., "Engineering of Unstable Requirements Using Agile Methods", International Conference on Time-Constrained Requirements Engineering, Essen, Germany, 2002.

[9] Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D., "Manifesto for Agile software Development", http://www.agilemanifesto.org/, retrieved: October, 2012.

[10] Babar, M., "An exploratory study of architectural practices and challenges in using agile software development approaches", http://ulir.ul.ie/handle/10344/1127, retrieved: October 2012.

[11] Sen, M. and Hemachandran, K., "Elicitation of Goals in Requirements Engineering Using Agile Methods", Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops, pp. 263-268, 2010.

[12] Jun, L., Qiuzhen, W., and Lin, G., "Application of Agile Requirement Engineering in Modest-Sized Information Systems Development", Proceedings of the 2010 Second World Congress on Software Engineering - Volume 01 Pages 207-210, 2010.

[13] Poppendieck, T. and Poppendieck, M., "Lean software development: An agile toolkit for software development managers", Addison-Wesley, London UK, 2003.

[14] Kotonya, G. and Sommerville, I., "Requirements Engineering Processes and Techniques", John Wiley & Sons, Chichester, UK, 2002.

[15] Ankori, R., "Automatic Requirements Elicitation in Agile Processes", the IEEE International Conference on Software - Science, Technology & Engineering, pp. 101-109, 2005.

[16] Macaseat, R., Chung, L., Garrido, J., Noguera, M., and Luisa, M.," An agile requirements elicitation approach based on NFRs and business process models for micro-businesses, 12th International Conference on Product Focused Software Development and Process Improvement, pp. 50-56, 2011.

An Evaluation Framework for Requirements Envisioning in Agile Methods

Waleed Helmy
Faculty of Computers & Information.
Cairo University
Cairo, Egypt

w.helmy@fci-cu.edu.eg

Amr Kamel
Faculty of Computers & Information.
Cairo University
Cairo, Egypt

a.kamel@fci-cu.edu.eg

Osman Hegazy
Faculty of Computers & Information.
Cairo University
Cairo, Egypt

o.hegazy@fci-cu.edu.eg

***Abstract*—A common agile practice is to perform some high-level requirements envisioning early in the project to gather and document business requirements during the initial phase of the project. The goals of requirements envisioning are to develop a common vision, identify the business goals, and identify the initial requirements for the system at a high-level. This paper presents an evaluation framework for the way the requirements envisioning can be done in the agile methods.**

Keywords-Requirements Envisioning; Agile Requirements Envisioning, Envisioning in Agile Methods.

I. INTRODUCTION

Information system development methodologies refer to a standard process followed in an organization to conduct all the steps necessary to analyze, design, implement, and maintain information systems [2]. They are developed to assure that software systems met established requirements. There are a number of methodologies used to develop and improve the systems such as the traditional waterfall, incremental development, prototyping, and spiral [13]. These methodologies impose a disciplined process upon software development with the aim of making software development more predictable and more efficient. They do this by developing a detailed process with a strong emphasis on planning aspired by other engineering methodologies. However, these traditional systems development methodologies sometimes fall short in the new business environment [14]. They are too "heavy" to keep up with the pace of new business software development projects.

In response to the problems of the traditional methodologies, the agile methodology has evolved in the mid-1990s. Highsmith and Cockburn in [12] wrote that "*what are new about agile methods is not the practices they use, but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and maneuverability. This yields a new combination of values and principles that define an agile world view*".

While Nerur and Balijepally in [3] defined agile methodologies as people-centric, that recognize the value of skilled people and their relationships bring to software development. They also explained in their paper that Agile methods focus on providing high customer satisfaction through three principles: quick delivery of quality software; active participation of concerned stakeholders; and creating and acting effectively toward changes. They added that big upfront designs, plans and extensive documentation are of little value to practitioners of agile methods. In [4] agile is defined as a software development method that is people-focused, communication-oriented, flexible (ready to adapt to expected change at any time), speedy (encourage rapid and iterative development of the product in small releases), lean (focuses on shortening timeframe and cost and on improved quality), responsive (reacts appropriately to expected and unexpected changes), and learning (focuses on improvement during and after product development).

However, this paper focuses only on the agile methods and the way the requirements envisioning can be done in the agile methods. A common agile practice is to perform some high-level requirements envisioning early in the project to help come to a common understanding as to the scope of what you're trying to accomplish. The goals at this point are to identify the business goals for the effort, develop a common vision, and quickly identify the initial requirements for the system at a high-level. This initial requirements envisioning effort is on the order of hours or days, not weeks or months, as we see on traditional projects.

The next section presents the agile software development life cycle. Section III gives an overview on agile requirements envisioning. Section IV discusses four agile methods namely XP, Scrum, Feature Driven Development, and Agile modeling. Section V presents an evaluation framework for requirements envisioning in four agile methods. The last section presents the research conclusion.

II. AGILE SDLC

Agile software development life cycle is comprised of six phases: Iteration -1/Pre-Planning, Iteration 0/Warm Up, Construction, Release/End Game, Production, and Retirement [11]. Here is a description of each phase.

A. Iteration -1: Pre-Project Planning

This phase includes the following activities:

- Define the business opportunities
- Identify a viable for the project
- Assess the visibility

B. Iteration 0/Warm Up: Project Initiation

The first week or so of an agile project is often referred to as “Iteration 0” (or "Cycle 0"). The goal during this period is to initiate the project by:

- Garnering initial support and funding for the project.
- Starting to build the team.
- Setting up the environment.

C. Estimating the project: Construction Iterations

During construction iterations agilists incrementally deliver high-quality working software which meets the changing needs of our stakeholders. This can be achieved by:

- Collaborating closely with both our stakeholders and with other developers
- Implementing functionality in priority order
- Analyzing and designing
- Ensuring quality
- Regularly delivering working software
- Testing, testing, and yes, testing.

D. Release Iterations(s): The "End Game"

During the release iteration(s), also known as the "end game", we move the system into production. There are several important aspects to this effort:

- Final testing of the system.
- Rework. There is no value testing the system if you don't plan to act on the defects that you find. You may not address all defects, but you should expect to fix some of them.
- Finalization of any system and user documentation.
- Training. We train end users, operations staff, and support staff to work effectively with our system
- Deploy the system

E. Production

The goal of the Production Phase is to keep systems useful and productive after they have been deployed to the user community. This process will differ from organization to organization and perhaps even from system to system, but the fundamental goal remains the same: keep the system running and help users to use it.

F. Retirement

The goal of the Retirement Phase is the removal of a system release from production, and occasionally even the complete system itself, an activity also known as system decommissioning. Retirement of systems is a serious issue faced by many organizations today as legacy systems are removed and replaced by new systems. You must strive to complete this effort with minimal impact to business operations. If you have tried this in the past, you know how complex it can be to execute successfully.

III. AGILE REQUIREMENTS ENVISIONING

Agile requirements activities are evolutionary (iterative and incremental) and highly collaborative in nature. Initially, requirements are explored at a high level via requirements envisioning at the beginning of the project and the details are explored on a just-in-time (JIT) basis via iteration modeling and model storming activities. The strategy is to take advantage of modeling, which is to communicate and think things through without taking on the risks associated with detailed specifications written early in the lifecycle, a traditional practice referred to as "Big Requirements Up Front" [1]. For the first release of a system you need to take several days, with a maximum of two weeks for the vast majority of business systems, for initial requirements and architecture envisioning. There are several models to envision the requirements which are:

High-level use cases (or user stories) - The most detail that we would capture would be point form notes for some of the more complex use cases, but the majority just might have a name. The details are best captured on a just-in-time (JIT) basis during construction.

User interface flow diagram - This provides an overview of screens and reports and how they're inter-related.

User interface sketches - Sketch out a few of the critical screens and reports to give your stakeholders a good gut feeling that you understand what they need.

Domain model - A high-level domain model shows major business entities and the relationships between them. Listing responsibilities, both data attributes and behaviors, can be left until later iterations.

Process diagrams - A high-level process diagram that shows some of the critical processes, are likely needed to understand the business flow.

Use-case diagram - Instead of a high-level process diagram you might want to do a high-level use case diagram instead.

IV. AGILE METHODS

Agile methods, generally, promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices that allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals [9].

A. Extreme Programming

XP uses story cards for elicitation [5]. A user story is a description of a feature that provides business value to the customer. Use cases, on the other hand, are a description of interactions of the system and its users and do not mandatory have to provide business value.

Before story cards can be written, customers have to think about what they expect the system to do. This process can be seen as brainstorming. Thinking about a specific functionality leads to more ideas and to more user stories.

Every story is discussed in an open-ended way before implementation. Initially, developers ask for enough details to be able to estimate the effort for implementing the story. Based on these estimates and the time available, customers prioritize stories to be addressed in the next iteration. XP emphasizes writing tests before coding. Acceptance tests are defined by the customer and are used to validate the completion of a story card. XP is based on frequent small releases. This can be compared with requirements review and with evolutionary prototyping.

B. Scrum

The main Scrum techniques are the product backlog, sprints, and daily scrums [5]. With regard to Requirements Engineering the product backlog plays a special role in Scrum. All requirements regarded as necessary or useful for the product are listed in the product backlog. It contains a prioritized list of all features, functions, enhancements, and bugs. The product backlog can be compared with an incomplete and changing (a kind of "living") requirements document containing information needed for development. For each sprint (= 30 day development iteration), the highest priority tasks from the backlog are moved to the sprint backlog. No changes are allowed to the sprint backlog during the sprint. I.e. there is no flexibility in the requirements to be fulfilled during a sprint but there is absolute flexibility for the customer reprioritizing the requirements for the next sprint. At the end of a sprint, a potentially shippable product is delivered and a sprint review meeting is held that demonstrates the new functionality to the customer and solicits feedback [10]

C. Feature Driven Development

As the name implies, features are an important aspect of Feature Driven Development (FDD). A feature is a small, client-valued function. Features are to FDD as use cases are to the Rational Unified Process (RUP) and user stories are to XP – they're a primary source of requirements and the primary input into your planning efforts.

FDD is a short iteration process for software development focusing on the design and building phase instead of covering the entire software development process [6]. In the first phase, the overall domain model is developed by domain experts and developers. The overall model consists of class diagrams with classes, relationships, methods, and attributes. The methods express functionality and are the base for building a feature list. A feature in FDD is a client-valued function. The items of the feature list are prioritized by the team. The feature list is reviewed by domain members [7]. FDD proposes a weekly 30-minute meeting in which the status of the features is discussed and a report about the meeting is written. Reporting can roughly be compared with requirements tracking.

D. Agile Modeling

The basic idea of Agile Modeling (AM) is to give developers a guideline of how to build models that help to resolve design problems but not 'over-build' these models [8]. Like XP, AM points out that changes are normal in software development. AM does not explicitly refer to any RE techniques but some of the practices support several RE techniques (e.g., tests and brainstorming). AM highlights the difference between informal models whose sole purpose is to support face-to-face communication and models that are preserved and maintained as part of the system documentation. The later are what is often found in RE approaches.

V. AN EVALUATION FRAMEWORK

Table 1 presents an evaluation framework for requirements envisioning in agile methods. The framework compares four agile methods: Scrum, extreme Programming, Feature Driven Development, and Agile Modeling Driven Development with respect to three evaluation criteria: Activities, Participants, and Time Frame.

TABLE 1: AN EVALUATION FRAMEWORK FOR REQUIREMENTS ENVISIONING IN AGILE METHODS

Criteria / Agile Method	Activities	Participants	Time Frame
XP	Initial Requirements Modeling, Initial Architecture Modeling	All Team Members	Hours or days
Scrum	X	All Team Members	Hours or days
FDD	Build an Object Model	All Team Members	Hours Or days
AMDD	Initial Requirements Envisioning, Initial Architecture Envisioning	All Team Members	Hours Or days

XP encompasses the initial requirements modeling and initial architectural modeling aspects of the agile software development lifecycle. This phase includes development of the architectural spike and the development of the initial user stories. From a requirements point of view it suggests that you require enough material in the user stories to make a first good release and the developers should be sufficiently confident that they can't estimate any better without actually implementing the system. Every project has a scope, something that is typically based on a collection of initial requirements for your system. Although the XP lifecycle does not explicitly include a specific scope definition task it implies one with user stories being an input into release planning. User stories are a primary driver of the XP methodology – they provide high-level requirements for your system and are the critical input into your planning process. The implication is that you need a collection of user stories, anywhere from a handful to several dozen, to get your XP project started. The second aspect of the exploration phase focuses on your system architecture. The architecture within an XP project is less formal than in traditional methodologies, with a preference for keeping your system flexible – XP recommends that you embrace change, whereas architecture-driven approaches advise you to build the skeleton the system first because some things are difficult to change. The XP approach is to identify a metaphor that describes how you intend to build your system. The metaphor acts as a conceptual framework, identifying key objects and providing insight into their interfaces. The metaphor is defined during an architectural spike early in the project, during the first iteration or during a pre-iteration that is sometimes referred to as a zero-feature release (ZFR).

In scrum project life cycle, there are no structured activities for requirements envisioning. The requirements are treated like a prioritized stack, pulling just enough work off the stack for the current iteration. At the end of the iteration, the system is demoed to the stakeholders to verify that the work that the team promised to do at the beginning of the iteration was in fact accomplished. But, where does the product backlog come from? It is actually the result of initial requirements envisioning early in the project.

An FDD project starts by building overall domain model to envision the requirements. The goal of envisioning is to identify the scope of the effort, the initial architecture, and the initial high-level plan. As with other agile software development processes, systems are delivered incrementally by FDD teams.

The envisioning activity in agile modeling includes two main sub-activities, initial requirements envisioning and initial architecture envisioning [4]. These are done during iteration 0, iteration being another term for cycle or sprint. The envisioning effort is typically performed during the first week of a project, the goal of which is to identify the scope of your system and a likely architecture for addressing it. To do this you will do both high-level requirements modeling and high-level architecture modeling. The goal isn't to write detailed specifications that prove incredibly risky in practice, but instead to explore the requirements and come to an overall strategy for your project. Finally and as shown in table 1, we need to take several days, with a maximum of two weeks for the vast majority of business systems, for initial requirements and architecture envisioning.

VI. CONCLUSION

Agile requirements envisioning aims to develop a high level understanding of the project. This allows the initial identifications of requirements for the system at the beginning of the project. This paper presented an evaluation framework for requirements envisioning in four agile methods: XP, Scrum, Feature Driven Development, and Agile Modeling. The results showed that clear specification of activities in the agile requirements envisioning process is missing and there is a lack of a set of activities and techniques that practitioners can choose from. Hence, there is a need to develop a structured approach that clearly outlines the activities of the agile requirements envisioning process and suggests techniques or practices that can be used.

REFERENCES

[1] Ambler, S., "Examining the "Big Requirements Up Front (BRUF) Approach", http://www.agilemodeling.com/essays/examiningBRUF.htm, retrieved: October, 2012.

[2] Hoffer, H., George, J., and Vlacich, J., "Modern Systems Analysis and Design", Pearson Prentice Hall, 2006.

[3] Nerur, S. and Balijepally, V., "Theoretical reflections on agile development methodologies", Communication of the ACM, vol. 50, pp. 79-83, 2007.

[4] Qumer, A. and Sellers, B., "An evaluation of the degree of agility in six agile methods and its implacability for method engineering", Information and Software Technology, 2007.

[5] Eberlein, A., Maurer, F., and Paetsch, F., "Requirements Engineering and Agile Software Development", Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.

[6] Pekka, A., Outi, S., Jussi, R., and Juhani, W., "Agile software development methods - Review and analysis", Publications, No. 478, 2002.

[7] Peter, C., Eric, L., and Jeff, L., "Java Modeling in Color with UML", Prentice Hall PTR, Chapter 6, 1999.

[8] Scott, W., "Agile Modeling", John Wiley & Sons, 2001.

[9] http://en.wikipedia.org/wiki/Agile_software_development, retrieved: October, 2012.

[10] Meng, X.,Wang, Y., Shi, L., and Wang, F., "A process pattern Language for Agile Methods", 14 th Asia-Pacific Software Engineering Conference, 2007.

[11] Ambler, S., "The Agile System Development Life Cycle", www.ambysoft.com/essays/agileLifecycle.html, retrieved: October, 2012.

[12] Highsmith, J. and Cockburn, A., "Agile Software Development: The Business of Innovation", IEEE Computer, Vol. 34, No. 9, pp. 120 – 127, 2001.

[13] Sommerville, I., "Software Engineering", Addison Wesley, 7th edition, 2004.

[14] Seyam, M. and Galal-Edeen, G., "Traditional versus Agile: The Tragile Framework for Information Systems development", the International Journal of Software Engineering (IJSE), Vol. 4, No. 1, pp. 63-93, ISSN: 1687-6954, 2011.

AgileKDD:

An Agile Business Intelligence Process Model

Givanildo Santana do Nascimento
Federal University of Sergipe
São Cristóvão, Brazil
gsnascimento@petrobras.com.br

Adicinéia Aparecida de Oliveira
Federal University of Sergipe
São Cristóvão, Brazil
adicineia@ufs.br

***Abstract* — In a knowledge-based society, transforming data into information and knowledge to support the decision-making process is a crucial success factor for all the organizations. In this context, one mission of Software Engineering is to produce systems able to process large volumes of data, transform them into relevant knowledge and deliver them to customers, so they can make right decisions at the right time. However, companies still face failures in determining the process model used in their Knowledge Discovery in Databases and Business Intelligence projects. This article introduces the AgileKDD, an agile and disciplined process for developing systems capable of discovering the knowledge hidden in databases, built on top of the Open Unified Process. A case study shows that AgileKDD can guide projects whose goal is to develop Knowledge Discovery in Databases and Business Intelligence applications, increasing success factor as well as customer satisfaction.**

Keywords – Business Intelligence; Knowledge Discovery in Databases; Agile Software Development; Software Process.

I. INTRODUCTION

In 1996, the Organization for Economic Cooperation and Development (OECD) redefined knowledge-based economies as: “economies which are directly based on the production, distribution and use of knowledge and information” [1]. In knowledge-based economies, the global competition is increasingly based on the ability to transform data into information and knowledge in an effective way. Knowledge is equated with the traditional factors of production - land, capital, raw materials, energy and manpower - in the process of wealth creation. Thus, data, information and knowledge constitute key assets for all organizations working in this economic model.

Knowledge management, Data Mining, Knowledge Discovery in Databases (KDD) and, more generally, Business Intelligence (BI) are key concepts in a knowledge-based economy. BI applications have vital importance for many organizations and can help them manage, develop and communicate their intangible assets such as information and knowledge, improving their performance. For instance, Continental Airlines’ investments in BI have a Return on Investment (ROI) of 1000%, attributed to increased revenue and reduced costs [2].

However, companies still face problems in determining the process model used to develop KDD and BI applications. As business requirements become more dynamic and uncertain, the traditional static, bureaucratic and heavy processes may not be able to deal with them. Recent researches have demonstrated that waterfall lifecycles and traditional software development processes are not successful in BI because they are unable to follow dynamic requirement changes in a rapidly evolving environment [3]. As software process is mandatory for KDD and BI development, one possible solution is to use an agile process, which is typically characterized by flexibility, adaptability, face-to-face communication and knowledge sharing.

This article discusses the importance of using an agile software process in KDD and BI applications development. Thus, the main objective of this paper is to present AgileKDD, an agile process able to guide the KDD and BI applications development in a manner compatible with the current ever-changing requirement environments. The next sections of this article are organized as follows: Section 2 describes BI and Knowledge Discovery in Databases as techniques for transforming raw data into information and knowledge. The 3rd section presents the agile software development processes. Section 4 presents the AgileKDD, an agile KDD and BI process model built on top of the Open Unified Process. Section 5 presents related work. Finally, Section 6 presents the conclusions.

II. TRANSFORMING DATA INTO INFORMATION AND KNOWLEDGE

The raw data evolve into information and knowledge as they receive degrees of association and meaning [4]. The knowledge gained from the interpretation of data and information drives the knower to action, so knowledge is an important asset for organizations that operate in knowledge-based economies and markets. BI, as well as KDD, has the goal of transforming raw data into information and knowledge, in order to support the decision making process.

A. Knowledge Discovery in Databases

Knowledge Discovery in Databases is a nontrivial process of identifying valid, novel, potentially useful, and understandable patterns in data [5]. The discovered knowledge must be correct, understandable by human users and also interesting, useful or new. In addition, the knowledge discovery method must be efficient, generic and flexible (easily changeable).

Data Mining (DM) is the main activity of KDD and consists of applying algorithms to extract models or patterns from data [5]. Data Mining is the process of searching for relationships and distinct patterns that exist in datasets but are hidden among the large amount of data. Its aim is to transform data apparently devoid of connection into relevant information for decision making and results evaluation. DM is used to find information without a prior formulation of hypotheses and search for something non-intuitive, transforming meaningless data into valuable strategic knowledge. DM tasks and tools include data classification, neural networks, clustering, regression analysis, correlation and predictive analysis. DM applications are characterized by the ability to deal with the explosion of business data and accelerated market changes. These characteristics provide powerful tools for decision makers. Such tools can be used by business users to analyze huge amount of data for patterns and discover trends [1].

The KDD systematization effort has resulted in a variety of process models, including the KDD Process [5] and the Cross-Industry Standard Process for Data Mining (CRISP-DM) [6], which are the most widely used in KDD projects, the most frequently cited and supported by tools. These two processes are considered the *de facto* standards in the KDD area. Several other process models were derived from KDD Process and CRISP-DM. Figure 1 shows the evolution of 17 KDD/BI process models and methodologies. KDD Process can be pointed out as the initial approach, and CRISP-DM as the central approach of the evolution diagram [7]. Most of the approaches are based on these process models.

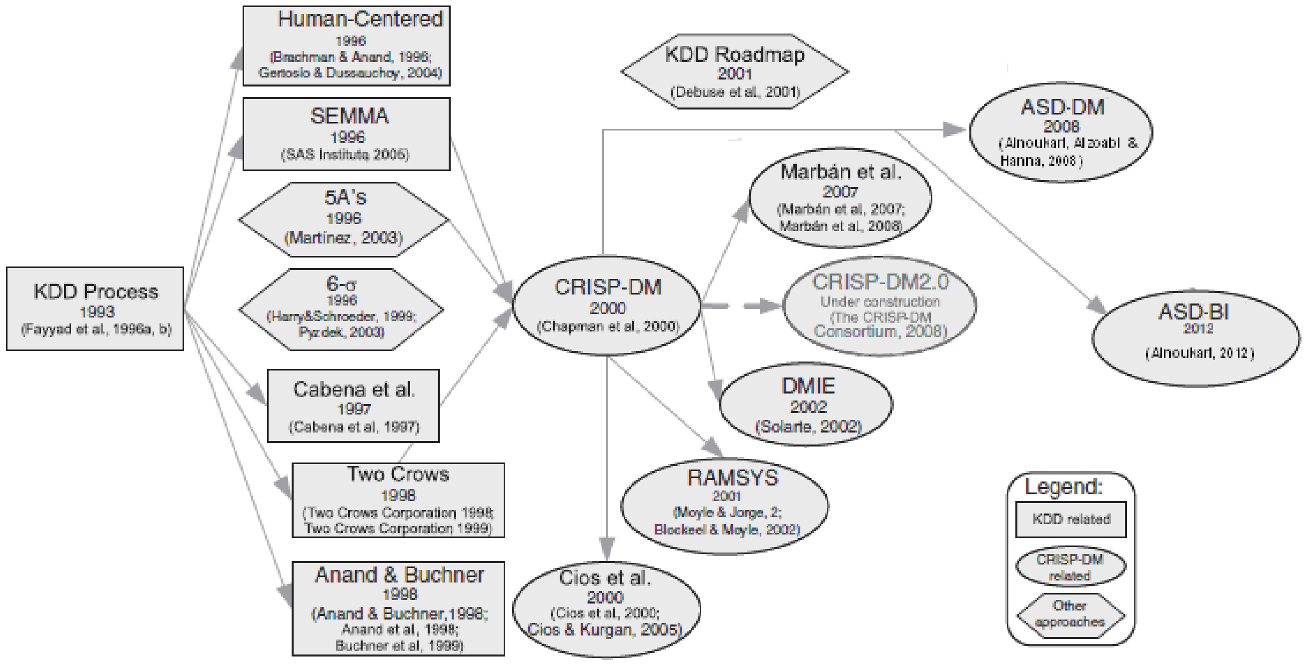


Figure 1. Evolution of KDD/BI process models (Source: Adapted from [7])

The KDD process models created between 1993 and 2008 were discussed in detail in a survey by [8] and then categorized by [7] into three groups:

- KDD related approaches – KDD Process (1993); Human-Centered (1996); 5A's (1996); 6-σ (1996); Cabena et al. (1997); Two Crowns (1998); and Anand & Buchner (1998).
- CRISP-DM related approaches – CRISP-DM (2000); Cios et al. (2000); RAMSYS (2001); DIME (2002); Marbán et al. (2007); and the CRISP-DM 2.0 initiative (not concluded).
- Other approaches – KDD Roadmap (2001).

Sometime later [9] continued the older surveys done by [8] and [7], and proposed a different categorization to the KDD process models:

- Traditional approach – Starting with KDD Process, many other process models used the same sequential steps: business understanding, data understanding, data processing, data mining, model evaluation, and deployment.
- Ontology-based approach – This approach is the combination of ontology engineering and traditional approach.
- Web-based approach – This approach is similar to the traditional approach, but it has some steps to deal with web log data analysis.
- Agile-based approach – Integrates agile processes and methodologies with traditional approaches. The process models in this category are Adaptive Software Development – Data Mining (ASD-DM) Process Model [10] and Adaptive Software Development – Business Intelligence (ASD-BI) Process Model [1].

Thus, the knowledge discovery process models are evolving from traditional to agile processes, becoming more adaptive, flexible and human-centered. However, these processes still lack software engineering capabilities such as requirements management, project management and changes management.

B. *Business Intelligence*

Business Intelligence assists in extracting information from the available data and using it as knowledge in developing innovative business strategies. BI is an umbrella term that combines architecture, tools, databases, applications, practices, and processes to organize, integrate and explore information, with the goal of developing understanding and knowledge, which can produce a better decision making process. Moreover, BI is an Information Technology (IT) framework vital for many organizations, especially those which have extremely large amounts of data, which can help organizations manage, develop and communicate their assets such as information and knowledge [2]. According to Mariscal et al. [7], BI is a broad category of applications and technologies for gathering, storing, analyzing, and providing access to data to help enterprise users make better business decisions, and DM is an important component of BI.

The number of BI projects has grown rapidly worldwide according to Gartner Group annual reports. BI has been on the list of the top ten priorities in IT since 2005 and was at the top of this list for four consecutive years, from 2006 to 2009. In a broader sense, companies understand that the information and knowledge provided by BI applications are essential to increase the efficiency and effectiveness, support competitiveness and innovation. Thus, investments into data mining BI applications grew by 4.8% from 2005 to 2006 and by 11.2% from 2007 to 2008 [7], [11], [12]. Continental Airlines' success case is the most expressive example of BI profitability. In 2006, the overall ROI of BI projects in that company was 1000% [2], [13] and in 2008 that initiative also reported very positive results [14].

However, not all KDD and BI results are positive. Regardless of the priority and budgets growth, neither all the projects results were delivered [7]. Further, the worsening of international financial crisis has led to significant cuts in IT budgets from 2008 on. In addition, many BI projects had failed to achieve their goals or were canceled because they were unable to follow the dynamic requirement changes in rapidly evolving environments. Because of this, BI left the top of the list of priorities in IT and, in 2010 and 2011, dropped to the fifth position. Technologies with higher productivity, lower risk and faster return on investment were priorized instead [12], [15].

The BI environment consists of the transaction processing applications, the Extraction, Transformation, Loading (ETL) and data integration processes, the Data Warehouse (DW) as well as the Data Marts, BI tools and analytic applications. The raw data are loaded by ETL processes into DW and data marts. During loading, the ETL processes also perform cleaning, completion, correction and integration of data. The DW and data marts are then explored by the user utilizing On-line Analytical Processing (OLAP) tools and data mining [2].

Many companies still develop KDD and BI applications without the guidance of a software process, but, as any software project, KDD and BI projects need a software process to succeed [16]. Also, the dynamic business requirements, the needs of quick ROI and fluid communication between stakeholders and the team led to agile process as one possible solution [3].

III. Agile Software Engineering Processes

A software process provides an ordered sequence of activities related to the specification, design and implementation as well as validation and deployment of software products, transforming user expectations into software solutions [17]. According to Pressman [18], the software processes set the context in which technical methods are applied, the work artifacts (models, documents, data, reports, forms) are produced, the milestones are established, quality is assured and changes are managed.

The traditional software development processes are characterized by rigid mechanisms with a heavy documentation process, which make it difficult to adapt to a high-speed, ever-changing environment [19]. Researchers have suggested that the complex, uncertain and unstable environment is pushing developers to adopt agile processes rather than traditional software processes. They claim that agile approach is the answer to the software engineering chaotic situation, in which projects are exceeding their time and budget limits, requirements are not fulfilled and, consequently, leading to unsatisfied customers [20].

The Manifesto for Agile Software Development [21] defines the values introduced by the agile software processes: individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; and responding to change over following a plan. Ultimately, by following these values, software development becomes less formal, more dynamic, and more customer-focused. Based on these values, agile processes are people-oriented and have the customer satisfaction as the highest priority through the early and continuous delivery of functioning software [20]. Also, the response to all types of changes and fluid communication between all projects participants become top priorities. In agile development, the main work product is the increment of functioning software, delivered to the customer within the fixed timeframes. Agile approaches are best fit when requirements are uncertain or volatile; this can happen due to business dynamics and rapidly evolving markets. It is too difficult to practice traditional plan-oriented software development in such unstable environments [19].

A. *Unified Process and Open Unified Process*

The Unified Process (UP) [22] is based on the Incremental Model [18], focuses on architecture and is use cases driven. Based on the use cases model, the analysis, design and implementation models are created to realize the use cases. The UP is focused on architecture, so it starts by the definition of an application skeleton (the architecture), which evolves gradually over development. The UP is also an iterative and incremental process because it offers an approach of partitioning the work into smaller portions or mini-projects. In UP, the architecture provides the framework to guide the system development into iterations, while the use cases define the targets and lead the work of each iteration.

Open Unified Process (OpenUP) is a variation of the UP that applies agile, iterative and incremental approaches within a structured lifecycle. OpenUP embraces a pragmatic, agile philosophy that focuses on the collaborative nature of software development. It is a low-ceremony process that can be extended to address a broad variety of project types [23]. OpenUP has compliance with the Manifesto for Agile Software Development, is minimal, complete and extensible. Moreover, it increases collaboration and continuous communication between project participants, more than formalities and comprehensive documentation [24].

The OpenUP process is divided into three layers, has four phases and six disciplines. The process applies intensive collaboration as the system is incrementally developed by a committed, self-organized team. OpenUP layers are illustrated by Figure 2. They are [23]:

- Project Lifecycle – structures the software project into four phases: Inception, Elaboration, Construction and Transition. A project plan defines the lifecycle and results in a released application.
- Iteration Lifecycle – OpenUP divides the project into iterations: planned, time-boxed intervals typically measured in weeks. Iterations focus the team on delivering incremental value to stakeholders in a predictable manner. OpenUP applies an iteration lifecycle that structures how micro-increments are applied to deliver stable, cohesive builds of the system that incrementally progresses towards the iteration objectives.
- Micro-increment – personal effort on an OpenUP project is organized in micro-increments. These micro-increments provide an extremely short feedback loop that drives adaptive decisions within each iteration.

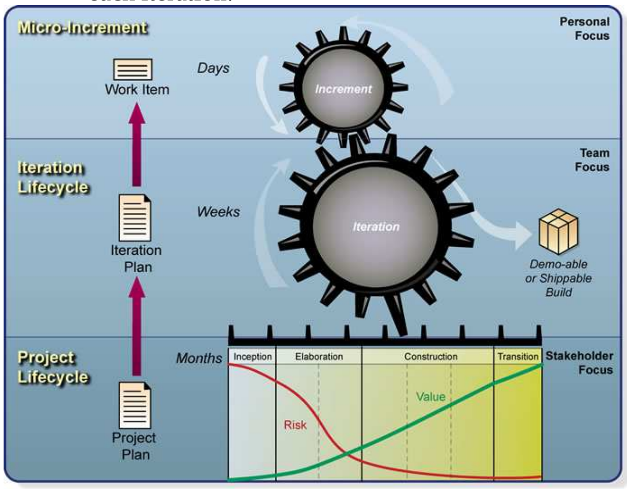


Figure 2. OpenUP layers and lifecycles (Source: [23])

The OpenUP phases are inception, elaboration, construction and transition. In inception phase the product vision is specified. The product architecture is defined in elaboration phase. The result of construction is a demonstrable or deliverable product version, deployed to the customer at the end of the transition phase. The OpenUP disciplines are requirements, architecture, development, testing, project management, and configuration and change management.

Again, the development of KDD and BI solutions must be guided by a software process. Therefore, it is mandatory to define processes that address aspects of KDD, BI, as well as the software engineering process disciplines, whose function is to order the software development. By the other hand, waterfall lifecycles and traditional processes are not successful in BI because they are unable to follow requirements in ever-changing environments [3]. Hence, one possible solution is to use an agile process, which is typically characterized by flexibility, adaptability, communication and knowledge sharing.

IV. AGILEKDD

AgileKDD is an agile and disciplined process for the development of KDD and BI applications. CRISP-DM and KDD Process provide the capabilities related to knowledge discovering. OpenUP provides to AgileKDD the lifecycle, phases and disciplines, which are requirements, architecture, development, test, project management and changes management. OpenUP also adds the agile software development core values and principles, without giving up the management disciplines. The personal effort on an AgileKDD project is organized in micro-increments. They represent small work units that produce measurable steps in the project progress, usually measured in days. The process applies intensive collaboration between the actors as the system is built incrementally. These micro-increments provide extremely short cycles of continuous feedback to identify and resolve problems before they become threats to the projects.

AgileKDD divides the projects in iterations with fixed time boxes, usually measured in weeks. The iterations drive the team to deliver incremental value to stakeholders in a predictable manner. Iteration plan defines what must be delivered during the iteration and the result is a demonstrable or deliverable piece of the KDD or BI solution. The AgileKDD lifecycle provides stakeholders and project team visibility and decision points at various milestones, until a working application is fully delivered to stakeholders. Figure 3 presents an overview of AgileKDD, highlighting its phases and activities.

The Inception phase has the aim of developing an understanding of the application domain and the relevant prior knowledge and identifying the goal of the BI project from the customer's viewpoint. In this phase the project vision and plans are defined and agreed by all project participants. Also, in inception the target data set, or subset of variables or data samples, is selected. The knowledge discovery processes will be performed on the selected target data set.

The Elaboration phase is responsible by the system's architecture, the data preprocessing and modeling. Data cleaning removes noise, collects the necessary information to model and decides on strategies for handling missing data fields. Data quality is a critical success factor for any KDD or BI project, so it is verified prior to the DW and data marts modeling.

Once DW and data marts are modeled, ETL processes are built to extract, integrate, transform and load the selected target data into DW and data marts. Thus, the data mining techniques that best fit to the data are selected and applied to the information stored in data marts.

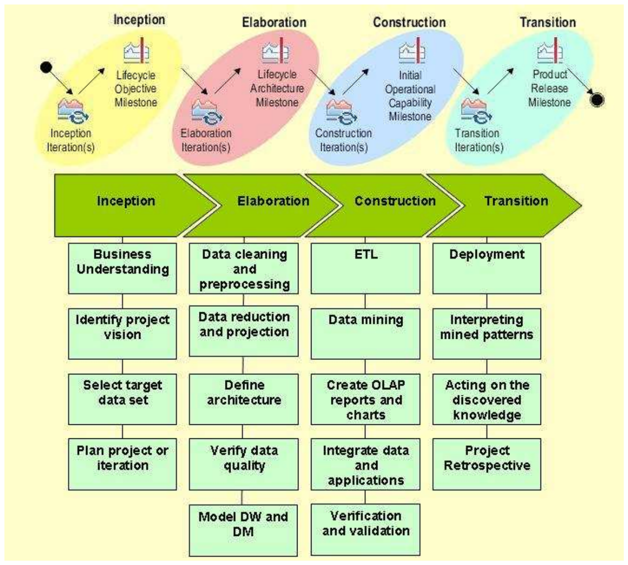

Inception
Elaboration
Construction
Transition
Lifecycle Objective Milestone
Lifecycle Architecture Milestone
Initial Operational Capability Milestone
Product Release Milestone
Inception Iteration(s)
Elaboration Iteration(s)
Construction Iteration(s)
Transition Iteration(s)
Inception
Elaboration
Construction
Transition
Business Understanding
Identify project vision
Select target data set
Plan project or iteration
Data cleaning and preprocessing
Data reduction and projection
Define architecture
Verify data quality
Model DW and DM
ETL
Data mining
Create OLAP reports and charts
Integrate data and applications
Verification and validation
Deployment
Interpreting mined patterns
Acting on the discovered knowledge
Project Retrospective

Figure 3. Initial AgileKDD lifecycle

Data mining tools search for meaningful patterns in data, including association rules, decision trees and clusters. The team can significantly aid the data mining method by correctly performing the preceding steps. The OLAP reports and charts as well as the dashboards are built to allow user data exploration. The verification and validation activities guarantee that the data was extracted, loaded and processed correctly, according to business objectives.

In Transition phase the deployment of both software and knowledge takes place, the discovered knowledge is discussed and interpreted by human beings, actions are created and the retrospective discusses lessons learnt during the project to promote continuous process improvement. Interpreting mined patterns involve visualization and storage of the extracted knowledge into knowledge bases, or simply documenting and reporting it to interested parties. This activity also includes checking for and resolving potential conflicts with previously believed knowledge. The AgileKDD process can involve significant iteration, interaction and can contain loops between any two phases.

A. *Case study and process refinement*

AgileKDD has been validated by a case study in oil and gas field. The process was applied to a KDD and BI project that dealt with Reservoir Evaluation data and afforded the early and continuous delivery of results to the customers. DM results in first iteration were delivered two months after the project kickoff. The second iteration delivered the performance indicators as a dashboard and the third iteration deployed the OLAP reports, graphs, and *ad hoc* exploration of the DW.

The case study showed that AgileKDD was able to guide the KDD and BI application development and helped to anticipate the project ROI. Moreover, the process was refined after the project retrospective. The refined AgileKDD lifecycle is represented by Figure 4.

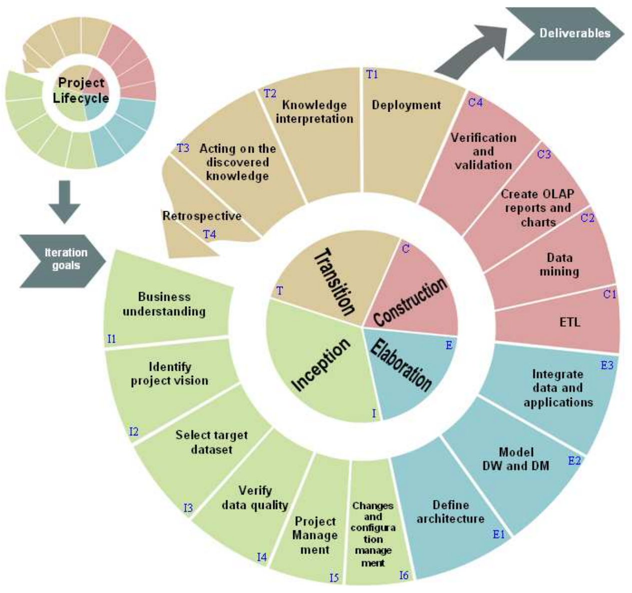

Deliverables
Project Lifecycle
Iteration goals
Transition
Construction
Elaboration
Inception
T
C
E
I
T1 Deployment
T2 Knowledge interpretation
T3 Acting on the discovered knowledge
Retrospective T4
C4 Verification and validation
C3 Create OLAP reports and charts
C2 Data mining
C1 ETL
E3 Integrate data and applications
E2 Model DW and DM
E1 Define architecture
I1 Business understanding
I2 Identify project vision
I3 Select target dataset
I4 Verify data quality
I5 Project Manage ment
I6 Changes and configura tion manage ment

Figure 4. Refined AgileKDD lifecycle after case study

As data quality is a critical success factor for any BI project, the verify data quality activity was moved from Elaboration to Inception phase because its result influences the project management. Then, data quality is verified in Inception phase to indicate the project feasibility and quality constraints. Project management activity consists on high level project planning, risks management and governance concerns. Changes and configuration management activity is related to the version control of all the project artifacts, including documentation, sources and binaries.

Under the architectural point of view, the Data Mart Bus Architecture [25] fits perfectly in agile approach. In this architecture, the data marts are built incrementally in response to iteration requirements. The Data cleaning and preprocessing and Data reduction and projection activities defined in Elaboration phase were performed as part of ETL activity in Construction phase. Since they are activities related to the transformation of the data, the best place to them is ETL activity.

All the documentation artifacts needed to develop the three iterations were composed by the vision document and the data models of operational sources and DW. There was no need to use cases and additional diagrams.

The data mining results verification against the operational data sources was crucial for the knowledge discovered acceptance by business experts. The rules proof using operational data gave no room for questioning the correctness of DM methods and tools used. The documentation of the discovered knowledge in an electronic presentation was sufficient to support communication with knowledge users. No other form of knowledge representation and storage of was required.

Requirements changes directly affected the project planning, but did not harm the product objectives achievement because they were discovered early, between project iterations.

B. *AgileKDD disciplines*

The refined AgileKDD disciplines are the same of OpenUP: requirements, architecture, development, test, project management and configuration and changes management. Table I shows the AgileKDD disciplines, their purposes and suggested work products.

TABLE I. REFINED AGILEKDD DISCIPLINES

Discipline	Purpose	Work products[a]
Requirements	Elicit, analyze, specify, validate and manage the requirements for the system being developed.	Vision document. Initial project glossary. Prototypes.
Architecture	Define an architecture for the system components.	Software architecture description. DW and DM models.
Development	Design and implement a technical solution adherent to the architecture that meets the requirements.	Software components. Integrated software increment.
Test	Validate system maturity through the design, implementation, execution and evaluation of tests.	Plan and test procedure. Test record.
Project management	Instruct, assist and support the team, helping them to deal with risks and obstacles faced when building software.	Project plan. Feasibility and risk evaluation.
Configuration and change management	Controlling changes in artifacts, ensuring a synchronized evolution of the set of artifacts that make a software system.	Work items list.

a. All the work products are optional. Only the necessary artifacts must be produced.

During a full project cycle, most of the requirements discipline effort is concentrated in the Inception phase. The architecture is the main discipline during the Elaboration phase. In the same phase, the development is intensified from the definition of the system architecture and continues as the main discipline of Construction phase. The tests occur mainly in verification and validation activity of Construction phase. The project management discipline is concentrated predominantly in the Inception phase. The configuration and change management has greater prevalence in Inception and Transition phases. Each discipline can be related to a set of work products created during the process phases, according to the project needs.

V. RELATED WORK

There are not many works related to agile software processes appropriate to the development of KDD and BI applications. The main work that applies agile methodologies to BI is [1]. Alnoukari [19] discusses Business Intelligence and Agile Methodologies for knowledge-based organizations in a cross-disciplinary approach. Alnoukari [26] introduces Adaptive Software Development – Business Intelligence (ASD-BI), a BI process model based on Adaptive Software Development agile methodology. Likewise, Alnoukari et al. [10] defined Adaptive Software Development – Data Mining (ASD-DM) Process Model. The main difference between this work and these is the fact that AgileKDD is a process, not a methodology. As a process, AgileKDD defines what to do instead how to do KDD and BI development. Also, the process proposed by this work defines lifecycle, roles, activities, inputs and outputs regarding agile KDD and BI application development. Moreover, AgileKDD contains management disciplines like project, changes and requirements management, which were inherited from OpenUP. Even in an agile process like AgileKDD, management is a crucial success factor for any software projects.

Three surveys about data mining and knowledge discovery process models and methodologies are discussed and compared by [7], [8] and [9]. All the process models and methodologies presented by these works focus on data mining and knowledge discovery, and don't consider databases like DW and data marts nor BI and OLAP. As BI is more comprehensive than data mining, this work focuses on an agile process modeled to address both KDD and BI software projects, in an adaptable, flexible and systematic manner.

VI. CONCLUSION

A software process is mandatory for KDD and BI developments; however traditional software development processes are not successful in KDD and BI because they are unable to fulfill dynamic requirement changes in an ever-changing environment. Agile processes fit in KDD and BI better than traditional processes because they are characterized by flexibility, adaptability, communication and knowledge sharing.

This work presented AgileKDD, a KDD and BI process based on KDD Process, CRISP-DM and Open Unified Process. AgileKDD has been validated by a case study and results indicate that software development organizations may apply AgileKDD in KDD and BI applications projects. The process bring benefits as more customer satisfaction through early and continuous delivery of functioning software, better communication between team members and reducing projects failures risks.

The main contribution of AgileKDD is its ability to guide the BI solutions development according to the practices present in agile software development processes. AgileKDD can increase the projects success factor and customer satisfaction through the early and continuous delivery of functioning software and useful knowledge. The process can be used to guide BI applications projects in scenarios of continuous requirements evolving and early ROI need.

Future work can validate AgileKDD by case studies in different areas and investigate the need of storing the knowledge discovered into ontology or knowledge bases.

REFERENCES

[1] A. El Sheikh and M. Alnoukari, Business Intelligence and Agile Methodologies for Knowledge-Based Organizations: Cross-Disciplinary Applications. IGI Global, 2012. pp. 1-370.

[2] M. Alnoukari, H. Alhawasli, H. Alnafea, and Amjad Zamreek, "Business Intelligence: Body of Knowledge" in *Business Intelligence and Agile Methodologies for Knowledge-Based Organizations: Cross-Disciplinary Applications*. IGI Global, 2012. pp. 1-13.

[3] D. Larson, "Agile Methodologies for Business Intelligence" in Business Intelligence and Agile Methodologies for Knowledge-Based Organizations: Cross-Disciplinary Applications. IGI Global, 2012. pp. 101-119.

[4] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, Sixth Edition. Pearson, 2010.

[5] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From datamining to knowledge discovery: an overview" in *Proc. Advances in Knowledge Discovery and Data Mining*, 1996, pp. 1–34.

[6] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, CRISP-DM 1.0: Step-by-step data mining guide, 2000.

[7] G. Mariscal, O. Marbán, and C. Fernández, "A survey of data mining and knowledge discovery process models and methodologies". *The Knowledge Engineering Review*, vol. 25, 2010, pp. 137-166.

[8] L. Kurgan and P. Musilek, "A survey of Knowledge Discovery and Data Mining process models". *The Knowledge Engineering Review*, vol. 21, 2006, pp. 1-24.

[9] M. Alnoukari and A. El Sheikh, "Knowledge Discovery Process Models: From Traditional to Agile Modeling" in *Business Intelligence and Agile Methodologies for Knowledge-Based Organizations: Cross-Disciplinary Applications*. IGI Global, 2012. pp. 72-100.

[10] M. Alnoukari, Z. Alzoabi, and S. Hanna, "Applying adaptive software development (ASD) agile modeling on predictive data mining applications: ASD-DM Methodology" in *IEEE Proceedings of International Symposium of Information Technology*, 2008, pp. 1083–1087.

[11] M. McDonald, M. Blosch, T. Jaffarian, L. Mok, and S. Stevens, "Growing It's Contribution: The 2006 Cio Agenda". Gartner Group, 2006.

[12] Gartner Group, "Gartner says more than 50 percent of data warehouse projects will have limited acceptance or will be failures through 2007". 2005. Available: http://www.gartner.com/it/page.jsp?id=492112 [retrieved: Oct., 2012].

[13] H. J. Watson, B. H. Wixom, J. A. Hoffer, R. A. Lehman, and A. M. Reynolds, "Real-Time Business Intelligence: Best Practices at Continental Airlines". *Information Systems Management*, Bristol, vol. 23, n. 1, pp. 7-18, Dec. 2006. Available: http://dx.doi.org/10.1201/1078.10580530/45769.23.1.20061201/91768.2 [retrieved: Jul., 2012].

[14] B. H. Wixom, H. J. Watson, A. M. Reynolds, and J. A. Hoffer, "Continental Airlines Continues to Soar with Business Intelligence". *Information Systems Management*, Bristol, vol. 25, n. 2, pp. 102-112, Mar. 2008. Available: http://dl.acm.org/citation.cfm?id=1451615 [retrieved: Jul., 2012].

[15] Gartner Group, "Gartner Executive Programs Worldwide Survey of More Than 2,000 CIOs Identifies Cloud Computing as Top Technology Priority for CIOs in 2011". 2011. Available: http://www.gartner.com/it/page.jsp?id=1526414 [retrieved: Oct., 2012].

[16] O. Marbán et al. "Towards data mining engineering: a software engineering approach". *Information Systems Journal*, vol. 34, n. 1, Mar. 2009. Available: http://dl.acm.org/citation.cfm?id=1458745 [retrieved: Oct., 2012].

[17] I. Sommerville, Software Engineering. Addison Wesley, 2006.

[18] R. Pressman, Software Engineering: A Practioner's Approach. McGraw-Hill, 2005.

[19] M. Alnoukari, "Business Intelligence and Agile Methodologies for Knowledge-Based Organizations: Cross-Disciplinary Applications" in *CEPIS UPGRADE: The European Journal for the Informatics Professional*, vol. 12, pp. 56–59, 2011. Available: http://www.cepis.org/upgrade/media/III_2011_alnoukari1.pdf [retrieved: Oct., 2012].

[20] Z. Alzoabi, "Agile Software: Body of Knowledge" in *Business Intelligence and Agile Methodologies for Knowledge-Based Organizations: Cross-Disciplinary Applications*. IGI Global, 2012. pp. 14-34.

[21] K. Beck et al Manifesto for Agile Software Development. 2001. Available: http://agilemanifesto.org [retrieved: Oct., 2012].

[22] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide. Addison Wesley, 1999.

[23] H. Hristov, Introduction to OpenUP. 2011. Available: http://epf.eclipse.org/wikis/openup/index.htm [retrieved: Oct., 2012].

[24] S. Santos, OpenUP: Um processo ágil. 2009. Available: http://www.ibm.com/developerworks/br/rational/local/open_up/index.html [retrieved: Oct., 2012].

[25] R. Kimball and M. Ross, Data warehouse toolkit: o guia completo para modelagem dimensional. Rio de Janeiro: Campus, 2002.

[26] M. Alnoukari, "ASD-BI: A Knowledge Discovery Process Modeling Based on Adaptive Software Development Agile Methodology" in *Business Intelligence and Agile Methodologies for Knowledge-Based Organizations: Cross-Disciplinary Applications*. IGI Global, 2012. pp. 183-207.

The Dilemma of Tool Selection for Agile Project Management

Gayane Azizyan
Ericsson AB
Stockholm, Sweden
gayane.azizyan@ericsson.com

Miganoush Magarian
SAP Innovation Center
Potsdam, Germany
miganoush.magarian@sap.com

Mira Kajko-Mattsson
KTH Royal Institute of Technology
Stockholm, Sweden
mekm2@kth.se

Abstract*—Even if agile project management tools grow in number and complexity, companies still face difficulties in selecting tools that fit their needs. One such company is Company A, a multinational company that has experienced a great need for a tool supporting their multi-layered management of requirements and projects. For this reason, they commissioned us to perform a detailed analysis of their tool usage needs, placing much stress on adherence to their processes and all the roles involved. In this paper, we present our journey towards selecting the right tool for Company A and the problems encountered when trying to reach our goal. The tool selection process was based on (1) a study of tool features such as usability and extensibility, and (2) observation of the company's process and elicitation of the company's tool usage needs. Our results show that even if there are many tools on the market, it was still difficult to find an appropriate tool. The reasons were the following: (1) a study of general tool features could not provide enough information to account for the company's needs, (2) even people possessing the same role in the company prioritized different features differently; hence, it was difficult to utilize their feedback for selecting a tool, and, finally, (3) even after having observed the company's process, it was still difficult to find an adequate tool. The tools simply could not support the company's multi-level requirements management process. Moreover, they suffered from poor usability as well as imposed their own terminology and process. Overall, the agile tools studied were not a good match for supporting the company's agile project management needs.

Keywords-components; agile; tool; process; adoption

I. Introduction

Agile methods sneaked into Company A's overall processes in an unofficial way about four years ago. No official decision for introducing them was made. Neither was there any process facilitating its introduction. Different teams simply started using agile practices on their own, mainly for implementation-level activities, such as tracking the status of tasks. With time, however, even management started using agile practices for managing and tracking requirements. As tool support, they mainly used MS Excel, Word, and PowerPoint for storing and managing requirements and product backlogs and MS PowerPoint for managing projects. The development teams, on the other hand, used simple physical tools such as paper, sticky notes, and whiteboards.

As the use of agile methods grew at Company A, the company experienced that the simple tools were insufficient for and unsupportive in managing large numbers of requirements and projects. Hence, Company A expressed a great need for a better tool. For this reason, they commissioned us to help them with agile tool selection. Our task was to perform a detailed analysis of their tool usage needs where great stress would be placed on adherence to their processes and all the roles involved.

In this paper, we present the results of our attempt to find an appropriate agile project management tool to meet Company A's tool support needs. The company wishes to stay anonymous in this paper, and therefore, we use a fictitious name, Company A, when referring to it. The remainder of this paper is organized as follows. Section 2 presents the research steps conducted during our study. Section 3 gives a brief description of the company. Section 4 lists and describes the criteria used for evaluating currently existing tools. Section 5 presents the tool evaluation results. Sections 6 and 7 describe the company needs based on our observations of its processes and current state of practice. Finally, Section 8 concludes with final remarks.

II. Research Steps

Our research was conducted in six main steps: (1) *Literature study*, (2) *Company need identification*, (3) *Tool selection, (4) Tool evaluation,* (5) *Company observation,* and (6) *Analysis of results.*

During the *Literature study* step, we went through the existing literature dealing with tool evaluations and adoptions in agile contexts. Although we looked through many scientific and non-scientific sources, we did not find any objective, detailed evaluations of agile project management tools. The articles we found were limited to high-level discussions of classes of agile tools to be used in different team types [1], tool evaluations aimed at meeting needs of some specific company [2], and tool evaluations solely focused on open-source tools [3]. Other resources included lists of agile tools, as well as some general discussions of their usage [4]-[6]. The *Literature study* step resulted in awareness that our study was unique. We could neither relate it nor base it on somebody else's results and experience. Based on our findings in this step, we understood that companies met their agile tool needs either by developing a custom tool [7][8], or by selecting an existing tool primarily based on factors such as the tool's popularity [9][10].

In the *Company need identification* step, we studied Company A and its agile environment. Our goal was to get acquainted with the company's process and tool support. To achieve this, we studied the company's processes, projects, and product documentation, as well as had a series of meetings with three company representatives possessing the roles of a Project Manager, a Scrum Master, and a Line Manager. This step resulted in the identification of a preliminary list of high-level tool support requirements and provided a basis for defining tool properties. To ensure that we had an exhaustive list of such properties, we made an additional literature study during which we elicited requirements that were imposed by Scrum practices [2]. Altogether, we collected 21 properties. We presented them to the company representatives and got them accepted as tool evaluation criteria to be used in the *Tool evaluation* step. The criteria are presented in Section 4.

In the third step, *Tool selection*, we looked through agile tools available on the market and selected six tools for a detailed study. The selection was influenced by the following criteria: (1) the company's interest in particular tools, (2) tool popularity [6][11][12], (3) support for the agile methods used at the company, (4) deployment options, (5) availability of integration options with other systems, (6) licensing, and (7) supported platforms. The selected tools were *VersionOne* [13], *Scrumworks* [14], *Rally* [15], *Scrum Desk* [16], *Silver Catalyst* [17], and *Agilo* [18]. Table 1 briefly summarizes their properties.

The *Tool evaluation* step was conducted in two sub-steps. First, the authors of this paper made their own evaluation of the selected tools. The goal was to rate the tools according to the selected criteria and gather an in-depth understanding of how well the chosen tools supported the criteria. The results of this sub-step would then be matched to the company's tool needs to be elicited in the next sub-step, during which the company representatives weighed the criteria by assigning a quantitative measure of how important each criterion was for the company's needs. A range of 0-10 was used for rating both the tools and the criteria. We chose this wide range of values in order to achieve richer granularity in the tool evaluation results, and to have more options for making our choice

In the first sub-step, when doing our own tool evaluation, we used demo versions of the selected tools. Here, we studied their documentation and executed sample test projects. We then individually rated each evaluation criterion for every tool. Finally, we discussed the ratings, removed all types of inconsistencies and ensured that we reached agreement on every rated value.

After reaching consensus on the rated values, we calculated the total and total normalized ratings for every tool. Here, as given by (1), we first summed up the rating u_i^t of each tool *t* over all criteria to yield a total rating U^t for each tool *t*. Further, as given by (2), we divided the total ratings by the number of criteria (*N*), to yield a total normalized rating U_N^t.

$$U^t = \sum u_i^t \tag{1}$$

$$U_N^t = \frac{U^t}{N} \tag{2}$$

The first sub-step resulted in a list of ratings for each criterion, which was used as a base for the evaluation in the second sub-step.

The second sub-step was conducted in form of interviews, with seventeen company representatives possessing the roles of *Developer, Scrum Master, Designer, Agile Project Manager, Program Manager, Product Manager, Development Manager,* and *Line Manager*. During the interviews, we first presented and explained the list of tool evaluation criteria. The representatives then assigned their weights to each criterion.

After having collected all the data from the company representatives, we summed up the assigned weights to

TABLE I. SUMMARY OF THE EVALUATED TOOLS

Tool name	Methods	Deployment	Integration	License	Platforms
VersionOne	Scrum, XP, DSDM	Hosted, Local	Connectors for a wide range of development tools. [19]	Commercial	Windows, Linux, Mac OS X
ScrumWorks Pro	Scrum	Hosted, Local	Bugzilla [20], JIRA [21], Eclipse IDE[22], MS Excel [23]	Commercial	Windows, Unix Mac OS X
Rally	Scrum	Hosted, Local	Connectors for a wide range of development tools. [24]	Commercial, Free	Windows, Linux, Mac OS X
ScrumDesk	Scrum	Hosted, Local	Microsoft Team Fuoundation Server[25], Mantis [26]	Commercial, Free	Windows
SilverCatalyst	Scrum, XP, FDD, Kanban.	Hosted, Local	SVN [27], Trac [28], Wikis	Commercial, Free	Windows, Linux, Mac OS X
Agilo Pro	Scrum	Hosted, Local	Trac, SVN, Eclipse IDE	Free	Windows, Linux, Mac OS X

calculate average weights $U_N^t\ w_{c_i}$. The goal was to calculate the final rating of each criterion c_i of tool t according to (3). Here u_i^t is the rating assigned by us in the first sub-step.

$$r_i^t = \frac{u_i^t \times w_{c_i}}{10} \qquad (3)$$

After studying the results, we realized that the second sub-step proved to be inadequate for appropriately identifying the company's needs. First, we observed that different roles and personalities had great impact on the assigned weights. Even interviewees with the same role provided completely different ratings. Second, due to their narrow view of the company's process, and due to their unawareness of the overall tool requirements, the interviewees had trouble in quantifying their needs by means of numbers.

All this made it meaningless to calculate averages. We became aware that this step would not help us in identifying the company's needs to be fulfilled by the selected tool. A deeper analysis was required. This had led us to the creation of the next research step, the *Company observation* step.

During the *Company observation s*tep, we conducted a detailed on-site observation of the company's agile process. The observation lasted for two months, during which we observed the process, conducted interviews, and studied the company's product documentation. All these tasks were performed in parallel. We followed the executed process and attended several meetings, such as Scrum of Scrums meetings, daily meetings, and management meetings. We conducted interviews with the company representatives, the same representatives that were involved in the *Tool evaluation* step.

The interviews took the form of informal discussions that were guided by a semi-formal and semi-structured questionnaire. We chose this over a more formal, structured approach because we felt that most of the interviewees did not have a clear a priori picture of the setbacks and hindrances in their daily work. Hence, informal discussions served better for studying the daily work, identifying pain areas, and eliciting wishes for improvements. The questionnaire used in this step is presented in Table 2. Last but not least, we studied the internal company documents, such as the product backlog, documents describing new requirements, process documentation, and the like. This had helped us to improve the quality and effectiveness of our observations and interviews.

Throughout the *Company observation* step, we continuously analyzed the gathered information and drew out a diagram of the existing process, its inputs, outputs, the roles involved, the tools used, and the difficulties in the tool usage. As a result of this step, we identified six main tool support needs. These six needs served as input to the final step, *Analysis*. Here we tried to match these needs with the features of the evaluated agile tools in an attempt to find adequate tool support for the company. The results of the *Analysis* step are presented in Section 7.

TABLE II. COMPANY INTERVIEW QUESTIONNAIRE

1	How do you work with the existing system?
2	What do you like about the existing system? What works well for you?
3	What pain areas do you see in your daily work? What is inconvenient or annoying for you?
4	Is there anything you would like to have changed or improved? How?
5	Is there anything you would like to have added to the tools you are using? Any specific features or capabilities?

III. COMPANY DESCRIPTION

Our case study was conducted at Company A, which is a large software development company with a complex structure. The department we collaborated with is spread over three locations: Stockholm in Sweden, Shanghai in China, and Rijen in Holland. The different sites collaborate on one large project. The company is hierarchically structured into six different nodes working on different product parts where each node is further divided into different teams spread over several locations. In total, the department has 85 people that are distributed over eight teams, out of which four are situated in Stockholm. It is these four teams that were involved in our case study.

Company A, just as any other company, has many different roles that are involved in management and development. By a "role" we mean a set of responsibilities that are assigned to an individual or a group of individuals. Below, we list and briefly describe the roles that are of interest to this paper. We would like to point out, however, that some of these roles lacked formal definitions at the company. For this reason, we describe them just as we had understood them. In our descriptions, we only list the responsibilities that lie within the scope of this paper.

- *Program Manager* responsible for the operational steering of activities within a program and accountable for deliveries. This role is also responsible for promoting agile ways of working, for release planning, keeping progress visible, as well as planning, assigning and following the program budget.
- *Solution Product Manager (SPM)* responsible for strategic planning, pricing, commercial packaging, marketing, and setting product and professional service requirements. This role acts as a business builder and maintains a competitive position for the product.
- *Solution Architect* responsible for ensuring that the product is scalable and reusable.
- *Agile Project Manager (APM) responsible for prioritizing* and managing the product backlog.

- *Release APM role* corresponding to an APM with the added responsibility of having an overall understanding of the product specifications, as well as presenting the product to the business owners.
- *User eXperience Designer (UXD*) corresponding to any person qualified in User eXperience Design [29].
- *Developer* corresponding to any development team member.
- *Scrum Master* responsible for maintaining and steering the Scrum process.
- *Line Manager* responsible for a particular product line.

IV. *EVALUATION CRITERIA*

The twenty-one criteria that were used for evaluating the tools are the following:

• ***Extensibility*** referring to whether the tool can be modified or extended. Here, we evaluated whether a tool provided access to the source code, and whether it was offered on a commercial or open source license.

• ***Usability*** concerning the general usability of the tool. Here, we rated the tools solely for their ease of use, also taking into account whether it was necessary to study tool documentation and tutorials.

• ***Connectivity*** describing the connectors, or plug-ins, provided by the tool vendors such as Integrated Development Environments (IDEs), bug-tracking systems or traditional project management tools. Here, we evaluated the availability of such connectors; we also took into account both their number and variety.

• ***Searching*** referring to the searching capabilities of the tool. Here, we evaluated the availability of searching options, taking into account the searching factors.

• ***Grouping*** standing for the capability to group items in a product backlog. We evaluated whether the tool enabled grouping of product backlog items.

• ***Simultaneous editing*** implying whether multiple users could simultaneously edit the same artifact in the tool. While this might seem a basic requirement for tools, we still consider it mainly due to the absence of such options in basic tools such as spreadsheets used for storing backlogs.

• ***Story status tracking*** referring to the opportunity to track the status of a user story. Here, we evaluated whether the tool allowed to record progress of the story. The status could simply be represented as a string,

• ***Group status tracking*** enabling grouping of product backlog items. We evaluated whether it was possible to track the status of the group.

• ***Overall status tracking*** referring to the options of viewing the overall project status. This could imply a high-level summary view of the Sprint backlogs, or a chart showing the number of completed product backlog items over time. For this criterion, we evaluated whether the tool provided feedback on project status and what kind of status reports it generated.

• ***Sorting/Filtering*** standing for the sorting and filtering options. Here, we evaluated whether the tool provided sorting and filtering services and whether it could sort by several criteria and filter by typing in keywords.

• ***Sprint backlog*** dealing with the ability to create and manage a Sprint backlog. Here, we evaluated whether it was possible to prioritize and order Sprint backlog items.

• ***Estimation*** concerning the estimation ability of user stories and tasks. For this criterion, we evaluated whether it was possible to enter estimations of user stories and tasks, and how flexible the estimation measures were.

• ***Stories.*** Here, we evaluated the options offered for creating and describing user stories. Specifically, we evaluated whether it was possible to group stories into epics.

• ***Tasks*** referring to the tasks required for implementing a user story. We evaluated whether it was possible to break down user stories into smaller tasks in a Sprint backlog.

• ***Testing.*** Here, we evaluated the support for testing tasks. This could be realized in form of connectors to testing tools or by enabling the creation of special testing tasks.

• ***Teams*** referring to team management. For this criterion, we evaluated whether it was possible to create teams within the tool, assign team members, and make changes to the team capacity over time.

• ***Planning*** covering the ability to support Sprint planning, that is, selecting items from the product backlog and entering them in a new Sprint. We evaluated whether this was possible, and whether the capacity of the created Sprint was automatically matched to the team assigned to that Sprint.

• ***Progress*** referring to the status of the story or task on a greater level of detail. Here, we evaluated whether the tool enabled entering the amount of work performed on a story or task, and whether it was also possible to see how much work remained.

• ***Board*** covering the provision of a virtual task board for storing user stories and tasks. Here, we evaluated the availability of a task board, and, if so, whether it was interactive and whether it was possible to drag and drop tasks and user stories on the board.

• ***Burndown*** describing whether the tool included Sprint burndown charts, whether they were updatable, and how visually clear they were.

• ***Remote workplace*** relating to the opportunity to access a tool remotely. Most often, this implies that a tool needs to be deployed as a web application so that the user can access the application even outside the office network. Here, we evaluated whether such an opportunity was available.

V. TOOL EVALUATION

In this section, we present the tool evaluation results. We first present the results of our evaluation according to the twenty-one criteria described in the previous section. We then describe the elicitation of the company's tool needs and motivate why it became unsuccessful.

V.1 Our Own Tool Evaluation

As a result of our evaluation, we discovered that all the evaluated tools focused on team-level rather than management-level aspects. The tools included features such as advanced virtual task boards for facilitating development within the teams, but provided only rudimentary requirements and project management support. Moreover, tools covering more features were complicated to use.

The results of our tool evaluation are summarized in Figure 1 and Table 3. The evaluation had led us to a number of conclusions. First, we noticed that the tools were targeted towards Scrum Masters and development teams rather than managers. Five out of the six tools studied provided virtual task boards, as well as possibilities to break down user stories into more detailed tasks and functions for storing and managing the tasks.

The second, more important conclusion concerned the usability of the tools studied. Tools with higher usability offered less features, and tools which provided many features had a notably lower usability. This can be seen in the pivot chart shown in Figure 2 where the ratings of *VersionOne* and *Silver Catalyst* are presented. On average, *VersionOne* provides more features such as Connectivity, Grouping, and Group status tracking, while *Silver Catalyst* has a rating of 0 for several criteria. However, *VersionOne* has a usability rating of 5, which is much lower compared to the rating of *Silver Catalyst,* which is 10. It took us great effort to get accustomed to working with *VersionOne* due to its many features and customization options. *Silver Catalyst*, on the other hand, had a very simple and intuitive interface that was pleasant to use.

V.1 Elicitation of Company's Tool Needs

During the elicitation of the company's tool needs, the company representatives assigned weights to each criterion. After studying their feedback, however, we had to reject all the results achieved in this step, for the following reasons. First, it became clear that even interviewees with the same role provided sometimes contradictory ratings. Second, due to their limited view of the company's process and unawareness of the overall tool requirements, the interviewees had trouble in quantifying their needs by means of numbers.

Since the ratings provided by the company representatives strongly varied, it is meaningless to show them all in this paper. For illustrative purposes, however, we show three sample responses in Table 4 and Figure 2. Table 4 shows the weights and average weights assigned to all the criteria by three different Scrum Masters whereas Figure 2 shows a pivot chart comparing the two contradicting weights assigned by two different Scrum Masters.

Looking at the presented values makes it obvious that different people possessing the same role assigned weights in radically different ways. For example, for the *Progress*

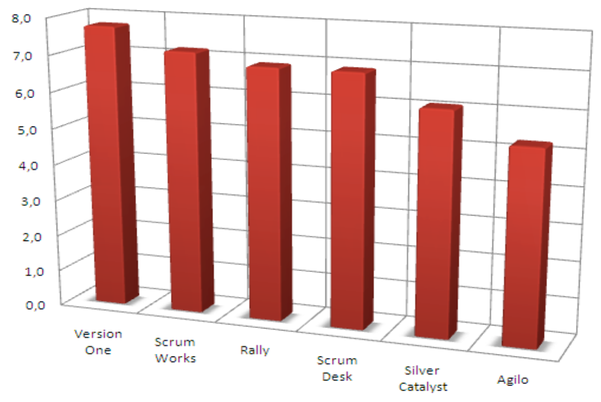


Figure 1. Summary chart of the total normalized rating U_N^t for each tool

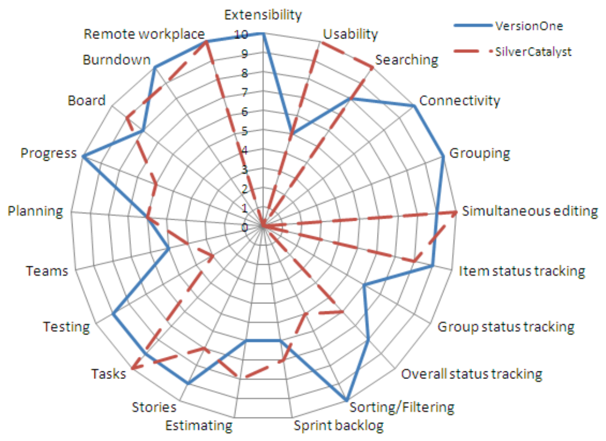


Figure 2. Comparison of the ratings of VersionOne and Silver Catalyst

TABLE III. RATINGS, TOTAL RATINGS, AND TOTAL NORMALIZED RATINGS FOR ALL CRITERIA AND ALL TOOLS

Criterion	Version One	Scrum Works	Rally	Scrum Desk	Silver Catalyst	Agilo
Extensibility	0,0	0,0	0,0	0,0	0,0	10,0
Usability	5,0	10,0	3,0	5,0	10,0	4,0
Searching	8,0	6,0	10,0	6,0	10,0	4,0
Connectivity	10,0	7,0	5,0	6,0	0,0	5,0
Grouping	10,0	9,0	7,0	0,0	0,0	4,0
Support for simultaneous editing	9,0	10,0	10,0	10,0	10,0	9,0
Story status tracking	9,0	10,0	9,0	9,0	8,0	4,0
Group status tracking	6,0	8,0	6,0	5,0	0,0	0,0
Overall status tracking	8,0	5,0	9,0	9,0	6,0	0,0
Sorting/Filtering	10,0	5,0	9,0	9,0	5,0	2,0
Sprint backlog	6,0	7,0	5,0	7,0	7,0	3,0
Estimating	6,0	5,0	8,0	9,0	8,0	6,0
Stories	9,0	10,0	7,0	7,0	7,0	5,0
Tasks	9,0	10,0	8,0	10,0	10,0	8,0
Testing	9,0	4,0	7,0	4,0	3,0	3,0
Teams	5,0	7,0	7,0	8,0	4,0	10,0
Planning	6,0	4,0	7,0	7,0	6,0	4,0
Progress	10,0	6,0	8,0	6,0	6,0	5,0
Board	8,0	10,0	0,0	9,0	9,0	6,0
Burndown	10,0	8,0	10,0	9,0	9,0	9,0
Remote workplace	10,0	10,0	10,0	10,0	10,0	10,0
Total rating	**163,0**	**151,0**	**145,0**	**145,0**	**128,0**	**111,0**
Normalized total rating	**7,8**	**7,2**	**6,9**	**6,9**	**6,1**	**5,3**

criterion (see Table 4), Scrum Master A assigned a weight of 0, while Scrum Master B assigned a weight of 10. The average weight for the *Progress* criterion is 5.3. This does not in any way reflect the company's overall need; neither does it reflect the individual needs. For instance, Scrum Master A saw no tool support need for tracking the team's progress since his team members were good at reporting the progress during stand up meetings every day, while Scrum Master B's team members were not good at reporting their progress and, therefore, it was necessary to track it via a tool. A similar conclusion can be made for the overall *Status Tracking* criterion.

It is interesting to compare the *Progress* criterion with the *Sorting/Filtering* criterion. The average ratings for these criteria were nearly the same – 5,3 and 5,7, respectively. However, unlike the greatly varying ratings assigned to the *Progress* criterion, all three Scrum Masters assigned weights of nearly the same value (5, 6, and 6) for *Sorting/Filtering*. Therefore, in this case we may draw a conclusion that the *Sorting/Filtering* criterion is of roughly the same importance to all Scrum Masters, and this is accurately reflected in the final average rating of 5,7. In contrast, the actual importance of the *Progress* criterion was lost after calculating its average value.

Calculating average weights for people of different roles, such as APMs and Scrum Masters, resulted in even less meaningful weights. Each role only saw the problems and requirements in their own area and daily work, and therefore, while rating they did not pay heed to the overall process and needs. In some cases, they made incorrect assumptions about the needs of others. For example, managers gave high ratings to virtual task boards, deeming it an important feature for the teams to have, while in reality the teams preferred to work with physical tools such as whiteboards. Thus, it became especially meaningless to calculate average ratings for different roles.

TABLE IV. WEIGHTS ASSIGNED BY THREE SCRUM MASTERS

Criteria	Scrum Master A	Scrum Master B	Scrum Master C	AVG
Extensibility	5	6	5	5,3
Usability	10	8	10	9,3
Connectivity	2	2	6	3,3
Searching	8	5	8	7,0
Grouping	10	10	8	9,3
Support for simultaneous editing	10	10	10	10,0
Story status tracking	10	4	7	7,0
Group status tracking	10	8	9	9,0
Overall status tracking	0	6	8	4,7
Sorting/Filtering	5	6	6	5,7
Sprint backlog	0	10	10	6,7
Estimating	10	10	9	9,7
Stories	0	6	8	4,7
Tasks	0	6	3	3,0
Testing	0	0	5	1,7
Teams	8	4	2	4,7
Planning	10	4	5	6,3
Progress	0	10	6	5,3
Board	0	4	4	2,7
Burndown	0	10	7	5,7
Remote workplace	0	6	8	4,7

The conclusion drawn from these results was that a similar evaluation method could not, and should not be used for determining the company's needs and for selecting tools.

VI. COMPANY OBSERVATION

In this section, we describe the results of our company observation. We first describe the status of the agile process at Company A. We then evaluate the process using the results of our interviews and our own observations.

VI.1 Status of the Agile Development Process at Company A

The overall process at Company A consisted of four main phases, (1) *Requirements definition*, (2) *Requirements management*, (3) *Project management*, and (4) *Development*. Figure 3 presents a simplified diagram of the company's process. We gathered an understanding of this process from the replies to interview *Question 1* presented in Table 2, as well as our own direct observations.

In the *Requirements definition* phase, new requirements were brought in from the customers in form of high-level specifications. They were all described according to a predetermined template and recorded in an MS PowerPoint file. For the sake of following the course of events and their related documents, we call this file the *Requirements Description File.* No specific reason was provided to why MS PowerPoint was being used. However, our impression was that the Program Managers, SPMs, APMs and Release APMs involved in the creation of requirements were accustomed to using MS PowerPoint. Once requirements had been created, they were then sent to the Release APM for approval

In the *Requirements management* phase, the new requirements were discussed, prioritized and decided upon. First, roles such as SPMs, Business Owners, APMs, UXDs, the Solution Architect, and the Release APM attended meetings during which the requirements were discussed and prioritized. To provide a basis for the meetings, the requirements from the *Requirements Description File* were put on a list that was stored in an MS Word document. We call this list *New Requirements List*. It contained a table with the requirement ID, a brief summary of some of the information taken from the *Requirements Description File*, and the requirement owner's name.

Once prioritization had been made, the prioritized requirements were manually added to a list of the existing requirements, which was stored in an MS Excel spreadsheet.

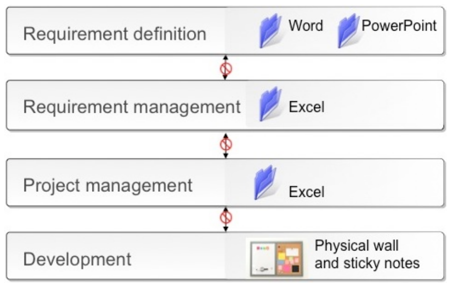


Figure 3. Overview of the company process

We call this list *Prioritized Requirements List*. It contained hundreds of requirements, along with a summary of information, such as, for instance, which requirements were defined for which customers. Still, the detailed descriptions of the requirements were stored in the *Requirement Description Files* in MS PowerPoint. Thus, at the end of this phase information about the same requirements was recorded in three different places: *Requirement Description Files, New Requirements List* and *Prioritized Requirements List*.

In the *Project management* phase, APMs and development teams broke down the requirements from the *Prioritized Requirements List* into lower-level, functional tasks. These tasks were then added to a list stored in another MS Excel spreadsheet, which we call *Product Backlog*. No links were provided to relate the backlog items to the high-level requirements, and vice versa. The backlog items were continuously updated, edited, and prioritized by different teams on different nodes. To make this possible, the *Product Backlog* was stored on a shared network drive, so that everyone on the internal company network could access it. At this point, requirements were described in four different places – the three files mentioned above and the *Product Backlog*.

Although the *Product Backlog* was accessible to everyone working on it, the company still encountered problems prioritizing the backlog items. In an ideal scenario, the prioritization should be made on a requirement level. In the company's scenario, however, this was not the case. Dependencies between different backlog items belonging to different requirements strongly affected the order in which the individual backlog items needed to be implemented. This complicated the process of prioritizing the requirements and monitoring their fulfillment. To add zest to it, some *Product Backlog* items belonged to two different requirements, creating many-to-many relationships.

In the *Development* phase, APMs together with the teams of different nodes performed Sprint planning. Here, they selected the highest prioritized items from the common *Product Backlog,* agreed on which node would implement them and included them in the team's respective *Sprint Backlog*s. The *Sprint Backlog* items were then recorded on a piece of paper affixed on a wall. They were further broken down into tasks and written on sticky notes. The information about the actual work that was done by the teams was not recorded anywhere else other than on the walls. At the end of the *Development* phase, the information about one requirement was stored in six different places, namely the four files that had been created at the end of the *Requirements management* phase, and two new places – the *Sprint Backlog* and sticky notes.

VI.2 Our Observations

The above-described scenario entailed a number of difficulties when managing and maintaining hundreds of requirements stored in different places. Using the feedback from *Question 3* in Table 2 (the question dealing with the pain areas of the company's process), we conclude that there were two main problems (1) *lack of visibility* and (2) *lack of traceability.*

Lack of visibility implied that different roles, especially the managerial ones, had no insight into the overall agile process. *Lack of traceability* implied that there was no link between the six artifacts storing information about the requirements. Both problems made it impossible to track the status of the requirements and to make sure that they had been completed.

The usage of different files for storing requirements created big difficulties. The files included hundreds of items, and it was difficult to detect similar requirements, group related requirements, as well as find their relationships, conflicts and duplicates. It was difficult to navigate among the files in order to get a complete overview of the requirements. In many cases, the contents in the files were not consistent. Changes made to one file were not always reflected in other files.

Many issues arose while using the *Product Backlog*, since it did not support simultaneous editing. Several nodes, teams, and APMs used the same *Product Backlog*, and they were thus unable to make any changes while somebody else was editing it. This naturally led to progress hindrance and frustration.

Using the answers to *Question 2* in Table 2 (dealing with the satisfaction with the current process), we discovered that the teams saw no problems with the process and its supporting tools. They were quite satisfied with it. They did not wish to change the process and to replace the physical walls with virtual task boards. This, however, was not the case with the mangers. The interviewed managers were not directly satisfied with the process and the tool supporting it.

Since the *Sprint Backlog* was only recorded on the walls, managers had no overview of the work progress in the teams. Moreover, they were usually too busy to attend the Sprint demos, which further meant that they were not always aware of what was completed and what was not completed. They did not always have good insight into the development process; they had no apprehension of team velocities, focus factors and other data. All this created difficulties in planning for future steps, managing resources, and dealing with customers.

Another difficulty faced by the managers was lack of access to release planning tools. Managers were forced to manually create release time plans by making drawings in MS PowerPoint and MS Word. To get an overview, they had to print them out on several sheets of paper and affix them on the walls. Due to the unavailability of a proper reporting tool, as well as lack of supporting data, they were also unable to create much needed reports providing various statistics on, for instance, number of incoming requirements and their rate of completion, task load on different nodes or teams, or number of requirements per customer. Finally, due to lack of support for tracking changes to requirements, managers had to manually find out who made these changes, when, and why.

In general, we conclude that except for the implementation phase, all the process phases lacked appropriate tool support for the reasons described above. All this had led to a very cumbersome and clumsy management process. Using this as feedback along with the answers to *Questions 4* and *5* in Table 2, (questions eliciting needs for change), we extracted the company's six primary tool support needs. They are all displayed in Table 5 and motivated and matched to the selected tools in the section to come.

VII. COMPANY NEEDS AND TOOL FEATURES

In this section, we describe the company's needs listed during the *Company observation* phase and presented in Table 5. We then match each need against the features offered by the tools that had been evaluated during the *Tool evaluation* phase of our research.

Need 1: Support for management levels

The first need experienced by the company was *support for management levels*. The development teams were satisfied with the used physical tools. The managers, on the other hand, lacked tool support in the three management phases – *Requirements definition, Requirements management*, and *Project management.* They needed to store all their requirements in one centralized location. They needed a tool that would enable them to track the status of the requirements throughout all the management phases as well as to create various status reports. A similar need has been observed in [30].

In order to adequately support the above-described need, it would be necessary – though by no means sufficient – for a tool to provide support for hierarchical, multi-level requirements management. The tool should make it possible to store and track all the information that is stored in the *Requirements Description,* the *New Requirements List,* the *Prioritized Requirements List*, and the *Product Backlog*.

Looking at our tool evaluation from the point of view of this need, we saw that three out of the six tools studied provided the option of grouping *Product Backlog* items. One of the tools even included an artifact equivalent to the *New Requirements List*. However, none of the tools supported the hierarchical structure of the company's process. In fact, none of the evaluated tools supported high-level requirement

TABLE V. SUMMARY OF THE COMPANY'S NEEDS

Company needs for Agile Tool Support	
1	Support for management levels
2	Simple and easy-to-use interface
3	Customized views for different roles
4	Support for the company's agile process
5	Adherence to company-specific terminology
6	Flexibility to adapt to process changes

definition, decision-making and prioritization in an agile process. Finally, none of the tools provided support for creating status reports.

The tools studied provided extensive support for the development level. All of them received a high rating in the provided options for breaking down *Product Backlog* items into tasks and for storing the tasks. They also provided detailed options for estimating the tasks and entering information regarding the work completed on the tasks. Finally, as many as five of the six evaluated tools provided virtual task boards. From the perspective of the company's need, however, these features were unnecessary and even undesirable, since the company wished to continue using physical walls for storing the *Sprint Backlog* and the team-level tasks. The company management had, however, a need to have an overview of the development phase in form of team velocities and completion of the *Product Backlog* items. This means that although there was no need to use virtual task boards, at least some information from the teams had to be channeled up for status tracking. Some of the evaluated tools had good support for status tracking, but none of them included artifacts for storing and handling the company's three levels of requirements. This made it impossible to use the tools to get a progress overview at the company.

Summing up, the tools studied focus on supporting team-level aspects of the agile process, such as virtual boards and support for Sprint planning. They do not provide enough coverage for the agile project management process that was desperately needed by the company.

Need 2: Simple and easy-to-use interface

The second need experienced by the company was *simple and easy to use interface* to be possessed by the tool. The tools used by the company were PowerPoint files, Word files, spreadsheets, slides, and sticky notes. All these are considered to be simple tools, and yet their usage became complicated because they did not adequately meet the company's needs. A similar need was reported in [31].

The company representatives expressed the need for "*just enough*" tool support, with simple, easy to use interfaces. This was valuable for the company since it had a complex structure, with multiple teams, roles, and process steps, and it was not desirable to introduce a tool that would further complicate daily work.

A requirement of simplicity implied that the tool had to be closely tailored to the company's process and it had to provide all the necessary features without providing the unnecessary ones. Our tool evaluation, however, revealed that the tools studied could not satisfy this need. They fell into two categories: (1) they were either simple and had pleasant interfaces, but did not offer advanced features, or (2) they were very complex and offered a wide array of options, but they were not easy to use. In general, we discovered that the inclusion of more features and options led to a decreased usability.

Need 3: Support for the company's agile process

The third need that the company experienced was *support for company's agile process*. The company had a complicated organizational structure and a multi-step requirements management process, where numerous people of different roles were involved.

The company wished to keep and have power over their process. It is highly undesirable for a company to be forced to introduce changes to their process in order to adopt a particular tool. Instead, the tool should be adapted to the company structure, the product it manages and the team setup. A tool should enable a workflow similar to that of the company, as well as store and display the information that is relevant to the company.

None of the tools satisfied this need. The tools studied were either simple and lightweight, or powerful and complex. The powerful and complex tools, such as *Rally*, imposed process adaptations. Even the simpler tools, such as *Silver Catalyst,* imposed their own process.

Need 4: Customized views for different roles

All roles were using the same spreadsheets and slides while working with requirements at the company. All the useful information had to be displayed in the same place. This created an overload of information for all the roles involved. For example, a UX designer did not need to see the same information as an APM performing a breakdown of *Product Backlog* items, while SPMs were mostly interested in looking at reports and charts.

The company needed to have different views for different roles in order to make their daily work simpler and more manageable. Hence, the fourth need identified concerned *customized views for different roles*.

Our tool evaluation revealed that the more advanced tools did account for a few different roles, but the views they provided and the information they displayed were not sufficient. The existing agile project management tools have failed to predict the need for all the views and custom reports that a company might need, especially in case of a large company with a large number of different roles.

Need 5: Adherence to company-specific terminology

The terms, concepts and abbreviations used at the company were quite complex and differed from the terminology used outside the company. For example, the company used the term “opportunity card” instead of “high-level requirement.” The term was used by a large number of people and in a large number of documents. Changing this term to fit a particular tool was not an option. The company needed a tool that made it possible to adhere to the terminology already in place.

Not surprisingly, the tools we evaluated imposed their own terminology. Hence, they did not fulfill this need. For example, some tools used the concept of a “feature” to describe groups of *Product Backlog* items. In the company, however, features were certain groups of functionalities, and the term did not coincide with the way it was used in agile project management tools.

Need 6: Flexibility to adapt to process changes

Changes in the process models and the ways of working were not an infrequent occurrence at the company. Adhering to the agile vision of continuous improvement, there was a drive to continuously learn from retrospectives and make process improvements. The company needed a tool that would not only support their current process, but would also accommodate process changes and an evolving company structure. Thus, the company's sixth need was *flexibility to adapt to process changes*.

Five out of six evaluated tools were of commercial availability and could not be extended to add or remove desired features. They simply did not support an evolving company process. Hence, they did not fulfill this need.

VIII. CONCLUSION

Despite the growing availability and complexity of agile tools, there is lack of tool selection guidelines and case studies. In this paper, we have attempted to shed some light on the matter by presenting the results of a case study of agile tool selection conducted at Company A – a company with a complex process and hierarchical requirements structure. As part of our research, we performed an evaluation of six agile tools available on the market using a detailed list of evaluation criteria.

During our attempt to select an adequate tool for Company A, we found out that the tool evaluation criteria, though well defined, proved to be insufficient for specifying the company's needs. We, however, do not reject evaluations of this type as an important aid in identifying needs. They have to be complemented with extensive observational studies and detailed interviews, similar to the study reported in this paper.

Even after having extracted and identified the company's needs, we had difficulties in finding an adequate tool. The tools focused more on team-level aspects of development and did not cater to the multi-layer requirements management process of the company. The tools available on the market imposed their own process and were cumbersome and difficult to use. They were not flexible enough to accommodate changes in the company's process, and they lacked support for the creation of reports. Further, the tools did not cater to the needs of all the roles present at the company, and, in most cases, imposed their own terminology. Overall, we conclude that the studied tools have not met Company A's agile project management needs.

It might be expected that other large companies with complex structures also face a similar dilemma. As future work, we plan to look into custom tools, or a combination of custom tools and tools from the market, in order to find out whether they might successfully meet the needs of such companies. We also plan to find out how current tools support distributed, multi-cultural agile environments that currently encounter many types of different problems [32].

REFERENCES

[1] M. Dubakov and P. Stevens, “Agile tools. The good, the bad and the ugly,” TargetProcess Inc., 2008.

[2] G. Dowst. *Reviewing agile process management tools. Part 1 and 2* [Online]. Available: http://consultingblogs.emc.com/gavyndowst [retrieved: October, 2012].

[3] B. Swanson (2009 Sep. 25). *Comparing open source agile project management tools.* [Online]. Available: http://olex.openlogic.com/wazi/2009/comparing-open-source-agile-project-management-tools [retrieved: October, 2012].

[4] CMC Media Inc. (2007 Apr.). *Agile Tooling: A Point, Counter-Point Discussion* [Online]. Available: http://agile.techwell.com/articles/weekly/agile-tooling-point-counter-point-discussion [retrieved: October, 2012].

[5] G. Goth, “agile tool market growing with the philosophy,” IEEE Software, 2009, pp. 88-91.

[6] Mountain Goat Software. *All products*. [Online]. Available: http://userstories.com/products [retrieved: October, 2012].

[7] S. H. Rayhan and N. Haque, “Incremental adoption of Scrum for successful delivery of an IT project in a remote setup,” in Proc. AGILE 2008 Conference, IEEE Computer Society, Toronto, Canada, 4-8 August 2008, pp. 351-355.

[8] M. Cottmeyer, “The goods and bad of Agile offshore development,” in Proc. AGILE 2008 Conference, IEEE Computer Society, Toronto, Canada, 4-8 August 2008, pp. 362-367.

[9] E. Uy and R. Rosendahl , “Migrating from SharePoint to a better Scrum tool” in Proc. AGILE 2008 Conference, IEEE Computer Society, Toronto, Canada, 4-8 August 2008, pp. 506-512.

[10] F. Cannizzo, G. Marcionetti, and P. Moser, “Evolution of the tools and practices of a large distributed Agile team” in Proc. AGILE 2008 Conference, IEEE Computer Society, Toronto, Canada, 4-8 August 2008, pp. 513-518.

[11] VersionOne Inc.. *State of Agile Development Survey 2009* [Online]. Available: http://pm.versionone.com/StateOfagileSurvey.html [retrieved: October, 2012].

[12] P. Behrens, “agile Project Management (APM) tooling survey results,” Trail Ridge consulting, December 2006.

[13] VersionOne Inc. [Online]. Available: http://www.versionone.com [retrieved: October, 2012].

[14] CollabNet Inc. *ScrumWorks.* [Online]. Available: http://www.open.collab.net/products/scrumworks/?q=scrumworks [retrieved: October, 2012].

[15] Rally Software Development Corp. *AgileZen* [Online]. Available: http://agilezen.com [retrieved: October, 2012].

[16] ScrumDesk Co. [Online]. Available: http://www.scrumdesk.com [retrieved: October, 2012].

[17] Silver Stripe Software Pvt. , Ltd.. *Silver Catalyst* [Online]. Available: http://toolsforagile.com/silvercatalyst [retrieved: October, 2012].

[18] Agile42 Gmbh. *Agilo* [Online]. Available: http://www.agile42.com/cms/pages/agilo [retrieved: October, 2012].

[19] VersionOne Inc. *Integrations* [Online]. Available: http://community.versionone.com/sdk/Documentation/Integrations.aspx [retrieved: October, 2012].

[20] Bugzilla org. [Online]. Available: http://www.bugzilla.org/ [retrieved: October, 2012].

[21] Atlassian Pty Ltd. *JIRA issue and project tracking* [Online]. Available: http://www.atlassian.com/software/jira [retrieved: October, 2012].

[22] Eclipse org. [Online]. Available: http://www.eclipse.org [retrieved: October, 2012].

[23] Microsoft Inc. *MS Excel.* [Online]. Available: http://office.microsoft.com/en-us/excel [retrieved: October, 2012].

[24] Rally Software Development Corp. *Rally Connectors* [Online]. Available: http://www.rallydev.com/agile_products/integrations/connectors/ [retrieved: October, 2012].

[25] Microsoft Inc. *Visual Studio Team Foundation Server* [Online]. Available: http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx [retrieved: October, 2012].

[26] Mantis Bug Tracker. [Online]. Available: http://www.mantisbt.org [retrieved: October, 2012].

[27] Subversion SVN [Online]. Available: http://subversion.apache.org [retrieved: October, 2012].

[28] Trac [Online]. Available: http://trac.edgewall.org [retrieved: October, 2012].

[29] Wikipedia. *User experience design* [Online]. Available: http://en.wikipedia.org/wiki/User_experience_design [retrieved: October, 2012].

[30] M. Kajko-Mattsson. “Problems in agile trenches”. In Proc. of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '08). ACM, New York, NY, USA, 2008, pp. 111-119.

[31] G. Azizyan, M. K. Magarian, and M. Kajko-Matsso “Survey of Agile Tools Usage and Needs” in Proc. AGILE 2011 Conference, IEEE Computer Society, Salt Lake City, UT, USA, 8-12 August 2011, pp. 29-38.

[32] M. Kajko-Mattson, G. Azizyan, and M. K. Magarian "Classes of distributed Agile problems" AGILE 2010 Conference, IEEE Computer Society, Orlando FL, USA, 9-13 August 2010, pp. 51-58.

Framework for Better Efficiency of Automated Testing

Work-in-Progress Paper

Martin Filipsky, Miroslav Bures and Ivan Jelinek
Department of Computer Science and Engineering
Czech Technical University
Prague, Czech Republic
{filipma2, buresm3, jelinek}@fel.cvut.cz

***Abstract*—The paper introduces our design of a framework for test automation, which utilizes both recording and scripting approaches to help quality-assurance engineers with their test automation efforts. Characteristic problems of test automation are a low efficiency and/or high maintenance costs. The framework cuts down those drawbacks using a technique of abstraction, a clever structure of test cases, and reusable common test case retrieval from recorded tests. Finally, the approach is robust to a test script ageing and is technology-independent as well as testing-tools independent.**

Keywords-automated testing; functional tests; quality assurance; test efficiency; test recording

I. INTRODUCTION

With the increasing importance and complexity of information systems, software vendors and system integrators are facing numerous challenges. Current technologies are rapidly changing. New technologies and ideas like Cloud computing [9], Virtualization [5], or Software as a service [4] are coming; but, budgets intended for software products and IT projects are cut down. On the other hand, customers require more and more features, a better performance, a higher reliability, and a first-rate availability for less money. Who wants to compete and be a market leader, has to be more efficient than his competitors. Since testing costs represent a significant part of total costs of development, we focused on areas where we see a potential to improve a testing process using test automation.

The area of automated testing of software applications is facing a number of issues and challenges like an insufficient expertise to automate tests, technological issues, and/or demand on a fast and effective test development and execution. One of the most serious problems revolving around test automation is that test scripts are getting inaccurate and obsolete with changes in an application under test. As the result, automated testing is limited mostly to regression tests, smoke tests and performance testing. A utilization of automated testing in a sense of a replacement of manual testing is not quite often.

Even solutions of some issues (automated test development/debugging, or maintenance [18], [13]) may increase the efficiency of the process of test automation. Moreover, a number of defects would decrease in final applications released to production environments. Some techniques used to speed up the process of test automation and to increase the efficiency are already adopted by many quality-assurance (QA) teams, like using Mockups [20] or generating test cases from application models [23]. Mockups may be very useful when QA teams have enough time to prepare and develop tests against an application prototype. Agile software development methodologies [17] bring additional requirements on a development of automated tests. Test recording approaches are very useful in such cases because they do not require a time-consuming preparation. Testers can start to develop automated tests immediately. Furthermore, if those test recordings are transformed into abstract tests organized in test suites, the resultant effort and costs will decrease significantly.

The main goal of our research is a definition and a validation of a meta-model among requirements on automated tests, an application under test (AUT) and test scripts to record test cases using the defined meta-model and to create a structure of tests having common parts of the recorded tests broken down into reusable pieces for an easier and cheaper test maintenance.

This paper is structured as follows: We start by giving an overview on related research in Section 2. In Section 3, we describe the concept of the framework. In Section 4, we conclude with a summary and an outlook.

II. RELATED WORK

Many research teams are interested in test automation of various kinds like unit testing, functional, GUI or regression testing, and performance testing. Unit test automation is being successfully theoretically covered. Some approaches focus on a generation of test data [8] or a generation of test cases [23] from models of AUTs. For example, Xu [23] introduced an approach based on high-level Petri nets as finite state test models for an automated test generation and a test execution. On top of that, he developed a model-based Integration and System Test Automation tool. This premise may be seen as a drawback in agile software methodologies where specifications and models frequently do not exist at the moment when the functional test automation is required. For a generation of test cases, a detailed model of AUT including use case diagrams and user scenarios is required prior the testing process can start. Those generated test cases might not be possible to execute without additional pre-processing. Missing object IDs may prevent from that as the objects cannot be identified in the AUT. For example, if a development team uses dynamic objects with insufficient unique object properties or with dynamically generated IDs. Approaches were introduced for web application modelling in [2], [16], and [22].

Koopman, Plasmeijer and Achten [15] introduce a model-based testing system for on-the-fly testing of thin-client web applications specified by Extended State Machines. The approach is proposed for plain HTML (no Flex, no JavaApplets). They do not discuss more general solutions for rich clients and/or other platforms, like mainframe panels (applications for terminal emulators), which leads to technological limitations (e.g., different objects, and application behavior) of the approach. Beek and Mauw [1] present an approach for a conformance, black-box testing of thin internet applications. Besson, Beder and Chaim [3] introduced an approach of test automation for acceptance testing. In general, approaches based on Finite State Machines (FSM) are quite often as presented Jia and Liu [14].

In comparison to test automation efforts based on model-based approaches, where test scripts are generated from models, Stepien, Peyton and Xiong [21] describe a testing of web applications using TTCN-3 language [6] and [19] which is a specification-based approach. An XML specification-based approach of testing of web applications is presented by Jia and Liu [14]. The specification-based approaches [6], [10], [19], and [21] are close to our intentions, but they did not utilize the test recording in their approaches.

The mentioned approaches are either limited by a technology or they require a model, or another detailed specification of AUT. In comparison to the mentioned approaches, we are focusing on a feature to build a structure of test cases on-the-fly while the user is recording tests. Furthermore, the proposed framework recognizes input data and uses them as test parameters. Both the features are a key to a reusability of tests or parts of tests, which might significantly decrease total costs of test maintenance.

García, Dueñas, and Parada introduced recently a couple of approaches for automated functional testing based on the navigation of web applications in [10], [11], and [12]. Their concept is to automate functional tests using UML diagrams [7] as an input for an automated test case generation driven by the navigation in the AUT. Compared with our proposed technique, a model or at least a part of the model of the AUT has to be available before the testing actually starts. They also presented an alternative to the test case generation for agile strategies. They can record tests in order to skip a phase of formal design. Unlike the presented approach, we lay emphasis on the feature of a detection of reusable test parts.

III. PROPOSED SOLUTION

Standard approaches of functional test automation are usually based on:

1. A plain test recording of test cases.
2. Test scripting using a programming language.
3. An automation framework.

A facility for test recording records a user activity while testing AUT and the resultant test is captured in any

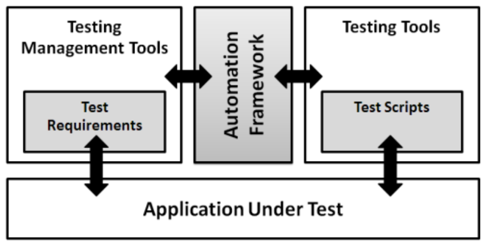


Figure 1. High-level architecture of the proposed solution.

programming language, e.g., in Java. If an advanced technique is used like an object repository, the tests are relatively fine but the maintenance is getting more difficult in comparison to projects utilizing scripting and/or different automation frameworks (code redundancy, no code optimization, object duplicity, and usually no parameters among tests).

Second approach is typically for experienced QA engineers. It is time-consuming and requires being well prepared. On the other hand, testers can fully utilize a power of object oriented programming and can develop highly reusable code (tests) with low maintenance costs.

The automation frameworks are driven by data (a flow is controlled by input data), by keywords (the flow usually does not depend on input data, test scripts completely control a test run) and by a model (the test flow depends on a model of AUT). The model-driven frameworks are worth in cases where the system changes dramatically and new test scripts can be easily regenerated from the model. Hybrid approaches are common (e.g., Data-Keyword, but not Recording-Scripting), and they combine the best features of single approaches.

The challenges described above lead us to several areas of our interest. We are working on a definition of a meta-model of test cases which is a key premise of an efficient test recording. The meta-model will be used in a process of building a smart structure of tests from the recorded test cases. Another field of our interest is a reusability of recorded tests. We are working on algorithms that will identify common test parts in the structure of tests and reorganize them.

Our intention is to integrate solutions of single problems in one test automation framework (Fig. 1) among test requirements, testing tools and AUTs. The framework is intended to be modular as well as platform-independent. We are focusing on a utilization of benefits of both the recording and scripting approaches (i.e., fast test automation and good maintenance costs). There are two options how the automation framework should support user efforts to automate test cases (Fig. 2). Either the user does not have a test case automated yet or the user has already automated a test case. In the first case, the user records the test case, which is converted on-the-fly into the meta-model, using the recording facility. All relevant objects (buttons, links, data, strings etc.) are currently captured into an object repository without additional duplicities. In the second case, the automation framework records a new sequence of

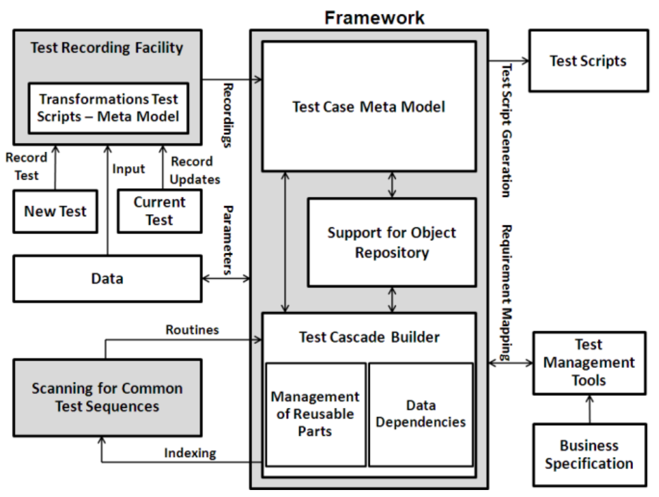


Figure 2. The concept of the framework based on test recording.

user activities while the user is executing altered parts of the test case. The user selects a relevant part of the test case to be updated and the new sequence of recorded steps is mapped into the current test and all relevant parts of other recorded tests, which use the updated part. If the user records more than two test cases in the test suite, the framework detects common parts, which are typically frequent and repeating activities like a login to the AUT. Therefore, we let the user record the complete test suite in order to run scanning algorithms, which will detect common test sequences in the recorded test cases. The detected common test sequences can be excluded from the test suites afterwards and represented as new subunits of test cases as it is presented on Fig. 3.

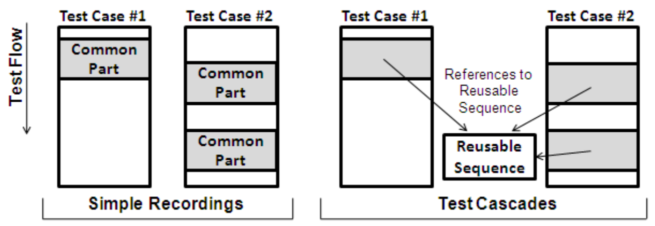


Figure 3. Test Cascades. Gray parts identified by scanning algorithms represent the common parts in test cases.

For instance, if each test case contains a login to a system, the procedure of logging in can be extracted and called from all test cases automatically. One change in the common test sequence (reusable part) takes effect in all calling tests. An example is shown on Fig. 4. Since we need to get an organized structure of test cases with reusable test parts for a better test maintainability, we have to find longest common subsequences of user activities in the test suite. In other words, we have to find a mapping of steps among all tests in the test suite. If such a mapping exists, those steps can be excluded from tests and the new reusable part, which is referenced from these tests, can be created. Obviously, a problem might to find a suitable level of a step count in the sequence. Remaining problems to be solved are questions

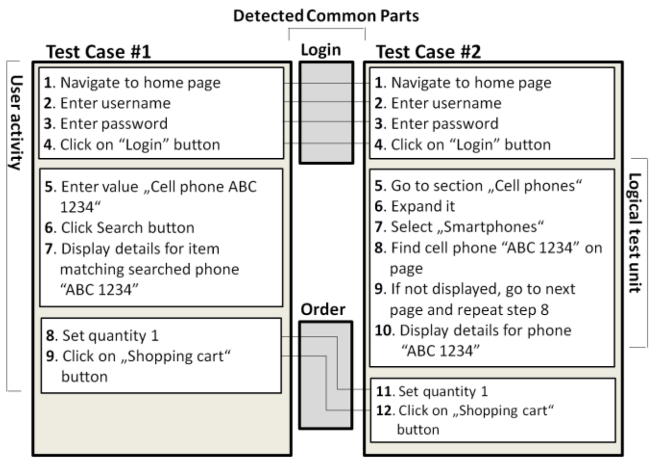

Detected Common Parts
Test Case #1
Login
Test Case #2
User activity
1. Navigate to home page
2. Enter username
3. Enter password
4. Click on "Login" button
5. Enter value „Cell phone ABC 1234"
6. Click Search button
7. Display details for item matching searched phone "ABC 1234"
8. Set quantity 1
9. Click on „Shopping cart" button
Order
1. Navigate to home page
2. Enter username
3. Enter password
4. Click on "Login" button
5. Go to section „Cell phones"
6. Expand it
7. Select „Smartphones"
8. Find cell phone "ABC 1234" on page
9. If not displayed, go to next page and repeat step 8
10. Display details for phone "ABC 1234"
11. Set quantity 1
12. Click on „Shopping cart" button
Logical test unit

Figure 4. An example of mapping steps in common test sequences between two test cases.

of handling data dependencies, a problem of a test parameterization and a problem of recursive calls to reusable units.

In addition to the recording, the user has still an option to design test cascades manually (consider test cascades as an automatically generated structure of test cases from test recordings) and to design and script test cascades in a domain-specific language (DSL). In DSL, we specify the test cases from the end user's point of view and it enables to work with the meta-model of test cases. Syntax of DSL comes out from simple English, but we are also planning a graphical version of the DSL, which will be identical to the standard text version represented by Tab. 1.

TABLE I. AN EXAMPLE OF THE DOMAIN-SPECIFIC LANGUAGE FOR A TEST CASE REPRESENTATION

Object Type	Object Name	Action	Parameters	Recovery Scenario
Browser	MyBrowser	Open	www.cvut.cz	CloseIfNot Available
Button	Submit	Click		StopTest
Table	MyTable	Validate		StopTest
TextBox	Search	Set	Test automation	SkipStep

A requirement on simple English is beneficial for non-experienced QA engineers who do not know standard programming languages like Java. Thus, they can immediately start to alter their recorded test cases. The second benefit of the DSL is a fact that a complexity of the solution will be hidden for end users. Objects are stored in an object repository. For this object repository, storage available in testing tools like, e.g., HP QuickTest Professional can be used. An alternative is to use an internal object repository of the framework.

The automation framework gives the QA engineers an option of inspection that helps them to detect common mistakes in the design of automated tests. For example, web applications may have different responses depending on system loads, which lead usually to synchronization issues.

The concept of test cases represented by the meta-model with the test cascades provides a robust solution to changes in the application. For example, if a new step is added into a current business process or if an object is altered in the AUT, a modular structure of test cascades supports an update in one place, which will take effect in all places. The higher level of abstraction of test cases together with a support of generation of test scripts for different testing tools helps to prevent tests from test script ageing.

IV. Conclusions and Future Work

We carried out a research of the current state-of-the-art and we have found out that the problem of building reusable test cascades from test recordings is covered insufficiently. There are either approaches for on-the-fly testing (usually limited by a technology like plain HTML), e.g., [1], [3] or [15] or approaches requiring to prepare a model of AUT like in [6], [19] or [20]. A more general concept utilizing the test recording is missing. Based on that, we have designed a framework for automation of functional tests with no need to have a model of the AUT. We have designed the structure of the meta-model, used in the proposed solution.

Currently, we are working on a detailed design of the meta-model. In parallel with that, we are conducting a research of a problem of transformation test recordings i.e., the transformation of test scripts from standard programming languages to the meta-model. Besides that, we are dealing with a problem of seeking out the longest common test sequences. The goal is to build test cascades from recordings with no need of a post-processing. As a consequent task, we are going to solve a problem of test parameterization i.e., to detect input parameters and dependencies among them.

Finally, we are preparing to conduct experimental observations on real industrial projects with comparisons of costs and results of the manual testing, the conventional automated testing using the test recording and/or the plain test scripting, and the automated testing using the proposed automation framework.

References

[1] H. M. A. van Beek and S. Mauw, “Automatic Conformance Testing of Internet Applications”. In *Lecture Notes in Computer Science*, 2004, Volume 2931, Formal Approaches to Software Testing, Page 1106.

[2] H. M. A. van Beek, “Specification and Analysis of Internet Applications”. In *PhD thesis*, Technical University Eindhoven, The Netherlands, 2005. ISBN 90-386-0564-1.

[3] F. M. Besson, D. M. Beder, and M. L. Chaim, “An Automated Approach for Acceptance Web Test Case Modeling and Executing”. *Lecture Notes in Business Information Processing*, 1, Volume 48, Agile Processes in Software Engineering and Extreme Programming, Part 2, Pages 160-165.

[4] G. Blokdijk, "SaaS 100 Success Secrets - How Companies Successfully Buy, Manage, Host and Deliver Software As a Service (SaaS)", Emereo Pty Ltd. USA, 2008. ISBN 978-0-9804-7164-9.

[5] M. Cafaro and G. Aloisio (Eds.), "Grids, Clouds and Virtualization". 1st Edition., Springer, 2011. ISBN 978-0-85729-049-6.

[6] ETSI ES 201 873-1: “The Testing and Test ControlNotation version 3”, Part1: TTCN-3 Core notation, V3.2.1, February 2007.

[7] M. Fowler, "UML Distilled: A Brief Guide to the Standard Object Modeling Language". Addison-Wesley Professional, 3rd Edition, 2003. IBSN-13: 978-0321193681.

[8] S. Fujiwara, K. Munukata, Y. Maeda, A. Katayama and T. Uehara, "Test data generation for web application using a UML class diagram with OCL constraints". In *Innovations in Systems and Software Engineering*, 2011, volume 7, Number 4, Pages 275-282.

[9] B. Furht and A. Escalante (Eds.), "Handbook of Cloud Computing". 1st Edition., Springer, 2010. ISBN 978-1-4419-6524-0.

[10] B. García, “Contribution to the Automation of Software Quality Control of Web Applications”. In *PhD thesis*, Universidad Politécnica de Madrid, Spain, 2011. ID code 9017.

[11] B. García, J. C. Dueñas, “Automated Functional Testing based on the Navigation of Web Applications”. WWV 2011, Reykjavik, Iceland, June 2011.

[12] B. García, J. C. Dueñas, H. A. Parada, “Functional Testing based on Web Navigation with Contracts”. IADIS International Conference (WWW/INTERNET09). Rome, Italy. Nov. 2009.

[13] D. Hoffman,"Cost Benefits Analysis of Test Automation". STAR West, 1999.

[14] X. Jia and H. Liu, “Rigorous and Automatic Testing of Web Applications”. In *Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA 2002)*, pages 280–285, Cambridge, MA, USA, Nov. 2002.

[15] P. Koopman, R. Plasmeijer, and P. Achten, “Model-Based Testing of Thin-Client Web Applications”. *Lecture Notes in Computer Science*, 2006, Volume 4262, Formal Approaches to Software Testing and Runtime Verification, Pages 115-132.

[16] F. Lanubile and T. Mallardo, “Inspecting Automated Test Code: A Preliminary Study”. In *Lecture Notes in Computer Science*, 2007, Volume 4536, Agile Processes in Software Engineering and Extreme Programming, Pages 115-122.

[17] R. C. Martin, "Agile Software Development, Principles, Patterns, and Practices". 1st Edition., Prentice Hall, 2002. ISBN: 978-0135974445.

[18] B. Pettichord, "Seven Steps to Test Automation Success". STAR West, 1999.

[19] R. L. Probert, B. Stepien, and P. Xiong, “Formal testing of web content using TTCN-3”. In *TTCN-3 User Conference 2005*, June 2005.

[20] J. M. Rivero, G. Rossi, J. Grigera, J. Burella, E. R. Luna, and S. Gordillo, “From mockups to user interface models: an extensible model driven approach”. In *Lecture Notes in Computer Science*, Volume 6385, Pages 13-24, 2010.

[21] B. Stepien, L. Peyton, and P. Xiong, “Framework testing of web applications using TTCN-3”. In *International Journal on Software Tools for Technology Transfer (STTT)*, 2008, Volume 10, Number 4, Pages 371-381.

[22] Y. Wu and J. Offutt, “Modeling and Testing Web-based Applications”. GMU ISE Technical ISE-TR-02-08, Information and Software Engineering Department, George Mason University, Fairfax, USA, Nov. 2002.

[23] D. Xu, “A Tool for Automated Test Code Generation from High-Level Petri Nets”. In *Lecture Notes in Computer Science*, 2011, Volume 6709, Applications and Theory of Petri Nets, Pages 308-317.

MobiTest:
A Cross-Platform Tool for Testing Mobile Applications

Ian Bayley, Derek Flood, Rachel Harrison, Clare Martin
Oxford Brookes University,
[ibayley, derek.flood, rachel.harrison, cemartin]@brookes.ac.uk

***Abstract*— Testing is an essential part of the software development lifecycle. However, it can cost a lot of time and money to perform. For mobile applications, this problem is further exacerbated by the need to develop apps in a short time-span and for multiple platforms. This paper proposes MobiTest, a cross-platform automated testing tool for mobile applications, which uses a domain-specific language for mobile interfaces. With it, developers can define a single suite of tests that can then be run for the same application on multiple platforms simultaneously, with considerable savings in time and money.**

Keywords – Mobile Application; Testing; MobiTest.

I. Introduction

The increasing prevalence of mobile applications (hereafter, apps) continues as the use of mobile phones becomes ubiquitous. By the end of 2010, there were an estimated 5.3 billion mobile subscriptions worldwide. In developed countries there are on average 116 subscriptions for every 100 inhabitants [1].

The applications that facilitate this, known as apps, are typically developed in a relatively short time span and on low budgets, often because the unit price of the app is very small or zero. This appears to greatly diminish the usability of many of the apps that are sold to users. This is unfortunate because a recent survey [2] has identified usability as being one of the most important factors when selecting a mobile app.

The annual cost of an inadequate infrastructure for testing in the US is estimated to range from $22.2 billion to $59.5 billion [3]. This cost is partly borne by users in the form of strategies to avoid and mitigate the consequences of errors. The remainder is absorbed by the software developers themselves, who have to compensate for inadequate tools and methods. The absorbed cost is even higher when one takes into account the damage that low software quality can bring to the reputation of the producer.

The problems noted above are further exacerbated by the need to target multiple platforms at once. In particular, a test suite for one platform must be rewritten for any other platform for which it is required. This problem has been addressed in the desktop domain through the use of the USer Interface eXtensible Markup Language (USIXML) [12], which allows developers to create a user interface using a common language that can then be translated to any platform.

This paper proposes a multi-platform testing tool that takes a description of the tests to be performed on an app and generates a test suite for every platform on which the app is to be tested. Consequently, the tests will only need to be specified once. They are described in a simple language, specialised to the domain of mobile devices. Here, we concentrate on GUI testing; but, the ideas expressed here could be extended to other forms of testing at a later date.

The rest of this paper is structured as follows. Section II details the related work of this research. Section III outlines our research objectives. Section IV provides an overview of the MobiTest tool. Section V highlights some of the challenges for implementation. In Section VI, the plan for progression is detailed and Section VII concludes this paper.

II. Related Work

A. Software Testing

In *The Mythical Man Month*, Brooks [4] says that he assigns half of his development time for testing. This includes both component testing (of individual elements of the system) and system testing (of the complete system). His advice highlights the importance of testing; since, if it is not done adequately, the results can be very serious or even (in safety critical systems) fatal.

The waterfall model [5], one of the first software development methodologies, proposed that the testing phase should happen after the implementation phase has been completed. In contrast, Beck [6] proposes that the two phases be more tightly coupled, advocating the use of Test Driven Development (TDD).

TDD involves the writing the tests before writing the code, then executing the tests, and then fixing the code if the test has failed. This enables the developer to know exactly where the failing code is (as code is written in small increments). It also forces the developer to think continually about the design of the system. The collection of tests thereby accumulated can be run automatically whenever retesting is required.

George and Williams [7] found that TDD produced software that passed 18% more black box tests than software built using the waterfall model. However, this higher percentage comes at the cost of development time, which is longer by 16%.

Whichever approach is adopted, the use of automation reduces the time taken for testing. The alternative of manual testing is not only time-consuming, but also error prone.

B. *Mobile applications*

Mobile applications are different from traditional applications in several ways. They are adversely affected by the limitations of mobile devices, some of which have been highlighted by Zhang and Adipat [8] as follows:

- **Mobile Context**: When considering mobile applications the user is not tied to a single environment. The environment will also include interaction with nearby people, objects and other elements which may distract a user's attention.
- **Connectivity**: With mobile devices connectivity is often slow and unreliable and therefore will impact the performance of mobile applications which utilise these features.
- **Small Screen Size & Different Display Resolution**: In order to provide portability mobile devices contain very limited screen size and so the amount of information that can be displayed is drastically reduced.
- **Limited Processing Capability and Power**: In order to provide portability, mobile devices often contain less processing capability and power. This has the effect of limiting the functionality of applications for mobile devices.
- **Data Entry Methods**: The input methods available for mobile devices are restricted and require a certain level of proficiency. This problem increases the likelihood of erroneous input and decreases the rate of data entry.

Thus, mobile applications typically contain less functionality than traditional desktop applications. This is mainly due to the limitations of the platform, but is also affected by the context in which these applications are used. Mobile applications are designed to be used while on the move, and, as such, complex interactions are undesirable as this negatively affects usability.

In addition to this, mobile applications tend to be developed in a short period of time. This has been facilitated by the availability of better source libraries and development tools for creating mobile apps.

III. RESEARCH AIM

The aim of this research is to develop a mobile application testing tool that can be applied to all mobile platforms. As each mobile platform contains different components, it is necessary to first understand the components on each platform and how these relate to one another. In order to do this, the following two research questions (RQ) have been defined:

- RQ1: What components are available on each mobile platform?
- RQ2: Which of these components are common across the platforms?

To answer RQ1, a thorough examination of each of the mobile platforms will be performed. During this examination each of the components together with their associated events and attributes will be identified. These will then be compiled into a comprehensive profile of the platform.

Using the platform profile produced by RQ1, RQ2 will be answered through a comparison of these profiles. This RQ will aim to identify how the components on one platform relate to those on the other platforms. For example, the TextView component on Android [9] is equivalent to the Label component on the iOS platform [11].

Additionally, a third research question has been defined to investigate how these components can be combined into a platform independent testing tool.

- RQ3: How should these components be modelled in a platform independent testing tool?

By investigating the third research question, we will bring together all components from all platforms into a single platform independent representation. This is in contrast to USIXML as RQ3 incorporates all components not just a subset of them. The common components identified in RQ2 will have a single representation with a mapping to the concrete components used by the underlying platforms. Using this representation it will then be possible to construct the platform-independent testing tool, which we call MobiTest.

IV. MOBITEST

The MobiTest tool is designed to address some of the difficulties associated with the automated testing of mobile applications, by using a single set of unit tests to test the application on multiple platforms.

The initial version focuses on testing through the interface as this should be similar (although not exactly the same) on all platforms. In this way, the need for platform-specific code can be minimised. In this section we present a sample app on which MobiTest can be used, and then outline the proposed system architecture.

A. *Sample Application*

The Log In screen illustrated in Figure 1, which could be used on a number of applications, such as apps for logging medical data, invites the user to enter a username and password, which is checked against a database, and displays a message indicating whether these credentials have been accepted or not.

Assuming that ("ian", "brookes") is a valid (username, password) pair, a possible test of this app is as follows:

1. click in the username text field
2. press the keys 'i', 'a', 'n'
3. click in the password text field
4. press the keys 'b', 'r', 'o', 'o', 'k', 'e', 's'
5. click the OK button
6. assert the text component of the lower label is "password accepted"

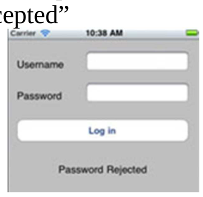


Figure 1. Sample Log In screen to be tested through MobiTest

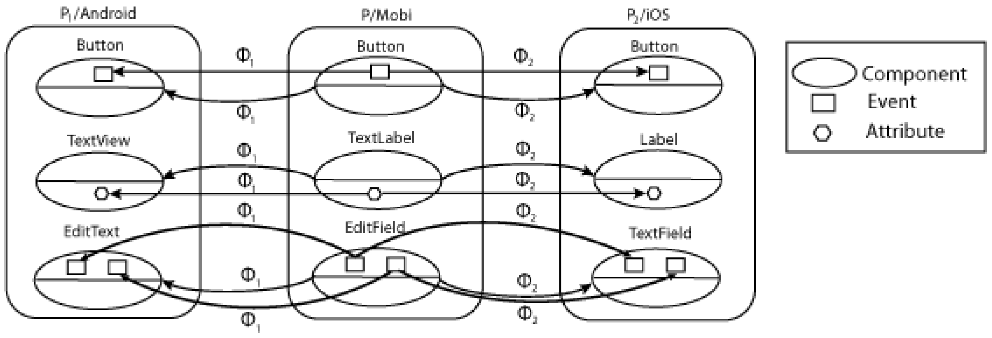


Figure 2. MobiTest System Architecture

When an assertion is false for any platform, this is reported to the user of the tool so that they can take action to correct the apparent error in the program. This is just one test that may be run on this application. In practice many more tests will be required. The benefit of MobiTest is that tests only need to be specified once. The test suites will be generated automatically for each platform, be it iOS, Android or Blackberry.

B. System Architecture

To generate automated tests for concrete platforms, such as Android or iOS, a virtual platform (called Mobi) will be defined. For Mobi, a number of GUI components will be defined through the answer to RQ3. For each such component, a list of valid attributes and events will be defined. Each concrete platform will have its components defined in a similar way. The available GUI components vary from platform to platform and even where the same component is available, the name may be different. Let P_i for i in 1..n denote the n different concrete platforms. To account for the naming differences, a function Φi can be defined that maps each Mobi component to its realisation in Pi. A similar function Φi maps each component attribute and event to its realisation in P_i. In the diagram below, n=2, P_1 represents Android and P_2 represents iOS.

Similarly, let p_i denote the set of actual components in the app written for platform P_i. Each version has six such components, as indicated in Figure 3.

Once a set of common components p is identified, a mapping φ_i from the components of *p* to those of p_i can thereby be deduced.

The tool MobiTest will operate as follows:

i. from the layout XML files of each version of the app, MobiTest will deduce the mappings φ_i and insert them into an empty XML file `tests.xml`

ii. the user will then add test cases in the form event$^+$ assert$^+$ where event and assert are given as tags with attributes and values in the normal manner and X^+ signifies one or more occurrences of X. It is anticipated that given the restricted nature of the language, GUI support can make this process exceptionally straightforward. This will be a major advantage, as it focuses attention on the interface which is unusual for conventional unit testing.

iii. MobiTest will produce a test class for each platform. In it, for each test case, MobiTest will translate each event into a piece of code that triggers that event and, similarly, each assert into a piece of code that tests that assert. This will be done using the definitions of φ_i from the tests.xml file to identify the components and using Φ_i to determine the events and attributes

iv. MobiTest will then run each test class on its associated platform, and present the test report to the user.

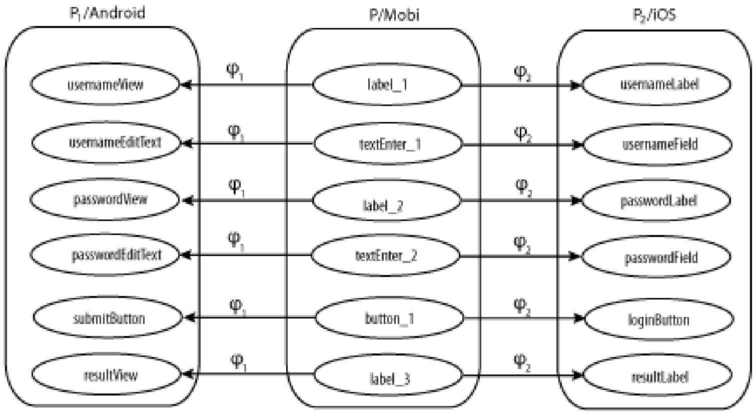


Figure 3. MobiTest view of the Log In screen

V. CHALLENGES

The creation of this tool presents a number of challenges:

- **Use of XML to specify components:** although iOS and Android can specify their layouts with XML, it may be that some platforms do not. In that case, it will be necessary to parse the code to obtain a list of the components used.
- **Incorrect assumptions about layout:** for example, suppose there are iOS and Android layout XML files that both specify two buttons. Based on the order in which components are specified in the file, it will be assumed that the first button from one file corresponds to the first button from the other. This assumption may not be valid.
- **Conflicting guidelines:** both iOS and Android provide a set of guidelines for user interfaces. These guidelines are not always compatible. Furthermore, adherence to such guidelines is often a requirement for the app to be distributed through the app store.
- **Platform-specific features**: each mobile platform has a number of components unique to that platform. For example, the “back” button, to return users to the previous screen, is physical for Android devices but it is a GUI component on iOS. The MobiTest tool will need to be able to identify these features and allow users to access and test them.
- **Inconsistent number of screens:** a single screen on one platform may correspond to multiple screens on another. MobiTest will therefore need to allow mappings between components on an application level, rather than at screen level.

VI. PLAN FOR FUTURE WORK

1. Determine components, events and attributes for a number of platforms by creating a compatibility matrix which identifies which components are available on which platform (RQ1) and how they correspond to components on other platforms (RQ2). For example, a Picker in iOS has no equivalent in Android but the ListView provides similar functionality.

2. Define the virtual platform Mobi and the functions Φi (RQ3). To help with this, an on-going study of existing multi-platform applications will be used to provide insight into how existing applications are created for cross-platform use and to help identify common conventions that are used in this context.

3. Write MobiTest for one platform, Android. To do this, a comprehensive examination of existing unit testing tools will be performed. This examination will focus mainly on the Android testing API [9], a specialisation of JUnit [10], and Logic Unit Tests for testing iOS applications [11]. Once this has been done, MobiTest will be generalised to multiple platforms.

VII. SUMMARY

This paper has proposed MobiTest, a testing tool for cross-platform mobile application development, which uses a domain-specific language for mobile interfaces. Short development cycles and the wide range of platforms mean that time available for testing is limited when developing applications for mobile devices. MobiTest will address this issue by allowing developers to specify a single set of tests for applications that can then be used with each platform on which the application is developed.

Conflicting guidelines and platform specific features are just some of the challenges when developing such a platform. If these challenges can be addressed, testing of mobile applications can be simplified and performed more easily leading to higher quality mobile applications and a much more enjoyable, satisfying and effective user experience.

Although this approach may not solve all of the issues associated with automated testing of mobile applications, we believe that it will help to address issues specifically relating to application development across multiple platforms.

VIII. ACKNOWLEDGEMENTS

This research is supported by Oxford Brookes University and the Science Foundation Ireland (SFI) Stokes Lectureship Programme, grant number 07/SK/I1299, the SFI Principal Investigator Programme, grant number 08/IN.1/I2030 (the funding of this project was awarded by Science Foundation Ireland under a co-funding initiative by the Irish Government and European Regional Development Fund), and supported in part by Lero - the Irish Software Engineering Research Centre (http://www.lero.ie) grant 10/CE/I1855.

IX. REFERENCES

[1] ITU, "The world in 2010 ICT Facts and Figures," ITU, 2010.
[2] D. Flood, R. Harrison, D. Duce, and C. Iacob "Using Mobile Apps: Investigating the usability of mobile apps from the users perspective," International Journal of Mobile HCI (In Press) 2012.
[3] G. Tassey, "The Economic Impacts of Inadequate Infrastructure for Software Testing," National Institute of Standards and Technology 2002.
[4] F. P. Brooks, *the mythical man-month*: Addison-Wesley Publishing Company, 1982.
[5] W. Royce, "Managing the Development of Large Software Systems," in *WESCO*, 1970.
[6] K. Beck, *Extreme Programming Explained: Embrace Change*: Addison-Wesley, Pearson Education, 2000.
[7] B. George and L. Williams, "An initial Investigation of Test Driven Development in Industry," in *ACM Symposium on Applied Computing*, Melbourne, FL, 2003.
[8] D. Zhang and B. Adipat, "Challenges, Methodologies, and Issues in the Usability Testing of Mobile Applications," *International Journal of Human-Computer Interaction,* vol. 18, pp. 293 - 308, 2005.
[9] A. D. Guide, "http://developer.android.com/." vol. 2011, 2011. (accessed 25/09/2012)
[10] JUnit, "http://www.junit.org/" 2011. (accessed 25/09/2012)
[11] i. D. Library, "http://developer.apple.com/." vol. 2011, (accessed 25/09/2012)
[12] http://www.usixml.org (accessed 25/09/12)

Requirement-based software testing with the UML: A systematic mapping study

Nesa Asoudeh
Carleton University, Dept. SCE
Ottawa, Canada
nasoudeh@sce.carleton.ca

Yvan Labiche
Carleton University, Dept. SCE
Ottawa, Canada
labiche@sce.carleton.ca

Abstract- Our goal is to determine the current state of the art in requirement based testing in a UML context. We combined an automated search in digital libraries with a manual search in related journal and conference venues. The search resulted in about 1,300 papers. After applying inclusion/exclusion criteria, we selected 100 papers as our final set of primary studies. Classification results based on several criteria lead us to interesting observations such as: A small proportion of the primary studies evaluate techniques through experiments and one-third of the techniques are simply illustrated with an example; More advanced selection criteria exist in literature than those used in primary studies.

Keywords- Requirement based testing; systematic mapping study; model driven development; requirement engineering; Unified Modeling Language.

I. INTRODUCTION

At least half of the effort to develop a working program is devoted to testing [4]. Undetected errors in software can cause substantial financial loss or even catastrophic results in safety critical systems. Detecting faults as early as possible during the software development is an effective means of reducing testing cost since the cost of fixing an error increases with the time between its introduction and detection.

Requirement-based testing (RBT) aims at starting testing-related activities as early as possible during software development, specifically deriving test cases (or test case specifications) from the requirements of the software under test [1]. RBT addresses two major issues: first, validating that the requirements are correct, complete, unambiguous and logically consistent; and second, designing a necessary and sufficient, from a black box point of view, set of test cases from those requirements to ensure that the design and code fully meet those requirements. Some of the benefits of and reasons for RBT are: (1) Creating tests early; (2) Allowing test engineers to find inconsistencies and ambiguities in requirements; (3) Leading to test data independent of any particular implementation; (4) Allowing conformance testing; (5) Reducing testing costs, since testing starts early; (6) Reducing software time to market.

These suggest that RBT can be an efficient and effective approach to software verification and validation. Note also that RBT is mandatory in some software development projects, e.g., airborne software developed according to the DO-178B/C standard [23].

There are numerous methods for representing software requirements (e.g., formal specification, plain text, the Unified Modeling Language) and more than one of them is typically used [7]. As a result, there are many approaches to RBT and, in spite of a great deal of research, to the best of our knowledge, there exists no systematic literature review [8] on this topic. There have been some surveys (e.g., [13, 14]) on model-based testing (MBT), a testing activity that aims at (automatically) deriving test cases from a model specifying the intended behaviour of a software [18]. However, RBT and model based testing are not interchangeable. Not any model used in MBT can be used for RBT. On the other hand, a model-based approach is one possible way to elicit requirements, and therefore RBT often uses models.

This paper reports on a systematic mapping study [8] on RBT, thereby identifying and classifying the available research (papers) in this area. A systematic mapping study is an important piece of work since it provides a wide overview of a research topic and establishes if research evidence exists in the area. It also provides an indication of the quantity of the evidence, prior to conducting a systematic literature review. During a systematic mapping study a classification scheme is defined and then used in an analysis phase to determine the coverage of the categories of the scheme. Petersen et al. discuss the differences between these two analyses [16].

More specifically, we are interested in RBT in the context of a UML-based software development [15]. The reason for reducing the scope of the research is threefold: (1) the UML is now the de-facto standard for analysis and design of object-oriented software [15], (2) not reducing the scope this way would lead to more research papers than what can likely be possibly managed, and (3) not reducing the scope would lead to testing techniques of widely different nature, which would complicate the review and the comparison (e.g., RBT from a Petri net model would have different capabilities than a technique based on UML sequence diagrams simply because of the more formal basis of that modeling notation).

Section II discusses the protocol we followed in our systematic mapping study. Section III presents descriptive statistics about the search of RBT techniques. Section IV discusses the comparison criteria we used to compare the identified testing techniques and the results of the comparison. Section V discusses threats to validity. Section VI concludes the paper.

We dedicate a fair amount of the paper to the search protocol and the comparison criteria. Our intent is to disclose enough details to readers' scrutiny and allow adequate conclusions to be drawn from the identified RBT techniques, to limit threats to the validity of the results, to allow replications, extensions and comparisons in future works (by us or others) [8].

II. REVIEW METHOD

We followed standard procedures [8, 16] whereby the first step is a planning activity. The most important

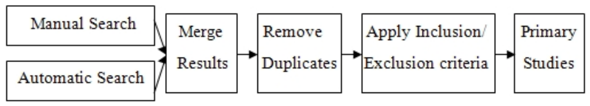


Figure 1. Paper selection process

planning activity is formulating the research questions. Our research questions are: RQ1—What are the current approaches to RBT? RQ2—What are the main characteristics of these approaches? Following planning, we identify relevant research works using an identification procedure (section II.A) and a selection/rejection procedure (section II.B).

A. Search Strategy

Most literature reviews, including mapping studies, involve automatically searching digital libraries with a set of keywords. Unfortunately, search engines embedded in digital libraries relevant to software engineering are not designed to support such reviews [5]. Therefore, one may miss relevant papers when only relying on such search engines. We complemented the automated search with a manual search, which is also recommended by others [5].

Manual Search. The focus was collecting recent papers on RBT, published between January 2006 and June 2011; We started in 2006 since UML 2.0 was officially released in July 2005 and we felt OMG's improvements to the UML specification would lead to increased opportunities for automated UML-based testing. This assumption is somewhat confirmed by our analysis (Section IV). We selected conference and journal venues (available in our online technical report [3]) that, from our past experience as researchers in the domain, we knew would likely publish work on RBT and UML based testing. Since the automatic search was conducted in parallel with the manual search, venues that had multiple relevant papers appearing in the automatic search were added to the list of relevant venues for the manual search [3].

Automatic Search. We searched five digital libraries that were highly relevant to software engineering and have been recommended by others [5, 8]: IEEE Xplore, ACM Digital Library, SpringerLink, Scopus and Inspec, during the period 1990-2011. Based on our research questions, we formulated the initial query string (Q0):

```
   Q0 "Requirement based testing" OR
"Requirement Driven Testing" OR "Specification
based testing" OR "Specification Driven
Testing"
```

Given our focus on RBT, we also performed a survey (though not as systematic as a mapping study) of requirement modeling techniques [10] and the definition of the word requirement itself (see [3] for details). We identified techniques as varied as natural language specifications to diagrammatic notations that specify system behavior (e.g., state machine) or interaction scenarios (e.g., sequence diagram). Considering that we are interested in testing techniques to be used in a UML context, we selected the following requirement modeling/specification techniques: use case, sequence diagram, activity diagram, state machine diagram, natural language. We selected this subset of UML diagrams, and not other diagrams like timing diagrams though they could be used to specify requirements, since these diagrams have been shown to be the most used by practitioners [6, 12]. We thus formulated the following additional queries:

```
Q1  "use case" AND test*
Q2  "sequence diagram" AND test*
Q3  "activity diagram" AND test*
Q4  ("statechart" OR "state machine" OR "state
    diagram") AND test*
Q5  "natural language" AND requirement AND
    test*
```

We kept word "requirement" in Q5 since otherwise query `"natural language" AND test*` was returning too many false positives, e.g., RBT techniques in other engineering disciplines than software engineering. We added this query because use case descriptions are usually written in natural language.

Because we had an interest in safety critical and embedded real-time systems we added the following two queries as well (in these two queries we also kept word "requirement" for the same reasons as previously):

```
Q6  "safety critical" AND requirement AND
    test*
Q7  "embedded" AND "real time" AND requirement
    AND test*
```

As illustrated above, we followed Kithenham's recommendations [8] and identified terms as well as synonyms and alternative spellings that were specific to our research questions, and then used ANDs and ORs to construct sophisticated queries.

We searched the selected five digital libraries with those queries, accounting for the slightly different formats the search engines required. The search with Q0 was performed in October 2010 and the one with Q1-Q7 was performed in June 2011. The search was performed within the title, abstract and keywords of papers.

B. Article Selection: Inclusion/Rejection Criteria

Our article selection entailed several steps (Figure 1). During the initial, manual plus automated search, any paper discussing an approach related to some testing activities from software requirements was added to the set of relevant papers.

Throughout the manual search, after scanning the list of papers in a conference proceeding or a journal issue, we considered all the papers that had a title relevant to software verification and validation. We then selected the ones that discussed any kind of RBT technique by reading the abstracts, introduction and conclusion sections. In a few cases we had to use the full text of the paper to make the final decision. For each selected paper, we also examined the list of references to find potential additional relevant papers.

We followed a similar procedure during the automatic search. However, we had to strengthen the exclusion criteria. Whenever possible, we used the filters provided in digital libraries to limit the scope of search to computer engineering or related areas. But, we still obtained many papers from other disciplines like mechanical engineering or civil engineering. We also excluded those papers.

After removing duplicates (obtained from different databases, from both the manual and automated searches) we defined a more precise set of inclusion/exclusion criteria to select the final set of primary studies:

1. We excluded papers on testing based on design artifacts, such as testing from design patterns specified in a class diagram;
2. We excluded papers that discussed testing model transformations;
3. We excluded papers discussing concepts like test case prioritization, test case optimization, test effort estimation, test planning and coverage analysis;
4. When we had several papers describing one single approach by the same author(s), we only considered the most recent one and if available the most recent journal paper, assuming that we would then obtain the most complete description of the approach;
5. We only included papers that discussed testing based on the UML diagrams mentioned previously.

III. RESULTS

The manual search resulted in a total of 147 papers, 51 of which are journal papers. Merging results of the manual and the automatic searches led to a set of 1,275 papers. After removing duplicates, we obtained a set of 702 papers. Applying our set of inclusion/exclusion criteria resulted in 100 unique papers as the final set of primary studies. The complete list of papers as well as descriptive results of automatic and manual searches are available in a technical report [3].

Table 1. Automatic Search (descriptive statistics)

	IEEE		Inspec		Scopus		ACM		Springer	
	T	R	T	R	T	R	T	R	T	R
Q0	64	49	123	86	133	87	26	11	20	12
Q1	124	38	290	83	589	93	46	12	52	22
Q2	30	19	61	38	154	74	20	10	8	6
Q3	25	21	45	33	98	46	20	13	6	3
Q4	405	130	NA	NA	NA	NA	21	7	114	37
Q5	38	10	194	38	89	15	18	4	14	3
Q6	75	12	286	42	209	26	20	3	21	3
Q7	113	9	399	16	189	10	21	2	21	4

IV. ANALYSIS

Our analysis entails comparison (Section IV.A) and classification (Section IV.B). We also observed evolutions over time (Section IV.C).

A. Comparison

We have defined seven criteria to compare primary studies, based on our research questions, to help us to extract from each paper data that we are interested in.

Requirement Model: The UML diagram used to model requirements is one of the most influencing factors when selecting and comparing testing techniques.

Test Model: In some papers, test cases are extracted directly from the requirement model in which case the test model is the same as the requirement model. In other papers, the requirement model is transformed into an intermediate model, which is used to generate test cases.

Level of Automation: We have defined four levels of automation: manual, partially automated, fully automated, and automatable. A partially automated technique entails some steps that require human expertise (e.g., expertise in the testing technique, in the system under test, in the domain of the system under test) while a fully automated technique does not. Automatable means that some steps of the technique are described with enough clarity and precision (e.g., an algorithm in pseudo code) to be automated. Should those steps be automated, the technique would then become either fully automated or partially automated (depending on whether some steps still require human expertise). If a primary study does not indicate that any of the steps can be automated, and therefore, the evaluation discussed in the paper (if any) is manual, then the technique is said to be manual.

Testing Level: We use a well known classification of testing levels [2]: acceptance testing, system testing, integration testing, module testing and unit testing.

Selection Criteria: These are the criteria being used to derive test cases from the test model. (Note that we make a difference between a selection criterion—a criterion being used to create tests, and a coverage criterion—a criterion to evaluate the coverage of an existing test suite, since creating technology and tool support for the latter is usually simpler than for the former.) Also we consider whether the selection criteria are based on the requirement model or some intermediate model when the test model is not the requirement model. This is important since in the latter case the mapping between the model element of the test model being exercised and elements of the requirement model is not necessarily straightforward.

Empirical Evaluation Technique: Since empirical evaluation has become an important part of software engineering research [20] and it is becoming more and more common to provide empirical data to support an idea, we compared primary studies according to the kind of empirical evaluation they provide. Different taxonomies of empirical evaluation techniques exist and we selected one that distinguishes between an experiment, a case study, and an example [22]. To distinguish between experiments and case studies we use the notion of a state variable [22]. In an experiment, the state variable can take on different values to recognize the differences between various situations, e.g., a controlled situation and the situation under investigation, whereas the state variable assumes only one value in a case study. On the other hand, an example covers only some parts of a technique.

Empirical Evaluation Material: We want to distinguish between papers that show application of the proposed approach in an industrial context from the ones that show application on a smaller application or fractions of it, or from the ones that illustrate application on a toy example.

Others, before us, have described criteria to compare software testing techniques. Neto and Travassos discuss criteria similar to ours but in the context of model-based testing [13, 14]. We report on a smaller amount of information in this paper since this is only a conference paper whereas they reported on the result of a complete literature review. More specifically, among the 18 different criteria they discuss, seven are similar to ours, two are not adequate to our situation (e.g., they consider whether a technique is structural or functional, whereas we only deal with the latter), and the remaining attributes will be used in a future publication. Our future publication will also compare the empirical results reported in primary studies: for instance, in the software testing community, researchers are typically interested in studying the cost (e.g., in terms of number of test cases generated) and effectiveness (at finding faults, either real or seeded). Our set of criteria and the one of Neto and Travassos are subsets of the larger, more extensive characterization schema for software testing techniques [21].

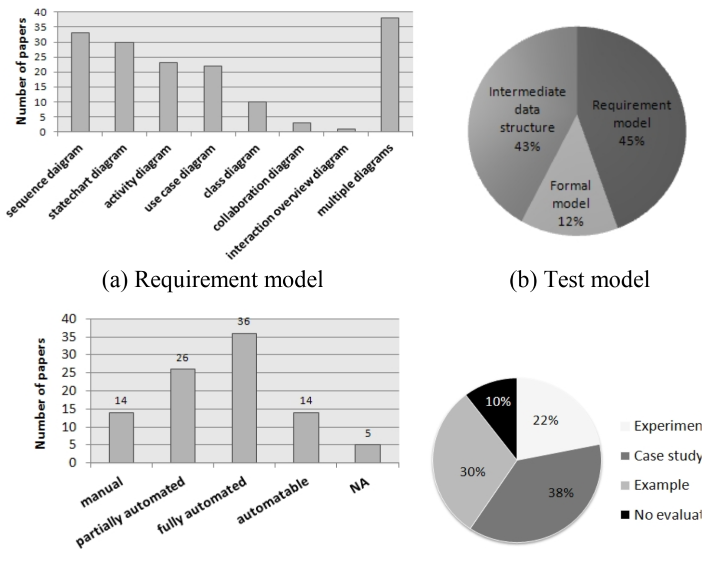
B. *Classification*

In this section, we classify the primary studies (papers) based on our seven comparison criteria.

Requirement Model. Figure 2 (a) summarizes the different UML diagrams that are used in primary studies. (We use only one term to refer to the UML 1.x and UML 2.x terminologies with respect to the state model.) Sequence and statechart diagrams have the highest number of hits. This is not surprising since these are probably the most used diagrams to specify behavior and interactions, and therefore functional requirements. Also, the statechart diagram is the most precisely defined UML diagram and therefore one of the most appropriate diagrams to derive tests from. We also noticed that 38 papers used more than one diagram as requirement model. This is a large amount and can be a result of the fact that usually more than one UML diagram is used during requirement engineering. The most common combination of models is use case diagram and either sequence diagram or activity diagram to model use case scenarios (17 papers). These 17 papers represent 77% of all the papers that rely on use cases for RBT. This is likely due to the fact that use cases and textual use case descriptions, although used a lot in requirement engineering, are not precise enough to be used alone during testing. Other common combinations are class diagram plus sequence diagram (six papers), sequence diagram plus statechart (four papers), and activity diagram plus statechart (three papers).

Test Model. 45% of the primary studies generate test cases directly from the requirement model: Figure 2 (b). The UML diagram that is the most used as a test model, either alone or in a combination with other diagrams, is the activity diagram (16 papers). The sequence diagram (13 papers) and statechart diagram (12 papers) come next. In 55% of the papers, the requirement model is transformed either into a formal model like Linear Transition System [17] or an intermediate data structure like communication tree [18] to generate test cases. This may be due to the lack of formality of UML or the desire to use a test model for which a testing technique already exists.

Level of Automation. Figure 2 (c). A lack of clear description for several steps of the proposed techniques



(a) Requirement model (b) Test model

(c) Level of automation (d) Empirical evaluation

Figure 2. Classification according to (a) the requirement model, (b) the test model, (c) the level of automation, and (d) the empirical evaluation

prevented us from classifying 5% of the primary studies. About 29% of the primary studies discuss automated tool support under the form of a single tool supporting several steps. These are a subset of the partially automated and fully automated primary studies. The rest of the (partially/fully) automated techniques discuss several pieces of tool support (but not a single bundle). Though not shown in Figure 2 (c), we note that all the techniques that use a formal model as test model are at least either partially automated or automatable. Equally interesting, 70% of the papers describing a manual process or that we were not able to classify are the ones that derive tests directly from the requirement model (i.e., the requirement model and the test model were the same), regardless of the kind or formal basis of requirement model used.

Testing Level. As expected, the majority of the primary studies (86%) describe a system testing technique. This is in accordance with the fact that system testing is designed to verify whether the assembled system meets its specification [2], and that requirements provide this specification. Other testing levels are integration testing (8%), unit testing (3%), and acceptance testing (2%). We also found one paper addressing non-functional testing (robustness testing). This small amount is not necessarily surprising since the diagrams we selected are more used to specify functional requirements than non-functional ones.

Selection Criteria. Since selection criteria depend on the test model, we do not classify primary studies based on this criterion: it would not be fair to say one technique does not use a criterion supported by another technique simply because they rely on different test models. However, we look at criteria used for similar test models. Some of the most used criteria for different diagrams are: state, transition for the statechart diagram; use case scenarios for the use case diagram; activity, transition, action for the activity diagram; message sequence path for the sequence diagram. We argue that, considering the extensive literature on test selection criteria, the criteria used in primary studies are among the simplest ones that exist. For instance, an activity diagram being very similar in structure (and purpose) to a control flow graph, it is surprising that other graph-based control- and data-flow criteria [2] are not experimented with. This also applies to sequence or statechart diagrams .

Four of the primary studies focus solely on empirical evaluation of different requirement based testing techniques either by means of case studies or experiments. We consider this to be a small proportion (4%) of empirical evaluations of different techniques, and this shows that there is room for more empirical studies in the area of RBT. Since these papers do not propose any new approach, we d not include them in the classification, but we have discussed them separately [3].

Empirical Evaluation. We found primary studies without any kind of empirical evaluation: Figure 2 (d). Only 22% of the primary studies have performed experiments to evaluate their approaches. This can be due to constraints in terms of length on what can be reported in a conference or workshop paper. Therefore we decided to analyze conference and workshop papers separately from journal papers and book chapters with respect to this comparison criterion (see figure 3). As expected experiments are used more often in journal papers while the most used empirical evaluation techniques in conference and workshop papers are case studies and examples. Note that we found two journal papers (out of 21) with no data on any empirical evaluation. The two journal venues are not listed in the Excellence in Research for Australia (ERA) journal and conference rankings; Nevertheless, the two papers were listed in the results of our automatic search.

Empirical Evaluation Material. The most occurring evaluation materials are toy example (34 papers), fraction of real world applications (18 papers), and industrial applications (13 papers). 16 studies include no discussion of the empirical evaluation material whatsoever: this includes the ten studies with no evaluation, indicating that six studies that report on an evaluation do not provide details on the experimental material.

C. Time Trends

Since the UML has evolved over time we speculated that our results would vary accordingly. Figure 4 (a) shows an increase in the number of primary studies around the year 2005. This can be caused by two factors. First, the more precise semantics of UML 2.0, released in 2005, might have made the UML more appropriate for testing. Second, 2006-2011 is the overlapping period between the manual and automatic searches. Therefore, the manual search could be the main cause of this increase. To better identify the root cause of the increase, we identified the papers found by only one or the other type of search and the papers found by both searches during the overlapping period 2006-2011: Figure 4 (b). The figure starts in 2005 to better present the trend for the automatic search. It shows that the trend is mostly due to the automated search.

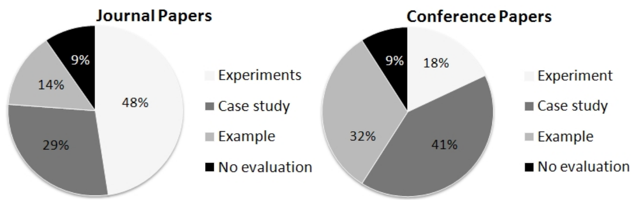


Figure 3. - Classification of journal and conference papers based on empirical evaluation method

We therefore conclude that the increase is due to the release of UML 2.0.

V. Threats to Validity

In any research work, there might be factors that can jeopardize validity. Our work is no exception and we need to discuss threats to validity. Systematic literature reviews and mapping studies are typically performed by a group composed of multiple researchers. In order to limit the threat of not having many researchers involved in the process, we followed guidelines [8] whereby we clearly defined and documented the important steps of the study including the review protocol, the research questions, the search strategy, the inclusion/exclusion criteria. These aspects are paramount to our study and results and this is the reason why we devoted a fair amount of space to describe them in this paper.

As mentioned before, both manual and automatic searches have advantages and drawbacks and we used both. One possible threat is that we might have missed a relevant (conference or journal) issue. However, our search was iterative and the dynamic search fed the manual one. Plus, the automatic search was not restricted in any way with that respect. Last, our knowledge of software testing research tells us that missing a venue is unlikely. A related threat is the possibility to miss a relevant paper from the identified venues because we manually selected them by only looking at the title, abstract, and keywords. However, when this information was not sufficient to make a decision we also looked at the introduction, the conclusion and the entire paper. This was, however, only necessary in a few cases, and we are confident we did not miss important papers. Another threat related to papers is the identification of primary studies. Although we tried to define a precise set of inclusion/exclusion criteria, there is always a chance of introducing a bias while applying them to different papers. Given the definition of those criteria, we however consider the risk is low. Note that we focused our study on UML-based RBT. We therefore excluded papers that describe testing techniques based on finite state machines (FSM) or extended finite state machines (EFSM). We do not consider this a threat, even though some of the (E)FSM techniques could be used in a UML context, since these techniques can be studied separately, there is a vibrant research community studying these techniques and there exists surveys describing them [9, 11].

The last threat to validity is about the classification of the primary studies. We tried to limit this threat by making our comparison criteria as precise as possible. We also reused existing classifications as much as possible. Nevertheless, there can always be cases for which the classification is affected by personal judgments (e.g., determining level of automation). In order to reduce the risks of introducing a bias in paper classification, 10% of the primary studies were selected randomly by the first author and submitted for classification to the second author. A few minor disagreements were noted and easily fixed.

VI. Conclusion and Future work

Requirement-based testing aims at starting testing-related activities as early as possible during the software development life cycle [1]. Since software testing is expensive, a great deal of research has been performed in the area of requirement based testing, with the hope to reduce testing costs. Unfortunately, as far as we know, to date, no systematic mapping study can help determine the current state of the art in requirement based testing.

The goal of our systematic mapping study was to identify published research works in the area of requirement-based software testing, and then provide a framework to evaluate them. Since the domain of requirement based testing is broad, such a mapping study is an important initial step before conducing more specific systematic literature reviews. In order to identify relevant papers we performed both an automatic search and a manual search, which resulted in about 1,300 papers. After removing duplicates and applying inclusion/exclusion criteria we obtained a set of 100 primary studies, which we compared by using seven complementary criteria.

The results of the classification lead us to make a number of interesting observations, including: (1) Almost all the primary studies (99%) discuss a functional testing technique. This indicates a lack of requirement-based testing approaches that address non-functional

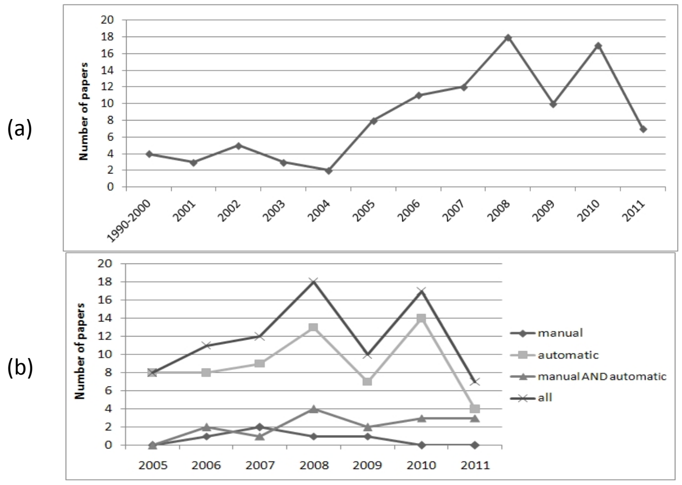


Figure 4. (a) Distribution of primary studies over time and (b) found by different search methods

requirements. This is however not entirely surprising since other UML diagrams than the ones we studied might be needed to specify and therefore test non-functional requirements; (2) Only a very small proportion of the primary studies (4%) evaluated different requirement-based testing techniques, suggesting a need for more empirical comparisons of requirement-based testing techniques; (3) A relatively small percentage of the primary studies (22%) provide a formal evaluation of the proposed testing technique under the form of an experiment. When only looking at primary studies published in journal venues, the proportion is higher (48%), though we observed that 23% of the journal papers only evaluate the proposed testing technique with an example or provide no evaluation; (4) The test selection criteria being used in the proposed requirement-based testing techniques seem to be among the simplest that have been suggested in the testing literature. For instance, we did not find a single primary study where data flow criteria are used, although data flow selection criteria can be used in sequence, activity, or state machine diagrams and they are known to complement control flow criteria.

Our future work includes an analysis of the primary studies in a qualitative way to answer questions like: What limits the application of the proposed techniques? What are the avenues for further research in UML-based, requirement-based software testing? We also plan to extend the scope of the review to other requirement modeling techniques than ones based on UML.

VII. Acknowledgment

This work was performed under the umbrella of a NSERC-CRD grant. The authors would like to thank NSERC, CRIAQ, CAE, CMC Electronics, and Mannarino Systems & Software for their financial support.

VIII. References

[1] G. Amit and B. Rajesh, “Testing functional requirements using B model specifications,” Software Engineering Notes, 35, 2010, pp. 1-7.

[2] P. Ammann and J. Offutt, Introduction to Software Testing, Cambridge University Press, 2008.

[3] N. Asoudeh and Y. Labiche, “Requirement based software testing in a UML context: A systematic literature review”, Tech. Rep. SCE-108,2011. Avialable from ,squall.sce.carleton.ca/ pubs/ tech_report /TR_SCE-11-08.pdf [retrieved: October, 2012]

[4] B. Beizer, Software Testing Techniques, International Thomson Computer Press, 1990.

[5] P. Brereton, A. B. Kitchenham, D. Budgen, M. Turner ,and M. Khalil, “Lessons from applying the systematic literaturereview process within the software engineering domain”, JSS, vol. 80, 2007, pp. 571-583.

[6] B. Dobing and J. Parsons, “How UML is used,” Com. of the ACM, vol. 49, 2006, pp. 109-113.

[7] C. Jones, Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies, McGraw-Hill, 2009.

[8] B. A. Kitchenham, “Guidelines for performing systematic literature reviews in software engineering,” Tech. Rep. EBSE-2007-001, 2007.

[9] R. Lai, “A survey of communication protocol testing,” JSS, vol. 62, 2002, pp. 21-46.

[10] A. V. Lamsweered, Requirements Engineering, Wiley, 2009.

[11] D. Lee and M. Yannakakis, “Principles and methods of testing finite state machines-a survey,” Proc. of the IEEE, vol. 84, 1996, pp.1090-1123.

[12] F. J. Lucas, F. Molina, and A. Toval, “A systematic review of UML model consistency management,” IST, vol. 51, 2009, pp. 1631-1645.

[13] A. Neto and G.H. Travassos, “Evaluation of model-based testing techniques selection approaches: An external replication,” Proc. ESEM, 2009, pp. 267-278.

[14] A. Neto and G.H. Travassos, “Model-based Testing Approaches Selection for Software Projects,” IST, vol. 51, 2009, pp. 1487-1504.

[15] T. Pender, UML Bible, Wiley, 2003.

[16] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, “Systematic mapping studies in software engineering,” Proc. EASE, 2008, pp. 68-77.

[17] S. Pickin, C. Jard, T. Jeron, J.M. Jezequel, and Y. Le Traon, “Test synthesis from UML models of distributed software,” IEEE TSE, vol. 33, 2007, pp. 252-269.

[18] A. Pretschner, “Model-based testing in practice,” Proc. Formal Methods, 2005, pp. 537-541.

[19] P. Samuel, R. Mall, and P. Kanth, “Automatic test case generation from UML communication diagrams,” IST, vol. 49, 2007, pp. 158-171.

[20] F. Shull, J. Singer, and D.I.K. Sjoberg, Guide to Advanced Empirical Software Engineering, Springer, 2008.

[21] S. Vegas and V. Basili, “A Characterization Schema for Software Testing Techniques,” ESE, vol. 10, 2005, pp. 437-466.

[22] C. Wholin, P. Runsen, M. Host, M. C. Ohlsson, B. Rengell, and A. Wessslen, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, 2000.

[23] RTCA: Software Considerations in Airbone Systems and Equipment Certification. Radio Technical Commission for Aeronautics (RTCA), Standard Document no. DO-178C. (2011)

Simulation-Based Management for Software Dynamic Testing Processes

Mercedes Ruiz
Department of Computer Science and Engineering
University of Cádiz
Cádiz, SPAIN
e-mail: mercedes.ruiz@uca.es

Javier Tuya
Department of Computing
University of Oviedo
Gijón, SPAIN
e-mail: tuya@uniovi.es

Daniel Crespo
Department of Computer Science and Engineering
University of Cádiz
Cádiz, SPAIN
e-mail: dani.crespobernal@alum.uca.es

***Abstract* - Managing software development projects requires the coordination of different processes which may be performed by different teams, e.g., a development team and a separate testing team. This coordination aims at optimizing the trade-off between cost, schedule and delivered quality. Simulation models are a powerful tool to explore what-if scenarios that help managers to achieve this trade-off and to fine-tune different project parameters. This paper presents a simulation model based on a multi-paradigm approach which connects development and testing processes. The testing process model is based on the process model described in the ongoing standard ISO/IEC DIS 29119-2. The simulation model is built using two different methods: the discrete-event approach, to simulate the execution of the dynamic testing processes, and the agent-based approach, to in-depth simulate defects life cycle. Results show how the simulation model is used to optimize the efficiency of the testing process.**

Keywords - software testing process; multiparadigm simulation; management

I. INTRODUCTION

Software testing is concerned with planning, preparation and evaluation of software products and related work products to: a) determine that they satisfy specified requirements, b) demonstrate that they are fit for purpose and c) detect defects [17]. In general, testing can be viewed as a means of improving the quality of a given product and mitigating risks due to poor quality.

Testing can be carried on using different approaches (e.g., scripted or exploratory), at different levels (e.g., unit, system, integration or acceptance), using different techniques and tools and with different degrees of independency (ranging from testing performed by the producer to third party testing). When testing entails the execution of the system under test, it is often referred to as dynamic testing. Testing exists in an organizational context and is carried on a given project or service. Therefore, the testing activities are tightly interrelated with the development ones, and both shall be planned, monitored and controlled. Problems of quality of the system under test or delays in the development hamper the testing process. Conversely, an inadequate or delayed testing endangers the development process. If not managed properly, both development and testing processes may jeopardize the goals of cost, schedule and quality of a project.

Both development and testing can be described as processes and take advantage of the use of simulation models for helping project and/or test managers in daily tasks of planning, monitoring and control.

Informally, a simulation model can be considered as an abstract view of a complex system comprised of a set of rules that tell how to obtain the next state of the system from the current state. Those rules can be of many different forms: differential equations, state charts, process flowcharts, schedules, etc. The outputs of the model are produced and observed as the model is running.

There is much research on simulation models of the software development process [12]. However there is lesser research on simulation models for the testing process, usually at the unit level. Furthermore, when testing is considered as part of a simulation model of the development process, it is often over simplified. The goal of this paper is to devise a multi-paradigm simulation model for the testing process to gain insights in how the testing process influences the goals of a given project.

The main contribution of this work is a multi-paradigm simulation model of the dynamic testing processes which combines a discrete-event model and agent-based model. The model can be used to simulate the testing process at the system level and to help in decision-making in the test managing processes.

The structure of the paper is as follows: Section II shows the works related to our proposal; Section III introduces the multi-layer process model proposed by the International Standards group upon which our simulation model is based; Section IV describes the simulation model; Section V shows some simulation runs. Finally, our conclusions and further work are given in Section VI.

II. RELATED WORK

The search string "simulation" AND "software testing process" AND "management" and others alike used in several digital libraries and citation databases of peer-reviewed literature retrieves only a few number of papers. In many of the papers retrieved, the term "simulation" is frequently used to describe experiences in which simulation is used as a tool for the testing process. In other works, the term "simulation" makes reference to a set of formulas that are solved by analytical means.

As an example of the first usage, in their collection of works, Lazić, Mastorakis and Velasĕvić [7]-[11] aim at raising awareness about the usefulness and importance of computer-based simulation in support of software testing. In their works, simulation is used to ease the design and execution of the testing processes of real military and defense systems.

Some analytical models of the software testing process can also be found. Zhang, Zhou, and Luo [19] propose a reward-Markov-chain-based quantitative model for sequential iterative processes and show how to use it to estimate the time for the software testing process. Similarly to this, Lizhi, Weiqin and Bin Zhu [12] propose an approach to model the testing process based on hierarchical time colored Petri Nets (HTCPN). However, while Petri-nets are good at modeling resources and parallel processing, simulation modeling models system components and their interactions, making it possible to conduct arbitrary time-related performance analysis, something which is not easy using Petri-nets.

Consequently to overcome the problems of analytical methods, simulation modeling can be applied in the context of testing processes mainly because: a) it enables to find solutions when analytical methods fail; b) it is a more straightforward process than analytical modeling since the structure of the simulation model naturally mimics the structure of the real system, and c) it is scalable, flexible, and easy to communicate since the modeling tools use visual languages.

However, despite these advantages there is a small number of contributions of simulation modeling in the field of software testing processes. Saurabh [15] presents a system dynamics (SD) model of software development with a special focus on the unit test phase. This work is partially based on Collofello's et al. [2] work about modeling the software testing process under the SD approach.

The motivation of these works is closely related to ours, but the models are built under a different simulation approach. System Dynamics approach operates at high abstraction level and is mostly used for strategic modeling. Hence, since a simulation model can only be used at the abstraction level in which it has been created, such a highly abstract model is not adequate for the operational and tactical levels in which decision-making regarding the testing processes takes place. In our case, since our main interest is to simulate the testing processes the discrete event (DE) modeling, with the underlying process-centric approach, has been selected. Furthermore, we have also selected the agent-based (AB) approach to be used together with the discrete-event one resulting in a multi-paradigm simulation model.

Generally, each simulation approach (SD, DE, AB) provides a set of different abstractions. If the system being modeled is complex enough, and software development is, then it is preferable to integrate different simulation methods than using one single approach, since the final model will represent the real system more realistically.

When we used the search string ("multi-paradigm" OR "multi-method") AND "simulation" AND "software testing process" and others alike in the digital libraries and citation databases, no single work was retrieved. Therefore, given the results of the systematic literature review performed, not fully documented here for space reasons, to the best of our knowledge our proposal is the first one that aims at using multi-paradigm simulation modeling to improve decision making in software testing management.

III. MULTI-LAYER TEST PROCESS MODEL

Testing processes include a variety of management and technical activities which are organized in a process model in part 2 of the draft ISO standard for software testing: ISO/IEC 29119-2 [4]. The purpose of this international standard is to define a generic process model for software testing that can be used by any organization when performing any form of software testing. Testing is structured in a multi-layer process model that defines the software testing processes at (1) the organizational level, (2) test management level and (3) dynamic test level. More specifically, the dynamic test level describes how dynamic testing is carried out within a particular phase of testing (e.g., unit, integration, system and acceptance) or type of testing (e.g., performance testing, security testing and usability testing). It is composed of four processes that are depicted in Figure 1.

- Test Design & Implementation Process: Describes how test cases and test procedures are derived; these are normally documented in a test specification, but may be immediately executed.
- Test Environment Set-Up & Maintenance Process: Describes how the environment in which tests are executed is established and maintained.
- Test Execution Process: Describes how the test procedures generated as a result of the Test Design & Implementation Process are run on the test environment established by the Test Environment Set-Up & Maintenance Process.
- Test Incident Reporting Process describes how the reporting of test incidents is managed.

The Test Execution Process is run after the tests have been specified and the environment has been established which leads to a strong dependency on the previous processes. This process may need to be performed a number of times as all the available test procedures may not be executed in a single iteration. Additionally, this process must be reentered as a consequence of detected failures after the underlying defects have been corrected (retesting).

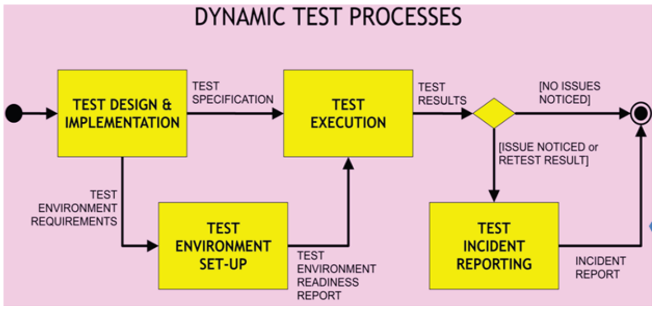


Figure 1. Dynamic Test Processes.

Besides, the Test Design & Implementation Process, and the Test Environment Set-Up & Maintenance Process may be reentered whether additional tests are needed after execution or some problems are detected in the testing environment. The Test Incident Reporting Process may be also reentered as a result of: a) the identification of test failures, b) something unusual or unexpected occurred during test execution, or c) retest activities.

IV. MODEL DESCRIPTION

The simulation model developed is described below in terms of its scope, result variables, process abstraction and input parameters. The description is organized following Kellner's proposal for describing simulation models [5].

A. *Model Proposal and Scope*

To determine a model proposal, the key questions that the model needs to address need to be identified. Then, model scope is set so that it is large enough to fully solve the key questions posed. In the context of this work, model proposal is to help in decision-making in software testing process management. Accordingly, the scope for this model will be a portion of the life cycle, with a short time span (i.e., the months in which the testing activities take place), one software product and two teams (i.e., development and testing teams) organizational breadth.

B. *Result Variables*

The result variables are the information elements needed to answer the key questions regarding the purpose of the model. In our model, several process metrics have been identified to help us understand the simulated process capability. According to this, process metrics have been classified into effectiveness and efficiency process metrics.

Effectiveness process metrics measure the extent to which a process produces a desired result [1].

The following result variables fall into this category:

- Defect Closure Period (DCP). The longer a reported defect takes to go from discovery to resolution, the higher the project risk associated with the underlying defect. Unresolved defects may: a) delay testing, b) make development less efficient or c) prevent the delivery of the software to the final customers. DCP measures the difference between the time required to repair a defect and the time required to confirm the defect is repaired.
- Defect Open Count. This measure tracks the number of times a defect report is opened. When the report is first submitted this count is set to one. This count is incremented each time the same defect report is reopened due to a failure in the confirmation test (retest).
- FixBacklog: Shows the percentage of defects closed per all the defects opened in a given time.
- Total Planned Test: The metric shows the evolution of the number of planned test cases along the testing project.
- Total Executed Tests: Shows the evolution of actual test cases which are executed along the project.
- Total Passed Tests: Shows the actual test cases which are executed and successfully passed (e.g., did not find any defects).
- Total Failed Tests: Shows the actual test cases that are executed and failed (e.g., did find defects).

Efficiency process metrics measure the extent to which a process produces its desired results in a not wasteful way and, ideally, minimizing the resources used [1].

Result variables in this category follow:

- Actual test time: Shows the total length of the testing process.
- Total team size and number of people per activity: Shows the total size of the testing team and the number of resources allocated to each activity of the process, respectively.
- Process efficiency: Shows the ratio of the number of defects closed per the number of defects found.

C. *Process Abstraction*

When developing a simulation model, the key elements of the process, their inter-relationships, and behavior need to be identified. The focus should be on those aspects of the process that are especially relevant to the purpose of the model, and believed to affect the result variables [5].

One of the decisions that need to be made in this phase is the simulation paradigm that it is going to be used to build the model. A simulation paradigm is a general framework for mapping a real world system to its model. The choice of paradigm should be based on the system being modeled and the purpose of the modeling. When modeling complex systems, it is frequent that different parts of the system are most naturally modeled using different paradigms. In this case, a multi-paradigm model is built.

In order to build our model, the multi-paradigm approach has been selected. First, to model and simulate the dynamic testing processes, the paradigm selected has been the discrete-event or process centric approach. Under this approach, the system being modeled is considered as a process, i.e., a sequence of operations being performed across entities, and this makes this paradigm the most natural and adequate to build process simulation models. The model is specified graphically as a process flowchart, where blocks represent the operations to be done along the process.

Although a simulation model following this approach allows us to analyze the evolution of the testing activities, the resource consumption and the number of defects detected, it would be interesting to add an extra functionality to the simulation model allowing the user to track the life of every defect since it is found until it is closed. It is important to notice that to achieve this aim the level of abstraction used needs to be changed from process-centric to individual-centric. Agent-based modeling is a simulation approach that allows the modeler to build a model under a bottom-up perspective, that is, describing the behavior of individuals (e.g., agents) and, if needed, their interactions. Frequently, the behavior of an agent is formalized by means of a state chart-like diagram. Therefore, this approach seems to be most natural and adequate to describe the lifecycle of defects found during the testing phase. As a consequence, a multi-paradigm simulation model was our choice for our modeling problem.

In summary, the model consists of two connected models. A description of each of these models follows:

1) Discrete event model (DE).

The discrete event model represents the Dynamic Test Processes in ISO/IEC 29119-2 [4] previously described in Section III.

The development process produces two main artifacts that are the input for the testing processes:

1. The test basis, usually the software specification, which is modeled as a set of features.
2. The executable code that is to be exercised by the tests.

The availability of the test basis enables the execution of the Test Design & Implementation Process which leads to a number of test cases. However, test cases are not ready to be executed until the test environment has been established (Test Design & Implementation Process) and the executable code released. Once the code is installed in the testing environment, the Test Execution Process can begin. Failed test cases are the input for the Test Incident Reporting Process and the results communicated to the development processes through the Agent-based model. Test execution reenters when previously detected defects have been fixed by development.

2) Agent-based model (AB).

During the software development process, each defect has a lifecycle in which it reaches different states. In order to simulate the different states that a defect reaches the agent-based paradigm has been used. Under this approach, we formalize the defects found as agents and their behavior as a state chart that reflects the different states and transitions of defect lifecycle. A description of each state in which the agents can be follows:

- *New*: An agent reaches this state when a defect is reported by the tester for the first time and is yet to be approved.
- *Analyzed*: Once a defect is reported, the manager has to analyze it in order to approve it as a genuine defect, reject or defer it. The agent remains in this state during the time in which this activity takes place. When the activity is done, the information for deciding what to do with the defect is available, and so, the agent moves to the next state which can be one of the following: a) *Rejected*: If a defect is found to be invalid, b) *Deferred*: If a defect is decided to be fixed in upcoming releases, and c) *Assigned*: If a defect is found to be valid and assigned to a member of the development team to fix it.
- *Fixed*: An agent moves to this state once the developer communicates the defect is fixed. The defect goes to the testing team for validation by injecting a task in the DE model to indicate that the test case that found this defect has to be executed again (retest). The result of this execution will determine the next state of the agent.
- *Closed*. If the tester finds that the defect is indeed fixed and is no more a cause of concern, the agent moves to the state Closed. Otherwise, if the defect is not fixed or partially fixed, the agent will go again to the state Assigned in which the work of a developer working on its fixing will be simulated again.

D. Input Parameters

The input parameters to include in the model largely depend upon the result variables desired and the process abstractions identified. Input parameters allow setting up different scenarios for simulation. The input parameters of the simulation model are the following:

- Software size: Size of the software product under development.
- FPA per Feature: Adjusted Functional Points per feature.
- Number of Test Cases per Feature: Number of test cases that need to be designed and executed per feature.
- Initial number of tasks in Environment Setup. Initial number of tasks that need to be done for the common and global environment setup.
- Estimated Time for Environment Setup. Time estimated to develop each environment setup task.
- Environment Setup Resources. Number of people allocated to the Environment Setup processes.
- Estimated Time for Test Design and Implementation. Time estimated to develop each task of the Test Design and Implementation processes.
- Test Design and Implementation Resources. Number of people allocated to the Test Design and Implementation process.
- Estimated Time for Test Execution. Time estimated to develop each task of the Test Execution processes.
- Test Execution Resources. Number of people allocated to the Test Execution processes.
- Estimated Time for Test Incident Reporting. Time estimated to develop each task of the Test Incident Reporting processes.
- Test Incident Reporting Resources. Number of people allocated to the Test Incident Reporting Processes.

- Estimated time to fix a defect. Time estimated to a fix a defect by a developer.
- Code released for Test Execution. Indicates when the code is released for testing. This value is provided as a percentage of delay measured regarding the initial estimated time for the testing project.
- Probability of finding a defect per Test Case Execution. Probability that a Test Case finds a defect when the test case is executed the first time.
- Probability of finding a defect per Test Case in Retest Execution. Probability that a Test Case finds a defect when the defect has been reported as fixed.

In order to achieve more realistic results, the model accepts a triangular distribution for most of the above input parameters.

V. SIMULATION RUNS

The model implementation and the simulation runs have been performed using Anylogic™ software [18] with the Enterprise Library. The model logic is written in Java.

a) Base scenario (BS). In this scenario the base simulation is run to determine the values of the result variables and, analyze the results of the process. In order to obtain a set of reasonable parameters we have estimated the costs of the different activities using a set of ratios observed in average risk profiles [6]. We consider functional testing for a system test phase in a project with waterfall development, experienced builders and a structured test approach driven by risk. Ratios of functional design, realisation and functional test are 1:2:1. In relation with the test processes, the ratios of test design & implementation, execution, reporting and environment set-up are 50:40:5:5, respectively. The values of the input parameters in this scenario are displayed in TABLE I.

TABLE I. SCENARIO CONFIGURATION

Input parameter	**Value**
Software size	800 FPA
FPA per Feature	5
Number of Test Case per Feature	(0.5, 2, 4)
Initial number of tasks in Environment Setup	5 tasks
Estimated Time for Environment Setup	(10, 14.4, 20) hours
Environment Setup Resources	1 person
Estimated Time for Test Design and Implementation	(3, 4.5, 6) hours
Test Design and Implementation Resources	4 people
Estimated Time for Test Execution	(1.5, 3.2, 4.5) hours
Test Execution Resources	4 people
Estimated Time for Test Incident Reporting	(1.5, 3, 4.5) hours
Test Incident Reporting Resources	1 person
Estimated time to Fix a defect	(3, 4.5, 6) hours
Code released for Test Execution	15%
Probability of finding a defect per Test Case Execution	(5%, 15%, 25%)
Probability of finding a defect per Test Case in ReTest Execution	(10%, 20%, 30%)

Figures 2 and 3 contain a summary of the main process metrics. Figure 2 shows that 160 test cases were planned, which were able to find 116 defects (see Figure 3), 6 of them were rejected and 9 of them deferred. 101 defects were closed and 26 reopened. The Test Efficiency reached with this setting is 87%, which is reasonable in practice, showing the consistency of the model when using the above parameters.

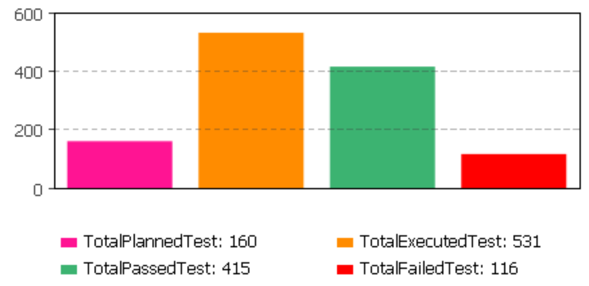


Figure 2. Different process output metrics in base scenario

b) Optimization. Even though simulation runs are useful to visualize the effect of different values of the input parameters in the process performance, that is, to execute what-if scenarios in managerial decision-making, a key benefit can be obtained when we use together simulation and metaheuristic optimization algorithms in a process called simulation optimization. In this case, it is possible to obtain which values need to take the input parameters in order to maximize or minimize an output variable. To show this

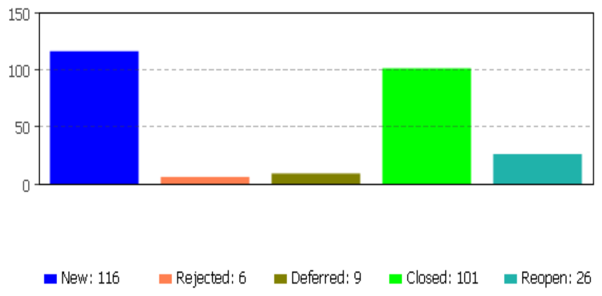


Figure 3. Number of defects in each state at the end of the simulation

application in our field of study we run an optimization experiment to determine wheter it is possible to improve the efficiency of the test process by controlling the moment in which the executable code is available for testing. The optimization will determine the distribution of the human resources that maximizes test efficiency when the code is released for testing in an range that varies from 5% to 50% from the moment the testing process begin. The optimizer OptQuest® [14] built-in Anylogic™ has been used for this optimization. Table II shows the input values for the control parameters of the experiment, the constraints impossed and the results obtained in the optimized process compared with the base case.

TABLE II. OPTIMIZATION CONTROL PARAMETERS AND OUTPUTS COMPARED WITH BASE SCENARIO

Input parameter	**Control Input**	**Result (Base scena.)**	**Result (Optim.)**
Initial number of tasks in Environment Setup	3-5 tasks	5	5
Environment Setup Resources	1-4 people	1	1
Test Design and Implementation Resources	1-4 people	4	2
Test Execution Resources	1-4 people	4	3
Test Incident Reporting Resources	1-4 people	1	1
Code released for Test Execution	5% - 50%	15%	27%
Constraints	**Value**		
Testing Team Size	<= **7** people		
Maximum Testing Time Overrun	<= 1 month		
Process Efficiency obtained		87%	97%

The results of the optimization experiment show that under the constraints imposed it is possible to achieve 97% of efficiency in the process allocating 7 people to the process and having a maximum delay of the code released for testing of 27% of the initial time estimated. This will result into a process which is 97% in closing defects but finishes one month later than the base scenario. The conclusion drawn from this particular experiment with regard to the base scenario is that if the project is adequately scheduled, it is possible to reduce the total number of test resources as well as increase the process efficiency. In other situations simulation can help to find the best input values for project schedule, resource allocation and quality objective from among all that lead to the optimization of the key process outputs.

VI. CONCLUSION AND FURTHER WORK

This paper presented a simulation model for the dynamic testing processes which allows a seamless integration between the testing and development processes. The model is devised as a multi-paradigm model composed by a discrete event simulation model, to simulate the execution of the dynamic test processes, and an agent-based simulation model, to in-depth simulate the defects life cycle. Results show that the simulation model can be effectively used to optimize different project parameters and then help managers to achieve a trade-off between cost, schedule and quality.

This is a first step in the use of multi-paradigm simulation models for testing. Further work will include, although not limited to, the consideration of agent-based models to simulate parts of the dynamic test processes, the integration into a more complex project development simulation model [3] and experimentation in different projects using different lifecycle models and including different test levels of testing. After calibrating and validating the model with historical data from the industry, it will be also possible to exploit it as an operating tool for decision-making in the industrial domain.

ACKNOWLEDGMENTS

This work has been partially supported by the Spanish Ministry of Science and Technology with ERDF funds under grants TIN2010-20057-C03-03 and TIN2010-20057-C03-01.

REFERENCES

[1] Black, R. Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. Wiley Publishing, 2002.

[2] Collofello J.S., Zhen Y., Tvedt J.D., Merrill D., and Rus, I. Modeling software testing processes (1996) Conference Proceedings - International Phoenix Conference on Computers and Communications, pp. 289-293.

[3] Crespo D. and Ruiz M. Decision Making Support in CMMI Process Areas using Multiparadigm Simulation Modeling. 2012 Winter Simulation Conference. Berlin, December 9-12, 2012. Accepted.

[4] ISO/IEC DIS 29119-2 Software and Systems Engineering - Software Testing – Part 2: Test Process. December 2011 (Draft International Standard).

[5] Kellner M.I., Madachy R.J., and Raffo D.M. Software Process Modeling and Simulation: Why, What, How?, Journal of Systems and Software, Vol. 46, No. 2/3 (15 April 1999).

[6] Koomen, T., van der Aalst, L., Broekman, B., and Vroon, M. TMap Next for result-driven testing. UTN Publishers, 2007.

[7] Lazić L. and Mastorakis N. The use of modeling & simulation-based analysis & optimization of software testing (2005) WSEAS Transactions on Information Science and Applications, 2 (11), pp. 1918-1933.

[8] Lazić L. and Mastorakis N. RBOSTP: Risk-based optimization of software testing process Part 2 (2005) WSEAS Transactions on Information Science and Applications, 2 (7), pp. 902-916.

[9] Lazić L. and Mastorakis N. RBOSTP: Risk-based optimization of software testing process Part 1 (2005) WSEAS Transactions on Information Science and Applications, 2 (6), pp. 695-708. Cited 3 times.

[10] Lazić L. and Mastorakis N. Integrated intelligent modeling, simulation and design of experiments for Software Testing Process (2010) International Conference on Computers - Proceedings, 1, pp. 555-567.

[11] Lazić L. and Velasěvić D. Applying simulation and design of experiments to the embedded software testing process (2004) Software Testing Verification and Reliability, 14 (4), pp. 257-282.

[12] Lizhi, C., Weiqin T., Bin Z., and Zhang J. Modeling Software Testing Process Using HTCPN, Fourth International Conference on Frontier of Computer Science and Technology, 2009. FCST '09, pp. 429-434, 17-19 Dec. 2009.

[13] Madachy, RJ. Software Process Dynamics. John Wiley & Sons, Inc., 2008.

[14] OpTek Systems, Inc. OptQuest$^{®}$. http:// www.opttek.com/ [retrieved: October, 2012]

[15] Saurabh, K. Modeling unit testing processes a system dynamics approach (2008) ICEIS 2008 - Proceedings of the 10th International Conference on Enterprise Information Systems, 1 ISAS, pp. 183-186.

[16] Soni R., Jolly A., and Rana A. Effect of residual defect density on software release management (2011) International Journal of Software Engineering and its Applications, 5 (4), pp. 151-158.

[17] van Veenendaal, E (ed). Standard glossary of terms used in Software Testing, Version 2.1. International Software Testing Qualifications Board. Oct. 2010.

[18] XJ Technologies. Anylogic^{TM}. http://www.anylogic.com/ [retrieved: October, 2012]

[19] Zhang W.M., Zhou, B.S., and Luo, W.J. Modeling and simulating of sequential iterative development processes (2008) Jisuanji Jicheng Zhizao Xitong/Computer Integrated Manufacturing Systems, CIMS, 14 (9), pp. 1696-1703.

A Systematic Approach to Risk-Based Testing Using Risk-annotated Requirements Models

Marc-Florian Wendland, Marco Kranz and Ina Schieferdecker
Fraunhofer Institute FOKUS
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
{marc-florian.wendland, marco.kranz, ina.schieferdecker}@fokus.fraunhofer.de

***Abstract*—Nowadays, software-intensive systems continuously pervade several areas of daily life, even critical ones, and replace established mechanical or manual solutions. Development and quality assurance methods have to ensure that these software-intensive systems are delivered both with adequate quality, optimized resources and within the scheduled time frame. The idea of risk-based testing is to prioritize testing activities to what is deemed critical for the software-intensive system. Although there is a common agreement that risk-based testing techniques ought to be rigorously applied, especially for safety- and security-critical systems, there is actually little knowledge available on how to systematically come to risk-optimized test suites. This paper presents a novel approach to risk-based testing that deals with the transition from risk management and requirements engineering to test design activities and test case generation by using models. The main contribution of the paper is the description of a methodology that allows an easy combination of test generation directives and risk level in order to generate risk-optimized test suites.**

Keywords - risk-based testing; behavior engineering; model-based testing; requirements model; safety-critical systems

I. Introduction

Already in 1999, Amland stated that IT projects are very rarely on time, schedule or budget, so when it comes down to testing, the time to delivery is extremely short and there is no budget left due to the development overrun [5]. This statement holds even for today. This requires test case design techniques to be able to identify the most important test cases to be carried out in view of limited time. Thus, the test cases need to be prioritized to be comparable with each other.

A well-known and highly recommended approach is *risk-based testing* ([24][28][29]). The idea of risk-based testing is as simple as intuitive: Identify prior to test case design and execution those scenarios that trigger the most critical situations for a system in production and ensure that these critical situations are both effectively mitigated and sufficiently tested. Following Bach, risk-based testing aims at testing the right things of a system at the right time [1]. He further states that each test process is actually carried out in a risk-based way due to its sampling characteristics. In most cases, the consideration of risk is rather made implicitly, though.

A *critical situation* is not necessarily dedicated to safety- or security-critical systems (though it is often used in the context of such systems), but applies actually to any kind of system. For example, the most critical situation for a text processor application might be the *save* functionality, since a malfunction may cause the user to switch to the product of a competitor. However, in the area of safety-critical systems, critical situations represent sensitive points in time during the execution of a system, where a malfunction may lead to harm of environment, human life, financial loss etc. No matter what kind of system is tested, the idea of risk-based testing remains the same, whereas the impact of a bad test case selection may differ dramatically, of course.

Even though risk-based testing is deemed helpful to deal with scarce resources, it is a matter of fact that there is only little literature available that provide the tester with a systematic and reproducible approach on how to actually get to a risk-optimized set of test cases. We seek to address the lack of well-founded methodologies for systematic and applicable risk-based testing approaches. Therefore, we are using semi-formal models to describe both the functional-related requirements and a risk-optimized test model. Furthermore, we show how formal test directives are coupled with risk to automate the risk-based test case generation.

The scientific contributions of this paper are:

- Outline of a coherent methodology that combines information from risk analysis and assessment activities with requirements engineering activities
- Use of formal requirements models to incorporate risk-information for further exploitation
- Specify how test case derivation strategies are being coupled to various risk levels in combination with a risk matrix using test directives
- Describe a prototype tooling landscape including complete set of modeling notations for the proposed methodology

However, this paper does not claim to be an industrial evaluation report. The remainder of the work is structured as follows: Section 2 summarizes the state of the art of relevant risk-based-testing approaches to the knowledge of the authors. Section 3 describes the main contribution of this paper, i.e., our methodology for model-based risk-based testing. At first, we give a definition of relevant terms in the realm of risk-based testing. We, then, briefly introduce Behavior Engineering as the basis of our approach. Next, we describe how risk information is incorporated into the resulting requirements models. Afterwards, we show how those risk-annotated requirement models are further exploited for systematic test case generation. Section 4 briefly summarizes a prototype tooling landscape and findings from a first application of the methodology. Section 5 and 6 provide an outlook to future work and conclude eventually.

II. Related Work

The principles of risk-based testing have been addressed by several often quoted publications before (such as [1][2][4]). These articles are mainly dedicated to the clarification of the terms and concepts that belong to risk-based testing. The authors provide justified arguments why risk should always be considered in structuring testing processes. Amland presented a concrete example how risk-based testing had been performed within a project in a financial domain [6]. None of these articles, however, provide precise statements or event suggestions how test design techniques shall be chosen due to an identified and given risk, which is a main contribution of our approach.

Stallbaum and Metzger made a first step towards automated generation of risk-based test suites based on previously calculated requirements metrics (e.g., [8][9]). A prototype research tool called RiteDAP has been presented as being able to generate test cases out of weighted activity diagrams in a two-stage process. At first, paths through the activity are derived in a non-risk based way. Secondly, the paths are ranked due to the risk they include. The traversal algorithm of the test case generator is predefined and not adjustable. The risk-based selection of test cases in that approach is a simple ordering of paths due to their subsumed risk exposures, what might be not sufficient. Our approach, in contrast, envisages that already the traversal algorithm shall be assigned to a certain risk level.

Bauer and Zimmermann have presented a methodology called *sequence-based specification* to express formal requirements models as low-level mealy machines for embedded safety-critical systems (e.g., [10] [11]). By doing so, they build a system model based on the requirements specification. Afterwards, the outcome of a hazard analysis is weaved into the mealy machine. The correctness of the natural language requirements is actually assumed to hold, so that there is no rigorous approach to verify or validate the natural language requirements prior to performing the hazard analysis. Finally, they describe an algorithm that derives test models that include critical transitions out of the system model for each single identified hazard in order to verify the implementation of a corresponding safety function. What they do not present is how to rank the critical transitions in the test models with respect to their risk priority. It is also not clear, whether and how the algorithm they present can be modified in order to vary the test case generation process. Apparently, this approach has ever produced a prototyping tooling beyond research projects.

Kloos has described an approach for transitioning from a fault tree as produced by a fault tree analysis (FTA) [12]. It is used in combination with a system model, expressed as mealy machine, to generate a test model. A test model is in their definition a system model with failure modes and critical transitions leading to the failure modes. As explicitly stated, this approach is dedicated to risk-based testing of safety functions for safety-critical systems. Although the authors claim their methodology to be risk-based, a clear method how the test case generation is actually influenced or guided by the identified risk is not provided.

The most recent approach to risk-based security testing using models is given by Zech for cloud environments [7]. The presented methodology is in a very early state, though. The author claims to fully automate the transition from system models over risk models to misuse cases and eventually to test code. The risk analysis is also planned to be carried out completely automatically by using a vulnerability repository. Neither one of the involved models has been described in greater detail, nor have the involved transformations been specified so far.

Chen discussed an approach for risk-based regression testing optimization [13]. In his approach the author applies a risk value to each test case to prioritize them. Based on these risk values, the test cases are comparable and can be prioritized to either be included in, or excluded from, a re-running regression testing process.

The Behavior Engineering (BE) methodology describes an approach to derive formalized requirements models, so called behavior trees (BT) out of informal, i.e. textual, requirements specifications. It was invented and firstly described in a series of paper (e.g., [17] [18]). Although BE is not related to risk analysis in the first place, we based our approach on it, since we are convinced that the rigorous methodology for requirements formalization is a good starting point to conduct testing in general, and risk-based testing in particular.

Although most of the literature presented above is dedicated to the ideas, principles or theories of risk-based testing, we do see a fundamental lack of concrete methodology on how to integrate the various pieces of information in a systematic way in order to guide the activities of test case derivation. We seek to provide testing experts with comprehensible instructions and a continuous toolset that is based on the principles of model-driven requirements engineering and model-based testing.

III. Risk-Based Testing in a Model-Based Way

Our approach strives to be generic and applicable for both systems that include functional-related risk mitigation measures (like counter-measures or safety, respectively security functions) and systems that do not include such measures. The latter kind of systems mostly refer to non-safety/security-critical systems, where certain execution paths are deemed critical nevertheless and need intensive testing as well.

In Figure 1, a high-level sketch of our methodology is depicted. In short, the steps are the following:

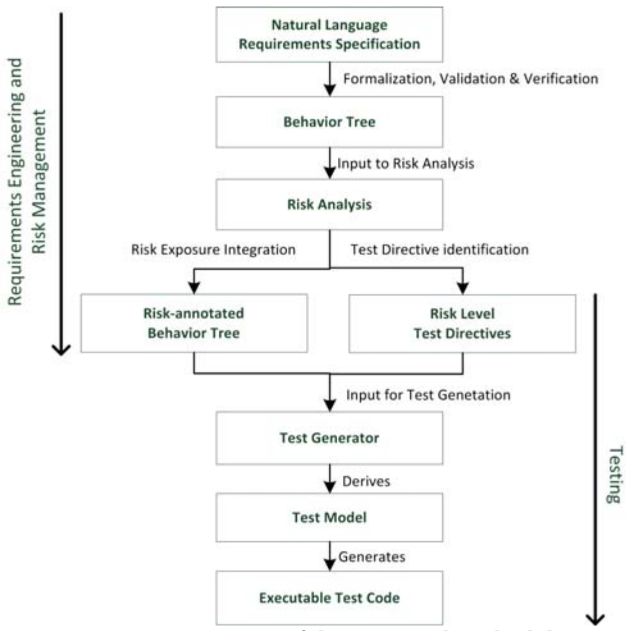
1. Formalize the requirements specification as integrated behavior tree
2. Augment the integrated behavior tree with risk information
3. Identify for each risk exposure in the integrated behavior tree an appropriate test directive and link them together
4. Pass both the risk-augmented integrated behavior tree and the test directive definition into a test generator

A. Definitions of risk, risky situation and risk exposure

Industry-relevant standards (such as [26][28][29]) for systems and software engineering or testing define *risk* (r) as a function of likelihood (l) or probability that a situation occurs during the execution of a system, which is deemed critical, multiplied by the severity (s) of the consequences that may happen if the risky situation is not mitigated, i.e.,

$f(R) = l * s$.

Beside the term *risk*, there are two other terms denoted in industry standards that apply to the above given definition: *Risk factor* [27] and *risk exposure* [26].


Natural Language Requirements Specification
Formalization, Validation & Verification
Behavior Tree
Input to Risk Analysis
Risk Analysis
Risk Exposure Integration
Test Directive identification
Risk-annotated Behavior Tree
Risk Level Test Directives
Input for Test Genetation
Test Generator
Derives
Test Model
Generates
Executable Test Code
Requirements Engineering and Risk Management
Testing
Figure 1. Overview of the presented methodology

As mentioned before, risk-based testing aims at testing the most critical situations at first and more thoroughly. However, we find that the term critical may cause misunderstandings since people may implicitly think of safety-critical systems. To prevent the reader's confusion, we rather use the term *risky situation* instead. A risky situation may occur in any kind of system (safety-critical or not) and describes a foreseeable sequence of events that leads to a situation where a failure of the systems causes an inacceptable loss (of data, reputation, life, money etc.) [25]. Finally, we use the term *risk* to denote an uncertain event or condition that, if occurs, has a negative effect on the system, and the term *risk exposure* as the comparable value that determines how risky a certain situation really is compared to other situations.

B. Towards risk-annotated requirements models

Risk management activities are performed on an information source that provides the risk experts with indications what might go wrong in the system in the field. Each system development project starts usually with a set of requirements that the intended system shall realize. Requirements are normally captured in software/system requirements specifications (SRS) that are structured in a certain way [30]. As already mentioned before, risk analysis activities or simple prioritization considerations are part of almost any development project. At the level of functional-related risk analysis, the SRS deemed to be one of the important information sources, risk experts should take into account. Once the risk analysis has been performed, the risk exposure is usually integrated with the SRS. This leads to a risk-annotated SRS.

Unfortunately, today's requirements specifications are mostly still textual. That entails some inherent problems such as ambiguity due to informal and imprecise textual specifications or the lack of human beings to grasp complex and comprehensive textual specifications. Most recent activities in the realm of requirements engineering strive to employ model-based techniques in order to produce less ambiguous and inconsistent SRSs. As mentioned before, one of these approaches is BE. It is an intuitive yet effective methodology to formalize the functional aspects of natural language requirements for further validation and/or verification activities. BE has been proven extremely beneficial in large-scale industry projects [19]. BE defines two core phases to transform informal requirements into formalized BTs, expressed with the Behavior Modeling Language (BML [36]). The first activity is called *requirements formalization* and provides a well-defined formalization strategy that is, each informal requirement will be translated into a *Requirements Behavior Tree* (RBT). A premise of BE is to stick as close as possible with the vocabulary which was used for expressing the natural language requirement with the advantage of removing any ambiguity being present in the natural language. An RBT comprises the behavioral flow of only one single requirement, namely the requirement it originated from. This keeps the complexity manageable, even of extremely large-scale systems. Hence, the complexity of an entire translation of large-scale requirement specifications is narrowed to the complexity of translating single requirements solely and in a repetitive manner. Myers [37] called this "… an approach with a minimized local problem space that remains constant regardless of the size of the global problem space."

After finishing the formalization of each requirement, the key phase for revealing and detecting flaws in the requirements specification takes place, the *Fitness-for-Purpose* (or simply *integration)* phase. Requirements commonly interact with other requirements, meaning, in a consistent requirements specification without gaps there are intersection points where requirements can be integrated with each other. To identify these points and to continually and rigorously integrate the RBTs with each other is the purpose of the integration phase. Integration is done by seeking parts in the RBTs which are logically identical. In practice, it is often the case that the root node of one RBT occurs elsewhere in another (or multiple) RBTs. Once such potential integration points are identified, the involved RBTs are integrated with each other [17]. The outcome of this activity is the so called *Integrated Behavior Tree* (IBT), expressing a behavioral and compositional overview of the requirements specification of the system. During the integration of RBTs, gaps and ambiguities within the requirements specification can be effectively identified, since a missing or ambiguously stated natural language requirement leads to situation in which an RBT cannot be integrated. If such a situation is detected, an issue has to be recorded and involved stakeholders have to decide how to resolve this situation. Thus, integration aims at improving the quality of the original requirements specification as well as creating an entire overview of the compositional and behavioral intention of the intended system. At this point in time, the requirements specification comprises all information being relevant for the development team to start their activities.

The starting point of our methodology is the outcome of the requirements integration phase with BE, which results in a requirements model expressed as integrated behavior tree (see Figure 1). The requirements model is passed afterwards to the risk experts, which also benefit from the integrated view on the requirements. It allows experts to consider potential failure (no matter if safety/security-critical or not) by traversing or even simulating the behavior of the system captured in a BT. For example, Grunske has presented approaches on how BTs can be leveraged for semi-automated hazard analysis [21]. How the actual risk analysis task is performed is not addressed by our methodology.

Risk analysis copes with risky situations by identifying risk mitigation actions. Risk mitigation may target several aspects of a development project such as organizational, process-related, functional or technical aspects. An organizational mitigation of risk might be to allocate only experienced and certified personnel to stem the project [2]; a process-related one that sufficient testing and manual inspections must be performed, whereas a technical mitigation action might be to rely on well-known and already established software technologies solely. Functional risk mitigation often includes the identification of functional counter-measures that reduce the risk exposure by the system, if correctly implemented. In case there have been functional counter-measures defined that shall mitigate risky situations, they have to be included into the requirements model together with the risk exposure. We end up with a functional-related requirements model that is augmented with risk exposures for risky situations. We call this a *risk-annotated behavior tree (*Figure 2).

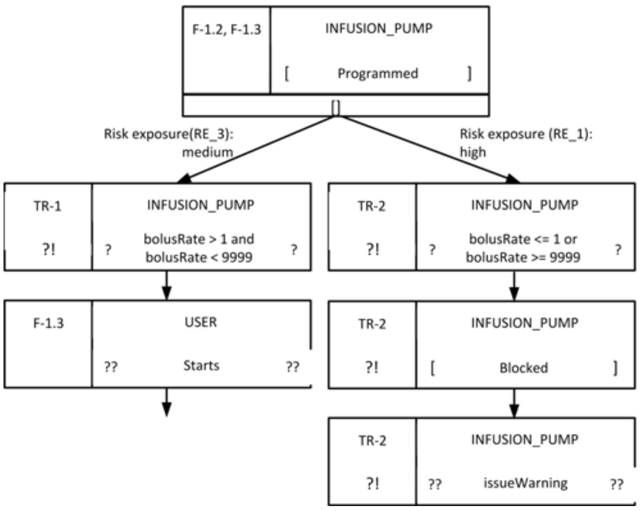


Figure 2. Risk-annotated BT (taken from [31])

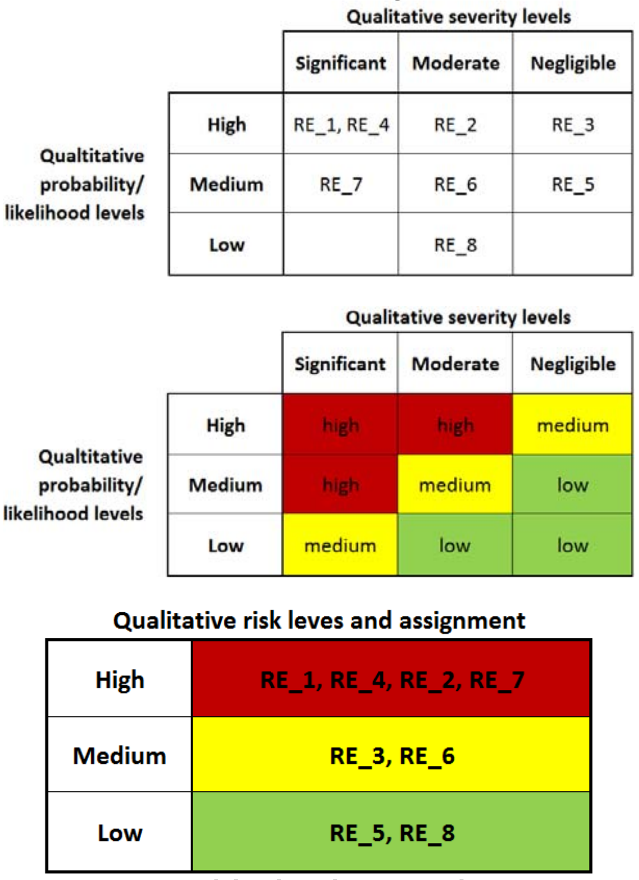
C. The role of risk levels and risk matrices

An important concept in our methodology of risk-optimized test case generation is *risk level*. Following the ISTQB glossary [28] a risk level indicates the importance of an identified and assessed risk, so it serves the purposes for comparing risks with other risks. Risk levels can be expressed either qualitatively or quantitatively. An often used qualitatively scale for risk levels is *low, medium, high*;however, there is actually no restriction on the number of risk levels being used.

One possible approach, especially in qualitative risk assessment, is to combine risk levels with a risk matrix. A risk matrix is a two-dimensional table for combining likelihood and severity of a risk. An example for risk levels and risk matrixes is shown in Figure 3.

The uppermost table depicts a risk matrix with qualified values for both likelihood and severity (see also [25]) and assigned risk exposures. The middle table assigns cells of the upper risk matrix with risk levels. For the sake of simplicity, we will stick with the aforementioned three risk levels *high*, *medium* and *low*.
Afterwards, the actual risk exposures to risk level assignment can be derived out of the two previous tables.

The result is kind of an instantiation of the risk matrix template plus the assigned risk levels for its cells. Risk exposures that are classified by the same risk level are considered to have an equally negative impact on the system if the corresponding risky situation leads to a system failure. This is shown in the lower table of Figure 3.



Qualtitative probability/ likelihood levels		Qualitative severity levels		
		Significant	Moderate	Negligible
	High	RE_1, RE_4	RE_2	RE_3
	Medium	RE_7	RE_6	RE_5
	Low		RE_8	

Qualtitative probability/ likelihood levels		Qualitative severity levels		
		Significant	Moderate	Negligible
	High	high	high	medium
	Medium	high	medium	low
	Low	medium	low	low

Qualitative risk leves and assignment

High	RE_1, RE_4, RE_2, RE_7
Medium	RE_3, RE_6
Low	RE_5, RE_8

Figure 3. Risk levels, risk matrix, risk assignment

D. *Systematic test case derivation using test directives*

A test directive is an additional piece of information within the test model that describes precisely what test derivation technique and strategy shall be used by the test generator to generate test cases for a certain risk level. This implies that test directives are bound to risk levels and transitively to the risky situation in the model. Utting [20] provides a good overiew of test derivation strategies for formal, graph-based models such as transition or state coverage.

In risk-based testing, we want to ensure that more risky situations are tested more thoroughly, because they are deemed more critical. It holds true for all risky situations that test cases try to provocate the risky situation to evaluate whether they appear or not. A simple and intuitive interpretation of *more thoroughly tested* for the example depicted in Figure 2 is that we want to ensure that test cases for the risk exposure RE_3 are more elaborated in terms of both structure and data than the ones for RE_3. There are many ways to come to those more elaborated test cases, for example by using different data coverage strategies (e.g., simple equivalence classes for RE_3 in opposite to boundary value analysis for RE_1) or structural coverage criteria (e.g., shortest path into the risky situation RE_3 in opposite to a path that is longer for RE_1). In our methodology, we capture the information on how to derive test cases for a certain risk level (and thus risk exposures) in a test directive. They are the fundamental means in our methodology to enable an automated derivation of risk-optimized sets of test cases as depicted in Figure 1. They make the entire test design activities more systematic, understandable and even more important reproducible. Additionally, the entire test generation process can be easily adjusted to changed needs by just re-defining a test directive's strategy.

The task of defining test directives for certain risk levels is most crucial in our methodology, since it has a crucial impact on the entire risk-based approach. This task requires the intellectual power of experienced personnel in both the current domain and testing. We believe it should not, or even cannot, be performed automatically. However, the actual test case generation approach according to the test directives bears an enormous automation potential. It allows the labor-intensive, error prone and time-consuming manual tasks to be outsourced to an automaton, such as a test case generator.

Eventually, after test case generation has been carried out automatically, a test model is created that contains all the risk-optimized test cases that adhere to the test directive. After execution, the test results analysis takes place. An important outcome of the result analysis is whether the initial assumptions on the risk were properly assessed. In any case, the test results have to be taken into account for a new development cycle, if there is one, in order to adjust the initial assessments with empiric data taken from the test process.

IV. Tools and Technology Landscape

For the implementation of our methodology, a consequent and integrated tooling landscape and modeling notation are required. In former research projects (e.g., [31]), we identified parts of that tooling landscape. Augmented with the needs for the further elaborated methodology presented in this article, the current tooling landscape and modeling notations are required:

- A language to specify behavior trees based on the Unified Modeling Language (UML) [24] and tooling to perform BE [1]
- Risk extension for behavior trees
- A language and tooling to capture risk matrices, risk levels and testing directives
- A language and tooling to express test models
- A language for executable test scripts

Our own premise for the implementation is to rely on established and well-known technologies and modeling notations instead of reinventing the wheel by using another proprietary solution. We decided to apply UML for all parts of the methodology by using so called *profiles*. A profile is a subset of the UML that adds domain-specific concepts and semantics to UML.

For the BE-related parts of the methodology, we specified a UML profile for the BML (called UBML) that already integrates risk information following the proposed method by Bran Selic [38]. A BT represents the behavioral description of single or integrated requirements. The tree itself is embedded into a surrounding, virtual frame that co-ordinates the process flow. Each node in a BT has a tight interlinking with a component contained in the composition tree, expressing that the behavior exhibited by that node will be executed on the linked component. There is an almost identical diagram type in the UML, namely the *activity diagram*. Activities describe control (and data) flows similar to BTs. Activities are constructed out of actions and edges. An action represents the fundamental and indivisible unit of an executable functionality that may operate on objects. Edges connect actions with each other. Given those ingredients, an activity appears appropriate to be customized for expressing BTs. To keep the analogy with BML, for each behavior node of BML [36] a direct counterpart stereotype has been created in UBML, such as *BehaviorTree, Selection*, *Guard*, *Event.* Structural aspects like components and messages are included, too, partially represented as stereotype (e.g. the stereotype *Message* extends the UML metaclass *Signal*) and partially reusing plain UML (e.g. UML metaclass *Component* and the component diagram are used to model BE components). As example, see the BT expressed as UBML in Figure 4. It is very similar to the original BML notation as depicted in Figure 2.

The test model artifact, as depicted in Figure 1, is expressed with the UML Testing Profile (UTP) [23]. UTP extends UML with test-relevant artifacts, which suits our needs. As test execution language we rely on the Testing and Test Control Notation version 3 (TTCN-3) [35]. All of these technologies are fostered by non-profit organizations (e.g., OMG [32], ETSI [34]), what guarantees vendor and methodological independence as well as continuous maintenance.

We have not yet specified precisely on how to express model risk matrices, risk levels and test directives in UML. In addition, the dependencies among risk exposure (as part of UBML), risk level and test directives have to be established as well. An early implementation of test directive guided test case generation has already been presented [15], and a more elaborated one will be presented in [16]. There is currently an ongoing discussion in the UTP working group [33] whether test directives might be incorporated into the specification. All modeling languages and tooling facilities mentioned above are or will be integrated into our test modeling environment Fokus!MBT, a UTP-based test modeling tool.

V. Conclusion and Further Work

In that paper, we presented an overview of a noval risk-based testing approach that relies on the principles of model-based testing. Our idea is based on test directives as interpretation of test case derivation techniques that support systematics, transparency and reproducibility of the test derivation task. We do not claim that our methodology is completely automated, because we do believe there is a need for intellectual creativity that can only be carried out manually, even if we rely on the principles of model-based engineering. We doubt the feasibility of just pressing a *magic button* and a risk-optimized set of test cases will be generated automatically. However, there is great potential in expressing suitable key artifacts of a system development process with semi-formal models. It allows capturing the intellectual power of experts in a computer-readable format, so that labor-intensive tasks can be carried out by an automaton.

We are going to apply this approach to more case studies in order to get empirical results for our methodology. What we have done so far was a proof-of-concept, so there have been some lessons learned that have impact on the refinement of the modeling methodology. Another important work to be done is to describe the entire modeling approach on a more technical level in order to explicitly show how things are interconnected with each other semantically and technically.

Further technical work will, in particular, address the target in particular the definition of a precise and stable methodology for doing BE with UBML.

The main focus, however, will be set to the combination and integration of test directives and risk levels, since this is the main contribution of our risk-based methodology and actually the most added value to the current state of the art in the realm of risk-based testing.

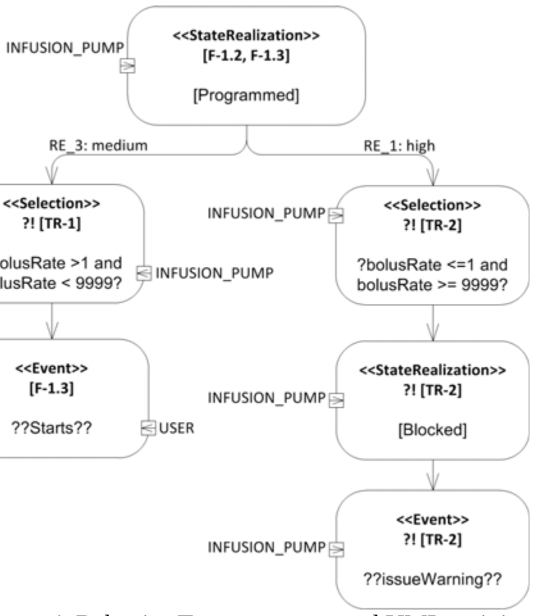


Figure 4. Behavior Tree as stereotyped UML activity diagram

ACKNOWLEDGMENT

This work was partially supported by the projects ROTESS [31] and BTTest (DAAD project number 50449190).

REFERENCES

[1] G. J. Bach, “Heuristic Risk-Based Testing”, Software Testing and Quality Engineering Magazine, November 1999, pp. 96-98.

[2] F. Redmill, “Exploring Risk-based Testing and Its Implications”, 1. Software Testing, Verification & Reliability, 14(1), 2004, pp. 3-15.

[3] F. Redmill, “Theory and practice of risk-based testing”, Software Testing, Verification & Reliability, 15(1), pp. 3-20 2005.

[4] S. Åmland, “Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study”, Journal of Systems and Software 53(3), 2000, pp. 287-295.

[5] S. Åmland, “Risk Based Testing and Metric”, 5th International Conference EuroSTAR 1999, Barcelona, Spain, 1999.

[6] S. L. Pfleegr, “Risky business: what we have yet to learn about risk management”, Journal of Systems and Software 53(3), Elsevier, 2000, pp. 265-273.

[7] P. Zech, “Risk-Based Security Testing in Cloud Computing Environments”, 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST), 2011, pp. 411-414.

[8] H. Stallbaum and A. Metzger, “Employing Requirements Metrics for Automating Early Risk Assessment”, In: Proceedings of the Workshop on Measuring Requirements for Project and Product Success, MeReP07, at Intl. Conference on Software Process and Product Measurement, Spain, 2007, pp. 1-12.

[9] H. Stallbaum, A. Metzger, and K. Pohl, “An Automated Technique for Risk-based Test Case Generation and Prioritization”, In: Proceedings of the 3rd Workshop on Automation of Software Test, AST'08, at 30th Intl. Conference on Software Engineering (ICSE), Germany, 2008, pp. 67-70.

[10] T. Bauer et al., “From Requirements to Statistical Testing of Embedded Systems”, In: Software Engineering for Automotive Systems, (ICSE), 2007, pp. 3-10.

[11] F. Zimmermann, R. Eschbach, J. Kloos, and T. Bauer, “Risk-based Statistical Testing: A Refinement-based Approach to the Reliability Analysis of Safety-Critical Systems”, In: Proceedings of the 12th European Workshop on Dependable Computing (EWDC), France, 2009.

[12] J. Kloos, T. Hussain, and R. Eschbach, “Risk-Based Testing of Safety-Critical Embedded Systems Driven by Fault Tree Analysis”, In: Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST 2011), IEEE Computer Society, Berlin, 2011, pp. 26-33.

[13] Y , Chen,. R. Probert, and P. Sims, “Specification-based Regression Test Selection with Risk Analysis”, In: Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research (CASCON '02), 2002, pp. 1.

[14] M.-F. Wendland, I. Schieferdecker, and A. Vouffo Feudjio, “Requirements-driven testing with behavior trees”,. In: Proceedings of the Fourth IEEE International Conference on Software Testing, Verification, and Validation Workshops (ICST 2011), IEEE Computer Society, 2011, Germany, 2011, pp. 501-510

[15] M.-F Wendland, J. Großmann, and A. Hoffmann, “Establishing a Service-Oriented Tool Chain for the Development of Domain-Independent MBT Scenarios”, In: Proceedings of 7th Workshop System Testing and Validation (STV’10), IEEE Press, 2010, pp. 329-334.

[16] M.-F. Wendland, M. Kranz, A. Hoffmann, and I. Schieferdecker, “Integration of arbitrary test case generators into UTP-based test models”, 2nd ETSI Model-Based Testing User Conference (MBTUC), Tallinn, Estonia, 2012.

[17] R. G. Dromey, “From Requirements to Design: Formalising the Key Steps” (Invited Keynote Address), In: IEEE International Conference on Software Engineering and Formal Methods (SEFM'03), Brisbane, Australia, 2003, pp. 2-11.

[18] R. G. Dromey, "Genetic Design: Amplifying Our Ability to Deal With Requirements Complexity", in S. Leue, and T.J. Systra, Scenarios, Lecture Notes in Computer Science, LNCS 3466, 2005, pp. 95 - 108.

[19] D. Powell, “Requirements Evaluation Using Behavior Trees: Findings from Industry”. In: Australian Conference on Software Engineering (ASWEC’07), Australia, 2007.

[20] U. Utting and B. Legeard, “PracticalModel-Based Testing – A Tools Approach”. Morgan Kaufmann Publ. (2007)

[21] L. Grunske, “An Automated Failure Mode and Effect Analysis based on High-Level Design Specification with Behavior Trees”, In: Proceedings of International Conference on Integrated Formal Methods (IFM), 2005, pp. 129-149.

[22] K-S. Soon, T. Myers, P. Lindsay, and M.-F. Wendland, “Execution of natural language requirements using State Machines synthesised from Behavior Trees”, The Journal of Systems & Software 85, Elsevier, 2012, pp. 2652-2664.

[23] Object Management Group (OMG): Unified Modeling Language (UML). http://www.omg.org/spec/UML/. Last visit: January 05, 2012

[24] Object Management Group (OMG): UML Testing Profile, Version 1.1 – Beta 1. URL: http://www.omg.org/cgi-bin/doc?ptc/2011-07-19, Last visit:

[25] International Organisation for Standardisation (ISO): ISO:14971 – „Medical devices -- Application of risk management to medical devices“, http://www.iso.org/iso, Last visit: January 05, 2012

[26] International Organisation for Standardisation (ISO): ISO/IEC 16085:2006– Sytems and software engineering, 2006.

[27] International Organisation for Standardisation (ISO): ISO/IEC/IEEE 24765:2010 – Sytems and software engineering, Vocabulary, 2010.

[28] International Software Testing Qualifications Board (ISTQB): ISTQB/GTB standard glossary for testing terms. http://www.software-tester.ch/PDF-Files/CT_Glossar_DE_EN_V21.pdf. Last visit: October 17, 2012.

[29] IEEE Standards Association (IEEE): 829-2008 – IEEE Recommended Practice for Software Requirements Specifications, 2008.

[30] Institute of Electrical and Electronics Engineers (IEEE): 830-1998 – IEEE Recommended Practice for Software Requirements Specifications, 1998.

[31] ROTESS project: Risk-oriented testing of embedded, safety-critical systems. http://www.fokus.fraunhofer.de/de/motion/projekte/laufende_projekte/ROTESS/index.html, last visit: October 17, 2012.

[32] Object Management Group (OMG): http://www.omg.org. Last visit: October 17, 2012.

[33] Object Management Group (OMG) UML Testing Profile Revision Task Force: http://www.omg.org/techprocess/meetings/schedule/UTP.html (access restricted). Last visit: October 17, 2012.

[34] European Telecommunications Standards Institute (ETSI): http://www.etsi.org. Last visit: October 17, 2012.

[35] Testing and Test Control Notation Version 3 (TTCN-3): http://www.ttcn3.org/. Last visit: October 17, 2012.

[36] Behavior Modeling Language (BML): Behavior Tree Notation v1.0, http://www.behaviorengineering.org/docs/Behavior-Tree-Notation-1.0.pdf. Last visit: October 17, 2012.

[37] T. Myers, “The Foundations for a Scaleable Methodology for Systems Design” , PhD Thesis, School of Computer and Information Technology, Griffith University, Australia, 2010.

[38] B. Selic, “A Systematic Approach to Domain-Specific Language Design Using UML”, In: Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07), USA, 2007, pp. 2-9.

An Automatic Security Testing approach of Android Applications

Stassia R. Zafimiharisoa
LIMOS - UMR CNRS 6158
Université Blaise Pascal, France
email: s.zafimiharisoa@openium.fr

Sébastien Salva
LIMOS - UMR CNRS 6158
Université d'Auvergne, France
email: sebastien.salva@udamail.fr

Patrice laurençot
LIMOS - UMR CNRS 6158
Université Blaise Pascal, France
email: laurencot@isima.fr

***Abstract*—In this paper, we propose a security testing approach which aims to check whether Android applications are not vulnerable to malicious *intents*. An intent is an IPC (Inter-Process Communication) mechanism which is used to compose Android components together to form a whole application. From Manifest files, which provide information about Android applications, and based on the vulnerabilities expressed within test patterns, we automatically generate JUnit test cases that can detect intent-based vulnerabilities. Using formal methods, executable security tests are then automatically generated from any Android applications.**

***Keywords*-security testing, Android applications, model-based testing.**

I. Introduction

As mobile usage grows, so should security: this sentence summarises well the conclusions of several recent reports [1] about mobile application security with platforms such as iOS, Android or Windows Phone. These reports also show that an alarming amount of malicious software is currently available. On the other side, end-users wish using trustworthy mobile applications; so, more and more developers have in mind that security must not be left aside. Nevertheless, eliminating security vulnerabilities in mobile applications is not so obvious since these ones depend on different concepts such as composition of software components, which have not been completely covered by security studies.

This paper proposes a work in progress about an automatic security testing method for Android applications. Since mobile application security is a tremendous research field, we focus here on the composition of Android components: most of mobile applications gather a set of components composed together statically or dynamically. With Android, these components are glued by means of the concept of *intent* which is an IPC mechanism. In reference to the Android documentation [2], an intent is an abstract description of an operation to be performed. Basically, an intent is a data structure holding an abstract description of an action to be executed by another component. This one is generally used to call or launch another component, e.g., an activity (a component which represents a single screen), or a service (component which can be executed in background). For example, an intent is used when a user wishes to search for a contact for later sending an email.

The use of intents introduces vulnerability issues (availability, integrity issues, etc.) when intents are composed of malicious data. The security testing method developed in this paper aims at detecting whether components are vulnerable to these malicious intents. The first step of this method is the generation of formal models ioSTS (input output Symbolic Transition Systems [3]) from Android applications, and more precisely from *Manifest* files. These Manifest files which can be found in Android application packages (apk), contain the list of Android components, permissions of these components and a list of intents that can be performed. More precisely, with introspection of the different compiled components, we derive an incomplete class diagram which will be used to refine and reduce the test case generation. We also parse Manifest files to derive one ioSTS for each component that describes the intent communication. Vulnerabilities (that would be tested) are also described formally with ioSTS called *test patterns*. We give an example of test pattern for availability testing in this short paper. However, our method can be also used to test other security concepts such as integrity or authorisation. Then, our method constructs test cases by combining the component models with test patterns. The resulting ioSTS test cases are finally translated into JUnit test cases to be later executed with classical development tools.

The paper is structured as follows: Section II presents some works dealing with Android security and security testing. The methodology is described in Section III. We conclude in Section IV.

II. Related work

Several works, dealing with Android security, have been proposed recently. Some works focused on the definition of a more secure Android framework. For instance, different actions were monitored to check the system integrity in [4]. These approaches offer a different point of view, in comparison to the work tackled in this paper which aims at detecting applications vulnerabilities, since they target the discovery of framework vulnerabilities. Analysis of IPC were studied by E. Chin et al. [5]. They described the permission system vulnerabilities that applications may exploit to perform unauthorised actions. We have exploited the described vulnerabilities to model test patterns which can be used to generate test cases. We have also completed this vulnerability list by means of the recommendations and vulnerabilities referenced by the OWASP (Open Web Application Security Project) communities [6]. At the moment, only Web applications are considered, but this gives some indications for mobile applications too. Then, our approach do not require Android framework extension and deal with a large coverage of Android applications vulnerabilities.

Security testing, based upon formal models, has been studied in several works [7]. Mouelhi et al. propose to produce test cases from security policies described with logic based languages(OrBAC). This ones permits to decribe specific properties such as access control [8].In these two previous works, test cases are generated from specification and invariants or rules describing security policies. Starting with a similar approach, we also use formal models to describe Android applications and vulnerabilities, and we propose to push one step beyond in the automatic generation of partial models to produce tests.

In [9], a threat model-based security testing approach is presented. This method produces security test cases from threat trees and transforms them into executable tests. Using trees is intuitive for industry but the use of formal languages offer several other advantages such as the description of the testing coverage.

III. Security testing methodology

In this section, we present an automatic security testing approach which can be used to detect vulnerability issues based on the intent mechanism. The different steps of the method are illustrated in Figure 1. Initially, several models are extracted from an Android application: from the compiled classes, we extract a partial class diagram by introspection. This one lists the components and gives the associations between classes. We also produce one ioSTS per component. These ioSTSs will be combined with ioSTS test patterns, which describe vulnerabilities, to produce test cases. We obtain symbolic test cases which need to be concretised with values. Finally, these latter are translated into JUnit test cases to be executed with classical development tools. These steps are more detailed in the remainder of the paper.

A. Model generation

1) Android components and interactions: Android applications are usually constructed over a set of components. A component belongs to one of the four basic components types that are *activities* (user interfaces), *services* (background processing), *content providers* (database management) and *broadcast receivers* (broadcast message handling). The communication between these components is performed by intents. An intent is a kind of bundle of information which gathers: the action which has to be performed, data specifies the type of data to operate on or the MIME

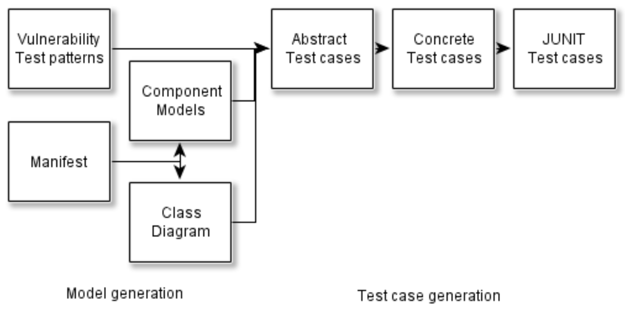


Figure 1. Test case generation.

(Multipurpose Internet Mail Extension) type, a category and eventually some extras which represent variable affectations which are required for the action [2]. Intents are divided into two groups: explicit intents where the target component is designated and must be launched and implicit intents (the most generally ones) where a component of another application is going to be used. The mapping of an intent to a component is expressed in Manifest files with *intent filters*. The latter are given for each component. Android Manifest file is mandatory in a project and it specifies system information about the application. In this paper, Manifest files are used to extract the list of Android components deployed in the application and their intent filters.

2) Component partial model extraction: A simplified class diagram, giving the Android components of the application, is initially computed. The relationships between components are established by applying reverse engineering based on introspection in Java. This step aims to later reduce the test case generation. For instance, we only produce test cases to check data integrity on components which have access to databases or to content-provider components only. Our approach combines introspection with the detection of specific Android component methods defined in [2], which give details about intents, and the access to data.

Then, we generate one ioSTS for each component to describe the intents that it can accept. An ioSTS is a kind of automaton model which is extended with a set of variables, with transition guards and assignments, giving the possibilities to model the system state and constraints on actions. Its complete definition can be found in [3]. An ioSTS is defined by the tuple $< L, l_0, V, V_0, I, \Lambda, \rightarrow>$, where: L is the finite set of locations, with l_0 the initial one, V is the finite set of internal variables, while I is the finite set of interaction ones. Λ is the finite set of symbols and $\rightarrow$ is the finite transition set.

IoSTS are generated with Algorithm 1. It constructs one partial ioSTS specification per component listed in the Manifest file. It is also based on the recommendations provided in [2] to describe the different responses that can be observed after the receipt of an intent by a component. In this paper, we focus on activity and service components only. Intents are expressed with the output action *!intent* composed of specific variables for the action, the data, the category, the type and the extras. IoSTS are constructed according to the intent action type. The set denoted A_o is composed of the list of Actions that requires response message after the intent, e.g.,the action PICK. A_q is the set composed of the list of Actions that do not need response, e.g.,the action VIEW.

An minimalist and straightforward example of specification is given below. It illustrates one intent composed of the action *VIEW* that is called to display information about the first person in the contact list of the mobile phone.

$$l_0 \xrightarrow[action=ACTION\_VIEW \wedge data=../contact/people/1]{?intent(action,data)} l_1 \xrightarrow{!display(ActivityA),[A.resp=isNull]} l_0.$$

Algorithm 1: Partial Specification set Generation

```
input : Manifest file $MF$
output: An ioSTS set $\{S_{ct} \mid ct \text{ is a component}\}$
foreach component $ct$ in $MF$ do
    foreach IntentFilter $it(a,d,c,v)$ of $ct$ in $MF$ do
        if $ct.type = activity$ then
            if $a \in A_o$ then
                $l_0 \xrightarrow{?intent(a,d,c,v),[a.in(A_o)]}_{S_{ct}} (l_{it,1})$
                $\xrightarrow{!display(ActivityA),[A.resp \neq Null]}_{S_{ct}} l_0$
            else if $a \in A_q$ then
                $l_0 \xrightarrow{?intent(a,d,c,v),[a.in(A_q)]}_{S_{ct}} (l_{it,1})$
                $\xrightarrow{!display(ActivityA),[A.resp = Null]}_{S_{ct}} l_0$
            else
                $l_0 \xrightarrow{?intent(a,d,c,v),[a.in(A_c)]}_{S_{ct}} (l_{it,1})$
                $\xrightarrow{!display(ActivityA)}_{S_{ct}} l_0$
        if $ct.type = service$ then
            if $a \in A_o$ then
                $l_0 \xrightarrow{?intent(a,d,c,v),[a.in(A_o)]}_{S_{ct}} (l_{it,1})$
                $\xrightarrow{!running(ServiceS),[S.resp \neq Null]}_{S_{ct}} l_0$
            else
                $l_0 \xrightarrow{?intent(a,d,c,v)}_{S_{ct}} (l_{it,1})$
                $\xrightarrow{!running(ServiceS)}_{S_{ct}} l_0$
```

B. Vulnerability modelling with ioSTS test patterns

We propose to define security vulnerabilities of Android components with ioSTSs in order to combine them later with ioSTS specifications. Our method can take different security concepts, e.g., availability, integrity or authorization in condition that they could be modelled with ioSTSs. One vulnerability is expressed by a tree whose final locations are labelled by a verdict in $\{pass, fail, inconclusive\}$: branches beginning from the initial location and ended by $fail$ express functional behaviours composed of malicious intents and responses which show the presence of the vulnerability. Branches ended by $inconclusive$ express functional behaviours which do not help to conclude on the presence or the absence of the vulnerability. Branches ended by $pass$ express functional behaviours which show the absence of the vulnerability.

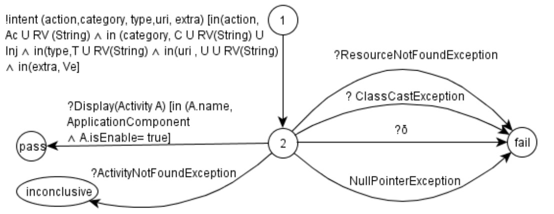

!intent (action,category, type,uri, extra) [in(action, Ac U RV (String) ∧ in (category, C U RV(String) U Inj ∧ in(type,T U RV(String) ∧ in(uri , U U RV(String) ∧ in(extra, Ve]
1
2
?Display(Activity A) [in (A.name, ApplicationComponent ∧ A.isEnable= true]
pass
?ActivityNotFoundException
inconclusive
?ResourceNotFoundException
? ClassCastException
?δ
NullPointerException
fail

Figure 2. Test pattern for availability

We only present a test pattern example to check availability in this paper. Availability of Android components means that they must respond despite the receipt of malicious intents. An availability test pattern for Activity components is shown in Figure 2. The other security properties are composed with more states. Availability issues are detected when exceptions such as *ClassCastException, NullPointerException, RessourceNotFoundException, etc.* are received [2]. So, the test pattern expresses that when one of these exceptions is observed, the fail location is reached. For instance, these exceptions can be received when there is no input validation of the URI path. System exceptions e.g., *ActivityNotFoundException, SecurityException, etc.* mean that the intent has been blocked directly by the Android system. This is expressed by transitions leading to the inconclusive location. Variables of the intent action take values in specific sets with the term *in*. These sets are required to target the test with specific values and above all to reduce the number of test cases. The set A_c is composed of the list of Actions found in the Android documentation [2]. In the same way, C is a specific set of categories, T a set of types, U a set of URI. Ve is a set of extras i.e., tuples (key, values) where the keys are extracted from the application package (apk) and values are chosen randomly. RV stands for a set gathering values known for relieving bugs and random values. For instance, for the type String, $RV(String)$ holds values such as ””, ”$”, ”&”, and random values. Inj gathers String values equal to XML and SQL injections.

C. Test case Generation

Test cases are generated by composing test patterns, modelling vulnerabilities with component specifications. As defined, a test pattern expresses one vulnerability in general

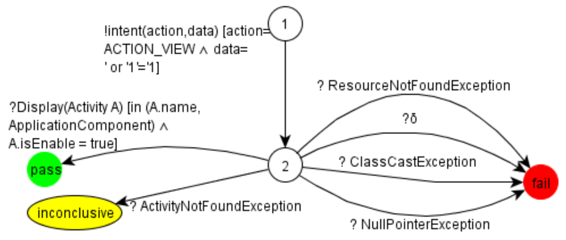

1
!intent(action,data) [action=ACTION_VIEW ∧ data=' or '1'='1]
2
?Display(Activity A) [in (A.name, ApplicationComponent) ∧ A.isEnable = true]
pass
? ActivityNotFoundException
inconclusive
? ResourceNotFoundException
?δ
? ClassCastException
? NullPointerException
fail

Figure 3. A test case example.

terms. This combination yields symbolic test cases which are specialised to test one vulnerability by means of the functional behaviours given in the specification. The combination of one test pattern TP with one specification S is performed by two steps: synchronous product of the specification with the test pattern to extract from the latter, the functional behaviours expressed in the specification, completion of this product with incorrect and inconclusive behaviours given in the test pattern by transition leading to fail or inconclusive respectively.

1) Test case concretisation: The resulting test cases are still composed of variables. To execute them, variables are replaced with values by using a pairwise technique [10] on the value sets provided by the test patterns (RV, Ac, etc.) instead of using a cartesian product. This technique helps to reduce the coverage of the variable domain by constructing discrete combinations for pair of parameters only. A final test case is shown in Figure 3. It illustrates a malicious intent transition composed of the classical SQL injection *' or '1'='1*.

2) Automatic unit test generation: IoSTS test cases are translated into JUnit test cases to be executed with Java development tools. This is done with an algorithm that can be summarised by: each ioSTS transition starting from the initial location corresponds to the launch of an intent on a component with its parameters defined in the ioSTS test case. Then, the following transitions that correspond to both observations and verdicts are translated into JUnit assertions. Since inconclusive verdict are not allowed with JUnit, it will be assigned to pass verdict and will be identified by specific annotations. The following example illustrates a JUnit test case.

```
public void testAvailability(){ ... setIntent(
    ACTION_VIEW, ' or
'1'='1}; try{ mActivity=getActivity();
assertTrue(currentView,isNotNull());
assertTrue(activityResult,isNull());}
catch (ClassCastException c) {fail(c.message);}
catch (ActivityNotFoundException a)
{assertTrue(a.message,true);} ... }
```

IV. Conclusion and Future Work

We have introduced a work in progress about Android security testing based on the notion of intent which can be considered as a glue of the components participating in applications. The intent concept offers some advantages and flexibility but is also a weak spot in security since attacks can be send to components. Our method generates models from Android Manifest files and constructs test cases from these models and vulnerabilities models. The originality of the method is to produce formal models from Android documentations and files extracted from Android applications automatically. The use of these formal models will be useful to express without ambiguity the coverage of the tests. The next step is to list all the possible intent-based vulnerabilities (we have collected five vulnerabilities at the moment) and to develop the corresponding testing tool to perform experiments on real applications.

References

[1] *IT Business: Android Security*, (June , 2012). [Online]. Available: http://www.itbusinessedge.com/cm/blogs/weinschenk/google-must-deal-with-android-security-problems-quickly/?cs=49291

[2] "Android developer page," (May 1, 2012). [Online]. Available: http://developer.android.com/index.html

[3] L. Frantzen, J. Tretmans, and T. Willemse, "Test Generation Based on Symbolic Specifications," in *FATES 2004*, ser. Lecture Notes in Computer Science, J. Grabowski and B. Nielsen, Eds., no. 3395. Springer, 2005, pp. 1–15.

[4] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically rich application-centric security in android," in *Proceedings of the 2009 Annual Computer Security Applications Conference*, ser. ACSAC '09, 2009, pp. 340–349.

[5] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proc. of the 9th International Conference on Mobile Systems, Applications, and Services*, 2011, pp. 239–252.

[6] OWASP, "Owasp testing guide v3.0 project," 2003, (accessed May 1, 2012). [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Testing_Project#OWASP_Testing_Guide_v3

[7] H. Marchand, J. Dubreil, and T. Jéron, "Automatic Testing of Access Control for Security Properties," in *TESTCOM/FATES 2009*, Nov. 2009.

[8] T. Mouelhi, F. Fleurey, B. Baudry, and Y. Traon, "A model-based framework for security policy specification, deployment and testing," in *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, 2008, pp. 537–552.

[9] K. H. S. K. Aaron Marback, Hyunsook Do and D. Xu, "A threat model-based approach to security testing," in *Softw. Pract. Exper*, 2012.

[10] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn, "Constructing test suites for interaction testing," in *Proc. of the 25th ICSE*, 2003, pp. 38–48.

An Integrated Process for Developing Safety-critical Systems using Agile Development Methods

Zhensheng Guo
Siemens AG, Corporate Technology
Erlangen, Germany
Joe.guo@siemens.com

Claudia Hirschmann
Siemens AG, Corporate Technology
Erlangen, Germany
Claudia.hirschmann@siemens.com

***Abstract* - This paper proposes a novel idea for developing safety-critical software-intensive systems by the use of agile development models. This idea comes from the dramatic increase of the complexity for constructing technical systems such as trains, cars and medical devices. Iterative and incremental development becomes more and more popular and effective in such fields. However, the development of safety-critical systems is strictly defined by relevant safety standards. A kind of finish-to-start relationship between the development phases is required implicitly. This paper explains an idea to integrate an iterative incremental development process into the strict safety development lifecycle. At the end of this short paper, an overview of the further activities is presented.**

Keywords-safety-critical software-intensive systems; IEC 61508; EN 50128; agile software development process; Scrum

I. Introduction

Software is replacing traditional mechanical and electrical components with an extremely high speed and huge extends. Many of such components are used in safety-critical systems, where the malfunctions of software could cause the damage of equipment, environmental pollution, even injury or death of human being. Typical examples of such systems are trains, cars, aircrafts, medical devices and nuclear power plants. To develop high quality software components, it is nowadays essential to use a suitable development process model. In [15], several typical process models are presented and compared for the suitability for developing safety-critical systems: One of the core results of the referenced paper is that strict finish-to-start process models such as the V- Model XT [14] are suitable to construct safety-critical systems, whereas agile process models, like Scrum, are not recommended.

Agile software development methods follow the principle of 'plan – build – revise' in short iterative cycles. At the end of each cycle they deliver incrementally ready product features. Agile methods provide good measures to handle the reality of software development with its late requirements, need for flexibility and fast reaction times; see [7].

The bases behind agile processes are the Manifesto for Agile Software Development [1] and the Principles behind the Agile Manifesto [2]. Agile processes welcome changing requirements, deliver working software frequently and in close cooperation with the customer trusting the motivated development team. But, agile methods, like e.g. Scrum [8], do not care about dedicated roles for quality management or safety management, or documentation.

One of the most used agile methods is Scrum (see [6]), which is continuously enhanced, like in "The Scrum Primer, version 1.2" [3] or "The Scrum Guide" [4].

Application of agile methods in safety-critical, regulated environments (as in medical technology) is discussed in science and industry, see the ScrumMed conference [5][9][10]. Application of agile methods in safety-critical systems with focus on IEC 61508 or EN 50128 is still a gap which we want to fill with our idea.

Nonetheless, many industry domains are using agile methods to reduce the project risk and increase better orientation, flexibility, transparency and even quality: So, there is a gap and practical need for finding out how to use iterative process models and agile methods save and effectively for developing safety-critical systems.

There is no specific process model required in the safety standards such as IEC 61508 [12] and EN 50128 [13]: in IEC 61508 a safety lifecycle is defined and required, in EN 50128 waterfall model and V-Model is referenced but not required. The safety standards only require certain activities and documentation with some quality according to the corresponding Safety Integrity Level (SIL). The appropriate process model can be decided by the individual project whereat the new version of EN 50128 even mentions the consideration of iterative development.

Therefore, an iterative process model for developing safety-critical systems becomes important: integrating the benefits without letting the drawbacks from safety view in.

II. Idea for developing safety-critical systems using agile development methods

Our idea is to map the activities of an agile process model into the safety lifecycle. The agile process model will be used, but not as the only process model. The traditional strict finish-to-start process model will be used as well. This limitation has the benefit that the required activities and documents of the individual phases are done in the required sequence of the safety standards.

The following section explains how the agile process model can be mapped into the safety lifecycle.

In order to make our idea general, we take the software safety lifecycle from the mother safety standard, IEC 61508, part 3. In the software safety lifecycle in IEC 61508 part 3, the activities and documents, which shall be done after each individual phase, are readily identifiable from the name of each phase of safety lifecycle.

Scrum shall contain the following activities and artifacts: Product backlog (user requirements) and Release plan; Sprint planning meeting and Sprint Backlog; Code including documentation after each sprint / increment; Review meeting and results; Retrospective.

Now, the Scrum activities and artifacts will be arranged in the phases of the safety lifecycle as described in Figure 1.

Figure 1 illustrates the idea of the solution for developing safety-critical systems using agile development methods from the Software safety lifecycle view point. It shows which agile elements would go into which step of the safety lifecycle: The Product Backlog from Scrum method will be applied for the SW safety requirements specification additionally. SW design & development will be performed according to Scrum's Sprints including the Sprint planning meeting, Sprint Backlog, the Daily Scrum Meetings, Sprint Review, and regular Sprint Retrospectives. The Sprint Review and Sprint Retrospective element from Scrum will be applied for the Software safety lifecycle step of SW aspects of system safety validation, as well, to facilitate stakeholder involvement and continuous improvement of the product and process.

The Sprints for the implementation of the different features could be performed in parallel and sequentially, depending on the dependability of the features that are implemented in the sprints and corresponding safety requirements respectively; so, the whole Software will be implemented iteratively through as many Sprints as needed.

Figure 2 illustrates the idea of the solution for developing safety-critical systems using agile development methods from the Scrum view point. It shows the Scrum method's framework form Product Backlog over Release Plan, Sprint Planning Meeting, Sprint Backlog into the Sprints ending with the Sprint Review Meeting and Sprint Retrospective Meeting, whose results flow back into the planning for the next Sprint. In order to ensure accurate consideration of safety issues, a Safety Manager will care for safety assurance throughout the whole process. So the Safety Manager will check the Product Backlog for certain safety goals and care for proper ranking and arrangement in the Release Plan,, Sprint Backlog and Sprint Review Result, and will monitor and track safety items during the Daily Scrum Meetings and the Sprint Retrospective Meetings. He supports the Scrum Master in all sprints, takes part in the Sprint Review and in the Sprint Retrospective to care for proper documentation, etc.

A second step of this integrated process is to integrate the Start and Done- criteria as a refinement of [11] into the software safety lifecycle. Such criteria define start and end of each sprint in the overall safety lifecycle.

The Done Criteria that are checked after each Sprint to determine whether a task of the Sprint is completely done or not, is steadily maintained and kept by the Safety Manager. These Done Criteria, which serve as checklist, will include for example safety relevant review activities, documentation, and safety measures.

One crucial factor for use of such an integrated process model is the correct definition of the sprints according to the separation of the non-safety-critical functions from the safety-critical functions and the decomposition of the safety functions regarding their criticalities and dependencies.

III. FUTHER ACTIVITIES

We are planning to use this integrated process model in the Scrum teams for developing safety-critical systems. Important is that the overall safety activities will be well-integrated and advised by the Scrum master and Safety manager. Of course, the recommended and required techniques according to the different SILs, Change Management like other required activities from the relevant safety standards will be used and integrated in each sprint.

IV. CONCLUSION

In this paper, we presented a novel idea to integrate agile methods / Scrum into the safety lifecycle to enable iterative incremental development in safety-critical systems.

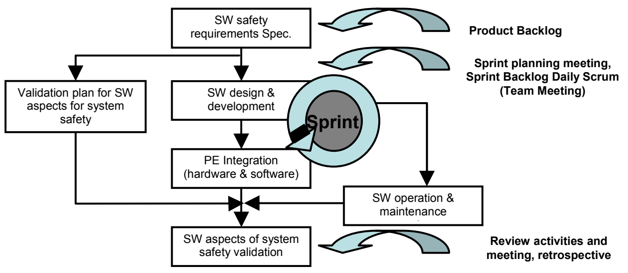

SW safety requirements Spec.
Product Backlog
Sprint planning meeting, Sprint Backlog Daily Scrum (Team Meeting)
Validation plan for SW aspects for system safety
SW design & development
Sprint
PE Integration (hardware & software)
SW operation & maintenance
SW aspects of system safety validation
Review activities and meeting, retrospective

Figure 1. Software safety lifecycle with Scrum elements.

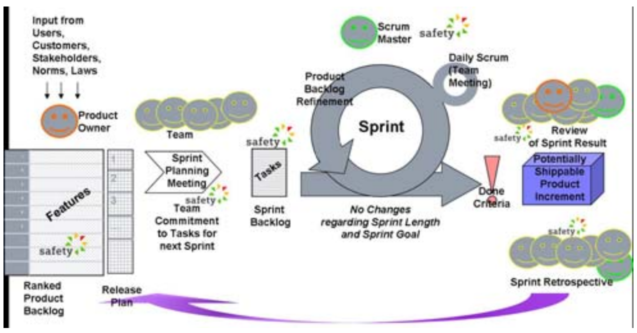

Input from Users, Customers, Stakeholders, Norms, Laws
Product Owner
Team
Scrum Master
safety
Daily Scrum (Team Meeting)
Product Backlog Refinement
Sprint
Review of Sprint Result
Potentially Shippable Product Increment
Sprint Planning Meeting
Team Commitment to Tasks for next Sprint
Tasks
Sprint Backlog
No Changes regarding Sprint Length and Sprint Goal
Done Criteria
Features
Ranked Product Backlog
Release Plan
Sprint Retrospective

Figure 2. Safety-oriented Scrum process framework.

V. REFERENCE

[1] http://agilemanifesto.org/. [retrieved: September, 2012].

[2] http://agilemanifesto.org/principles.html. [retrieved: September, 2012].

[3] Pete Deemer, Gabrielle Benefield, Craig Larman, and Bas Vodde: "The Scrum Primer, version 1.2", 2010, pp. 4-16.

[4] Ken Schwaber and Jeff Sutherland: “The Scrum Guide, The Definitive Guide to Scrum: The Rules of the Game”, 2011, pp. 3-15.

[5] Jörg Bindner and Claudia Hirschmann: “Agil in Medical Technology? Hand in Hand with Quality Management” (Transl.), published in OBJECTspectrum, Agility/2009, pp. 1-5.

[6] Results from Scott Ambler’s March 2006 ‘Agile Adoption Rate Survey’ posted at www.ambysoft.com/surveys/. [retrieved: September, 2012].

[7] Computerwoche: „Agile Methods in Comparison“ (Transl.), http://www.computerwoche.de/software/software-infrastruktur/2352712/ , April 2012. [retrieved: September, 2012].

[8] Roman Pichler: „Scrum – Using Agile Project Management Successfully” (Transl.), 2008, pp. 7-123.

[9] ScrumMed, Conference for Scrum in Medical Technology.

[10] Andrea Heck: „How is a large software project in medical technology getting agile“ (Transl.), Colloquium at Georg Simon Ohm University of Applied Sciences.

[11] Jeff Sutherland: Sprint Ready and Done threshold, 2009.

[12] IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related systems, International Electrotechnical Commission, 2010, pp. 19-20.

[13] EN 50128, European Standard of Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems. 2011, pp. 10-26.

[14] V-Modell XT Version 1.3 English, Federal Republic of Germany, 2006, pp. 10- 20.

[15] Adrien Mouaffo, Zhensheng Guo, Mahmudul Huq, Dieter Rombach, and Peter Liggesmeyer, Tool support for a safety- and security- based assessment model for software engineering processes, in Software Process Improvement And Capability dEtermination (SPICE) Conference, 2010.

ESAC-BPM: Early Security Access Control in Business Process Management

Mahmoud F. Ayoub
Computer and Systems Engineering
Alexandria University
Alexandria, Egypt
Email: mfayoub@alexu.edu.eg

Riham Hassan
Computer Science
Virginia Tech
Blacksburg, VA, USA
Email: rhabdel@cs.vt.edu

Hicham G. Elmongui
Computer and Systems Engineering
Alexandria University
Alexandria, Egypt
Email: elmongui@alexu.edu.eg

***Abstract*—Business process modeling notations do not provide explicit means to model security aspects such as access control, integrity and confidentiality. Business analysts who are not typically security experts are incapable of modeling security aspects that could not be modeled in business process modeling notations. In this paper, we propose systematic means to model access control explicitly in business process models. More specifically, we used Business Process Modeling Notation (BPMN) as a graphical notation to represent processes. Our proposed technique exploits BPMN by employing business rule activities to carry the access control logic as If-Then rules with conflict detection capabilities. We prove the validity of ESAC-BPM formally. Further, we demonstrate the technique using a case study for a reservation process for a movie store by telephone, that needs data access control policies to be applied on the process model.**

***Index Terms*—Business process management, security data access control, business rule activities.**

I. Introduction

Business Process Management (BPM) is the process of optimizing business processes and aligning all the organizational aspects with the requirements needed to be in a software. BPM concerns about the validity, performance and agility of business processes. A business process is a network of activities done by collaborators to achieve a business goal.

Business Process Modeling Notation (BPMN) [1] is used to represent business processes. With this unified graphical representation, all business stakeholders can easily understand business processes and can adjust any business modifications quickly and in a standard way. It is very similar to activity diagrams of Unified Modeling Language (UML). Therefore, it is intuitive to business users and technical engineers, and hence it can enclose the gap between them.

BPMN consists of graphical constructs and objects that can be mapped to execution languages like Business Process Execution Language (BPEL) as in [2], [3], and [4]. As a result, the pace of processes development has increased and process management became easier.

While modeling, a business analyst finds problems in expressing all the business requirements. Many of such requirements are about security. Engineering a secure software is a challenging problem. In industry, almost security is the last aspect to be considered and it is added to the software in an adhoc manner. The analysts are the best ones in the software cycle to know about the security holes and how to handle them. However, they do not have a direct control on security policies. Instead, they have to forward the required changes to software engineers.

One important aspect of software security is the data access control, which is a necessary and crucial design element for any secure application. In general, an application should protect its data and system resources against unauthorized access by implementing access control restrictions on what users can do. Access Control refers to the much more general way of controlling access to data, including restrictions based on things like the time of day, age, gender, the IP address of the HTTP client browser, or any other derived variables that can be extracted or calculated easily. Simpler access control models often cannot adequately meet the complex access control requirements that such relationships require, and so more granular, powerful, dynamic models and mechanisms are needed to address these new realities. In short, increasingly complex data access and sharing drive the need for increasingly complex access control models and mechanisms.

Attribute Based Access Control (ABAC) is one model of access control. It uses attributes in a structured language to define access control policies. There are 3 kinds of attributes: subject, object and environment. 1) The subject represents who requests data access. In typical applications, It can be a user, application, or process. Subject related attributes are like age, name, gender or role. 2) The object is the target identity to be secured. It can be a file, database, data object or web resource. Object related attributes are like name, size or URL. 3) The environment is the context where the access request happens. Environmental attributes are like time of day, weather, season or place of request.

Process modeling languages and security policies languages are both used to document organizational policies and procedures. While process modeling languages describe a procedural sequence of activities, security policy languages often rely on a declarative description of security constraints. Understanding the relationship between the two languages would maximize benefit, avoid content duplication, and reduce their overall effort [5]. In this paper, we present a provably systematic and easily deployable approach for embedding security access control, as one example of security aspects, that is designed for BPMN diagrams. Our approach is based on a novel usage of business-rule activities which are in the BPMN specification. So, we do not need to modify the BPMN meta-model. It is based on putting all the security logic as If-Then rules. Additionally, we integrate our approach with an option to detect conflicts [6] between access control policies. Access control policy conflicts usually happen as a result of their complexity. We adopt ABAC as it is the most flexible access control model, and it can be a replacement to any access control model as shown later. In ABAC, access control policies are given as boolean expressions, which are easily written and understood. They are comprised of conditional attributes and boolean operators ($\neg$, $\wedge$ and $\vee$). They are mapped to business rules and are wrapped by a business rule activity. Upon activation, a business rule activity activates the associated business rules and on their completion, the activity terminates. A business rule activity hides all the security logic in its underlying properties. As a result, all complex logic is not visible in the graphical view of a process model, and so it is still easily understood. In such a way, we provide a complete software solution by considering early security while modeling business processes. Our evaluation shows that the proposed approach can achieve significant secrecy gain with minimal additions compared to the state of the art.

The rest of the paper is organized as follows. In Section II, we discuss the related work. In Section III, we present our approach. In Section IV, we prove the validity of our approach. In Section V, we provide a complete test case example examining our approach. And, finally, the conclusion and future work are in Section VI.

II. Related Work

Concerning different related work of embedding security in business processes, there are many ways to do so with different approaches. A first popular approach is to add text annotations to different BPMN constructs. These annotations contain a formal specification of security policies and need to be parsed afterwards at runtime to enforce security requirements. There were many security policy specifications languages used and are presented here. Second approach changes the meta-model of BPMN to add new constructs for security requirements. These constructs are translated into security policies to be enforced at runtime. Our approach does not change the meta-model, but it creates new fragments based on business rule activities and some gateways. So it is easily deployable.

Mülle et al. [14] and Darnianou et al. [15], propose languages for security policies specification. A business analyst should study it for usage. In [18], security elements and process models are integrated. The language constructs can be inserted as a text annotation associated with BPMN constructs. Annotations are to be compiled and enforced at the process execution stage. The language supports access control by providing authorization, delegation, information filtering and refrain policies. Additionally, they provide obligations, basic constraints, meta policies and policy composition. After integrating the process model with security elements, the graphical representation of the process is complex and hard to be understood. It is better to externalize all the security logic.

In [16], a Policy Description Language (PDL) is proposed. It is declarative and consists of 3 categories: 1) Set of events. 2) Set of actions. 3) Set of functions to evaluate the environment. Policy rule propositions take the following form.

event **causes** *action—event(s)* **if** *condition*

This means that if *event* occurs in a situation where *condition* is true, then the action or consequential events specified will be executed. The language is a generic language that can be used to describe any type of policies and not only security policies. This type of formalization will allow representing obligations, and prohibitions.

Rodríguez et al. [10] and Menzel et al. [11], concern about the graphical representation of security constructs. Additionally, they add to the meta-model of BPMN. Security requirements are included in the extended meta-model. They do not cover how the underlying layer works for enforcing such security constraints. Additionally, they map each security requirement to a BPMN construct and add a mark to it. Supported security aspects are non-repudiation, attack harm detection, integrity, privacy, access control, security role, and security permissions. In contrast to our proposition, there are no clues about how security constraints are to be enforced. They model the requirements and leave it up to the developer to decide how to implement them, which may introduce security holes.

Wolter et al. [12] and [13], constitute their generic security model that specifies security goals, policies, and constraints based on a set of basic entities, such as objects, attributes, interactions, and effects. It is a general description regardless the notation used. For BPMN, they visualize the security requirements as artifacts like a text annotation to a message link for message confidentiality. They use the group artifact on some activities to express the separation of duties (SoD) security aspect.

In [17], a similar methodology, to our approach, for modeling security requirements in business processes is proposed. They cover both the design-time modeling and run-time enforcement of security requirements for business processes. Such requirements are translated to XACML policies which are enforced by generated Policy Enforcement Points (PEPs). Additionally, Policy Decision Points (PDPs) are generated to decide if a certain request is granted or not. A prototype is implemented based on Activiti (http://www.activiti.org/), extending the Eclipse designer and process engine. Some security requirements are represented as new constructs in BPMN and others like access control are represented by domain-specific user interface.

In [19], security needs are supported by the commitments view, which consists of a set of commitments between actors. The conversation and choreography diagrams of BPMN 2.0 are targeted. An overview of intercompany processes between several partners is given. Hence, annotations are used to show which conversations and related participants the requirements apply.

III. Formal Method of Security Access Control in BPMN

While accessing data objects, we authorize the incoming read/write request based on the access control policies of the requested object. For accessing data base object, we do the same for the CRUD operations. Our proposed approach adopt ABAC because of its flexibility and context awareness. Access control policies are represented by boolean expressions comprised of conditional attributes and boolean operators ($\neg$, $\wedge$ and $\vee$). Boolean expressions are transformed to business rules. Our approach does not modify the meta-model of BPMN's, because there is a native support for business rules via business rule activities. When a model token activates such activity, the associated business rules are called. On completion, the business rule activity completes and the result is the actions specified in the body section of rules. In our case, for handling access control policies, the action is either permit or deny the incoming request.

Our approach uses system-wide rules by business analysts to model security aspects (mandatory access control policies). As illustrated in Figure 1, when the regional manager receives a message of a loan application, it is reviewed using the loan data file, and then the manager notifies the customer with the result of application either by acceptance or refusal. In this example, a business analyst is incapable of adding constraints on accessing the data file like restricting the access time in a certain duration of a day ($6am \leq time \leq 6pm$) or other constraints.

Our proposed approach solves this problem by adding a construct to be activated before accessing data objects. As depicted in Figure 2, the secure read sub process is inserted before accessing loan data file. This process can throw an error event if it denies the incoming access request, and the business analyst should handle the thrown error in a proper way. In Figure 3, we give a further insight into our construct. It contains a business rule activity that has all the logic of the access control policy. The activity is followed by a gateway to differentiate between a granted access and a denied one. If the access is accepted, the subprocess finishes normally. If it is denied, the subprocess throws an intermediate error event to the calling process.

It is better to externalize the decision logic in an external decision table like business rules rather than organizing them among gates that control the model's token path. In the former approach, business analyst can express complex policies, easily review and validate the business rules controlling a certain data object. On the other hand, the latter approach is subject to policy changes and requires significant maintenance.

IV. Validity Proof of Formal Method

To prove the validity of our approach, we split the proof into 3 steps as follows. 1) Traditional access control models can be converted to ABAC. 2) We can transfer any ABAC policy

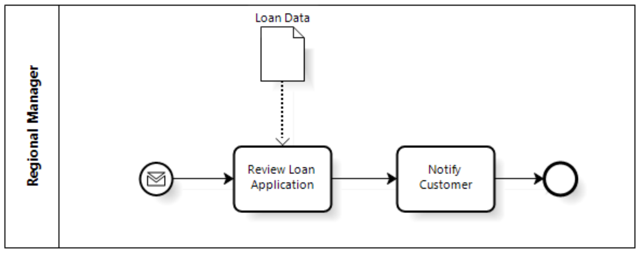


Fig. 1. Example before secure read activity

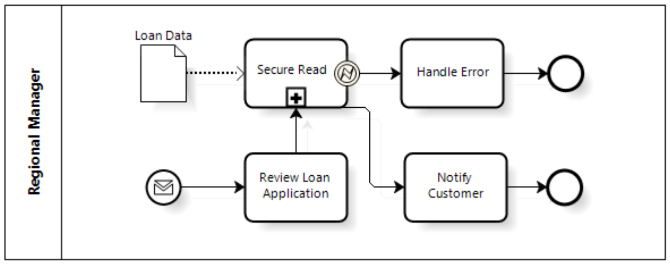


Fig. 2. Example after secure read activity

to some business rules. 3) Conflict detection. We present each of these steps in detail in the following subsections.

A. Traditional Access Control Models to ABAC

ABAC is flexible enough among all other traditional models, as presented in [7] and [8]. So we can express various access control policies. Because they all depend on attributes of 3 domains: subject, object and environment. 1) Subject represents the identity who requests for a data access. Typical attributes are like age, name, gender or role. 2) Object is the target identity to be secured. It can be a file, database, data object or web resource. 3) Environment is the context where the access request happens. So for using ABAC, we have to convert the given domain to its attributes and then create the ABAC policies. We do a proof by complete induction on all access control models from [9] and convert each one to the corresponding ABAC model as follows.

1) Role Based Access Control (RBAC) to ABAC: RBAC only concerns about roles in the system. So one of the inherent

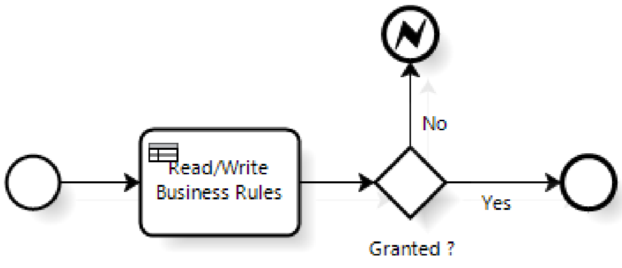


Fig. 3. Collapsed secure read subprocess

TABLE I
MODELING RBAC RULES

Conditions	Result
Role	Access Granted
Manager	Yes
Clerk	No
TA	Yes

TABLE II
MODELING BLP READ RULES. THE SIMPLE SECURITY PROPERTY (NO READ-UP). EXPRESSION: $SubjectLabel \in \{Owner, Manager\}$ AND $Operation = "Read"$

Conditions	Conditions	Result
Subject Label	Operation	Access Granted
Owner	Read	Yes
Manager	Read	Yes
Clerk	Read	No
Customer	Read	No

TABLE III
MODELING BLP WRITE RULES. THE STAR PROPERTY (NO WRITE-DOWN)

Conditions	Conditions	Result
Subject Label	Operation	Access Granted
Owner	Write	No
Manager	Write	Yes
Clerk	Write	Yes
Customer	Write	Yes

limitations of RBAC is the single dimension of roles. As a result, if we want to model multiple attributes, the number of roles needed to encode these attributes will grow exponentially. Additionally, RBAC does not support environment attributes and can not model BellLaPadula model. Table I shows how to model an RBAC to an ABAC one.

2) BellLaPadula (BLP) Model to ABAC: BLP is used in military organizations where there are object security labels and subject clearances. They build the finite state machine of the model. Assume that we have a model with the following specifications. Object labels are ordered from top secured to lower level as follows.

1) TOP SECRET
2) SECRET
3) CONFIDENTIAL
4) PUBLIC

The subject clearances are ordered in the same way as follows.

1) Owner
2) Manager
3) Clerk
4) Customer

Tables II and III show how to model this BLP model to an ABAC one assuming we have a SECRET object to be secured.

This policy can be expressed by two attributes: $SubjectLabel$ and $Operation$ as follows

$$(SubjectLabel \in \{Owner, Manager\} \wedge Operation = "Read") \vee (SubjectLabel \in \{Manager, Clerk, Customer\} \wedge Operation = "Write") \quad (1)$$

TABLE IV
BUSINESS RULES FOR BPL MODEL

Condition	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**
Subject Label	Owner		Manager		Clerk, Customer	
Operation	Read	Write	Read	Write	Read	Write
Access Granted	Yes	No	No	Yes	No	Yes

B. ABAC Policy to Business Rules

In this subsection, we have an input ABAC policy expressed as a boolean expression. We want to transform it to some business rules that satisfy the expression. We deal with business rules because BPMN has a complete support to them. This is our main contribution that we make use of business rules in embedding security in business processes. So there is no need to modify the meta-model of BPMN.

The input boolean expression will consist of attribute variables each of which belongs to a certain range of the whole attribute domain and logical operators ($\neg$, $\wedge$ and $\vee$). A family of disjoint sets of each attributes can be detected from the given boolean expression. Union of these sets is the whole attribute domain and there is no intersection between them. These sets make up the business rules to be put in business processes. The cardinality of the number of business rules is given as follows:

$$NumberOfBusinessRules = \prod_{i=1}^{N} |A_i| \quad (2)$$

where N is the number of attributes in policy, A_i represents the i^{th} attribute and $||$ returns the number of disjoint sets of the given attribute. For example, in Equation 1, there are three sets for the $SubjectLabel$ attribute: {Clerk, Customer}, {Owner} and {Manager}. For $Operation$ attribute, there are two sets: {Read} and {Write}. Hence we have six business rules and we can put them in a decision table as in Table IV. Decision trees can be used for visualizing business rules. But they are very brittle when rules change and require significant maintenance. Additionally, they are more complex when each parameter potentially has a large number of different values where each possible parameter value becomes a node at a branching point in the tree. On the other hand, decision tables, as in Table IV, can be used instead which are more compact and intuitive when many rules are needed to analyze many combinations of attribute values.

C. Conflict Detection

Several policies control access to each object. Each policy consists of some business rules that adheres what the policy specifies. Each rule has an action (whether permit or deny) to access the object. Due to enterprise business processes and complex access control policies, some conflicts may arise. A conflict arises when two policies having some conditional attributes in common but different in their actions.

We adopt the approach in XACML access control policies [6], as in Algorithm 1, by representing each policy in d-dimensional space, where d is the number of conditional attributes. The plane sweep algorithm, as in [20], is used for detecting pairwise conflicts. It has three steps: 1) project each policy on a specified dimension plane and sort the projected values. 2) sweep the plane. 3) report intersections.

Algorithm 1 Conflict Detection Algorithm

```
1: map all policies to the d-dimensional space
2: determine start and end of the range every policy covers
3: for each dimension do
4:     determine all intersections via plane sweep algorithm
5:     prune all policies that cannot conflict another one
6: end for
7: report all pairwise conflicts
```

The complexity of the plane sweep algorithm is $O(n * log(n))$. And due to the fact that it is used d times in the conflict detection algorithm. So the overall complexity of the conflict detection algorithm is as in Equation 3

$$= d * n * log(n) \tag{3}$$

where n is the number of policies and d is the number of dimension.

We have developed many techniques to resolve the conflicts among the rules. However, we omit their details due to the space limitations. These techniques depend on how the business rule engine works. Some techniques can call the business rules in serial manner and finish when it takes a decision from the calling business rules. Other techniques can call all the business rules in parallel, and then aggregates the results by voting algorithm, permit dominates, deny dominates or others. A business analyst can do the proper modifications to remove any conflict.

V. Case Study

Our illustrative example describes a typical business process for a movie rental shop. Reservations are done via telephone calls by customers to store clerks. There are two requirements to be considered for business security and management. 1) Basic requirement in which access control is based on customers' age and the movies content ratings. Ratings are Restricted (R), Parented Guidance Strongly Cautioned (PG-13) and General Audience (G). 2) Advanced requirement which introduces membership classes (Premium, Regular), which enforces a new policy that only Premium users can view new releases. Figure 4 shows the details of what a movie store clerk does to securely create a right order for the incoming call request according to the store policies. The diagram also handles any possible attack for the data objects.

Figure 4 describes a business process of a movie store that is triggered by a phone call. The clerk responds, and then review the customer data. Before accessing *Customer Data* file, we can apply access control policies on the subject role, username, time of request, or type of operation. As shown in Table V, it restricts the accessing of file to either clerks and managers. Additionally, it puts additional time constraint for each role. On activation the *Secure Read* business rule activity, the associated business rules in Table V are activated and produce the result, depending on the business rule engine, either with permit or deny the incoming access control request. If the business rule denies the request, the process terminates to prevent an unauthorized access.

TABLE V
Decision table for accessing the customer data object file

Conditions		Result
Role	**Time**	**Granted**
Clerk	$8am \geq time \leq 4pm$	Yes
Manager	Any	Yes

In case of granting the access control request, the process execution continues and checks whether the customer already exists in the system or not. If not, the process registers the customer into the system. In this case, we may not need any access control policies because they are already granted at first. If the customer already exists, the process proceeds with writing the incoming order. The process should enforce the management requirements. So, the *Secure Order* business rule activity is inserted before accessing the movies data file. The ABAC policies for the basic and advanced requirements are represented as boolean expressions in Equations 4 and 5, respectively.

$$(Age \geq 21 \land Rating \in \{R, PG13, P\}) \lor (21{>}Age \geq 13 \land Rating \in \{PG13, P\}) \lor (Age{<}13 \land Rating \in \{P\}) \tag{4}$$

$$(MemberType = "Premium") \lor (MemberType = "Regular" \land MovieType \neg = "NewReleased") \tag{5}$$

The *Secure Order* business rule activity either permits or denies the incoming order request. If it denies the request, it throws an intermediate error event and *Deny Request* task is activated. If it permits the request, it continues normally and activate the *Accept Request* task. In both directions, the process terminates.

Finally, business analysts can model the security and management policies using our approach. They are easy deployable into a process model definition. It is easy to add new policies or attribute values without adding any constructs to the BPMN diagram. This is because we put all the security and management logic in business rule activities. In this way, we still keep the diagram easily understood without embedding complex constructs, artifacts and fragments.

VI. Conclusion and Future Work

We presented a novel approach for embedding security in business processes in a systematic manner. Therefore, we can provide a complete software solution without the need of post-adhoc security to be considered. We consider data access control as an aspect of security. It is crucial in most business

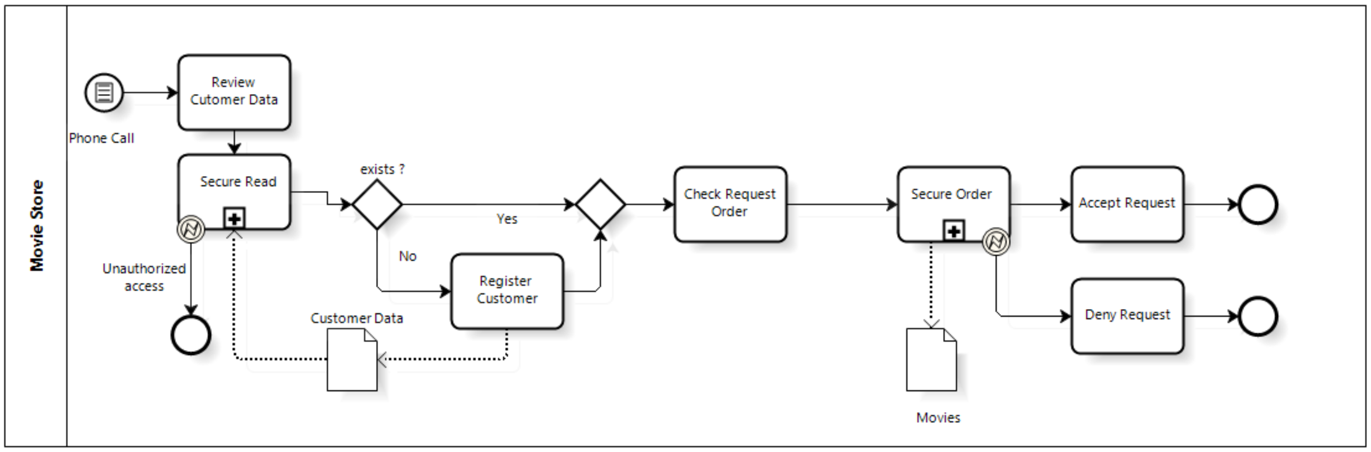


Fig. 4. Movie store example

TABLE VI
BASIC REQUIREMENT DECISION TABLE

Conditions		Result
Age	**Movie Rating**	**Granted**
≥ 21	{R, PG13, P}	Yes
$21 > age \geq 13$	{PG13, P}	Yes
<13	{P}	Yes

TABLE VII
ADVANCED REQUIREMENT DECISION TABLE

Conditions		Result
Member Type	**Movie Type**	**Granted**
Premium	Any	Yes
Regular	Not new released	Yes

processes where business analysts do not know either about access control models or how to add them to their processes while modeling. We used Business Process Modeling Notation (BPMN) as a graphical notation to represent processes. In this approach, we make use of business rule activities of the notation via putting all the security logic to be put as If-Then rules with conflict detection. This comes with minimal overhead for business analysts. Business processes diagrams are still readable and easily understood. We prove the validity of the approach.

Our ongoing work is to consider other aspects of security like confidentiality, integrity, and availability. We plan to transform the new inserted fragments to BPEL in a systematic way in order to provide a complete software solution with early security consideration from the beginning.

REFERENCES

[1] BPMN, Business Process Model and Notation (BPMN). *In http://www.omg.org/spec/BPMN/2.0/PDF/*, 01.7.2012.

[2] C. Ouyang, M. Dumas, A. H. M. ter Hofstede, and W. M. P. van der Aalst, "From BPMN process models to BPEL web services," In *Proceedings of the 4th International Conference on Web Services (ICWS06)*, pp. 285-292, Chicago, Illinois, USA, September 2006. IEEE Computer Society.

[3] J. Recker and J. Mendling, "On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages," In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering*, pp. 521-532, 2006.

[4] J. Blox, "BPMN 2 BPEL," *Eindhoven: Eindhoven University of Technology*, 2009.

[5] M. zur Muehlen and M. Indulska, "Modeling languages for business processes and business rules: A representational analysis," *Information Systems*, pp. 379-390, Elsevier, 2010.

[6] F. Huonder, "Conflict Detection and Resolution of XACML Policies," *Master Thesis, University of Applied Sciences Rapperswil*, 2010.

[7] E. Yuan and J. Tong, "Attributed Based Access Control (ABAC) for Web Services," In *ICWS05: IEEE International Conference on Web Services, Orlando*, pp. 561-569, 2005.

[8] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "A Flexible Attribute-Based Access Control Method for Grid Computing," *Journal of Grid Computing*, vol. 7, no. 2, pp. 169-180, 2009.

[9] P. Samarati and S. D. C. di Vimercati, "Access Control: Policies, Models, and Mechanisms," In *FOSAD II*, pp. 137-196, Springer-Verlag, 2001.

[10] A. Rodríguez, E. Fernández-Medina, and M. Piattini, "A BPMN Extension for the Modeling of Security Requirements in Business Processes," *IEICE Transactions on Information and Systems*, pp. 745-752, 2007.

[11] M. Menzel, I. Thomas, and C. Meinel, "Security requirements specification in service-oriented business process management," In *ARES*, pp. 41-48, 2009.

[12] C. Wolter, M. Menzel, and C. Meinel, "Modelling security goals in business processes," *Proc. GI Modellierung*, pp. 197-212, 2008.

[13] C. Wolter and A. Schaad, "Modeling of task-based authorization constraints in bpmn," In *BPM*, pp. 64-79, 2007.

[14] J. Mülle, S. von Stackelberg, and K. Bohm, "A Security Language for BPMN Process Models," *Technical Report, Karlsruhe Institute of Technology (KIT)*, 2011.

[15] N. Darnianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder Policy Specification Language," *International Workshop, Policies for Distributed Systems and Networks*, pp.29-31, 2001.

[16] J. Lobo, R. Bhatia, and S. Naqvi, "A Policy Description Language," *Proc. of National Conference of the American Association for Artificial Intelligence*, pp. 291-298, July 1999.

[17] A. D. Brucker, I. Hang, G. Lückemeyer, and R. Ruparel, "SecureBPMN: Modeling and Enforcing Access Control Requirements in Business Processes," In *ACM Symposium on Access Control Models and Technologies*, pp. 123-126, 2012

[18] J. Mülle, S. von Stackelberg, and K. Bohm, "Modelling and Transforming Security Constraints in Privacy-Aware Business Processes," In *SOCA*. pp. 1-4, 2011.

[19] E. Paja, P. Giorgini, S. Paul, and P. H. Meland "Security Requirements Engineering for Secure Business Processes," In *10th International Conference BIR 2011*, LNBIP, pp. 77-89, 2012.

[20] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. "Computational Geometry, Algorithms and Applications," *Springer*, 1998.

Intrusion Detection Using Ensembles

Alexandre Balon-Perin[*†]
abalonpe@ulb.ac.be

Björn Gambäck[*‡]
gamback@idi.ntnu.no

Lillian Røstad[*]
lillianro@idi.ntnu.no

[†]*Ecole Polytechnique*
Université libre de Bruxelles
Brussels, Belgium

[*]*Department of Computer and Information Science*
Norwegian University of Science and Technology
Trondheim, Norway

[‡]*SICS — Swedish Institute*
of Computer Science AB
Kista, Sweden

***Abstract*—The paper discusses intrusion detection systems built using ensemble approaches, i.e., by combining several machine learning algorithms. The main idea is to exploit the strengths of each algorithm of the ensemble to obtain a robust classifier. Network attacks can be divided into four classes: probe, remote to local, denial of service, and user to root. Each module of the ensemble designed in this work is itself an ensemble using bagging of decision trees and is specialized in the detection of one class of attacks. Experiments highlighted the efficiency of the approach and showed that increased accuracy can be obtained when each class of attacks is treated as a separate problem and handled by specialized algorithms. In all experiments, the ensemble was able to decrease the number of false positives and false negatives.**

***Keywords*-intrusion detection; ensemble approaches; bagging**

I. Introduction

Intrusion detection systems (IDSs) are monitoring devices that have been added to the wall of security in order to prevent malicious activity on a system. Here we will focus on network intrusion detection systems mainly because they can detect the widest range of attacks compared to other types of IDSs. Network IDSs analyse traffic to detect ongoing and incoming attacks on a network. Additionally, they must provide concise but sound reports of attacks in order to facilitate the prevention of future intrusions and to inform the network administrators that the system has been compromised. Current commercial IDSs mainly use a database of rules (*signatures*), to try to detect attacks on a network or on a host computer. This detection method is presently the most accurate, but also the easiest to evade for experienced malicious users, because variants of known attacks (with slightly different signatures) are considered harmless by the IDS and can pass through without warning. New attacks and attacks exploiting zero-day vulnerabilities can also slip through the security net if their signatures are unknown to the IDS. (A *zero-day vulnerability* is a software weakness unknown by the system developers which potentially could allow an attacker to compromise the system. 'Zero-day' refers to the first day, day zero, that the vulnerability is observed.)

Hence there is a need for mechanisms allowing the IDS to learn by itself how to detect previously unseen attacks or variants of known attacks. However, the problem is further complicated by the extreme requirements of robustness of the IDS. It must be able to detect all previously seen and unseen attacks without failure, it must never let an attack pass through unnoticed, and it must never deliver unwanted warnings when the traffic is in fact legitimate. For a summary of the main challenges that machine learning has to overcome to be useful for intrusion detection, see [1].

Despite this, several attempts have been made to build automatically adaptable intrusion detection systems using various machine learning algorithms. So far though, the machine learning classifiers trigger too many false alarms to be useful in practice. Part of the problem is the lack of labelled datasets to train the classifiers on. The only labelled dataset available is the KDD99 dataset (www.sigkdd.org/kddcup) which is an adaptation of the DARPA98 dataset created in 1998 by the Defense Advanced Research Projects Agency (DARPA). To address these problems, new machine learning paradigms have been introduced in the field of intrusion detection, and in general the machine learning community has in recent years paid more attention to ensemble approaches, i.e., combinations of several machine learning algorithms.

Network attacks can be divided into four classes: probe, remote to local, denial of service, and user to root. Most previous machine learning-based solutions include a single algorithm in charge of detecting all classes of attacks. Instead, in this work, one module of an ensemble is specialised on the detection of attacks belonging to one particular class. The main idea is to exploit the strengths of each algorithm of the ensemble to obtain a robust classifier. Ensembles are particularly efficient in cases like this, when a problem can be segmented into parts, so that each module of the ensemble is assigned to one particular subproblem. The modules in turn include one or more algorithms cooperating together.

Furthermore, each class of attacks is characterized by very specific properties, observable through the values of certain features on instances in the dataset belonging to a specific class of attacks. However, even though feature selection is often applied in IDSs using machine learning techniques, often only one set of features is selected for *all* classes of attacks. In this work, one set of features is selected for *each* class of attacks according to their relevance to the corresponding class. The corresponding algorithm(s) is then fed with the appropriate set of features. The system can, in theory, reach a very high accuracy with a small cost, and the ensemble processing can potentially be parallelized.

The rest of the paper is laid out as follows: First Section II gives an overview of the state-of-the-art by introducing the resources and methods used in the experiments and describing related work, in particular focusing on previous efforts in applying ensemble-based methods to intrusion detection. The core of the paper is Section III which details two rounds of experiments carried out, on feature selection for ensembles resp. on feeding an ensemble of machine learning algorithms with the most successful sets of features identified. Section IV then discusses the results of the experiments at length and points to ways in which the present work could be extended. Finally, Section V sums up the previous discussion.

II. Ensemble-based Intrusion Detection

The ensemble method is a way to build different types of approaches to solving the same problem: the outputs of several algorithms used as predictors for a particular problem are combined to improve the accuracy of the overall system. The difficulty of ensemble approaches lays in the choice of the algorithms constituting the ensemble and the decision function which combines the results of the different algorithms. Often, the more algorithms the better, but it is important to take into account the computational expense added by each new algorithm. The decision function is often a majority vote which is both simple and efficient, but alternatives should be analysed to obtain an optimal combination. Another advantage of ensemble approaches is their modular structure, unlike hybrid constructions which are engineered with algorithms having non-interchangeable positions. Consequently, the ensemble designer can easily replace one or more algorithms with a more accurate one.

Bagging and boosting are the two main techniques used to combine the algorithms in an ensemble. In an ensemble using the boosting technique, the algorithms are used sequentially. The advantage of this technique is that the most difficult examples can be classified correctly without adding too much computational overload. In an ensemble using the bagging technique all algorithms of the ensemble are used in parallel. In this case, each algorithm builds a different model of the data and the outputs of all predictors are combined to obtain the final output of the ensemble. In order to build different models, either each algorithm of the ensemble, or the data fed to each algorithm, or both, can be different. Since all algorithms perform in parallel, each of them can be executed on a different processor to speed up the computation.

A. The KDD99 Dataset

As observed in the introduction, part of the problem of automatically creating good intrusion detection systems is the lack of labelled datasets to train on. The only one available is the KDD cup 99 dataset, which was used for the first time in the 3rd International Knowledge Discovery and Data Mining Tools Competition in 1999. It is based on the DARPA98 dataset (built during the DARPA98 IDS evaluation program) which includes seven weeks of data from traffic passing through a network engineered for the purpose, i.e., the traffic was generated in a simulated and controlled environment.

Table I shows the distribution of instances of the KDD cup 1999 training and test sets over the different classes. All the examples are separated into the class `Normal` and four different classes of attacks: `Probe`, `R2L` (remote to local), `DoS` (denial of service), and `U2R` (user to root). Each entry in the sets is represented by 41 features such as `duration`, `src_bytes`, `dst_bytes`, etc., and a label. The training set contains 4,898,431 entries and is highly unbalanced. Whereas the `DoS` class contains 3,883,370 instances, the classes `U2R` and `R2L` are represented by only 52 and 1,126 instances, respectively. With such a small number of examples to train on, it can be expected that it will be difficult for the classifiers to predict the correct classes of unseen examples.

Table I
Types of attacks in the KDD cup 99 data sets

Set	`Normal`	`Probe`	`R2L`	`DoS`	`U2R`
Training	972,781	41,102	1,126	3,883,370	52
Test	60,593	4,166	16,347	229,853	70

The test set is composed of 311,029 entries with a distribution of the examples over the different classes similar to that in the training set. However, the number of examples belonging to the class `R2L` is more than ten times higher, so that in order to perform well on the test set, the predictor must acquire a very high power of generalisation. Most importantly, the number of unseen attacks added in the test set is huge: for the classes `U2R`, `R2L` and `Probe`, it is respectively 44.29%, 63.34% and 42.94%. Furthermore, the attacks "spy" and "warezclient" belonging to the class `R2L` are not represented in the test set. In particular, "warezclient" attacks count for more than 90% of the `R2L` training set. Finally, two entries in the test set erroneously have a `service` value of `ICMP`, as also previously reported [2]. Those were removed from the test set before the experiments reported in Section III.

The major criticisms of the KDD99 dataset include the unbalanced distribution of the data, the redundant records which can introduce a bias in the learning phase because of their frequency, that the dataset includes old attacks which have been mostly mitigated, and that the data were captured from a controlled environment somewhat different from what is observed in the wild. The first two issues can be addressed by sampling appropriate sets of examples in each class. However, the distribution of `R2L` attacks in the training set and the test set is a problem which is difficult to overcome. Nevertheless, the KDD99 dataset is far from useless. Firstly, if an IDS using machine learning does not perform well on old attack provided that the data are well sampled, why would it on newer ones? Furthermore, most of the research in the field of machine learning applied to intrusion detection uses the KDD99 dataset, making it a vector of comparison between different approaches. The controlled nature of the environment in which the data were captured is probably the most problematic. For example, the high number of attacks in comparison to normal traffic does not reflect the reality of a network in which almost all traffic is normal.

B. Related Work

Intrusion detection systems have been around since the 80s. In the late 90s researchers in artificial intelligence started to incorporate their algorithms to improve IDSs. An intrusion detection system should be able to autonomously recognize malicious actions in order to defend itself against variants of previously seen attacks and against attacks exploiting zero-day vulnerabilities. Misuse-based IDSs can only detect attacks whose signatures are available in their signature database. Signatures of attacks are very specific, and a slight variation of the attack can make it unnoticeable for the IDS. That is why learning mechanisms must be implemented to detect and prevent these attacks without having to wait for an update of the signature database or a patch for the vulnerable system. Still, machine learning algorithms are designed to recognize examples similar to those available in the training set used to build the model of the data. Consequently, an IDS using machine learning would have a hard time detecting attacks which patterns are totally different from the data previously seen. In other words, even though machine learning is a suitable candidate to detect variants of known attacks, detecting completely new types of attacks might be out of reach for these kinds of algorithms.

For a summary of most research involving machine learning applied to IDSs until 2007, see [3] which covers a range of techniques, including fuzzy sets, soft computing, and bio-inspired methods such as artificial neural networks, evolutionary computing, artificial immune systems, and swarm intelligence; comparing the performance of the algorithms on the KDD99 test set and showing that all algorithms perform poorly on the `U2R` and `R2L` classes. The best results reported are by genetic programming with transformation functions for `R2L` and `Probe` and by linear genetic programming (LGP) for `DoS` and `U2R` (with 80.22%, 97.29%, 99.7% and 76.3% accuracy, respectively). However, since ensemble-based methods is a fairly new technique applied to intrusion detection, their description in the review is somewhat limited. The works on the topic date from 2003 and many papers were written in 2004 and 2005. Recently, there has been a renewed interest of ensembles in this field [4]–[6].

Abrahams *et al.* have performed three types of ensemble-based experiments, all on a subset of the DARPA98 dataset composed of 11,982 randomly selected examples: First, in [7], an ensemble composed of different types of artificial neural networks (ANN), support vector machines (SVM) with radial basis function kernel, and multivariate adaptive regression splines (MARS) combined using bagging techniques was compared to the results obtained by each algorithm executed separately. SVM used alone outperformed the other single algorithms, but was totally outperformed by the ensemble.

Second, in [8], the combination of classification and regression trees (CART) and bayesian networks (BN) in an ensemble using bagging techniques was explored. Feature selection was applied to speed up the processing: the performance on the set of 41 features was compared to a set of 12 selected by BN, 17 selected by CART and 19 features selected by another study. The final ensemble was composed of three CART to detect `Normal`, `Probe` and `U2R` examples, respectively; one ensemble of one CART and one BN to detect `R2L` examples; and one ensemble of one CART and one BN to detect `DoS` examples — with each classifier trained on its resp. reduced set of features; an approach quite similar to the one used in the present paper. This was then extended by adding a hybrid model composed of SVM and decision trees (DT) to the ensemble [9]. However, the hybrid model did not seem to help much.

Third, in [10], fuzzy rule-based classifiers, linear genetic programming (LGP), DT, SVM, and an ensemble were evaluated using feature selection to reduce the number of variables of the dataset to 12. The fuzzy rule-based classifier outperformed the other methods when trained on all 41 features, while LGP seemed more appropriate when using a smaller feature set. The ensemble was composed of one DT in charge of the `Normal` instances, one LGP each for `Probe`, `R2L` and `DoS`, and one fuzzy set of rules for `U2R`. The results obtained with the ensemble were very encouraging with accuracy $> 99\%$ for all classes (on the subset data).

Folino *et al.* [11] instead used the entire KDD99 dataset and examined the performance of a system composed of several genetic programming ensembles distributed on the network based on the island model. The system showed average performance for the `Normal`, `Probe` and `DoS` classes, but very low for the `U2R` and `R2L` classes.

The key conclusion from all these works is that ensemble approaches generally outperform approaches in which only one algorithm is used. An ensemble is a very efficient way to compensate for the low accuracy of a set of weak learners. Moreover, feature selection should provide specific subsets to train algorithms specialised in the detection of one particular class of attacks. Mukkamala *et al.* [12]–[14] identified the five most important features for each class of attacks. The features were selected using SVM, LGP and MARS. Surprisingly, neither `protocol_type` nor `service` was selected by the three algorithms for the `DoS` class. In contrast, Kayacik *et al.* [15] concluded that those features were the most significant for that classs, even though their experiments were conducted on hierarchical self-organizing maps (SOM).

III. EXPERIMENTS

The problem of intrusion detection can be divided into five distinct subproblems, one for each class of instances (`Normal` and the four types of attacks: `Probe`, `U2R`, `R2L`, and `DoS`). Here each problem will be handled by one or more algorithms of an ensemble, allowing each subproblem to be treated separately in the experiments and to join the solutions to the subproblems into a general solution for the problem of intrusion detection. A dataset for each attack subproblem was built by sampling a number of examples in one class of attacks and the same number in the class `Normal` in order to have a balanced dataset with 50% anomalous and 50% normal examples (no algorithm was explicitly designed to detect normal traffic). A balanced dataset is necessary to avoid the problem of skewed classes where the accuracy of the predictor can be made artificially high by increasing the number of instances from one of the classes.

For the classes of attacks with few examples, `R2L` and `U2R`, the entire set was selected. For the `Probe` class, 10,000 instances were selected randomly. This number was chosen to have a significant sample with as many different examples as possible without affecting the training time too much. The

DoS training set contains 3,883,370 instances, with 'neptune' and 'smurf' attacks counting for the majority (with resp. 1,072,017 and 2,807,886 instances). The other types of attacks have much smaller number of examples, e.g., the type of DoS called 'land' is represented only 21 times. For this reason, samples of 5,000 examples each were selected randomly from the 'neptune' and 'smurf' sets. All examples of the other types of attacks were included for a total of 13,467 DoS instances. For all four classes, the same number of Normal instances was selected. The experiments performed are in direct continuity of the work done by Mukkamala *et al.* [12]–[14], which identified the key features relevant to each of the four classes of attacks. The first step of the experiments was to assess the sets of features selected in [12]. Then in a second round of experiments those sets were fed to an ensemble of machine learning algorithms. All models were evaluated by 10-fold cross-validation.

A. Experimental Setup

Figure 1 shows the model for the ensemble used in this work. First, the network packet being analysed is sent to four different detector modules, one each for Probe, R2L, U2R, and DoS. Each module executes a preprocessing step to extract a number of features from the packet; the set of features varies depending on the module (as further described in Section III-B). The extracted features are then dispatched to different decision trees which have been previously trained with the same features on the training set, as shown at the top of the figure for the Probe detector. Each decision tree is a binary classifier which outputs 0 if the packet is considered normal traffic and 1 if the packet is classified as anomalous. A vector of dimension n containing the output of n classifiers is then fed to the module decision function. In the figure $n = 4$, but it could be any number of algorithms.

Finally, a vector of dimension 4 containing the output of each module is fed to the ensemble decision function which combines the results and outputs a value describing if the packet is considered normal or anomalous, and if anomalous from which class of attacks. The easiest situations are when all modules, or all modules except one, output Normal. In the former case, the system classifies the packet as normal. In the latter, the system classifies the packet as anomalous and is able to unambiguously identify the class of attack concerned. If more than one module classify the packet as anomalous, it will be more difficult for the network administrator to understand which class of attack the anomalous packet belongs to.

The resulting model is an ensemble of ensembles with feature selection applied independently for each module. However, in this work, we will not be concerned with the decision functions for each module. Instead, we will evaluate the intersection of the sets of false positives and false negatives produced by the four algorithms in each module. This will give us the optimal performance that each module could achieve. The most important advantages of this model is the possibility to execute the algorithms in parallel and the modularity allowing the exchange of any algorithm of the ensemble without any modification of the rest.

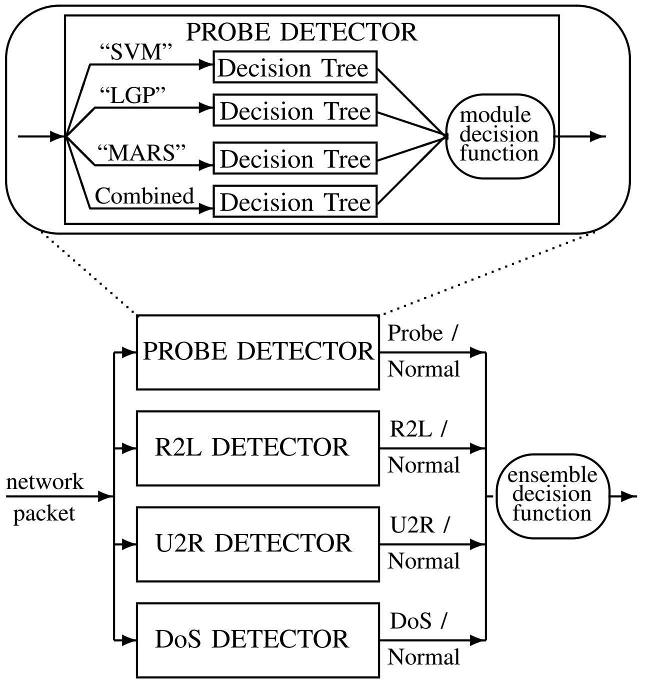


Figure 1. Model of the ensemble

B. Feature Selection Assessment

In the first experiment, several classifiers were trained with different number of features. The goal of the experiment was not to find the best algorithm possible and fine-tune it, but rather to conclude on how well an algorithm performs with a smaller set of features. In this case, it is only natural to use exactly the same setting for the algorithms and to compare the performance based only on the sets of features. Five decision trees were trained with different sets of features. The results obtained represent the performance of the algorithms on the cross-validation set which is extracted from the training set. The second experiment assessed performance on the test set.

The first classifier was trained with all 41 features in the dataset. The next three were trained with 5 features selected in [12] for each class of attacks by the three algorithms support vector machines (SVM), linear genetic programming (LGP) and multivariate adaptive regression splines (MARS). The last classifier was trained on a "combined" set of features: the union of the feature sets selected by the three algorithms. The number of features in the "combined" set is 11 for Probe, 14 for U2R, 11 for R2L and 12 for DoS. These additional sets help bringing down the number of false positives and false negatives, as we will see in the results of the experiments.

As displayed in Figure 1, the algorithm used as a classifier was decision tree (DT). Attempts were also made to use an SVM with a Gaussian radial basis function kernel. However, it could have given an advantage to features picked by SVM when used for feature selection, and it seemed that the choice of SVM greatly affected the set of features selected by MARS. Furthermore, SVM was much slower than DT, roughly two orders of magnitude both for training and classification. In particular classification time is an important criterion to take into account when building a real-world application.

Table II
ACCURACY OF THE FEATURE SELECTION ASSESSMENT

Classifier	Probe	U2R	R2L	DoS
DT: 41 features	99.865	93.000	99.022	99.948
DT: 5 SVM features	99.815	96.000	98.578	93.346
DT: 5 LGP features	99.320	90.000	97.378	98.689
DT: 5 MARS features	99.750	97.000	98.044	99.863
DT: 11–14 combined features	99.895	96.000	98.933	99.948

Table III
FEATURE SELECTION ASSESSMENT: FALSE POSITIVES AND NEGATIVES

	Probe		U2R		R2L		DoS	
Classifier	FP	FN	FP	FN	FP	FN	FP	FN
DT: 41 features	12	17	4	3	17	10	6	8
$\texttt{ensemble}_{\max}$	0.7	3	0.3	0.3	6.6	0.5	0	1.6

The results obtained in terms of accuracy are shown in Table II and can be compared to those obtained with 41 features by [9] using the same decision tree. For the class Probe, the accuracy is exactly the same as in [9]: 99.86%. The classifiers trained with sets of 5 features are not far behind the one trained with all 41. The reduced feature sets seem to be a good choice when the algorithms are trained using decision trees. However, the classifier fed with the 5 features selected by LGP performs slightly worse than the others and could be replaced by a more accurate algorithm.

The results for U2R are worse than for Probe, but this was expected: each false positive and false negative has a larger impact on the general accuracy due to the small number of examples. The results are much better than the 68.00% accuracy obtained by [9] on U2R. However, the classifier trained on features selected by LGP again perfomed poorly. Interestingly, the algorithms trained on the features selected by SVM and MARS outperformed the one trained on all features. This is probably since 41 features are too many to generalize from given the small number of examples.

The results for R2L are similar to those obtained for Probe, even though the number of instances in the dataset is much smaller. The results are also much better than [9] who obtained 84.19% accuracy on this class. This experiment clarifies that classifying Probe attacks and R2L attacks are two very distinct problems, even if they are both intrusions, which is why they should be treated separately. Again, the selected features seem to be a good choice even if a small drop of accuracy can be observed compared to Probe. The classifier trained on the features selected by MARS has a high rate of false positives and the one trained on features selected by LGP has the lowest accuracy, but also a lower false positive rate which implies a higher false negative rate.

DoS also shows better results than [9] who obtained 96.83% accuracy. The classifier trained on features selected by SVM obtained the worse score, whereas features selected by MARS gave the best score after the set of all features and the combined feature set. This is important since there is a set of 5 features that can perform almost as well as the full feature set even on larger number of training examples.

The overall numbers of false positives (FP) and false negatives (FN) drop significantly when using more than one algorithm, as Table III shows. For the FP and FN analysis, we call $\texttt{ensemble}_{\max}$ the number of examples wrongly classified by all three algorithms trained on sets of 5 features and the one trained on the "combined" feature set. This is the maximum an ensemble composed of these four algorithms could achieve if the combination of their individual results was optimal; here calculated by taking the intersection of the set of examples misclassified for each algorithm. The experiment was run ten times for each class of attacks to ensure accuracy of the results and to find the types of attack in each class misclassified most of the time by $\texttt{ensemble}_{\max}$.

All types of Probe attacks appear at least once as an FN, however, 'satan' and 'portsweep' seem to be the most difficult attacks to detect. When comparing the problematic instances of 'satan', 'portsweep' and 'ipsweep' with true instances of the same attack types, it seems that src_bytes is the feature that gives the classifiers most trouble. In fact, for probe attacks, src_bytes should be very small if not equal to zero; when an example of these attacks has a high value for src_bytes, it goes undetected. This is a big problem since an attacker could easily fill the packets of the attack with random bytes to evade the IDS. It could seem like a good idea to get rid of this feature; however, src_bytes is very important to detect Probe attacks: the only classifier that performs poorly is the one trained on the features selected by LGP, a feature set that does not include src_bytes.

For the U2R class, in general either one FP or one FN appears in each test run. The FP can be explained by the small number of examples in the dataset, only 52 Normal examples are present. The FN is always a 'rootkit' attack which is wrongly classified as normal traffic, but it is not always the same instance, indicating that some information is missing for the decision tree to classify 'rootkit' attacks. These can be any kind of malware such as worm, Trojan or virus with the ability to hide its presence and actions to the users and processes of a computer; this is called a stealth attack. The diversity found in malware probably has a huge impact on the problem. Moreover, there are only 10 'rootkit' attacks in the dataset, increasing the difficulty. Examining the values of these examples for the 14 features of the combined algorithm revealed that almost all 10 instances have very different values for those features. The $\texttt{ensemble}_{\max}$ performs perfectly in most cases, but it is difficult to conclude anything with such a small dataset: One FP or FN out of 10 instances of the cross-validation set is quite a bad score.

The combination of all algorithms helps to bring down the number of false positives and false negatives also for R2L, but these numbers are again too high for a real-world application. There are eight different types of R2L attacks represented in the training set. After running the experiments ten times, only three types of these attacks trigger false negatives for the $\texttt{ensemble}_{\max}$: 'spy', 'imap' and 'phf'. There is not

Table IV
ACCURACY OF THE MODEL ASSESSMENT

Classifier	Probe	U2R	R2L	DoS
DT: 41 features	93.087	90.000	50.000	79.345
DT: 5 SVM features	77.628	40.000	50.000	87.698
DT: 5 LGP features	87.482	83.571	61.033	76.105
DT: 5 MARS features	84.037	85.000	50.000	82.200
DT: 11–14 combined features	79.969	94.286	50.000	85.361

Table V
MODEL ASSESSMENT: FALSE POSITIVES AND FALSE NEGATIVES

	Probe		U2R		R2L		DoS	
Classifier	FP	FN	FP	FN	FP	FN	FP	FN
DT: 41 features	86	490	3	11	0	16,347	69	7,268
$\texttt{ensemble}_{\max}$	11.4	524	1.6	1	1	7,779	16.6	688

much documentation about 'spy' attacks which are not even represented in the test set. However, the signatures of 'imap' and 'phf' are described in [16]. Detection of these attacks requires very specific features. In the case of an 'phf' attack, the IDS "must monitor http requests watching for invocations of the phf command with arguments that specify commands to be run" [17]. None of the 41 features in the KDD99 dataset gives any information about a specific command being run on the system. It would be impractical to do so for each specific command vulnerable to an attack. However, this could be the reason why the machine learning algorithms are incapable of detecting these kind of attacks with certainty. There are two ways to solve this problem, either new features could be added to the dataset or an IDS using signatures of attacks should perform the detection for these particular types of attacks. In the former case, the new features should not be too specific to ensure that new attacks could also be identified. In the second case, the IDS loses its ability to detect similar attacks but its accuracy increases. To detect an 'imap' attack, an IDS should be "programmed to monitor network traffic for oversized Imap authentication strings" [17]. This seems more within reach of our IDS, since `service` and `src_bytes` are both represented in the feature set.

$\texttt{ensemble}_{\max}$ was highly successful on the `DoS` class, returning zero FP. Table III shows that the number of FN is reduced as well. Three types of attacks trigger FN: 'smurf', 'neptune' and 'back'. The first two rarely appear in the list; however, the third seems to be the most difficult type to handle. This is not a surprise, since to detect a 'back' the IDS must look for a big number of frontslashes ("/") in the request URL [16]. There are no features in the dataset taking this particularity into account. Consequently, the model has to rely on other features to make up for the lack of information, leading to an imperfect result. Nevertheless, as expected, $\texttt{ensemble}_{\max}$ brings robustness to the accuracy of the IDS.

C. Model Assessment

In the second round of experiments, several classifiers were trained with different number of features on examples from the training set. Again the algorithm used as classifier was decision tree. The goal of the experiment was to evaluate the model used in the previous experiment on the test set after training on the same number of examples as selected for the training set for each class in the first experiment. As discussed in Section II-A, the test set is composed of many examples of unseen attacks (attacks that are not represented in the training set). The experiment aimed to assess if the ensemble was capable of generalizing to new types of attacks belonging to the same classes as the ones previously seen.

In most cases, the accuracy of all algorithms degraded drastically in comparison to the first experiment as shown in Table IV, where the values represent one run of the program. In particular, the set of features selected by SVM obtains the worst results, and does not seem to generalize well to new types of attacks. The set selected by LGP managed to keep a respectable accuracy on the `Probe` class, while all classifiers except SVM showed results very similar to those in the feature selection experiments on `U2R`, with the "combined" set of features being the best one, outperforming even the algorithm trained with all 41 features in the same way that was observed in the feature selection experiment.

Particularly bad results could be expected for `R2L` because of the poor distribution of attacks in the training set, and Table IV confirms this: the accuracy of all algorithms is equal or close to the 50% guessing baseline. Most of the attacks are 'warezclient' (1020 out of 1126 in total for the `R2L` training set) leaving only 106 instances of all other attack types (seven different types) to train on — and 'warezclient' is not even represented in the test set. There is no chance that the models built would perform well on new attacks (or even on old) with this limited training set. Also the results for `DoS` were much worse than in the first experiment, with the set of features selected by LGP obtaining by far the worst results. Nevertheless, all other algorithms performed better than the one trained with all features.

As Table V shows, the $\texttt{ensemble}_{\max}$ is able to handle part of the new attacks, but does not recognize them as easily as the old ones, and the number of false negatives is very high for most classes. For `Probe`, the most surprising fact is that the attack 'ipsweep' seems to go undetected almost all the time. This result is unusual because 'ipsweep' was available in the training set and did not cause any trouble in the previous experiment. One reason for this could be if the examples of 'ipsweep' from the test set were very different from the ones in the training set. However, after examining the training set carefully, typical values for the features of an 'ipsweep' attack were observed, and it appears that the values of 'ipsweep' in the test set are in the same range as those in the training set. To conclude, the results are not as bad as they look. First, almost all old attacks are perfectly detected, especially 'portsweep' and 'satan' which triggered FN in the first experiment are now absent from the attacks triggering FN. The new attacks are detected most of the time, but the number of FN is still too high to be useful in a real-world application. Finally, solving the problem of 'ipsweep' would substantially bring down the number of FN.

For U2R, the $\texttt{ensemble}_{\text{max}}$ brings down the number of FP to 1 and the number of FN to 0 with an average value of 1.6 and 1.0, respectively, over five runs of the program. As expected, sometimes a ‘rootkit’ attack goes undetected, just as in the first experiment. Besides, ‘ps’ also rarely appears as an FN. The most surprising result comes from undetected ‘buffer overflow’ even though it never happened in the previous experiment. However, ‘xterm’ and ‘sqlattack’ are detected all the time which is good because it means that the $\texttt{ensemble}_{\text{max}}$ generalizes well for the U2R class.

The number of FN for R2L explodes. Old and new types of attacks are similarly misclassified. The only conclusion that can be drawn is that the R2L training set contains too few examples of each type of attack to be of any help.

For DoS, the major part of FN are due to new attacks. ‘pod’ is the only old attack that regularly triggers a few FN, while other old attacks such as ‘smurf’ and ‘neptune’ sometimes trigger FN, but the number of FN for those are very low. New attacks are more problematic, with ‘mailbomb’, ‘apache2’, ‘processtable’ and ‘udpstorm’ recurrently triggering FN, even if most of these attacks are detected in general. Even though its generalization power is limited, $\texttt{ensemble}_{\text{max}}$ performed quite well overall on unseen DoS attacks and helped bring down both FP and FN. This is quite an improvement, but again not enough for a real-world application.

IV. Discussion and Future Work

The Feature Selection Assessment experiments showed that the ensemble approach is indeed a very powerful paradigm that can be used to bring down the number of FP and FN. The lower accuracy observed by individual algorithms is countered by the union of their results. Even with sets containing only five features, the results are very encouraging. Moreover, treating each class of attack as a different problem solved by a specialised algorithm seems to work well when compared to strategies using one algorithm to detect all classes of attacks. “Divide and conquer” and “Unity is strength” seem to be opposite views, but they are actually both applied in this work with impressive results. In general, algorithms using fewer features have slightly lower accuracy and prediction time but much lower training time. The results obtained by Mukkamala *et al.* [7] seem to be correct. However, the features selected by LGP give the worst result in most cases except for DoS where it is the feature set selected by SVM which performs poorly. Consequently, the sets of features selected by LGP should be reconsidered for all classes except DoS, while the set of features selected for DoS by SVM should be replaced. The number of different types of attacks that go undetected is very small and only few examples of these attacks are problematic. Most of the time, the problem lays in the lack of information contained in the dataset. Some attacks require very specific features and should probably be handled by specialized programs or signature-based IDSs. The class Probe is a bigger problem since most of the attacks belonging to this class exploit a legitimate feature used by network administrators; as a result, all types of Probe attacks trigger FN at some point, even though ‘portsweep’ and ‘satan’ are the most problematic.

A smaller feature set means that less information must be extracted from a network packet in the data preprocessing phase. Since the accuracy is not lowered too much in the best cases, this is a huge improvement that could be used in real IDSs. Moreover, the union of all algorithms using fewer features tremendously improves the accuracy: on average over ten runs of the program, only 0.7 FP and 3 FN were observed for Probe over 20,000 examples, 6.6 FP and 0.5 FN for R2L over 2,252 examples, 0.3 FP and 0.3 FN for U2R over 104 examples, and 0 FP and 1.6 FN for DoS over 20,000 examples. Even though these results are much better than what could be achieved with a single algorithm, they are still quite far from being useful in a real-world application where the false positives and negatives should be < 1 for some 15 millions examples in a 10Gb/s Ethernet network. Arguably, 90% of the 15 millions examples will be normal traffic containing no attack at all, but $\texttt{ensemble}_{\text{max}}$ still has to be improved to stand a chance against clever hackers.

The results described above are the best that an ensemble composed of these algorithms and sets of features could achieve. In its current state, there is no point in building an experiment to assess a real combination of the results of the individual algorithms in the $\texttt{ensemble}_{\text{max}}$. Further work will have to be carried out to find the best suitable algorithms and sets of features. Nevertheless, it is interesting to see how well this $\texttt{ensemble}_{\text{max}}$ can perform when predicting previously unseen attack types. That was the topic of the second round of experiments, on Model Assessment. Even if $\texttt{ensemble}_{\text{max}}$ in general helps tremendously to bring down the numbers of false positives and false negatives, it is still far from reaching the accuracy appropriate for a real-world application. In particular, datasets which are not carefully designed are proven to be useless in building accurate models of the attacks. This is the case with the R2L training set which contains mainly examples of the ‘warezclient’ attack which is not even represented in the test set and very few examples of all other types of attacks. The performance of $\texttt{ensemble}_{\text{max}}$ was acceptable for the classes of attacks U2R and DoS. The performance on the Probe class was also standard, even though ‘ipsweep’ attacks went undetected for unknown reasons. Overall, we can say that the results of this second round of experiments were not very satisfying, but once again proved the usefulness of the ensemble approach.

In the future, particular attention has to be paid to the features relevant to each attack. New features carrying meaningful information about the attacks must be designed to help the machine learning algorithms to successfully classify all types of attack. The DoS and Probe classes are mostly characterized by time-related features, whereas R2L and U2R mostly are characterized by content-related features extracted from the payload of the network packets.

V. Conclusion

The aim of this work was to show that ensemble approaches fed with appropriate features sets can help tremendously in reducing both the number of false positives and false negatives. In particular, our work showed that the sets of relevant features are different for each class of attacks which is why it is important to treat those classes separately. We developed our own IDS to evaluate the relevance of the sets of features selected by Mukkamala *et al.* [12]. This system is an ensemble of four ensembles of decision trees. Each of the four ensembles is in charge of detecting one class of attacks and composed of four decision trees trained on different sets of features. The first three decision trees were fed with sets of five features selected in [12]. The last decision tree was fed with the union of these three sets of five features from which the redundant features were removed.

The experiments showed that these sets were appropriate in most cases. In the first experiment, the set of features selected by linear genetic programming gave the worst results, except for the class `DoS` for which the set of features selected by SVM performed poorly. The second experiment gave less interesting results because of the inappropriate distribution of examples between the training and test sets of the KDD99 data. In particular, the ensemble could not generalize properly on the `R2L` class because the training set mainly contains a type of attack that is not represented in the test set. In both experiments, we looked at the number of instances that were misclassified by all four algorithms in order to obtain a result from the best combination of these algorithms. Further work would be required to develop a real decision function combining the results of the different algorithms. However, since the accuracy obtained here was not good enough for a real-world application, designing decision functions was unnecessary. Nevertheless, we are convinced that this work is heading in the right direction in order to overcome the limitations of current intrusion detection systems.

Finally, a thorough analysis of the examples that were misclassified by the ensemble was performed, in particular highlighting the types of attacks that were systematically misclassified by the ensemble. By looking at the signatures of these attacks, we were able to find the reasons for the classification errors. In most cases, the attacks displayed very specific features not captured by the set of variables in the dataset. These attacks should probably be handled by a specialized system or new variables should be developed to train the classifiers.

Acknowledgements

The authors would like to express their thanks to Esteban Zymanyi, Olivier Markowitch and Liran Lerman (all at Université Libre de Bruxelles) for valuable comments.

References

[1] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, Washington, DC, Jun. 2010, pp. 305–316.

[2] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *Proceedings of the 2nd International Conference on Computational Intelligence for Security and Defense Applications*, Ottawa, Ontario, Canada, Jun. 2009, pp. 53–58.

[3] S. X. Wu and W. Banzhaf, “The use of computational intelligence in intrusion detection systems: A review,” *Applied Soft Computing*, vol. 10, no. 1, pp. 1–35, 2010.

[4] E. Bahri, N. Harbi, and H. N. Huu, “Approach based ensemble methods for better and faster intrusion detection,” in *Proceedings of the 4th International Conference on Computational Intelligence in Security for Information Systems*, Torremolinos-Málaga, Spain, Jun. 2011, pp. 17–24.

[5] S. González, J. Sedano, A. Herrero, B. Baruque, and E. Corchado, “Testing ensembles for intrusion detection: On the identification of mutated network scans,” in *Proceedings of the 4th International Conference on Computational Intelligence in Security for Information Systems*, Torremolinos-Málaga, Spain, Jun. 2011, pp. 109–117.

[6] P. Zhang, X. Zhu, Y. Shi, L. Guo, and X. Wu, “Robust ensemble learning for mining noisy data streams,” *Decision Support Systems*, vol. 50, no. 2, pp. 469–479, Jan. 2011.

[7] S. Mukkamala, A. H. Sung, and A. Abraham, “Intrusion detection using an ensemble of intelligent paradigms,” *Journal of Network and Computer Applications*, vol. 28, no. 2, pp. 167–182, Apr. 2005, special issue on computational intelligence on the internet.

[8] S. Chebrolu, A. Abraham, and J. P. Thomas, “Feature deduction and ensemble design of intrusion detection systems,” *Computers & Security*, vol. 24, no. 4, pp. 295–307, 2005.

[9] S. Peddabachigari, A. Abraham, C. Grosan, and J. P. Thomas, “Modeling intrusion detection system using hybrid intelligent systems,” *Journal of Network and Computer Applications*, vol. 30, 2005.

[10] A. Abraham, R. Jain, J. P. Thomas, and S.-Y. Han, “D-SCIDS: Distributed soft computing intrusion detection system,” *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 81–98, 2007.

[11] G. Folino, C. Pizzuti, and G. Spezzano, “An ensemble-based evolutionary framework for coping with distributed intrusion detection,” *Genetic Programming and Evolvable Machines*, vol. 11, pp. 131–146, 2010.

[12] S. Mukkamala, A. Sung, and A. Abraham, “Cyber security challenges: Designing efficient intrusion detection systems and antivirus tools,” in *Enhancing Computer Security with Smart Technology*. USA: CRC Press, 2005, pp. 125–161.

[13] S. Mukkamala and A. H. Sung, “Identifying significant features for network forensic analysis using artificial intelligent techniques,” *International Journal of Digital Evidence*, vol. 1, no. 4, pp. 1–17, 2003.

[14] A. H. Sung and S. Mukkamala, “The feature selection and intrusion detection problems,” in *Proceedings of the 9th Asian Conference on Advances in Computer Science*. Chiang Mai, Thailand: Springer-Verlag, 2004, pp. 468–482.

[15] H. Gunes Kayacik, A. Nur Zincir-Heywood, and M. I. Heywood, “A hierarchical SOM-based intrusion detection system,” *Engineering of Applied Artificial Intelligence*, vol. 20, no. 4, pp. 439–451, Jun. 2007.

[16] K. Kendall, “A database of computer attacks for the evaluation of intrusion detection systems,” in *DARPA off-line intrusion detection evaluation, proceedings DARPA information survivability conference and exposition*, 1999, pp. 12–26.

[17] ——, “Intrusion detection attacks database,” Webpage (last accessed: June 24, 2012), 2007, http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/docs/attackDB.html.

Linear Constraints as a Modeling Language for Discrete Time Hybrid Systems

Federico Mari, Igor Melatti, Ivano Salvo, and Enrico Tronci
Department of Computer Science – Sapienza University of Rome
Via Salaria 113, 00198 Rome, Italy
Email: {mari,melatti,salvo,tronci}@di.uniroma1.it

***Abstract*—*Model Based Design* is particularly appealing in embedded software design where system level specifications are much easier to define than the control software behavior itself. Formal analysis of *Embedded Systems* requires modelling both continuous systems (typically, the plant) as well as discrete systems (the controller). This is typically done using *Hybrid Systems*. Mixed Integer Linear Programming (MILP) based abstraction techniques have been successfully applied to automatically synthesize correct-by-construction control software for *Discrete Time Linear Hybrid System*, where plant dynamics is modeled as a linear predicate over state, input, and next state variables. MILP solvers requires constraints represented as conjunctive predicates. In this paper we show that, under the hypothesis that each variable ranges over a bounded interval, any linear predicate built upon conjunction and disjunction of linear constraints can be automatically transformed into an equisatisfiable conjunctive predicate. Moreover, since variable bounds play a key role in this transformation, we present an algorithm that taking as input a linear predicate, computes *implicit variable bounds*.**

***Keywords*-Model-based software design; Linear predicates; Hybrid systems**

I. Introduction

Many Embedded Systems are *Software Based Control Systems* (SBCSs). An SBCS consists of two main subsystems: the *controller* and the *plant*. Typically, the plant is a physical system consisting, for example, of mechanical or electrical devices, while the controller consists of control software running on a microcontroller. In an endless loop, each T seconds (sampling time), the controller, after an *Analog-to-Digital* (AD) conversion (quantization), reads sensor outputs from the plant and, possibly after a *Digital-to-Analog* (DA) conversion, sends commands to plant actuators. The controller selects commands in order to guarantee that the *closed loop system* (that is, the system consisting of both plant and controller) meets given safety and liveness specifications (*System Level Specifications*).

Software generation from models and formal specifications forms the core of *Model Based Design* of embedded software [1]. This approach is particularly interesting for SBCSs since in such a case system level specifications are much easier to define than the control software behavior itself. Correct-by-construction software generation as well as formal verification of system level specifications for SBCSs requires modelling both the continuous subsystem (the plant) and discrete systems (the controller). This is typically done using *Hybrid Systems* (e.g., see [2][3]).

Discrete Time Linear Hybrid Systems (DTLHSs) [4][5] provide an expressive model for closed loop systems: a DTLHS is a discrete time hybrid system whose dynamics is defined as a linear predicate (i.e., a boolean combination of linear constraints) on its continuous as well as discrete (modes) variables. A large class of hybrid systems, including mixed-mode analog circuits, can be modeled using DTLHSs. System level safety as well as liveness specifications are modeled as set of states defined, in turn, as linear predicates.

In [6], stemming from a constructive sufficient condition for the existence of a quantized sampling controller for an SBCS modelled as a DTLHS, we presented an algorithm that, given a DTLHS model $\mathcal{H}$ for the plant, a quantization schema (i.e., how many bits we use for AD conversion) and system level specifications, returns correct-by-construction quantized feedback *control software* (if any) meeting the given system level specifications. The synthesis algorithm rests on the fact that, because of the quantization process, the plant P is seen by the controller as a *Nondeterministic Finite State Automaton* (NFSA) $\hat{P}$, that is an abstraction of P. The NFSA $\hat{P}$ is computed by solving *Mixed Integer Linear Programming* (MILP) problems, and thus it requires the DTLHS dynamics given as a conjunctive predicate, i.e., a conjunction of linear constraints.

This paper is motivated by circumventing such a limitation, by showing that, under the hypothesis that each variable ranges over a bounded interval, any linear predicate can be represented by an equivalent conjunctive predicate.

Bounds on variables that describe DTLHS behaviour is a reasonable hypothesis. Usually, control software drives the plant towards a goal, while keeping it inside a given bounded admissible region. Bounds on present state variables essentially model the *sensing region*, that is the range of values observable by the sensors, that usually is a bounded rectangular region (i.e., the Cartesian product of bounded intervals). Bounds on controllable input variables model the *actuation region*, that is the range of values of commands that the actuators may send to the plant and it is also typically a bounded rectangular region. Non-state variables may model both non-observable plant state variables and uncontrollable inputs (i.e., disturbances). Therefore, bounds on such variables are usually implied by bounds on state variables or by reasonable assumptions about disturbances.

1) Our Main Contributions: In this paper we give an algorithm to transform any linear predicate into an equisatisfiable conjunctive predicate, under the hypothesis that each variable ranges over a bounded interval. This allows a MILP based abstraction technique to be applied on a wider class of DTLHSs (Section III) with respect to [6].

We consider predicates built upon linear constraints (i.e., inequalities of the shape $\sum_{i=1}^{n} a_i x_i \leq b$, Section II), conjunctions and disjunctions. First, we show that, at the price of introducing fresh boolean variables, a predicate can be transformed into an equisatisfiable *guarded predicate* (Section IV), that is a conjunction of guarded constraints, i.e., constraints of the shape $y \to (\sum_{i=1}^{n} a_i x_i \le b)$. Then, assuming that each variable ranges over a bounded interval, we show that any guarded constraint can be in turn transformed into a *conjunctive predicate*, i.e., a conjunction of linear constraints (Section IV-A). Conjunctive predicates are the input language of MILP solvers. Finally, in Section V, we give an algorithm that computes bounds for a variable x in a given guarded predicate $G(X)$, i.e., either it returns two values $m_x, M_x \in \mathbb{R}$ such that if $G(X)$ holds, then $m_x \le x \le M_x$, or it concludes that such values do not exist. An evaluation of such algorithm is in Sections VI and VII.

A. Related Work

Mixed Integer Linear Programming (MILP) solving based abstraction techniques have been designed for the verification of Discrete Time Hybrid Automata (DHA) [4] and implemented within the symbolic model checker HYSDEL [7]. A MILP based DTLHS abstraction algorithm is the core of automatic control software synthesis from system level specifications in [6], and it requires DTLHS dynamics modeled as a conjunctive predicate. The same limitation occurs in abstraction techniques based on the Fourier-Motzkin procedure for existential quantifier elimination [8].

The automatic procedure that we present here to transform any linear predicate into an equisatisfiable conjunctive predicate is reminiscent of Mixed Integer Programming modeling techniques [9] in Operations Research and boolean formula transformations involved in the conversion of a formula into a conjunctive or disjunctive normal form [5][10].

Finally, an automatic convertion procedure targeting a MILP formulation for automatic synthesis of schedules is presented in [11], where the starting point is a deterministic finite automaton rather than a linear predicate.

II. Basic Definitions

An initial segment $\{1, \ldots, n\}$ of $\mathbb{N}$ is denoted by $[n]$. We denote with $X = x_1, \ldots, x_n$ a finite sequence of distinct variables, that we may regard, when convenient, as a set. Each variable x ranges on a known (bounded or unbounded) interval $\mathcal{D}_x$ either of the reals (continuous variables) or of the integers (discrete variables). The set $\prod_{x \in X} \mathcal{D}_x$ is denoted by $\mathcal{D}_X$. Boolean variables are discrete variables ranging on the set $\mathbb{B} = \{0, 1\}$. If x is a boolean variable we write $\bar{x}$ for $(1 - x)$. The sequence of continuous (discrete, boolean) variables in X is denoted by X^r (X^d, X^b).

The set of sequences of n boolean values is denoted by $\mathbb{B}^n$. The set $\mathbb{B}^n_k \subseteq \mathbb{B}^n$ denotes sequences that contains exactly k elements equal to 1. Given $a, b \in \mathbb{B}^n$ we say that $a \le b$ if a is point-wise less or equal to b, i.e., if for all $i \in [n]$ we have that $a_i \le b_i$. Given a set $B \subseteq \mathbb{B}^n$ and $a \in \mathbb{B}^n$ we write $a \le B$ if there exists $b \in B$ such that $a \le b$ and $a \ge B$ if there exists $b \in B$ such that $a \ge b$. We denote with $J(b)$ be the set of indexes such that $b_j = 1$, i.e., $J(b) = \{j \in [n] \mid b_j = 1\}$.

A. Predicates

A *linear expression* $L(X) = \sum_{i=1}^{n} a_i x_i$ is a linear combination of variables in X with rational coefficients. A *constraint* is an expression of the form $L(X) \le b$, where b is a rational constant. We write $L(X) \ge b$ for $-L(X) \le -b$, $L(X) = b$ for $(L(X) \le b) \wedge (-L(X) \le -b)$, and $a \le L(X) \le b$ for $(L(X) \le b) \wedge (L(X) \ge a)$.

Predicates are inductively defined as follows. A constraint $C(X)$ is a predicate over X. If $A(X)$ and $B(X)$ are predicates, then $(A(X) \wedge B(X))$ and $(A(X) \vee B(X))$ are predicates over X. Parentheses may be omitted, assuming usual associativity and precedence rules of logical operators. A *conjunctive predicate* is a conjunction of constraints.

A *valuation* over X is a function v that maps each variable $x \in X$ to a value $v(x)$ in $\mathcal{D}_x$. We denote with $X^* \in \mathcal{D}_X$ the sequence of values $v(x_1), \ldots, v(x_n)$. Given a predicate $P(Y, X)$, $P(Y, X^*)$ denotes the predicate obtained by substituting each occurrence of x with $v(x)$. We call valuation also the sequence of values X^*. A *satisfying assignment* to a predicate $P(X)$ is a valuation X^* such that $P(X^*)$ holds. We denote with P also the set of satisfying assignments to the predicate P. $P(X)$ and $Q(X)$ are *equivalent*, notation $P \equiv Q$, if they have the same set of satisfying assignments. $P(X)$ and $Q(Z)$ are *equisatisfiable*, notation $P \simeq Q$, if P is satisfiable if and only if Q is satisfiable.

B. Mixed Integer Linear Programming

A *Mixed Integer Linear Programming* (MILP) problem with *decision variables* X is a tuple $(\max, J(X), A(X))$ where X is a list of variables, $J(X)$ (*objective function*) is a linear expression over X, and $A(X)$ (*constraints*) is a conjunctive predicate over X. A *solution* to $(\max, J(X), A(X))$ is a valuation X^* such that $A(X^*)$ and $\forall Z\, (A(Z) \to (J(Z) \le J(X^*)))$. $J(X^*)$ is the *optimal value* of the MILP problem. A *feasibility* problem is a MILP problem of the form $(\max, 0, A(X))$. We write also $A(X)$ for $(\max, 0, A(X))$. In algorithm outlines, MILP solver invocations are denoted by function *feasible*($A(X)$) that returns TRUE if $A(X)$ is satisfiable and FALSE otherwise, and by function *optimalValue*($\max$, $J(X)$, $A(X)$) that returns either the optimal value of the MILP problem ($\max$, $J(X)$, $A(X)$) or ∞ if such MILP problem is unbounded. We write $(\min, J(X), A(X))$ for $(\max, -J(X), A(X))$.

III. Discrete Time Linear Hybrid Systems

Discrete Time Linear Hybrid Systems (DTLHSs) provide a suitable model for many embedded control systems since they can effectively model linear algebraic constraints involving both continuous as well as discrete variables. In Ex. 1, we present a DTLHS model of a buck DC-DC converter, i.e., a mixed-mode analog circuit that converts the DC input voltage to a desired DC output voltage.

Definition 1: A *Discrete Time Linear Hybrid System* is a tuple $\mathcal{H} = (X, U, Y, N)$ where:

$X = X^r \cup X^d$ is a finite sequence of real and discrete *present state* variables. X' denotes the sequence of *next state* variables obtained by decorating with $'$ variables in X.

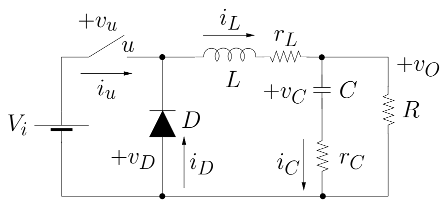


Figure 1. Buck DC-DC converter

$U = U^r \cup U^d$ is a finite sequence of *input* variables.

$Y = Y^r \cup Y^d$ is a finite sequence of *auxiliary* variables. Auxiliary variables typically models *modes* (switching elements) or *uncontrollable inputs* (e.g., disturbances).

$N(X, U, Y, X')$ is a predicate over $X \cup U \cup Y \cup X'$ defining the *transition relation* (*next state*) of the system.

Example 1: The buck DC-DC converter is a mixed-mode analog circuit (Figure 1) converting the DC input voltage (V_i in Figure 1) to a desired DC output voltage (v_O in Figure 1). Buck DC-DC converters are used off-chip to scale down the typical laptop battery voltage (12-24) to the just few volts needed by the laptop processor as well as on-chip to support *Dynamic Voltage and Frequency Scaling* (DVFS) in multicore processors. Because of its widespread use, control schemes for buck DC-DC converters have been widely studied (e.g., see [12][13][14]). The typical software based approach is to control the switch u in Figure 1 (typically implemented with a MOSFET) with a microcontroller. The circuit in Figure 1 can be modeled as a DTLHS $\mathcal{H} = (X, U, Y, N)$. The circuit state variables are i_L and v_C. However we can also use the pair i_L, v_O as state variables in $\mathcal{H}$ model since there is a linear relationship between i_L, v_C and v_O, namely: $v_O = \frac{r_C R}{r_C + R} i_L + \frac{R}{r_C + R} v_C$. Such considerations lead us to the following DTLHS model $\mathcal{H}$: $X = X^r = i_L, v_O$, $U = U^d = u$, $Y = Y^r \cup Y^d$ where $Y^r = i_u, v_u, i_D, v_D$ and $Y^d = q$. Note how $\mathcal{H}$ auxiliary variables Y stem from the constitutive equations of the switching elements (i.e., the switch u and the diode D in Figure 1). From a simple circuit analysis (e.g., see [15]) we have the following equations:

$$\dot{i_L} = a_{1,1} i_L + a_{1,2} v_O + a_{1,3} v_D \quad (1)$$
$$\dot{v_O} = a_{2,1} i_L + a_{2,2} v_O + a_{2,3} v_D \quad (2)$$

where the coefficients $a_{i,j}$ depend on the circuit parameters R, r_L, r_C, L and C as follows: $a_{1,1} = -\frac{r_L}{L}$, $a_{1,2} = -\frac{1}{L}$, $a_{1,3} = -\frac{1}{L}$, $a_{2,1} = \frac{R}{r_c + R}[-\frac{r_c r_L}{L} + \frac{1}{C}]$, $a_{2,2} = \frac{-1}{r_c + R}[\frac{r_c R}{L} + \frac{1}{C}]$, $a_{2,3} = -\frac{1}{L}\frac{r_c R}{r_c + R}$. Using a discrete time model with sampling time T and writing x' for $x(t+1)$, we have:

$$i'_L = (1 + T a_{1,1}) i_L + T a_{1,2} v_O + T a_{1,3} v_D \quad (3)$$
$$v'_O = T a_{2,1} i_L + (1 + T a_{2,2}) v_O + T a_{2,3} v_D. \quad (4)$$

The algebraic constraints stemming from the constitutive equations of the switching elements are the following:

$$v_D = v_u - V_i \quad (5) \qquad (u = 1) \vee (v_u = R_{\text{off}} i_u) \quad (7)$$
$$i_D = i_L - i_u \quad (6) \qquad (u = 0) \vee (v_u = 0) \quad (8)$$
$$((i_D \geq 0) \wedge (v_D = 0)) \vee ((i_D \leq 0) \wedge (v_D = R_{\text{off}} i_D)) \quad (9)$$

The transition relation N of $\mathcal{H}$ is given by the conjunction of the constraints in Eqs. 3–9.

IV. From Linear to Conjunctive Predicates

As shown in [6], MILP solvers can be used to build a suitable discrete abstraction of a DTLHS. As mentioned in Section II-B, MILP solvers require constraints represented as conjunctive predicates. In this section, we show how this limitation can be circumvented. We proceed in two steps. First, in Section IV, we introduce *guarded predicates* and we show that each predicate can be transformed into an equivalent guarded predicate at the price of introducing new auxiliary boolean variables. Then, in Section IV-A, we show that, under the hypothesis that each variable ranges over a bounded interval, each guarded predicate can be in turn transformed into an equivalent conjunctive predicate.

1) Guarded Predicates:

Definition 2: Given a predicate $P(X)$ and a fresh boolean variable $z \notin X$, the predicate $z \to P(X)$ (resp. $\bar{z} \to P(X)$) denotes the predicate $(z = 0) \vee P(X)$ (resp. $(z = 1) \vee P(X)$). We call z the *guard variable* and both z and $\bar{z}$ *guard literals*. If $P(X)$ is a constraint $C(X)$, a predicate of the form $z \to C(X)$ or $\bar{z} \to C(X)$ is called *guarded constraint*. A *generalized guarded constraint* a predicate of the form $z_1 \to (z_2 \to \ldots \to (z_n \to C(X))\ldots)$ A *guarded predicate* (resp. *generalized* guarded predicate) is a conjunction of either constraints or guarded constraints (resp. generalized guarded constraints).

To simplify proofs and notations, without loss of generality, we always assume guard literals distinct: a conjunction $z \to C_1(X) \wedge z \to C_2(X)$ is equisatisfiable to the guarded predicate $z_1 \to C_1(X) \wedge z_2 \to C_2(X) \wedge z_1 = z \wedge z_2 = z$ (z_1, z_2 fresh boolean variables). Moreover, in algorithm outlines, conjunctive predicates will be regarded as sets of constraints.

By applying standard propositional equivalences, we have the following facts.

Fact 1: A predicate of the form $z \to \bigwedge_{i \in [n]} P_i(X)$ is equivalent to the guarded predicate $\bigwedge_{i \in [n]} (z \to P_i(X))$.

Fact 2: A generalized guarded constraint $z_1 \to (z_2 \to \ldots \to (z_n \to C(X))\ldots)$ is equisatisfiable to the guarded predicate $(z - \sum_{i \in [n]} z_i \geq 1 - n) \wedge (z \to C(X))$, where z is a fresh boolean variable.

Proof: Let z be a fresh boolean variable. We have:

$$\begin{aligned} & z_1 \to (z_2 \to \ldots \to (z_n \to C(X))\ldots) \\ & \equiv z_1 \wedge z_2 \wedge \ldots \wedge z_n \to C(X) \\ & \simeq (z_1 \wedge z_2 \wedge \ldots \wedge z_n \to z) \wedge (z \to C(X)) \\ & \equiv (\bar{z}_1 \vee \bar{z}_2 \vee \ldots \vee \bar{z}_n \vee z) \wedge (z \to C(X)) \\ & \equiv (1 - z_1) + (1 - z_2) + \ldots + (1 - z_n) + z \geq 1 \wedge (z \to C(X)) \\ & \equiv (z - \textstyle\sum_{i \in [n]} z_i \geq 1 - n) \wedge (z \to C(X)) \end{aligned}$$ ■

Lemma 3: Any predicate $P(X)$ is equisatisfiable to a predicate $Q(X, Z) = G(X, Z) \wedge D(Z)$, where G and D are generalized guarded predicates and Z is the set of boolean variables that occur positively as guards in G.

Proof: By induction on the structure of the predicate $P(X)$. If $P(X)$ is a constraint or a conjunction, the statement easily follows from inductive hypothesis.

Let $P(X)$ be the disjunction $P_1(X) \vee P_2(X)$. By inductive hypothesis, there exist two generalized guarded predicates $Q_1(X, Z_1) = G_1(X, Z_1) \wedge D_1(Z_1)$ and $Q_2(X, Z_2) =$

$G_2(X,Z_2) \wedge D_2(Z_2)$ such that $P_1(X) \simeq Q_1(X,Z_1)$ and $P_2(X) \simeq Q_2(X,Z_2)$. We can always choose auxiliary boolean variables in such a way that $Z_1 \cap Z_2 = \varnothing$.

Taken two fresh boolean variables y_1 and y_2, the predicate $y_1 \to Q_1(X,Z_1) \wedge y_2 \to Q_2(X,Z_2) \wedge y_1 + y_2 \geq 1$ is equisatisfiable to $P(X)$. The predicate $y_1 \to Q_1(X,Z_1)$ has the form $y_1 \to (\bigwedge_{i\in[n]} G_1^i(X,Z_1) \wedge \bigwedge_{j\in[p]} D_1^j(Z_1))$ and therefore it is not a generalized guarded constraint. By Fact 1, it is equivalent to the predicate $\bigwedge_{i\in[n]}(y_1 \to G_1^i(X,Z_1)) \wedge \bigwedge_{j\in[p]}(y_1 \to D_1^j(Z_2))$. By applying Fact 1 also to $y_2 \to Q_2(X,Z_2)$, the statement follows by taking $Z = Z_1 \cup Z_2 \cup \{y_1, y_2\}$, $G(X,Z) = \bigwedge_{i\in[n]} y_1 \to G_1^i(X,Z_1) \wedge \bigwedge_{i\in[m]} y_2 \to G_2^i(X,Z_2)$, and $D(Z) = \bigwedge_{j\in[p]} y_1 \to D_1^j(Z_2) \wedge \bigwedge_{j\in[q]} y_2 \to D_2^j(Z_2) \wedge (y_1 + y_2 \geq 1)$ ■

Proposition 4: Any predicate $P(X)$ is equisatisfiable to a predicate $Q(X,Z) = G(X,Z') \wedge D(Z)$, where G and D are guarded predicates and $Z' \subseteq Z$ is the set of boolean variables that occur positively as guards in G.

Proof: By Lemma 3, any predicate $P(X)$ is equisatisfiable to a generalized guarded predicate $G_1(X,Z_1) \wedge D_1(Z_1)$. By Fact 2, $D_1(Z_1)$ is equisatisfiable to a guarded predicate $D_2(Z_1,Z_2)$. Let $G_1(X,Z_1) = \bigwedge_{i\in[n]} z_1^i \to (z_2^i \to \ldots \to (z_{n_i}^i \to C_i(X)) \ldots) \wedge G_1'(X,Z_3)$, where $G_1'(X,Z_3)$ is a guarded predicate ($Z_3 \subseteq Z_1$). By Fact 2 $G_1(X,Z_1)$ is equisatisfiable to the guarded predicate $\bigwedge_{i\in[n]} w_i \to C_i(X) \wedge \bigwedge_{i\in[n]}(w_i - \sum_{j\in[n_i]} z_j^i \geq 1 - n_i) \wedge G'(X,Z_3)$. The statement follows by taking $Z' = Z_3 \cup \{w_1,\ldots,w_n\}$, $Z = Z' \cup Z_1 \cup Z_2$, $G(X,Z) = \bigwedge_{i\in[n]}(z^i \to C_i(X)) \wedge G''(X,Z')$, and $D(X,Z) = \bigwedge_{i\in[n]}(z^i - \sum_{j\in[n_i]} z_j^i \geq 1 - n) \wedge D''(Z',Z'')$ ■

The function *PtoG* (Alg. 2) summarizes the predicate transformations given in the proof of Prop. 4. It calls function *PtoGG* (Alg. 1) that performs predicate transformations given in the proof of Lemma 3. The function *fresh*( ) returns at each invocation a (globally) fresh variable.

Algorithm 1 From predicates to generalized guarded pred.

```
Input: $P$ predicate over $X$
Output: $\langle G, D, Z\rangle$ where $G$ is a general. guarded predicate,
    $Z$ is the set of its (fresh) guard variables,
    $D(Z)$ is a generalized guarded predicate over Z
function $PtoGG(P, X)$
 1. if $P$ is a constraint $C(X)$ then return $\langle C(X), \varnothing, \varnothing\rangle$
 2. let $P = P_1 \diamond P_2 \quad (\diamond \in \{\wedge, \vee\})$
 3. $\langle G_1, D_1, Z_1\rangle \leftarrow PtoGG(P_1)$
 4. $\langle G_2, D_2, Z_2\rangle \leftarrow PtoGG(P_2)$
 5. if $P = P_1 \wedge P_2$ then return $\langle G_1 \cup G_2, D_1 \cup D_2, Z_1 \cup Z_2\rangle$
 6. if $P = P_1 \vee P_2$ then
 7.     $y_1 \leftarrow fresh()$, $y_2 \leftarrow fresh()$, $Z' \leftarrow Z_1 \cup Z_2 \cup \{y_1, y_2\}$
 8.     $D' = \{y_1 \to \gamma | \gamma \in D_1\} \cup \{y_2 \to \gamma | \gamma \in D_2\} \cup \{y_1 + y_2 \geq 1\}$
 9.     $G' = \{y_1 \to \gamma \mid \gamma \in G_1\} \cup \{y_2 \to \gamma \mid \gamma \in G_2\}$
10.     return $\langle G', D', Z'\rangle$
```

Example 2: Let $\mathcal{H}$ be DTLHS in Ex. 1. Given the predicate N that defines the transition relation of $\mathcal{H}$, function *PtoG* computes the following guarded predicate equisatisfiable to N. Constraints 3–6 remain unchanged, as they are linear constraints in a top-level conjunction. The disjunction 9 is replaced first by the following predicates:

$$z_1 \to (i_D \geq 0 \wedge v_D = 0) \quad (10) \qquad z_2 \to (i_D \leq 0 \wedge v_D = R_{\text{off}} i_D) \quad (11)$$

and then by constraints 13–16 below, obtained by moving arrows inside the conjunctions, as shown by Fact 1. Similarly, disjunctions 7 and 8 are eliminated by introducing four boolean fresh variables. Summing up, disjunctions 7–9 in Example 1 are replaced by the conjunction of the following (guarded) constraints:

$$\begin{array}{lrlrlr} z_4 \to (v_u = R_{\text{off}} i_u) & (12) & z_1 \to (i_D \leq 0) & (16) & z_1 + z_2 \geq 1 & (20) \\ z_2 \to (v_D = R_{\text{off}} i_D) & (13) & z_3 \to (u = 1) & (17) & z_3 + z_4 \geq 1 & (21) \\ z_1 \to (i_D \geq 0) & (14) & z_5 \to (u = 0) & (18) & z_5 + z_6 \geq 1 & (22) \\ z_1 \to (v_D = 0) & (15) & z_6 \to (v_u = 0) & (19) & & \end{array}$$

With respect to the statement of Proposition 4, we have that $Z = \{z_1, z_2, z_3, z_4, z_5, z_6\}$, $G(X,Z')$ is the conjunction of guarded constraints 12–19 and original constraints 3–6, and $D(Z)$ is the conjunction of constraints 20–22.

Algorithm 2 From linear to guarded predicates

```
Input: $P$ predicate over $X$
Output: $\langle G, D, Z', Z\rangle$ where $G$ is a guarded predicate,
    $Z' \subseteq Z$ set of its guard variables,
    $D(Z)$ is a guarded predicate over Z
function $PtoG(P, X)$
 1. $\langle G, D, Z\rangle \leftarrow PtoGG(P, X)$
 2. $G' \leftarrow \varnothing$, $D' \leftarrow \varnothing$, $Z' = \varnothing$
 3. for all $\gamma \in G \cup D$ do
 4.   if $\gamma \equiv z_1 \to (\ldots \to (z_n \to C(W)) \ldots)$ then
 5.     $w \leftarrow fresh()$, $Z \leftarrow Z \cup \{w\}$
 6.     if $W \subseteq X$ then $G' \leftarrow G' \cup \{w \to C(W)\}$
 7.         else $D' \leftarrow D' \cup \{w \to C(W)\}$
 8.     $D' \leftarrow D' \cup \{w - \sum_{i\in[n]} z_i \geq 1 - n\}$
 9.   else if $vars(\gamma) \subseteq X$ then
10.       $G' \leftarrow G' \cup \{\gamma\}$ else $D' \leftarrow D' \cup \{\gamma\}$
11. return $\langle G', D', Z', Z \setminus Z'\rangle$
```

A. From Guarded to Conjunctive Predicates

Definition 3: Let $P(X)$ be a predicate. A variable $x \in X$ is said to be *bounded* in P if there exist $a, b \in \mathcal{D}_x$ such that $P(X)$ implies $a \leq x \leq b$. A predicate P is bounded if all its variables are bounded. We write $\sup(P,x)$ and $\inf(P,x)$ for the minimum and maximum value that the variable x may assume in a satisfying assignment for P. When P is clear from the context, we will write simply $\sup(x)$ and $\inf(x)$.

Given a bounded predicate $P(X)$, a real number a, and a variable $x \in X$ we write $\sup(ax)$ for $a\sup(x)$ if $a \geq 0$ and for $a\inf(x)$ if $a < 0$. We write $\inf(ax)$ for $a\inf(x)$ if $a \geq 0$ and for $a\sup(x)$ if $a < 0$. Given a linear expression $L(X) = \sum_{i=1}^n a_i x_i$ over a set of bounded variables, we write $\sup(L(X))$ for $\sum_{i=1}^n \sup(a_i x_i)$ and $\inf(L(X))$ for $\sum_{i=1}^n \inf(a_i x_i)$.

Proposition 5: Each bounded guarded predicate $P(X)$ is equivalent conjunctive predicate $Q(X)$.

Proof: The conjunctive predicate $Q(X)$ can be obtained from the guarded predicate $P(X)$ by replacing each guarded constraint φ of the shape $z \to (L(X) \leq b)$ in $P(X)$ with the constraint $\varphi' = (\sup(L(X)) - b)z + L(X) \leq \sup(L(X))$. If $z = 0$ we have $\varphi \equiv \varphi'$ since φ holds trivially and φ' reduces to $L(X) \leq \sup(L(X))$ that holds by construction. If $z = 1$ both φ and φ' reduce to $L(X) \leq b$. Along the same line of reasoning, if φ has the form $\bar{z} \to (L(X) \leq b)$ we pick φ' to be $(b - \sup(L(X)))z + L(X) \leq b$. ■

Together with Prop. 4, Prop. 5 implies that any bounded predicate can be transformed into an equisatisfiable conjunctive predicate, at the cost of adding new auxiliary boolean variables, as stated in the following proposition.

Proposition 6: For each bounded predicate $P(X)$, there exists an equisatisfiable conjunctive predicate $Q(X, Z)$.

Example 3: Let $\mathcal{H}$ be the DTLHS in Examples 1 and 2. We set the parameters of $\mathcal{H}$ as follows:

$$r_L = 0.1\Omega \quad R = 5\Omega \quad V_i = 15V \quad L = 2 \cdot 10^{-4}H$$
$$r_C = 0.1\Omega \quad R_{\text{off}} = 10^4 \quad T = 10^{-6}\text{secs} \quad C = 5 \cdot 10^{-5}F$$

and we assume variables bounds as follows:

$$-2 \cdot 10^4 \leq v_u \leq 15 \quad -4 \leq i_L \leq 4 \quad -1 \leq v_O \leq 7 \quad -4 \leq i'_L \leq 96$$
$$-2 \cdot 10^4 \leq v_D \leq 0 \quad -1.1 \leq v'_O \leq 17 \quad -4 \leq i_u \leq 4 \quad -2 \leq i_D \leq 4$$

By first decomposing equations of the shape $L(X) = b$ in the conjunctive predicate $L(X) \leq b \wedge -L(X) \leq -b$ and then by applying the transformation given in the proof of Prop. 5, guarded constraints 14–19 are replaced by the following linear constraints:

$$2z_1 - i_D \leq 2 \quad (23)$$
$$4 \cdot 10^4 z_4 + v_u - 10^4 i_u \leq 4 \cdot 10^4 \quad (24)$$
$$6 \cdot 10^4 z_4 - v_u + 10^4 i_u \leq 6 \cdot 10^4 \quad (25)$$
$$-2.10^4 z_1 - v_D \leq 2 \cdot 10^4 \quad (26)$$
$$2.10^4 z_2 + v_D - 10^4 i_D \leq 2.10^4 \quad (27)$$
$$6.10^4 z_2 - v_D + 10^4 i_D \leq 6.10^4 \quad (28)$$
$$2 \cdot 10^4 z_6 + v_u \leq 15 \quad (29)$$
$$2 \cdot 10^4 z_4 - v_u \leq 2 \cdot 10^4 \quad (30)$$
$$v_D \leq 0 \quad (31)$$
$$4z_2 + i_D \leq 4 \quad (32)$$
$$z_5 + u \leq 1 \quad (33)$$
$$-u \leq 0 \quad (34)$$
$$15z_6 + v_u \leq 15 \quad (35)$$
$$z_3 - u \leq 1 \quad (36)$$
$$u \leq 1 \quad (37)$$

V. Computing Variable Bounds

In this section, we present an algorithm that checks if a variable x is bounded and that computes an over- and under-approximation of $\sup(x)$ and $\inf(x)$.

Given a guarded predicate $G(X, Z)$, where Z is the set of guard variables, for any valuation Z^*, $G(X, Z^*)$ is equivalent to a conjunctive predicate (Prop. 7). A naïve algorithm to find bounds for a variable x for any valuation Z^* solves the MILP problems *optimalValue*$(x, \max, G(X, Z^*))$ and *optimalValue*$(x, \min, G(X, Z^*))$. If, *for all* $Z^* \in \mathbb{B}^n$, x is bounded in $G(X, Z^*)$ or $G(X, Z^*)$ is unfeasible, then x is bounded in $G(X, Z)$. Vice versa, if *for some* $Z^* \in \mathbb{B}^n$ $G(X, Z^*)$ is feasible and x is not bounded, then x is not bounded in $G(X, Z)$. Unfortunately, this exhaustive procedure requires to solve $2^{|Z|}$ MILP problems.

The function *computeBounds* in Alg. 3 refines such idea in order to save unnecessary MILP invocations. If all guard literals are positive, if an assignment Z_1^* makes true more guards than an assignment Z_2^*, then the conjunctive predicate $G(X, Z_1^*)$ has more constraints than $G(X, Z_2^*)$ and therefore if x is bounded in $G(X, Z_2^*)$ then it is also bounded in $G(X, Z_1^*)$, and if $G(X, Z_2^*)$ is unfeasible, then also $G(X, Z_1^*)$ is unfeasible (Prop. 7). In the following we establish the correctness of function *computeBounds*.

Proposition 7: Let $Z = z_1, \ldots, z_n$ and let $G(X, Z) = \bigwedge_{i \in [n]}(z_i \to C_i(X))$ be a conjunction of guarded constraints, where head variables occurs positively. Then:

1) For any $Z^* \in \mathbb{B}^n$, $G(X, Z^*)$ is equivalent to the conjunctive predicate $\bigwedge_{j \in J(Z^*)} C_j(X)$.
2) If $Z_1^* \leq Z_2^*$, then $G(X, Z_2^*) \Rightarrow G(X, Z_1^*)$.

Proof: Statement 1 easily follows by observing that a guarded constraint $z \to C(X)$ is trivially satisfied if z is assigned to 0 and it is equivalent to $C(X)$ if z is assigned to 1. Statement 2 follows from the observation that $a \leq b$ implies $J(a) \subseteq J(b)$ and hence $G(X, b)$ has more constraints than $G(X, a)$. ■

Definition 4: We say that a set $C \subseteq \mathbb{B}^n$ is a *cut* if for all $b \in \mathbb{B}^n$ we have $b \leq C$ or $b \geq C$. Let $D(Z)$ be a predicate over a set boolean variables $Z = Z_1 \cup Z_2$ and let $|Z_2| = n$. A cut $C \subseteq \mathbb{B}^n$ is (D, Z_2)-minimal, if for all $c \in C$ $D(Z_1, c)$ is satisfiable, and for all $b < C$ $D(Z_1, b)$ is not satisfiable.

To verify that a variable is bounded $G(X, Z') \wedge D(Z)$, where G is a guarded predicate with positive guards in the set $Z' \subseteq Z$ and $D(Z)$ is a conjunctive predicate, it suffices to check if it is bounded in the conjunctive predicate $G(X, c)$, for all c that belong to a (D, Z')-minimal cut.

Algorithm 3 Computing variable bounds in predicate

```
Input: ⟨$G, D, X, Z', Z, x$⟩ where $G$ is a guarded predicate,
    $Z' \subseteq Z$ set of its guard variables, $x \in X$ a variable,
    $D(Z)$ is a conjunctive predicate over Z
Output: ⟨$b, \inf, \sup$⟩, where $b \in \{B, \neg B, \neg F\}$.
    If $b$ = B, $G(X, Z) \Rightarrow \inf \leq x \leq \sup$
function computeBounds($G, D, X, Z', Z'', x$)
 1. $C \leftarrow \varnothing$, $r \leftarrow |Z'|$, $\inf \leftarrow +\infty$, $\sup \leftarrow -\infty$, $f \leftarrow$ FALSE
 2. $r' \leftarrow$ optimalValue$(\min, \sum_{i \in [r]} z_i, D(Z))$
 3. $r'' \leftarrow$ optimalValue$(\max, \sum_{i \in [r]} z_i, D(Z))$
 4. for $k = r'$ to $r''$ do
 5.     $end$ = TRUE
 6.     for all $b \in \mathbb{B}^r_k$ do
 7.         if $C \not\leq b$ then $end \leftarrow$ FALSE else continue
 8.         if feasible$(D(Z, c))$ then $C \leftarrow C \cup \{b\}$ else continue
 9.         if feasible$(G(X, b))$ then
10.             $f \leftarrow$ TRUE
11.             $M \leftarrow$ optimalValue(max, $x$, $G(X, b)$)
12.             $m \leftarrow$ optimalValue(min, $x$, $G(X, b)$)
13.             if $M = \infty$ or $m = \infty$ then return ⟨¬B, _, _ ⟩
14.             sup ← max(sup, $M$), inf ← min(inf, $m$)
15.     if $end$ then break
16. if $f$ then return ⟨B, inf, sup⟩ else return ⟨¬F, _, _⟩
```

Proposition 8: Let $Q(X, Z) = G(X, Z') \wedge D(Z)$, where G is a guarded predicate such that guard variables in $Z' \subseteq Z$ occur positively and D is a conjunctive predicate. Let C be a (D, Z')-minimal cut and $x \in X$. If, for all $c \in C$, x is bounded in $G(X, c)$ then x is bounded in $Q(X, Z)$.

Proof: Since C is a (D, Z')-minimal cut, any satisfying assignment (X^*, Z^*) to Q is such that $C \leq Z'^*$. As a consequence, there exists $c \in C$ such that $c \leq Z'^*$. Prop. 7.2 implies that $\max\{x \mid G(X, Z^*)\} \leq \max\{x \mid G(X, c)\}$ and $\min\{x \mid G(X, Z^*)\} \geq \min\{x \mid G(X, c)\}$. Therefore if x is bounded in $Q(X, c)$ for any $c \in C$, then it is bounded in $Q(X, Z)$. ∎

Stemming from Proposition 8, function *computeBounds* (Alg. 3) checks if a variable x is bounded in a guarded predicate by finding a minimal cut. To limit the search space, in line 2 (resp. line 3) it is computed the minimum (resp. maximum) number of 1 that a satisfying assignment to the predicate $D(Z)$ must have. The loop in lines 4–16 examines possible assignments to guard variables in Z, keeping the invariant $\forall b < C \neg \textit{feasible} G(X, b) \wedge \forall b \geq C \max\{x \mid G(X, Z)\} \leq \max\{x \mid G(X, b)\} \wedge \min\{x \mid G(X, Z^*)\} \geq \min\{x \mid G(X, b)\}$. In the loop in lines 6–14, if the assignment c under consideration is greater than an assignment in C, no further investigation are needed (by Prop. 8 x is bounded in $Q(X, c)$). If $D(Z \setminus Z', b)$ is unfeasible, the assignment c is not relevant, because $c \leq C$, for any (D, Z')-minimal cut C. Otherwise, c is a relevant assignment and it is added to C (line 8). If x is unbounded in $Q(X, c)$ (lines 11 and 13) we can immediately conclude that x is unbounded in $Q(X, Z)$. Otherwise, we update the approximations computed for $\inf(x)$ and $\sup(x)$ (line 14). If for all assignments in $c \in \mathbb{B}^n_k$ we have $c \geq C$ ($\mathbb{B}^n_k$ is a cut) we are done, C is a (D, Z')-minimal cut, and $\inf$ and $\sup$ computed so far are over-approximation of x bounds in $Q(X, Z)$ (line 15).

Algorithm 4 From predicates to conjunctive predicates

```
Input: $P$ predicate over $X$
Output: $\langle b, C\rangle$, $b \in \{$B, ¬B, ¬F$\}$.
    If $b$ = B, then $C \simeq P$
function PtoC($(P, X)$)
  1. $\langle G, D, Z', Z''\rangle \leftarrow$ PtoG$(P, X)$
  2. $D' \leftarrow$ GtoC$(D, Z' \cup Z'', \langle \mathbf{0}, \mathbf{1}\rangle)$
  3. for all $x \in X$ do
  4.     $\langle \mu, m_x, M_x\rangle \leftarrow$ computeBounds$(G, D', X, Z', Z'', x)$
  5.     if $\mu \neq$ BOUNDED then return $\langle \mu, \varnothing\rangle$
  6. return $\langle \mu,$ GtoC$(G, X \cup Z' \cup Z'', \langle m, M\rangle)$
```

Example 4: In Ex. 3 we assumed bounds for each variable in the DTLHS $\mathcal{H}$ introduced in Example 1. Such bounds has been obtained by fixing bounds for state variables i_L and v_O and for variables v_D and i_D, and by computing bounds for variables i'_L, v'_O, i_u, and v_u using Alg. 3.

The function *PtoC* in Alg. 4 presents the overall procedure that transforms a bounded predicate into an equisatisfiable conjunctive predicate. It calls functions in Algs. 1–3 and the function *GtoC* that performs predicate transformations given in the proof of Prop. 5. As a first step, Alg. 4 translates a predicate $P(X)$ into an equisatisfiable guarded predicate $G(X, Z') \wedge D(Z', Z'')$ by calling the function *PtoG*. Since boolean variables are trivially bounded (bounds are vectors $\mathbf{0} = \langle 0, \ldots, 0\rangle$ and $\mathbf{1} = \langle 1, \ldots, 1\rangle$), the guarded predicate D can be transformed into a conjunctive predicate D' by calling the function *GtoC* on D. To apply function *GtoC* on $G(X, Z')$, we need bounds for each variable in X. These bounds are computed by calling $|X|$ times the function *computeBounds* and are stored in the two arrays m, M. If the function *computeBounds* finds that G' is unfeasible or some x is not bounded in G', the empty constraint is returned together with the failure explanation. Otherwise, the desired conjunctive predicate is returned.

VI. Modeling Issues

The disjunction elimination procedure given in Alg. 4 returns a guarded predicate that may contain a large number of fresh auxiliary boolean variables and this may heavily impact on the effectiveness of control software synthesis or verification. On the other hand, guarded predicates are themselves a natural language to describe DTLHS behavior: assignments to guard variables play a role similar to modes in hybrid systems and, by using negative literals as guards, we can naturally model different kinds of plant behavior according to different commands sent by actuators.

Example 5: Disjunctions 7–9 in Ex. 1 can be replaced by the conjunction of the following (guarded) constraints:

$$q \to v_D = 0 \quad (38) \qquad u \to v_u = 0 \quad (40) \qquad \bar{q} \to v_D = R_{\text{off}} i_D \quad (42)$$
$$q \to i_D \geq 0 \quad (39) \qquad \bar{q} \to v_D \leq 0 \quad (41) \qquad \bar{u} \to v_u = R_{\text{off}} i_u \quad (43)$$

The resulting model for the buck DC-DC converter is much more succinct than the guarded model in Ex. 2 and it has two guard variables only, rather than six as in Ex. 2.

Alg. 3 cannot be directly applied to guarded predicates with both positive and negative guard literals. This obstruction can be easily bypassed, by observing that a guarded constraint $\bar{z} \to C(X)$ can is equisatisfiable to the guarded predicate $(z' \to C(X)) \wedge (z' + z = 1)$. This transformation may double the number of guard variables and hence make the application of Alg. 3 less effective than an exhaustive algorithm on the original model with positive and negative guard literals (see experimental results in Section VII). Summing up, guarded predicates turn out to be a powerful and natural modeling language for describing DTLHS transition relations. We end this section by proposing a syntactic check, that most of the time may be used to compute variable bounds avoiding to use the function *computeBounds*.

Definition 5: A variable x is *explicitly bounded* in a predicate $P(X)$, if $P(X) = B(x) \wedge P'(X)$, where $B(x) = x \leq b \wedge x \geq a$, for some constants a and b.

Proposition 9: Let $\mathcal{H} = (X, U, Y, N)$ be a DTLHS such that each variable $v \in X \cup U \cup Y$ is explicitly bounded in N, and for all $x' \in X'$ there are in N at least two constraints of the form $x' \geq L_1(X, U, Y)$ and $x' \leq L_2(X, U, Y)$. Then N is bounded.

Proof: Since all variables in X, U, and Y are explicitly bounded in N, they are also bounded in N. Therefore $\inf(L_1(X, U, Y))$ and $\sup(L_2(X, U, Y))$ are finite. Since N is guarded, it is a conjunction of guarded constraints and for all $x' \in X'$ it can be written as $x'_1 \geq L_1(X, U, Y) \wedge x'_1 \leq L_2(X, U, Y) \wedge N'(X, U, Y, X')$ for a suitable predicate N'.

This implies $\inf(L_1(X, U, Y)) \leq x' \leq \sup(L_2(X, U, Y))$, which in turn implies that x' is bounded in N. ∎

Example 6: Let $\mathcal{H}_1$ be the DTLHS $(\{x\}, \{u\}, \varnothing, N_1)$, where $N_1(x, u, x') = (0 \leq x \leq 3) \wedge (0 \leq u \leq 1) \wedge (x' = x + 3u)$. By Proposition 9, $\mathcal{H}_1$ is bounded with $\inf(x') = 0$ and $\sup(x') = 6$. All other variables are explicitly bounded in N. Explicit bounds on present state and input variables do not imply that next state variables are bounded. As an example, let us consider the DTLHS $\mathcal{H}_2 = (\{x\}, \{u\}, \varnothing, N_2)$, where $N_2(x, u, x') = (0 \leq x \leq 3) \wedge (0 \leq u \leq 1) \wedge (x' \geq x + 3u)$. Since, for any value of x and u, x' can assume arbitrary large values, we have that $\mathcal{H}_2$ is not bounded.

VII. Experimental Results

In this section, we evaluate the effectiveness of our predicate transformation algorithm *PtoC*. We implemented Alg. 4 in C programming language, using GLPK to solve MILP problems. We present the experimental results obtained by using PtoC on a n-inputs buck DC-DC converter, that we model with three DTLHSs $\mathcal{H}_i = (X_i, U_i, Y_i, N_i)$, with $i \in [3]$, s.t. $X_1 = X_2 = X_3$, $U_1 = U_2 = U_3$, $Y_1 \subset Y_2 \subset Y_3$, N_1 is a *predicate* (Section II-A), N_2 and N_3 are *guarded predicates* (Section IV) and guards in N_3 are positive only.

We then run PtoC on $\mathcal{H}_i$ for increasing values of n (which entails that the number of guards increases), in order to show effectiveness of PtoC. Namely, in Section VII-A1 we show experimental results for the whole algorithm in Alg. 4. Furthermore, in Section VII-A2 we show that exploiting knowledge of the system and modeling it with guarded predicates we obtain better results than those in Section VII-A1. To this aim, we suppose that predicates G, D' and variables sets X, Z', Z'' in Alg. 4 may be directly given as an input to function *PtoC* (thus lines 1 and 2 in Alg. 4 are skipped).

Both in Section VII-A1 and VII-A2 we compare the computation time of function *PtoC* against function *PtoCexh*, which may be obtained from Alg. 4 by replacing the call to function *computeBounds* (our bottleneck here) in line 4 with the naïve algorithm which exhaustively checks all possible assignments to guard variables (see Section V). To this aim, also *PtoCexh* has been implemented inside PtoC. As for *PtoC*, also for *PtoCexh* it is possible to directly specify predicates G, D' and variables sets X, Z', Z''.

A. Multi-Input Buck DC-DC Converter

A Multi-Input Buck DC-DC converter [16] (Figure 2), consists of n power supplies with voltage values $V_1 < \ldots < V_n$, n switches with voltage values $v_1^u, \ldots, v_n^u$ and current values $I_1^u, \ldots, I_n^u$, and n input diodes $D_0, \ldots, D_{n-1}$ with voltage values $v_0^D, \ldots, v_{n-1}^D$ and current values $i_0^D, \ldots, i_{n-1}^D$ (in the following, we will also write v_D for v_0^D and i_D for i_0^D). As for the converter in Ex. 1, the state variables are i_L and v_O, whereas action variables are $u_1, \ldots, u_n$, thus a control software for the n-input buck dc-dc converter has to properly actuate the switches $u_1, \ldots, u_n$. Constant values are the same given in Ex. 3.

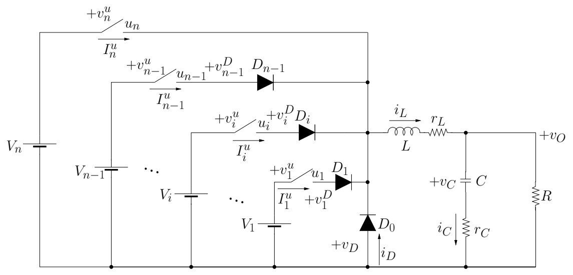


Figure 2. Multi-input Buck DC-DC converter

Table I
PtoC performances (predicates)

n	r	r'	r''	k	$\lvert cut\rvert$	CPU$_r$	CPU$_e$
2	12	6	12	11	64	1.48e+00	1.13e+02
3	18	9	18	17	512	8.33e+01	1.35e+04
4	24	12	24	23	4096	8.73e+03	>1.38e+06

1) Multi-Input Buck as a Predicate: We model the n-input buck DC-DC converter with the DTLHS $\mathcal{H}_1 = (X_1, U_1, Y_1, N_1)$, where $X_1 = i_L, v_O$, $U_1 = u_1, \ldots, u_n$, and $Y_1 = v_D, v_1^D, \ldots, v_{n-1}^D, i_D, I_1^u, \ldots, I_n^u, v_1^u, \ldots, v_n^u$. From a simple circuit analysis (e.g., see [15]), we have that state variables constraints are the same as Eqs. (3) and (4) of the converter in Ex. 1. Analogously, as for the algebraic constraints, we have that Eq. (9) in Ex. 1 also holds for the n-inputs converter. In addition to Eqs. (3), (4) and (9) of Ex. 1, the Eqs. (45)–(48) below must hold.

$$\bigwedge_{i \in [n]} (u_i = 0) \vee (v_i^u = 0) \quad (44) \qquad \bigwedge_{i \in [n]} (u_i = 1) \vee (v_i^u = R_{\text{off}} I_i^u) \quad (45)$$

$$\bigwedge_{i \in [n-1]} ((I_i^u \geq 0) \wedge (v_i^D = 0)) \vee ((I_i^u \leq 0) \wedge (v_i^D = R_{\text{off}} I_i^u)) \quad (46)$$

$$i_L = i_D + \sum_{i=1}^{n} I_i^u \quad (47) \qquad \bigwedge_{i \in [n-1]} v_D = v_i^u + v_i^D - V_i \wedge v_D = v_n^u - V_n \quad (48)$$

N_1 also contains the following explicit bounds: $-4 \leq i_L \leq 4 \wedge -1 \leq v_O \leq 7 \wedge -10^3 \leq i_D \leq 10^3 \wedge \bigwedge_{i=1}^{n} -10^3 \leq I_i^u \leq 10^3 \wedge \bigwedge_{i=1}^{n} -10^7 \leq v_i^u \leq 10^7 \wedge \bigwedge_{i=0}^{n-1} -10^7 \leq v_i^D \leq 10^7$.

We call function *PtoC* with parameters $N_1, X_1 \cup U_1 \cup Y_1$ for increasing values of n, and we compare its computation time with that of function *PtoCexh*. Table I shows our experimental results. In Table I, column n shows the number of buck inputs, column r shows the number of guards (see line 1 of Alg. 3), columns r', r'' have the meaning given in lines 2 and 3 of Alg. 3, column k gives the value of k at the end of the for loop of Alg. 3, column $|cut|$ gives the size of cut at the end of the for loop of Alg. 3, and column CPU$_r$ (resp. CPU$_e$) shows the computation time in seconds of function function *PtoC* (resp. *PtoCexh*). Table I shows that heuristics implemented in function *computeBounds* greatly speeds-up variable bounds computation.

2) Multi-Input Buck as a Guarded Predicate: We modify the DTLHS $\mathcal{H}_1$ of Section VII-A1 by defining $\mathcal{H}_2 = (X_2, U_2, Y_2, N_2)$, where $X_2 = X_1$, $U_2 = U_1$, $Y_2 = Y_1 \cup Y_2' = Y_1 \cup \{q_0, \ldots, q_{n-1}\}$ and N_2 is obtained from N_1 by replacing Eqs. (9) and (45)–(48) with Eqs. (38)–(43) (where $q = q_0$, see Section VI), and by adding the following ones ($i \in [n-1]$):

Table II
PtoC performances (guarded predicates)

n	r	r'	r''	k	$\|cut\|$	CPU$_r$	CPU$_e$
4	16	8	8	8	256	1.17e+01	1.24e+01
5	20	10	10	10	1024	1.55e+02	6.93e+01
6	24	12	12	12	4096	2.65e+03	3.78e+02

$$q_i \to v_i^D = 0 \quad (49) \qquad u_i \to v_i^u = 0 \quad (51) \qquad \bar{q}_i \to v_i^D = R_{\text{off}} I_i^u \quad (53)$$
$$q_i \to I_i^u \geq 0 \quad (50) \qquad \bar{q}_i \to v_i^D \leq 0 \quad (52) \qquad \bar{u}_i \to v_i^u = R_{\text{off}} I_i^u \quad (54)$$

Finally, we define $\mathcal{H}_3 = (X_3, U_3, Y_3, N_3)$, where $X_3 = X_2 = X_1$, $U_3 = U_2 = U_1$ and N_3 is obtained from N_2 by eliminating negative guards as described in Section VI. This introduces $2n$ additional auxiliary variables to manage negations of $q_0, \ldots, q_{n-1}, u_1, \ldots, u_n$, thus $Y_3 = Y_2 \cup Y_3' = Y_2 \cup \{q_0', \ldots, q_{n-1}', u_1', \ldots, u_n'\}$.

For $i = 2, 3$, let constraints in N_i be partitioned in G_i and D_i s.t. G_i contains all guarded constraints in N_i. We call function *PtoC* with parameters $G_3, D_3, X_3 \cup U_3 \cup Y_1, Y_2' \cup Y_3', \varnothing$ for increasing values of n, and we compare its computation time with that of function *PtoCexh* with parameters $G_2, D_2, X_2 \cup U_2 \cup Y_1, Y_2', \varnothing$ Note that G_3 only contains positive-guarded constraints, thus it is possible to call function *PtoC* on it. On the other hand, G_2 also contains negative-guarded constraints, thus it cannot be passed to function *PtoC*, whilst it can be managed by function *PtoCexh*.

Table II shows our experimental results. Columns meaning in Table II are the same as of Table I. Predicate translation on the multi-input buck dc-dc model given as guarded predicate is much faster due to a smaller number of auxiliary variables (and constraints). The negative impact of auxiliary boolean variables is clearly showed by the fact that function *PtoCexh*, much slower than function *PtoC* on a model of the same size, performs better than *PtoC* in this case, because it can work on a model with half of the variables. This phenomenon would be greatly amplified in a verification or control software synthesis procedure. These results strongly support guarded predicates as modeling language.

VIII. Conclusions

The results presented in this paper contribute to Model Based Design of embedded software by proposing an expressive modelling language for discrete time linear hybrid systems. Indeed, MILP based abstraction of a DTLHS have been used to synthesize correct-by-construction control software that implements a quantized controller. They require DTLHS dynamics modeled as a conjunctive predicate over state, input, and next state variables.

In this paper, we circumvented such a limitation, by giving an automatic procedure that transforms any predicate into an equisatisfiable conjunctive predicate, provided that each variable ranges over a bounded interval. Moreover, we have presented an algorithm that, taking a linear predicate P and a variable x, verifies if x is bounded in P, by computing (an over-approximation of) bounds for x.

Finally, our experimental results show the effectiveness of our algorithms. Most notably, they show that guarded predicates may turn out to be a natural language to describe succinctly DTLHS dynamics.

References

[1] T. A. Henzinger and J. Sifakis, "The embedded systems design challenge," in *FM*, ser. LNCS 4085, 2006, pp. 1–15.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3 – 34, 1995.

[3] R. Alur, T. A. Henzinger, and P.-H. Ho, "Automatic symbolic verification of embedded systems," *IEEE Trans. Softw. Eng.*, vol. 22, no. 3, pp. 181–201, 1996.

[4] A. Bemporad and M. Morari, "Verification of hybrid systems via mathematical programming," in *HSCC*, 1999

[5] F. Mari and E. Tronci, "CEGAR based bounded model checking of discrete time hybrid systems," in *HSCC*, 2007

[6] F. Mari, I. Melatti, I. Salvo, and E. Tronci, "Synthesis of quantized feedback control software for discrete time linear hybrid systems," in *CAV*, ser. LNCS 6174, 2010, pp. 180–195.

[7] F. Torrisi and A. Bemporad, "HYSDEL — A tool for generating computational hybrid models for analysis and synthesis problems," *IEEE Transactions on Control System Technology*, vol. 12, no. 2, pp. 235–249, 2004.

[8] S. K. Jha, B. H. Krogh, J. E. Weimer, and E. M. Clarke, "Reachability for linear hybrid automata using iterative relaxation abstraction," in *HSCC*, ser. LNCS 4416, 2007, pp. 287–300.

[9] F. S. Hillier and G. J. Lieberman, *Introduction to operations research*. McGraw-Hill Inc., 2001.

[10] D. Sheridan, "The optimality of a fast cnf conversion and its use with sat," in *SAT*, 2004.

[11] A. Kobetski and M. Fabian, "Scheduling of discrete event systems using mixed integer linear programming," in *Discrete Event Systems*, 2006

[12] W. Kim, M. S. Gupta, G.-Y. Wei, and D. M. Brooks, "Enabling on-chip switching regulators for multi-core processors using current staggering," in *ASGI*, 2007.

[13] W.-C. So, C. Tse, and Y.-S. Lee, "Development of a fuzzy logic controller for dc/dc converters: design, computer simulation, and experimental evaluation," *IEEE Trans. on Power Electronics*, vol. 11, no. 1, pp. 24–32, 1996.

[14] V. Yousefzadeh, A. Babazadeh, B. Ramachandran, E. Alarcon, L. Pao, and D. Maksimovic, "Proximate time-optimal digital control for synchronous buck dc–dc converters," *IEEE Trans. on Pow. El.*, 23(4), 2008

[15] P.-Z. Lin, C.-F. Hsu, and T.-T. Lee, "Type-2 fuzzy logic controller design for buck dc-dc converters," in *FUZZ*, 2005, pp. 365–370.

[16] M. Rodriguez, P. Fernandez-Miaja, A. Rodriguez, and J. Sebastian, "A multiple-input digitally controlled buck converter for envelope tracking applications in radiofrequency power amplifiers," *IEEE Trans on Pow. El.*, 25(2), 2010

ME-DiTV: A Middleware Extension for Digital TV

An Architectural Proposal of A Middleware Extension based on Dynamic Context Changes for Distributed System

Victor Hazin da Rocha[1] [2], Felipe Silva Ferraz[1] [2],
Heitor Nascimento de Souza[1], Carlos André Guimarães Ferraz[2]

[1]CESAR – Recife Center for Advanced Studies and Systems
{vhr,fsf,hns}@cesar.org.br

Informatics Center
[2]Federal University of Pernambuco (UFPE) Recife – PE, Brazil
{vhr,fsf3,cagf}@cin.ufpe.br

Abstract—**This paper aims at providing a trustworthy architecture for a middleware extension, based on geolocalized context information, focused on the development of distributed interactive applications for digital TV. The proposed solution was built using the middleware Ginga. Although it has been implemented for the Brazilian Digital TV System, the architecture described in this paper can be applied to other existing Digital TV middleware with the same benefits. Among those, this work presents a project as a study case to demonstrate the solutions viability and performance analyses on its implementation furthermore this works aims to create an easier way to build distributed, context-sensitive applications.**

Keywords-Digital TV;Distributed System; Middleware .

I. Introduction

Most recent data from Brazilian Institute of Statistics shows that 97,2% of Brazilian homes have a Television Device instead of that only 39,3%[1] residences that have a computer. In this scenario, it is possible to realize that the popularity of the television system plays an important role in integration and distributed solutions..

The TV was not originally designed to provide an infrastructure that enables applications and the challenge is increased when we think about distributed applications, whose development is more complex and requires mastery and expertise by the developers [2].

The web pages or applications are usually available 24 hours a day. This is different from the scenario of television programs, which are transmitted only at predefined times by the broadcaster. Therefore, an interactive application sent by the broadcaster is only available to the viewers during the time in which the program is displayed. Thus, depending on the audience of this program and the attractiveness of the application, the application can have millions of simultaneous accesses, overloading broadcaster servers.

This work´s main purpose is to present a middleware extension that can be compatible with different systems and will make development of distributed application easier.

This paper will first present concepts related to Middleware architecture, followed by the proposition of a Middleware Extension. Next, we present and discuss a case study that uses Brazilian Digital Television infrastructure to create a distributed voting system.

The outline of the rest of the paper is organized as follows. Section 1 gives an introduction to the paper. Section 2 describes some middleware concepts. Section 3 describes a few characteristics of Digital TV. Section 4 illustrates the architecture of the proposed solution. Section 5 presents the study case and Finally, Section 6 finishes the paper by explaining a couple of conclusions.

II. Middleware

Distributed systems create new problems that do not exist in centralized systems, like connections problem or network saturation [2]. The question is how to facilitate the development or implementation of distributed applications in such a way that is possible to solve additional problems created by the distribution itself.

In principle, there are different options - from hardware support level to the extension of programming languages to enable support of distributed applications. Software solutions typically provide flexibility because of their suitability for integrating existing technologies (such as operating systems and programming languages). These conditions lead to the concept of Middleware.

Middleware offers general services to support distributed applications execution. The term Middleware suggests that it is software situated between the operating system and the application. Viewing abstractly, Middleware can be envisaged as a “tablecloth” that spreads itself over a heterogeneous network, abstracting the complexity of the underlying technology from the application using it [3].

There are several ways to categorize Middleware. In this paper, we will use the four main types of Middleware found in the literature. These are: transactional, procedural, message-oriented and object-oriented middleware [4].

A. Transactional Middleware

Transactional Middleware supports transactions involving components that run on distributed hosts. This kind of Middleware was designed in order to support distributed synchronous transactions. It should be used when transactions need to be coordinated and synchronized over multiple databases [4].

B. Procedural Middleware

Remote Procedure Calls (RPCs) were designed by Sun Microsystems in the early 1980s as part of the Open Network Computing (ONC) platform. Sun provided remote procedure calls as part of all their operating systems and submitted RPCs as a standard to the X/Open consortium, which adopted it as part of the Distributed Computing Environment (DCE) [5]. RPCs are now available on most Unix implementations and also on Microsoft's Windows operating systems.

According to Pinus [4], RPCs could be used in small, simple applications with primarily point-to-point communication. RPCs are not a good choice to use as the building blocks for enterprise-wide applications where high performance and high reliability are needed.

C. Message-oriented Middeware

Message-oriented middleware (MOM) bear the communication between distributed system components by facilitating message exchange. According to Pinus [4], there are two different types of MOM: message queuing and message passing.

Message queuing is defined as indirect communication model, where communication happens through a queue. A message from one program is sent to a specific queue, identified by name. After the message is stored in this infrastructure, it will be sent to a receiver.

In message passing - a direct communication model - the information is sent to the interested parties. One flavor of message passing is publish-subscribe (pub/sub) middleware model. In pub/sub clients have the ability to subscribe to the interested subjects. After subscribing, the client will receive any message corresponding to a subscribed topic. MOM should be used in the applications where the network or all-components availability is not trustable [4].

D. Object-oriented Middleware

Object-oriented Middleware (OOM), evolved from RPCs, extends them by adding object-oriented concepts. These concepts are: inheritance, object references and exceptions. OOM allows referencing of remote objects and can call operations on them. OOM should be considered for applications where immediate scalability requirements are somewhat limited. These applications should be part of a long-term strategy towards object orientation [4].

III. Digital TV

Digital TV is popular because of the the quality of the image provided by the broadcaster. However, this concept is minimalist. There are three deep concepts of Digital TV: Interactivity, Portability and Connectivity; these concepts, supported by software definitions, are the core of Digital TV [6].

The interactivity and connectivity allows digital TV viewers to submit content and to get a reaction from it. This means it is possible for the viewer to interact with a particular broadcast content [7].

A. Middleware Ginga

Ginga is the name of the middleware specification for the Nipo-Brazilian Digital TV System (SBTVD, from the Portuguese *Sistema Brasileiro de TV Digital*). It consists of a set of standard technologies and innovations which make the most advanced middleware specification and the best solution for the brazilians requirements [6].

The middleware is divided into two main integrated subsystems, which allow the development of applications following two different programming paradigms. Those subsystems are called Ginga-NCL (for declarative NCL applications) and Ginga-J (for imperative Java applications). The use of any of these two paradigms depends on the requirements of each application [6].

In addition to making it possible to send applications to compatible TVs, Ginga provides information about content transmitted to the receiver through a set of tables, called SI (*Service Information*). Among the tables that compose this group we highlight the EIT (*Event Information Table*) and NIT (*Network Information Table*). The EIT is responsible for delivering information related to the program schedule, while the NIT contains information about the network that the content is being made[8].

Ginga-J was chosen to be used in this article because of the support to the network layer of the Ginga middleware.

1) Ginga-J

Ginga-J is designed to provide an infrastructure for the implementation of applications based on Java language, with features aimed specifically for the digital TV environment [9].

Ginga-J, as the name suggests, supports Java procedural language. According to [10] *" it is the logical subsystem of the Ginga middleware responsible for processing imperative applications written using the Java language*".

IV. Architecture

This section aims at describing the architecture of the solution proposed by this work.

The first important project decision was the choice for building a message-oriented middleware. This choice was made because MOM systems can provide distributed communication on the basis of asynchronous interaction model allowing the system to continue processing once a message has been sent [11].

Figure 1 shows the macro architecture for the implemented solution. The middleware was divided into three separate layers, which will be detailed below.

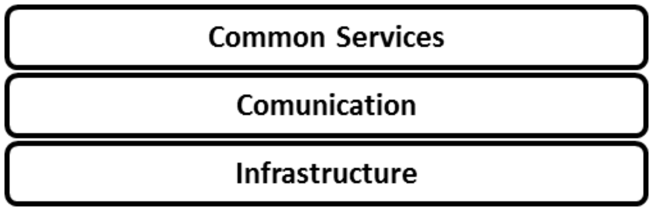


Figure 1. Macro Architecture

The described architecture is compatible with existing middleware such as MHP [12] and Ginga [8]. The goal is to facilitate the creation of distributed interactive applications for Digital TV which can have millions of simultaneous accesses, causing an application to work as a distributed system, dividing the access to the servers based on the context of the device responsible for TVs connection, called as set-top box.

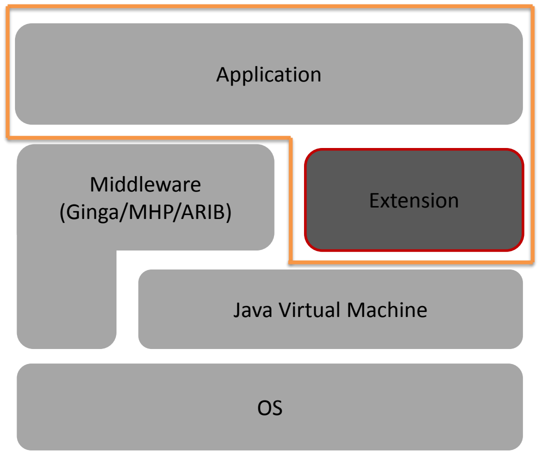


Figure 2. Potential Middleware extension.

To make possible for the extension to be used by existing middleware, the module presented in this paper was developed in Java. This choice was made because Java is the language used on the main Digital TV middlewares, such as MHP, ARIB (Association of Radio Industries and Businesses) and Ginga-J. Despite the fact that the extension is not be incorporated into any TV middleware, it can still have access to Middleware Features since it is presented in the same level as other applications.. Figure 2 shows how the extension is positioned connecting the Middleware and the application, considering as basis a mixed architecture of the MHP/ARIB/Ginga-J. The module presented in this paper should include together with the application as is highlighted in the Figure 2, this is required for not be necessary to change the existing middleware implementations.

For a possible adoption of this extension we choose to use Ginga middleware. Figure 3 shows the usage scenario of the proposed extension as part of a bigger structure.

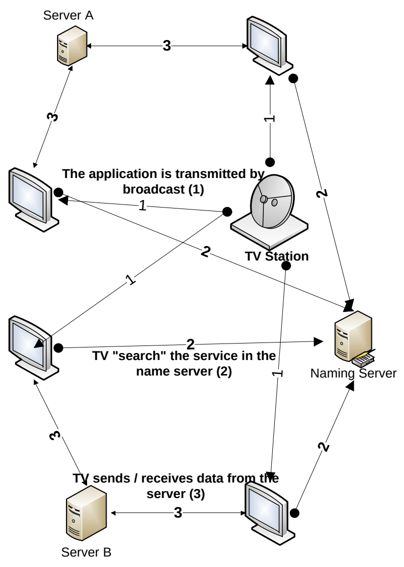

Server A
3
3
1
2
The application is transmitted by broadcast (1)
1
2
TV Station
1
2
TV "search" the service in the name server (2)
1
Naming Server
3
2
TV sends / receives data from the server (3)
3
Server B

Figure 3. Usage scenario

The first step is to build an application that makes use of the feature of distribution based on location provided by the extension; this application must be registered in the naming service and sent by the broadcaster to viewers via broadcast (1). When the application is received by the TV middleware, a query is made to the naming service (2) to discover what is the most appropriate server based on the location to perform information exchanges. When the application receives the reply of the naming service, it can finally exchange information via messages with the broadcaster server (3).

A. *Infrastructure*

Figure 4 shows the class diagram of this layer.

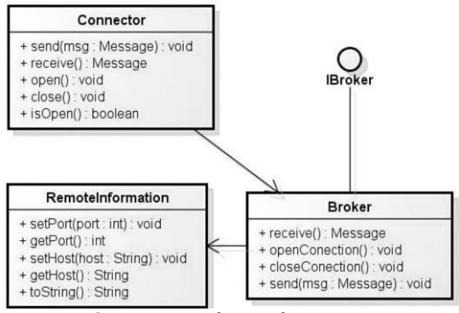


Figure 4. Class Diagram from Infrastructure Layer

This layer is responsible for sending messages to the network layer within the operating system, making transparent communication between processes and applications that uses the middleware and hide the use of sockets from the layers above.

Classes and methods of communication layer cannot be called directly by the developer. It is only used by others layers to send messages over the network.

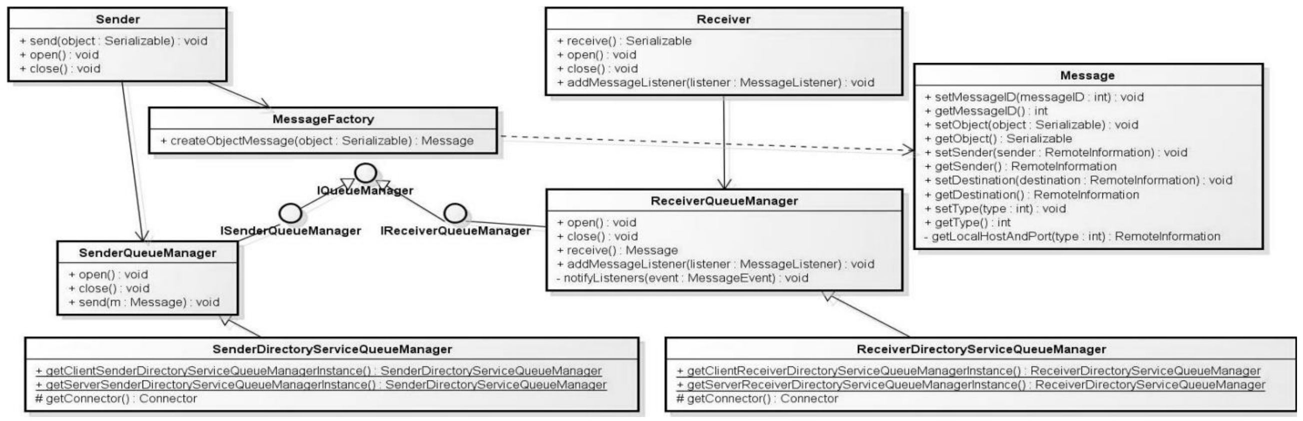

Sender
+ send(object : Serializable) : void
+ open() : void
+ close() : void
Receiver
+ receive() : Serializable
+ open() : void
+ close() : void
+ addMessageListener(listener : MessageListener) : void
Message
+ setMessageID(messageID : int) : void
+ getMessageID() : int
+ setObject(object : Serializable) : void
+ getObject() : Serializable
+ setSender(sender : RemoteInformation) : void
+ getSender() : RemoteInformation
+ setDestination(destination : RemoteInformation) : void
+ getDestination() : RemoteInformation
+ setType(type : int) : void
+ getType() : int
- getLocalHostAndPort(type : int) : RemoteInformation
MessageFactory
+ createObjectMessage(object : Serializable) : Message
IQueueManager
ISenderQueueManager
IReceiverQueueManager
ReceiverQueueManager
+ open() : void
+ close() : void
+ receive() : Message
+ addMessageListener(listener : MessageListener) : void
- notifyListeners(event : MessageEvent) : void
SenderQueueManager
+ open() : void
+ close() : void
+ send(m : Message) : void
SenderDirectoryServiceQueueManager
+ getClientSenderDirectoryServiceQueueManagerInstance() : SenderDirectoryServiceQueueManager
+ getServerSenderDirectoryServiceQueueManagerInstance() : SenderDirectoryServiceQueueManager
# getConnector() : Connector
ReceiverDirectoryServiceQueueManager
+ getClientReceiverDirectoryServiceQueueManagerInstance() : ReceiverDirectoryServiceQueueManager
+ getServerReceiverDirectoryServiceQueueManagerInstance() : ReceiverDirectoryServiceQueueManager
# getConnector() : Connector

Figure 5. Class Diagram from Communication Layer

B. *Communication*

The communication layer is the responsible for the creation of each message that will be send through the network and for the creation and management of queues.

This layer also makes abstract the sending of messages to the application. Figure 5 has the diagram that represents this layer.

Only the Sender and Receiver classes can be called by the application developer, providing the mechanisms of transparency of communication.

C. *Common Services*

This layer is responsible for providing naming service and the location transparency. Furthermore, others services could be provided such as, security service.

The services of this layer are available for use by both the middleware and the application. The class diagram that best describes the architecture can be seen in Figure 6.

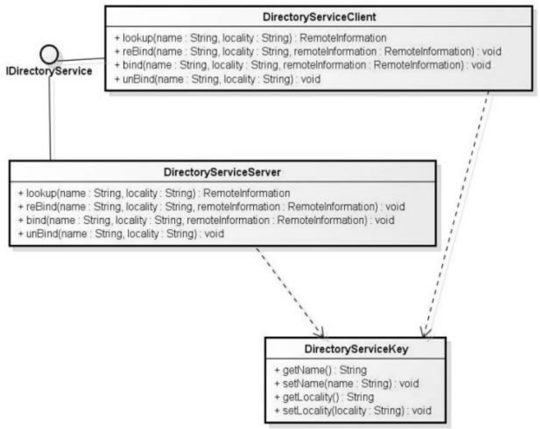

DirectoryServiceClient
+ lookup(name : String, locality : String) : RemoteInformation
+ reBind(name : String, locality : String, remoteInformation : RemoteInformation) : void
+ bind(name : String, locality : String, remoteInformation : RemoteInformation) : void
+ unBind(name : String, locality : String) : void
IDirectoryService
DirectoryServiceServer
+ lookup(name : String, locality : String) : RemoteInformation
+ reBind(name : String, locality : String, remoteInformation : RemoteInformation) : void
+ bind(name : String, locality : String, remoteInformation : RemoteInformation) : void
+ unBind(name : String, locality : String) : void
DirectoryServiceKey
+ getName() : String
+ setName(name : String) : void
+ getLocality() : String
+ setLocality(locality : String) : void

Figure 6. Class Diagram from Common Services Layer

The service name of the proposed extension provides four methods for application developers: *bind*, *reBind unBind*, *lookUp*. The bind/rebind/unbind methods are responsible to register/deregister a server/service in the naming service, together with that, lookup method is responsible for naming service and find server/service address using the service name and the information about who is transmitting the application, obtained from the network. Since the information contained in the network is part of the context of the set-top box (or the device), we assume the naming service provided by the solution is context-sensitive

V. STUDY CASE

To validate the architecture proposed in this paper, we implemented a version for an extension compatible with the Ginga middleware, using the Java 1.3. This is the Java version compatible with Ginga [13]. Ginga was chosen because it is the middleware of the Brazilian Digital TV.

The current implementation includes all the features that were described in the architecture section, it contains a total of 24 classes. This implementation contains more classes that were explained in Figure 4, Figure 5 and Figure 6, because some helper classes were created.

A. *Voting System*

To better evaluate the architecture two identical applications were developed. The difference between them is restricted to how they send objects across the network. The App01 is the application that uses the middleware extension built in this work, while App02 does not use the extension.

The application chosen to be developed was a voting system for reality shows. The system receives a vote given by the user / viewer through some input device, in the present case we use remote control and since the input is received, the Middleware takes care of sending to the broadcast server.

As the focus of this work is to facilitate the development of applications making transparent the communication layer, the GUI was not implemented.

The application works by pressing one of the input keys in the remote control. The information of which button was pressed is detected and then sent to the server via message.

```
String vote = "participant01";
Sender.getInstance().open();
Sender.getInstance().send(vote);
Sender.getInstance().close();
```

Figure 7. App01 Code

In Figure 7, we can see the code of App01, which is responsible for sending the vote of set-top-box/television to the server, this application uses the module constructed in this work.

Figure 8 shows the code of App02 that is responsible for doing the submission of the votes. Note that in App02 the server address should be passed with the application, so their location cannot be changed dynamically. In App01, only the parameters defined in this paper must be sent together with the application, and the server location could change dynamically.

```
Socket clientSocket;
clientSocket = new Socket(Constants.APPLICATION_HOST,
Constants.APPLICATION_PORT);

ObjectOutputStream outToServer;
outToServer = new ObjectOutputStream(
                clientSocket.getOutputStream());

String vote = "participant01";
outToServer.writeObject(vote);
outToServer.flush();
clientSocket.close();
```

Figure 8. App02 Code

Looking at the code responsible for the communication of the two applications, one can observe that the code of App01 is much simpler and transparent than the App02. Figure 9 shows the server code responsible for receiving the votes from the App01, while Figure 10 contains the server code of App02.

```
RemoteInformation ri = new
RemoteInformation(Constants.APPLICATION_PORT,
Constants.APPLICATION_HOST);

DirectoryServiceClient.getInstance().reBind(Constants.S
ERVICE_NAME,Constants.LOCALITY, ri);

MainReceiver r = new MainReceiver();
r.init();
public MainReceiver() {
    receiver = new Receiver();
}
public void init() throws InterruptedException {
receiver.open();
receiver.addMessageListener(new MessageListener() {
    public void onMessageReceived(MessageEvent event)
     {
          countVotes(receiver.receive());
     }
  });
}
```

Figure 9. App01 Server Code

Differently from clients, the server of App01 has more code lines than the App02 server. This happens because the middleware extension proposed in this paper enables transparent error handling, and offers a names service, allowing the server to changes its IP address dynamically.

```
int port = Constants.APPLICATION_PORT;
welcomeSocket = new ServerSocket(port);
Socket connectionSocket = welcomeSocket.accept();
ObjectInputStream input = new ObjectInputStream(
connectionSocket.getInputStream());
countVotes(input.readObject());
connectionSocket.close();
welcomeSocket.close();
```

Figure 10. App02 Server Code

In the next subsection, we will present an experiment to evaluate the performance of the applications built here to validate the proposed extension.

B. *Validation and Results*

To analyze the middleware impact, tests were executed to measure the performance and reliability of the two applications. The tests were made in a laboratory of Digital TV with a television embedded with Ginga and a Playout EITV, which is a complete TV broadcast station that can perform transmissions containing TV programs in high definition and interactive content [14]. The configuration of the testing environment is illustrated in Figure 11.

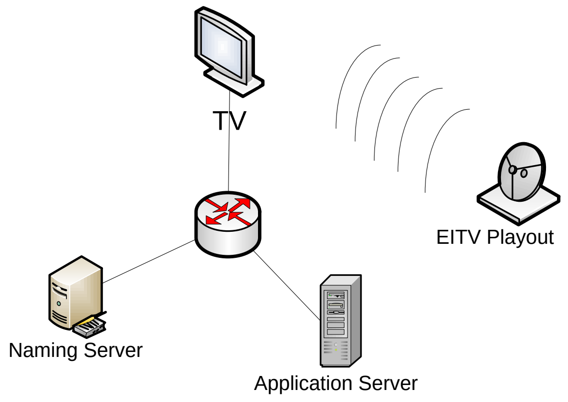


Figure 11. Test Environment

The test was done as follows: for each application, we added a function to send 100 votes consecutively when one of the colored keys on the remote control was pressed.

The modified applications were transmitted one at a time, to the TV using the EITV Playout. For each application, the time between the arrival of the first and last on the server was measured. Each vote was sent separately, a new connection was opened to send the vote, after send it, the connection was closed. The experiment was done to simulate an environment where a user wants to vote more than once. The experiment was repeated five times only due to the stability of the local network of the test environment, and the results are shown in Table I.

TABLE I – TIME IN SECONDS BETWEEN THE ARRIVAL OF THE FIRST AND LAST VOTE ON THE SERVERS

Id	**Time(s) to complete send action on App01**	**Time(s) to complete send action on App02**
1º	32,57	31,17
2º	31,99	31,48
3º	32,49	31,58
4º	32,26	31,73
5º	32,19	31,68
Avg Function	32,30	31,53

Analyzing the results presented in Table I, we can realize that App01 had a delay of less them 2,4% comparing to App02. App01 shows its potential even losing in performance, by using the localization and communication transparency provided by this work proposal.

VI. Conclusion

This work proposed an extension that provides a context-sensitive feature to applications, in a way to make easier and more abstract communication for adopted implementations. After a brief explanation about the architecture, a proof of concept was developed and validated using performance tests.

Even though the study case presented was developed for Brazilian Digital Television Middleware, the proposed solution can be adopted in different Middleware, or as a solo API.

Through the analysis of the results, it can be seen that the extension can decrease the performance in less than 3%, but shows its power by creating an easier way to build distributed, context-sensitive applications. Furthermore, it guarantees a more dynamically and network-error free environment since it abstract those scenarios.

Acknowledgment

This work is supported by the CAPES (process 23038-023577/2008-23, AUX-PE-RH-TVD 385/2008).

References

[1] IBGE, *Síntese de indicadores sociais : uma análise das condiçoes de vida da populaçao brasileira*. IBGE, 2010.

[2] A. S. Tanenbaum and M. V. Steen, *Distributed Systems - Principles and Paradigms*. Prentice Hall, 2002,.

[3] A. Puder, K. Römer, and F. Pilhofer, *Distributed Systems Architecture: A Middleware Approach*. Morgan Kaufmann, 2005.

[4] H. Pinus, “Middleware: Past and present a comparison” , 2004. [Online]. Available: http://www.research.umbc.edu/~dgorin1/451/middleware/middleware.pdf. [Accessed: 20-Sep-2012].

[5] The Open Group, “DCE 1.1: Remote Procedure Call.” [Online]. Available: http://www.opengroup.org/public/pubs/catalog/c706.htm. [Accessed: 20-Sep-2012].

[6] “Ginga Digital TV Middleware Specification,” 2012. [Online]. Available: http://www.ginga.org. [Accessed: 10-Sep-2012].

[7] L. Cosentino, “Software: a essência da TV digital,” in *TV Digital Qualidade e Interatividade*, Brasília: IEL/NC, 2007, pp. 41–49.

[8] “ABNT NBR 15603-1:2007. Televisão digital terrestre - Multiplexação e serviços de informação (SI) - Parte 1: Serviços de informação do sistema de radiodifusão.” .

[9] “Site Oficial da TV Digital Brasileira,” 2012. [Online]. Available: http://dtv.org.br. [Accessed: 10-Sep-2012].

[10] L. F. Soares, “Ambiente para desenvolvimento de aplicações declarativas para a TV digital brasileira,” in *TV Digital Qualidade e Interatividade*, Brasília: IEL/NC, 2007, pp. 51–62.

[11] E. Curry, Message-Oriented Middleware, in Middleware for Communications (ed. Q. H. Mahmoud), John Wiley & Sons, Ltd, Chichester, 2004.

[12] “MHP,” 2012. [Online]. Available: http://www.mhp.org/. [Accessed: 05-Sep-2012].

[13] “ABNT NBR 15606-6: Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 6: Java DTV 1.3.” 2010.

[14] “EITV Playout - Estação completa de TV digital interativa para os padrões SBTVD, ISDB e DVB,” 2012. [Online]. Available: http://www.eitv.com.br/playout.php. [Accessed: 15-Sep-2012].

The Dynamic Composition of Independent Adaptations including Interferences Management

Sana Fathallah Ben Abdenneji, Stéphane Lavirotte, Jean-Yves Tigli, Gaëtan Rey and Michel Riveill
Laboratory I3S (University of Nice-Sophia Antipolis / CNRS)
B.P. 145 06903 Sophia-Antipolis Cedex - France
{fathalla, stephane.lavirotte, tigli, gaetan.rey, riveill}@unice.fr

***Abstract*—Nowadays, software systems are becoming ever more complex and are likely to evolve quickly. A challenge is being able to adapt these systems and thus to integrate, swap or remove new functionalities. Compositional adaptation provides a way to tackle this at runtime. This situation, leads to satisfy new requirements in software adaptation. Moreover, adaptations entities may be developed independently and even can be specified by different designers. Accordingly, the set of all adaptations that may be deployed cannot be anticipated at design time. In such case, they may interfere when they are composed. In this paper, we propose an associative composition operation to support software adaptation. In our research, we consider that the adaptation of a running system is assimilated to the composition of the model of the initial application (called initial graph) with new model elements (graphs of adaptations).**

***Index Terms*—software composition; adaptive software; interference management; graph transformation.**

I. Introduction

Nowadays, software systems must be developed with the capacity of being able to evolve quickly [1]. Software systems must react to environments changes and consequently integrate, swap or remove new functionalities. The compositional adaptation approach [2] allows to change software units. In compositional adaptation, the applications must be based on a modular architecture with a loose coupling between the software units composing it. The loose coupling between component and its execution environment control facilitates the dynamic reconfiguration of components assembly. Accordingly, Component Based Software Engineering is well-suited for compositional adaptation [3]. We will use throughout the paper the more general term *adaptation* or *adaptation entities* to point out the modification that will be done on the application.

In this paper, we present an adaptation mechanism that allows the *dynamic* composition of adaptation entities. In order to support such composition, existing approaches consist either in defining *explicitly* dependencies between adaptation entities or either in calculating all possible combinations between them in order to choose the most appropriate [4]. Since the result of the composition may depend on the order in which adaptations are made, the number of combinations to be calculated from n adaptations may be $n!$. In this paper we propose an associative and a commutative composition mechanism that allows to minimize the number of combination by construction. The application resulting from adaptations composition will be always the same, regardless of their order of composition. Accordingly, there is no need for a designer to explicit the order when deploying adaptation entities or to calculate combinations. Using this mechanism, adaptations are added or removed independently of each other. More specifically, all resulting **interferences** (*interactions*) between adaptations entities are handled automatically at the composition step. Interference is defined as *"a conflicting situation where one adaptation that works correctly in isolation does not work correctly anymore when it is composed with other adaptation"* [5]. These interferences should be managed in order to ensure the consistency of the application after the adaptation process. Moreover, since adaptation mechanism will modify only the structure of the application, we model software applications and adaptations entities by graphs. The main reason to choose graph as a basis formalism is that it can define software architecture easily as shown in [6]. The interference resolution step will operate on the graph by applying a set of graph transformation rules.

The remainder of this paper is organized as follows: next section presents some related works. Then, Section III introduces the model of our applications and details the example that will be used all along the paper to illustrate our approach. In Section IV, we detail formally our composition process and how it addresses the issue of interference management. Particularly, we show the associativity property that allows us to offer a deterministic solution when we compose adaptation entities. Then, we describe our implementation showing some experimental results (Section V). Finally, we conclude in Section VI on the contribution of this work and its perspectives.

II. Related Work

Systems adaptation can be triggered by several causes. Researchers have proposed a variety of methodologies for the development of self-adaptive systems. This objective can be achieved using reflective architecture-based mechanism [7]. To fulfill software adaptation objective, there is several representation of adaptations. As a consequence, the process of the detection and the resolution of interference are related to the used method. In this section, we mention some of works that have contributed to tackle the interference problem, all other related works are out of the scope of this paper. This problem is considered as the violation of the constraints defined at design time of the system [7]. A system change is valid only if the system satisfies the constraints after the change. Otherwise, the change will be canceled. *Barresi et al.* [8] define explicitly at design time the order in which adaptation should be applied. This requires the knowledge of all system adaptation at design time. But we have seen that we need unanticipated adaptation entities that will be performed at runtime and that can be loaded and unloaded at runtime. This requirement has been included in K-Component Model [9]. *Dowling et al.* use graph transformation rules to separate the adaptation code from the computational code and to perform architecture adaptation. Furthermore, they are not interested in the problem of interference. Similarly, *Guennum et al.* [10] use graph transformation rules to perform software adaptation and propose tree operators to compose rules. These operators allow defining an order between rules or select only one rule to be executed. These operators need to be defined at design time and as a consequence require the knowledge of all possible adaptations (contradictory with our constraint).

Another related area of research that relies on software adaptation is the use of AOP (Aspect Oriented Programming). *David et al.* [11] consider that software adaptation is a crosscutting concern of the application and use aspects to encapsulate adaptations code. In this area, the problem of interference was well defined and several solutions were proposed. Greenwood et al. [3] investigate a solution to interferences in the context of AO-Middleware platform. To do that, they define *interaction contract* which is used at runtime. These contracts express several strategies to resolve interferences such as priority, precedence and logical operators (to combine contracts). Despite the use of these contracts at runtime, their specifications are made by the developer who should include all dependent relationships between the adaptations. This will be a complex task because we consider multi designers approach. Moreover, the strategy of interference resolution may depend on the runtime state of application. *Dinkelaker et al.* [12] define an extensible ordering mechanism which can be modified at runtime. Such approach still suffers limitation in term of software adaptation because interference management at runtime needs to be anticipated. In that direction, *Cheung et al.* [13] propose a composition mechanism that repair interference problem in an anticipate manner. Since interference can occurs at input and/or output of components, they propose two composition mechanisms, based on two different languages for adaptations, to handle these problems separately. But, we have shown previously that these problems need to be resolved together and not separately. Their solution can resolve interference either for the output of component or for the input of component but not for both interferences types.

Graphs is not only intuitive representation of software architecture, but are also used to identify errors on the analysis level. The integration of the paradigm of graph with the aspect-oriented paradigm has been proposed by *Cirarci et al.* [14]. They use the graph formalism to identify interference. Graphs represent the several states of a program, according to different order of aspect weaving. Interference is detected if the final state changes according to the selected order.

Although all of previously described approaches propose software adaptations mechanism that support interferences detection and resolution, none considers a merge process for computing reasonable system adaptation from a large set of possible adaptation, which is the main focus in this article.

III. Preliminaries

To be able to present our composition mechanism, we introduce in this section the model of our software systems.

A. Modelling Software architecture Assemblies Using Graphs

In order to represent structural specifications of software systems, several researchers use Business Process Model (BPM), which embodies also the strategies for accomplishing software evolution. It provides high-level specification that is independent of the target platform. There exist many notations to represent BPM [15]. In this paper, we abstract from any specific notation and represent a process model as a directed graph as per in the definition 1.

Definition 1. *A process graph G is a set of vertices V and a set of directed edges E $(G = (V, E))$. A vertex v_i of V is defined by a tuple $(Id(v_i), Typ(v_i))$ where $Id(v_i)$ is the identifier of v_i and the attribute $Typ(v_i)$ is its type. An edge e_j of E is written as $e_j = (v_i, v_k, l_j)$ where l_j is a label.*

definition 2 (Successor and Predecessor). *Let $G = (V, E)$ be a directed graph. For each vertex $v_i \in V$: we define the set of its predecessor vertices as $[v_{0i}, ..., v_{li}]\bullet v_i$ where $\{\forall v_{li} \in V, (v_i, v_{li}) \in E\}$, and the set of its successors (or output) vertices $v_i\bullet[v_{0i}, ..., v_{mi}]$ where $\{\forall v_{mi} \in V, (v_{mi}, v_i) \in E\}$.*

Each vertex has a type and depends on the target platform language that we represent. In the remainder of this paper we consider that applications are created as component assemblies (component can be instantiated when a device appears and destroyed when the device disappears). Vertices are classified into two subclasses: (i) *Black box* vertices ($Typ(v)$='*Port*' and $id(v)$='*ComponentName:PortName*'), representing component ports (event and method call); (ii) *White box* vertices (or *connectors*), which determine the flow of the execution when events are triggered. The attribute $Typ(v)$ of a white box vertex indicates the kind of connectors. We introduce **5** basic connectors: *PAR, SEQ, IF, CALL, DELEGATE.* $PAR\bullet[v_i, v_j]$ connector performs the concurrent execution of vertices v_i and v_j. $SEQ\bullet[v_i, v_j]$ defines the order of execution; v_i before v_j. $IF\bullet[v_c, v_i, v_j]$ is used to choose a path between v_i and v_j. If the vertex v_c has the value true, v_i will be performed (or selected) otherwise v_j will be executed. $CALL$ connector allows rewriting an existing edge. The connector $DELEGATE\bullet[v_i]$ specifies that the link to the vertex v_i will be unique in case of interference. $CALL$ and $DELEGATE$ are special connectors that will never be instantiated in the final application because they are used only to modify some links. Figure 1 shows an example of the application according to our model. In order to facilitate the comprehensibility of this paper, we use a lightweight representation. Each black box component will be represented by a rectangle on which we add a label

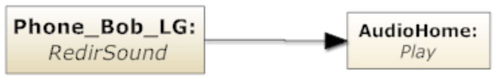


Figure 1. An example of component assembly graph

on the form $ComponentinstanceName : PortName$. The white box vertex will be represented by a rectangle on which we add a label on the form : $ConnectorName$.

B. Running Example

We will now present the scenario that will be used throughout this paper to present our approach. As an example, we define a ubiquitous application that relies on variable and communicating devices which define its software infrastructure. This software infrastructure appears dynamically populated by the functionalities of these devices. As a consequence, the application has to be adapted during execution time in order to consider these changes. *As all young people, Bob listens to music all the time on his Smartphone. Inside his home, he has an adaptation that redirect the sound from his phone to any available audio device (Home cinema, speaker, etc.). Bob lives alone. Today his mother visits him. Bob's mother had recently hearing problems; she cannot withstand the high tones. For this reason, she has an adaptation (reconfiguration) that will specify the threshold of the sound level for audio devices in her surrounding physical environment. Bob doesn't know that his mother had hearing problems. He increases the level of music.*

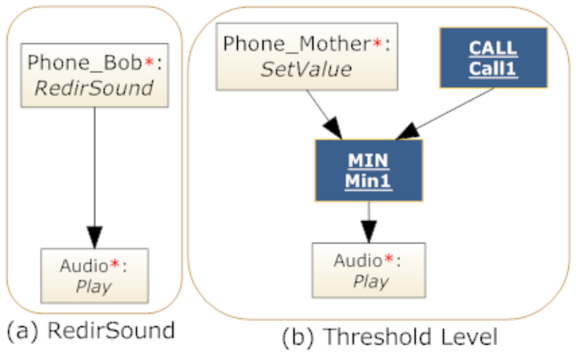


Figure 2. The graphs of adaptations defined in the running example

We consider that the original application (called the base assembly) is build from a physical audio device, which is represented by a Blackbox component *AudioHome* at software level. In our use case, there are two sub-assemblies (adaptations) linked to the *AudioHome* device. The *SoundRedirect* adaptation graph is depicted in Figure 2. In order to be applied, this adaptation requires that there are instances of *PhoneBob** and *Audio** components in the initial assembly (for example, if no component with audio capability is found, this adaptation will not be woven at all). A simple wildcard '*' is used in this example, but more complex regular expression can be defined (Component instances' types and component ports' names can both be wild-carded). This wildcard will be replaced by the real device type of the infrastructure. It allows these adaptations to be applied to real application although its configuration is not completely known at design time. This adaptation adds a new edge between the port *RedirSound* of the component *PhoneBob** and the port *Play* of the component *Audio**. Through this link the *Audio** device will receive the music to play. Figure 1 shows an example of instance graph of this adaptation.

The *Threshold Level* adaptation is depicted in Figure 2. The adaptation that is not relevant to the application at the beginning, can become relevant only when its required components appear in the application assembly (components tagged in Figure by a '*'). This adaptation specifies that when an Audio device is detected, the user can define using his phone the threshold for the sound level. The Min vertex is a new connector, which will be defined in the next section. It is used to forward the minimal received values from its input vertices. The *CALL* vertex allows rewriting existing links.

In the final system both adaptations are running in parallel. Even though the adaptation of Bob's mother has delineated the threshold, when Bob will increase the sound, this threshold will be exceeded. This example illustrates a problem of a concurrent access to a shared resource (audio device). In this case, if we apply classical resolution approaches, such as a precedence strategy, the problem will not be resolved. If we apply the *Threshold Level* adaptation before or after the *SoundRedirect* adaptation the problem will persist.

IV. A GRAPH-BASED APPROACH FOR ADAPTATIONS COMPOSITION

In this section, we detail our approach for interference resolution and the proof of the symmetry property.

A. Approach Overview

Until now, we have presented the adaptation entities graphs. Since each adaptation entity is based on a set of required components, we need to determine the set of relevant adaptations (adaptations that can be applied). After that, we transform the abstract description of relevant adaptations into a concrete one (replacing '*' wildcard by the type of real components in the base assembly). If various components satisfy the requirements of an adaptation, then it can be applied as many times as there are combinations of components instances. The *combination function* computes all the places in the assembly where adaptations can be applied according to different strategies (build all possible combinations, combine according to components names, etc.). Subsequently, all these graphs need to be composed with the graph of the base assembly. The **composition** mechanism considers then these graphs, in order to generate a single graph representing the adapted assembly. Despite the order of graphs composition the final graph will be the same due to the **commutativity** and the **associativity properties**. There are two sub steps in the composition process: the first sub step *superimposes* all graphs and also identifies potential interferences. The second sub step is accomplished by the

Merging engine, which resolves these problems using *graph transformation rules*. The final graph that represents the new configuration of the system, will be exported to the adaptive execution platform.

The interference problem is defined through two patterns. The interference Type 1 is detected when there is a Black Box vertex that has two or more outgoing edges. The interference Type 2 is denied as a Black Box vertex that has at least two incoming edges. In our previous work [16], we have considered interference Type 1. In this paper, we introduce new *White box* vertices and we focus on a new type of interference that occurs when several adaptation try to access to a shared component's ports (Type 2).

B. Superimposition and Interference Detection process

The superimposition operation builds a unique graph $G_T = (V_T, E_T)$ from the graph representing the initial assembly $G_{initial}$ and several graphs resulting from the *adaptation instantiation* step (G_{Adap1},...,G_{AdapN}). $G_T = (((G_{Base} \cup G_{Adap1}) \cup G_{Adap2})... \cup G_{AdapN})$ is the graph with $V_T = V_{Base} \cup V_{Adap1} \cup ... \cup V_{AdapN}$ and $E_T = E_{Base} \cup E_{Adap1} \cup ... \cup E_{AdapN}$. When two graphs g_1 and g_2 share a vertex ($vg1$ and $vg2$ have the same Typ and Id) the superimposition operation: (1) keeps one vertex $vg1$ in the resulting graph (2) copies the incoming edges (respectively the outgoing edges) of the $vg2$ to $vg1$ (by modifying their target vertex and their source vertex to be $vg1$). Starting from this point, interferences may appear in the G_T graph.

Interference will occur only when adaptations share at least one port. This is due to the following constraints, which are needed to guarantee the associativity and the commutativity properties: (1) In order to preserve the independency of adaptations entities, these later can only share vertices of the base assembly, (2) Adaptations cannot remove Blackbox vertices or links explicitly (This may lead to the loss of associativity and commutativity properties). Vertices are removed only if the adaptation is withdrawn or components are not available. Thus, the execution of an adaptation cannot prevent or enable another adaptation to be performed. Thus, the potential for interference between adaptations is greatly reduced. We defined two types of interferences: (1) Control flow interference occurs when adaptation entities share an output port of the same component (2) Concurrent method call occurs when adaptation entities share a method call of a component. In [16] we have already detailed the process of resolution of *interference Type 1*. In this paper, *(i)* we extend the set of white box vertices, *(ii)* we proof the associativity and commutativity properties for these new connectors, and *(iii)* we detail the process of resolution of interference of Type 2.

We have seen in the previous example that the *SoundRedirect* adaptation and the *Threshold Level* adaptation provide an interference problem. These two adaptations share the port $Play$ of the component $AudioHome$ with the base assembly graph. Each adaptation sends a different value to this device. The superimposition of these graphs illustrates an example of interference of type (2) presented above. To tag this point in the graph G_T, a special vertex has been added in Figure 3.

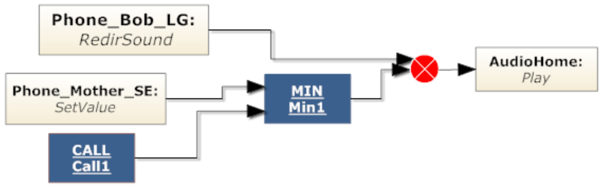


Figure 3. The Interference detection step adds a $\otimes$ vertex to mark the problem

Thus, what is the resulting behavior when both adaptations are relevant to be applied? In this case, the system will respect the threshold defined by the mother? Or will let Bob increases the level of sound independently of the threshold. In the next section we detail the resolution process of this problem.

C. Interference management using Graph transformation rules

The interference management operation addresses the automatic resolution of interferences with the *associativity* and the *commutativity* properties. To preserve this property, it has been seen that adaptation entities are applied on the base assembly G_{Base}, which is free of any adaptation. Consequently, the result of the interference management will be the same whatever the order of adaptations composition. The resolution mechanism builds its solution from $\otimes$ vertices (added previously in G_T graph). We have seen that graphs are build from two subclasses of vertices: Black box and White box. Basically, the algorithm used to resolve interferences browses all $\otimes$ components in order to run a *merging engine* on them [16]. Since we work on graphs, the merging will be achieved by applying *graph transformation rules*.

A graph transformation rule has the form of $p : L \rightarrow R$ and is applicable to a graph G_{Base} if there is an occurrence of L in G_{Base}. Using rules definition, the designer can control the manner in which adaptation graphs will be merged. Thus, the semantics of the deployed application will be affected by the logic of merging given by the designer. Given two vertices, $v1$ and $v2$ (they must share a $\otimes$ successor or predecessor vertex), we define a set of graph transformation rules that shows how they will be merged according to different possible configurations (for example if $v1$ and $v2$ share a predecessor vertex). Thus, we have defined the set of rules to merge each white box vertex with all other vertices. Compared with our previous work, we have extended the set of White box vertices and accordingly their graph transformation rules.

1) Concurrent method call interference resolution: This kind of interference occurs when at least two ports are linked to a port of another component in the same time. To resolve concurrent method call interference, we have defined new connectors (which have 2 predecessors vertex and one successor vertex). The first connector is $[v1, v2] \bullet FW$. When a data come from v1 and/or v2, this connector forward it to its successor vertex(the method call). In other cases, it would be necessary to add an $[v1, v2]\bullet AND$ or $[v1, v2]\bullet XCHoice$ connectors (the $XCHoice$ connector can be instantiated according several strategies such as Min, Max, XOR,.. It allows to choose one data from its input vertices). Similarly to the Control flow interference resolution, adaptations entities should specify connectors that should be added for each method call. Otherwise the merging engine will apply its default solution (FW connector). The merging algorithm is defined in algorithm 1.

In order to merge interfering vertices, we first need to get

Algorithm 1 $Merging(Graph\ G_T)$

```
$ListFusvertex$ : is used to save the list of $\otimes$ vertex of $G_T$
$k$ : the number of $\otimes$ vertex in $G_T$

for $s = 0$ to $k = ListFusvertex.getSize()$ do
    $Pre$= $GetPredecessor(ListFusvertex(s))$
    Vertex $v1$= $Predecessor(0)$ ;
    Vertex $v2$= $Predecessor(1)$ ;
    Rule $r$=$SelectRule(v1, v2)$
    Boolean $res$=$ExecuteRule(r)$
    $ListFusvertex = getFusVertex(G_T)$
end for
```

the list of $\otimes$ vertices ($ListFusvertex$) from G_T graph. Each $\otimes$ vertex has exactly two predecessor' vertices (saved at Pre). Then, we set $v1$ to the first predecessor vertex and $v2$ to the second predecessor vertex. Given two vertices $v1$ and $v2$, the $SelectRule(v1, v2)$ function starts by searching the graph transformation rule in the rule database in order to merge these vertices (using $Typ(v1)$ and $Typ(v2)$ attributes). If there is no graph transformation rule able to solve this problem, then the default solution will be applied. For example, if a designer has defined a *permissive policy*, all adaptations will be met at the same time (parallel execution). This is similar to applying a Forward *FW* connector to each adaptation graph. The default solution will replace the $\otimes$ vertex (which cannot be resolved) by a *FW* vertex. Next the function $ExecuteRule(r)$ applies the selected rule r to G_T. This rule merges $v1$ and $v2$ and removes the current $\otimes$ vertex. After this, the merging engine selects the next $\otimes$ vertex from the list $ListFusvertex$. Using our case study, we will show that the merging operation can be propagated to the successors vertices of $v1$ or $v2$ in order to solve the interference for each vertex. In such case we need to get the new list $ListFusvertex = getFusVertex(G_T)$ because some $\otimes$ vertices have been added to the graph.

Figure 3 shows an interference on the port $Play$ of the component *AudioHome*. Thus, we will apply our merging algorithm to resolve this problem. The $\otimes$ vertex has two predecessors vertices $v1 = Min$ and $v2 = PhoneBobLG$: $RedirSound$. The rule r that will be applied is given in the top of Figure 4.

Throughout this rule the designer has defined that he wants rewrite an existent link (due to the CALL use). The trivial

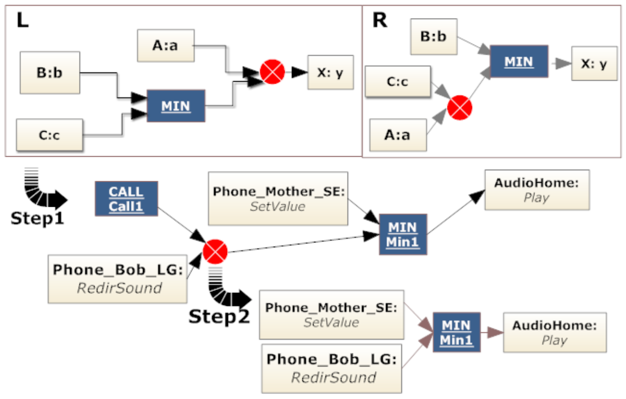


Figure 4. The Graph transformation rule (R1) and the two steps for interference resolution

way to do this is achieved by propagating $v2$ into the second branch of the Min connector. The rule will reconnect these vertices using a new $\otimes$ vertex. R1 rule can be applied in our example because there is an occurrence of the left graph L on G_T. The matching step will unified the L graph variables as following: *A:a* is $PhoneBobLG$: $RedirSound$; *B:b* is $PhoneMotherSony$: $SetValue$ port.

Figure 4 shows the result of R1 execution (Step1). First, $\otimes$ vertex is propagated in Min branch. Since there are still $\otimes$ vertices, the merging algorithm will continue the resolution process. In this example, there are a CALL connector and a port. In that case, the merging will apply a graph transformation rules that will connect the port to the output vertex of the *CALL* connector. We have seen that this connector allows rewriting an existent link. The graph of components assembly of the final application is depicted in Figure 4, Step 2. We note that $CALL$ connector will not be instantiated in the final assembly because it is used only for the interference resolution. When we define adaptation entities, this connector should be used carefully.

D. The symmetry propriety

Associativity and commutativity properties guarantee a good independence between adaptation entities, that is to say that there is no need to define some explicit dependency between them since the composition process ensures the consistency of the result. The result of adaptations weaving will be the same whatever the order of composition. The interference resolution process must also guarantee this property when it merges interfering adaptations. Let $\boldsymbol{C}$ be the set of connectors. The symmetry is defined via three sub-properties:

- *Idempotent*: $\forall c1 \in \boldsymbol{C}$ and c1$\otimes$c1=c1. Merging a connector with it self should return the same connector. This property is achieved by construction. We have specified a graph transformation rule that keep only one connector when we compose the same connector (i.e., *SEQ•[a,b]*$\otimes$*SEQ•[a,b]=SEQ•[a,b] when a*$\neq$*b*).
- *Commutativity* : $\forall c1, c2 \in \boldsymbol{C}$; c1$\otimes$c2=c2$\otimes$c1. It should not matter in which order connectors are merged. The

function *SelectRule* presented previously, select the same graph transformation rule independently of the order of its parameter (it means *SelectRule*(c1,c2)= *SelectRule*(c2,c1)). This property is ensured at composition level by construction.

- *Associativity*: $\forall c1, c2, c3 \in$ ***C***; ((c1⊗c2)⊗c3) =(c1⊗(c2⊗c3)). If the merge operation is associative, then generalization to more than two connectors can be achieved merely by repeated merges, in any order.

To grantees the determinism of the computed application and also ensures the consistency of the composed system at the implementation level, we need to prove the associativity property. In order to resolve interferences at input port, we have extended the set of White Box vertices to support new semantic. Thus, for each new connector, we must prove the *associativity property of its merging rules*. A fundamental question to answer is that *"how associativity can be proved?"* To answer to this question we define steps to be done for that end. We choose as example the *new* connector Min (we use the notation $Newc$). Connectors do ***not necessarily need to be symmetric*** (for example *SEQ* connector is not symmetric because: when *a≠b,* ***SEQ•[a,b]≠ SEQ•[b,a]*** but its composition with itself and other connector is symmetric). For the proof, we will require demonstrating that the merging of $Newc$ with any other connector is associative. The first step to do is the definition of graph transformation rules that specify the way of merging of the $Newc$ connector with each other connectors (we have already done the proof for all existent connectors). After that we need to prove that:

- The merging of $Newc$ with itself is associative. It means that ($Newc$⊗ $Newc$)⊗ $Newc$ = $Newc$⊗($Newc$⊗ $Newc$). For example, using our graph transformation rules, we obtain {*[a,b]•Min*⊗ *[a,c]•Min*}⊗ *[c,b]•Min=[a,b]•Min* ⊗ {*[a,c]•Min*⊗ *[c,b]•Min*}= *[a,[b,c]•Min]•Min* when *a≠b, a≠c and b≠c* (there are also other combinations to compute such as [b,c]•Min, [a,d]•Min...).
- The merging of $Newc$ with all the existent connectors is also associative. That is to say: $(Newc \otimes Oldc_1) \otimes Oldc_2 = Newc \otimes (Oldc_1 \otimes Oldc_2)$ *where both* $Oldc_1$ *and* $Oldc_2$ ∈{*PAR, IF, SEQ, DELEGATE, CALL, FW*}

In this paper, we will demonstrate this property for the rule R1 presented previously. This rule shows the merging of a port with a *MIN* connector. Let $G1 = [a, CALL]\bullet Min, G2 = c, G3 = d$ be the set of input graphs of the composition operation where a, b, c and d are ports' components. We will compose theses graphs as fellows: $Comb1 = (G1 \otimes G2) \otimes G3$ and $Comb2 = G1 \otimes (G2 \otimes G3)$ and we will show that the resulting graph is the same. There are several configuration to consider during composition:

case1: $c = a$ and $d \neq a$ and $d \neq c$
$Comb1 = ([a, CALL]\bullet Min \otimes a) \otimes d = [a, CALL \otimes a]\bullet Min \otimes d = [a, (CALL \otimes a) \otimes d]\bullet Min = [a, a \otimes d]\bullet Min = [a, [a, d]\bullet FW]\bullet MIN$
$Comb2 = [a, CALL]\bullet Min(a \otimes d) = [a, CALL]\bullet Min \otimes [a, d]\bullet FW = [[a, CALL \otimes [a, d]\bullet FW]\bullet Min = [a, [a, d]\bullet FW]\bullet Min$
we conclude that $comb1 = Comb2$

case2: $c = d$ and $c \neq a$
$Comb1 = ([a, CALL]\bullet Min \otimes c) \otimes c = [a, CALL \otimes c]\bullet Min \otimes c = [a, (CALL \otimes c) \otimes c]\bullet Min = [a, c]\bullet Min$
$Comb2 = [a, CALL]\bullet Min(c \otimes c) = [a, CALL]\bullet MIN \otimes c = [[a, CALL \otimes c]\bullet Min = [a, c]\bullet MIN$
we conclude that $comb1 = Comb2$

case3: $c \neq a$ and $d \neq a$ and $d \neq c$
$Comb1 = ([a, CALL]\bullet Min \otimes c) \otimes d = [a, CALL \otimes c]\bullet Min \otimes d = [[a, (CALL \otimes c) \otimes d]\bullet Min = [[a, c \otimes d]\bullet Min\ [a, [c, d]\bullet FW]\bullet MIN$
$Comb2 = [a, CALL]\bullet Min(c \otimes d) = [a, CALL]\bullet Min \otimes [c, d]\bullet FW = [[a, CALL \otimes [c, d] \bullet FW]\bullet Min = [a, [c, d]\bullet FW]\bullet Min$
we conclude that $comb1 = Comb2$

We conclude that the rule R1 is associative. The proofs of the associativity property of the merge function on other new graph transformation rules are completely identical to the proofs of the associativity shown for the fusion rule R1. We do not detail the proof of other rules. To demonstrate that the merging of each new connector is associative, we compute all the possible composition combinations (connector with itself and with existent connectors). In order to facilitate this task, we have implemented a tool for associativity automatic proof. The input of this tool is the definition of the new connector and its graph transformations rules (we defined the graph of Min connector and all graph transformation rules of its merging). As output, it will show in which case this property is failed (if it exists). Thus, the designer can modify the set of rules that cause problems to guarantees the property. Using this tools we have proved that the composition of Min connector with all previous operators is associative.

V. Practical Issues

The compositional adaptation mechanism presented in this paper has been implemented in the WComp platform, which is a middleware for adaptation based on a dynamic composition of services based on the SLCA (Service Lightweight component Architecture) model [17] . Components allow the management of the black box properties of devices. The interaction is limited to the use of their required and provided ports (the direct access to implementation is forbidden). In order to manage adaptation in a transversal way to produce simultaneous modifications in different points of the application, we use a paradigm named Aspect of Assembly(AA)[18] (based on AOP principles). It produces components assembly (graphs) that will be composed. The merging engine implementation has been presented in [16] and [19]. The merging engine uses graph transformation mechanism in order to merge vertices where interferences have been detected. Various graph transformation tools exist but the most used is AGG (Attributed Graph Grammar) [20].

Performance is a decisive factor in self adaptive systems (ubiquitous systems need to be quickly adapted to consider their infrastructure changes). For that reason, we measured the execution time of the composition operation. Then, we can conclude in which case our solution can be used. The ruling that the response time is acceptable or not depends on application's domains. The results of our experiments are briefly presented in the rest of this section. Figure 5 shows the execution time of the merging step. From these experimentations, we conclude that our approach can be used in application's domain that don't require a very low response time to be adapted. For example, our composition process can be used in the area of domestic application since it composes adaptations about 1 second.

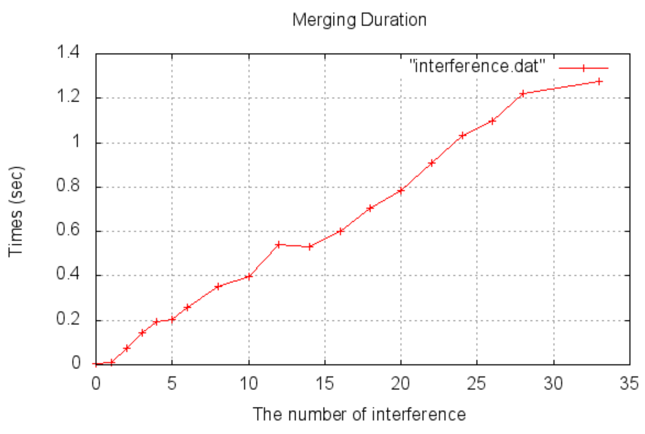


Figure 5. The execution time of the Merging process

VI. Conclusion and Future Work

The work described in this paper is derived from our experience in composing independently developed adaptations (but jointly deployed). In this paper, we introduced a new connector Min that enables our composition mechanism to handle the concurrent method call interference problems. Using graph representation, we formally defined the model of the running system and the set of its adaptations. Our approach performs some predefined graph transformation rules that will merge special vertices of the graph in order to resolve problems. We provide that this operation is associative as the order of adaptations compositions does not matter.

Immediate perspective of this work is to provide a well defined representation of connectors behavior. Model checking techniques will be applied to ensure that the composition of synchronous connectors does not violate the behavior of the target application.

VII. Acknowledgments

This work is supported by the French ANR Research Program VERSO in the project ANR-08-VERS-005 called CONTINUUM.

References

[1] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, and B. Cukic, "Software engineering for self-adaptive systems: A research roadmap," *Software Engineering for Self-Adaptive Systems*, pp. 1–26, 2009.

[2] P. McKinley, S. Sadjadi, E. Kasten, and B. Cheng, "A taxonomy of compositional adaptation," *Rapport Technique numéroMSU-CSE-04-17, juillet*, 2004.

[3] P. Greenwood, B. Lagaisse, F. Sanen, G. Coulson, A. Rashid, and E. Truyen, "Interactions in ao middleware," in *Proc. Workshop on ADI, ECOOP*, 2007.

[4] F. Munoz and B. Baudry, "Validation challenges in model composition: The case of adaptive systems," *ChaMDE 2008*, p. 51.

[5] F. Sanen, E. Truyen, and W. Joosen, "Modeling context-dependent aspect interference using default logics," in *Fifth workshop on Reflection, AOP and Meta-data for Software Evolution*, no. 5, 2008, pp. 1–5.

[6] M. Wermelinger, A. Lopes, and J. Fiadeiro, "A graph based architectural (re) configuration language," in *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 5. ACM, 2001, pp. 21–32.

[7] D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[8] L. Baresi, R. Heckel, S. Thöne, and D. Varr´ o, "Style-based modeling and refinement of service-oriented architectures," *Software and Systems Modeling*, vol. 5, no. 2, pp. 187–207, 2006.

[9] J. Dowling, V. Cahill, and S. Clarke, "Dynamic software evolution and the k-component model," in *Workshop on Software Evolution*, 2001.

[10] M. Guennoun, "Architectures dynamiques dans le contexte des applications à base de composants et orientées service," 2006.

[11] P. David and T. Ledoux, "An aspect-oriented approach for developing self-adaptive fractal components," in *Software Composition*. Springer, 2006.

[12] T. Dinkelaker, M. Mezini, and C. Bockisch, "The art of the meta-aspect protocol," in *Proceedings of the 8th ACM international conference on Aspect-oriented software development*. ACM, 2009, pp. 51–62.

[13] D. Cheung, J. Tigli, S. Lavirotte, and M. Riveill, "Wcomp: a multi-design approach for prototyping applications using heterogeneous resources," in *Rapid System Prototyping. Seventeenth IEEE International Workshop on*, 2006, pp. 119–125.

[14] S. Ciraci, W. Havinga, M. Aksit, C. Bockisch, and P. van den Broek, "A graph-based aspect interference detection approach for uml-based aspect-oriented models," *Transactions on aspect-oriented software development VII*, pp. 321–374, 2010.

[15] M. La Rosa, M. Dumas, R. Uba, and R. Dijkman, "Merging business process models," *On the Move to Meaningful Internet Systems: OTM 2010*, pp. 96–113, 2010.

[16] S. Fathallah Ben Bbdenneji, S. Lavirotte, J. Tigli, G. Rey, and M. Riveill, "Mergeia: A service for dynamic merging of interfering adaptations in ubiquitous system," in *UBICOMM 2011, The Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, 2011, pp. 34–38.

[17] V. Hourdin, J. Tigli, S. Lavirotte, G. Rey, and M. Riveill, "Slca, composite services for ubiquitous computing," in *Proceedings of the International Conference on Mobile Technology, Applications, and Systems*. ACM, 2008, p. 11.

[18] J.-Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, D. Cheung-Foo-Wo, E. Callegari, and M. Riveill, "WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services," *Annals of Telecommunications (AoT)*, vol. 64, Apr 2009.

[19] S. Fathallah, S. Lavirotte, J.-Y. Tigli, G. Rey, and M. Riveill, "Adaptations interferences detection and resolution with graph-transformation approach," in *the 6th International Conference Sciences of Electronic, Technologies of Information and Telecommunications(SETIT)*, ser. , Sousse, Tunisia, Nov.

[20] G. Taentzer, "Agg: A graph transformation environment for modeling and validation of software," *Applications of Graph Transformations with Industrial Relevance*, pp. 446–453, 2004.

Mission-oriented Autonomic Configuration of Pervasive Systems

Guillaume Grondin, Matthieu Faure, Christelle Urtado and Sylvain Vauttier
LGI2P / Ecole des Mines d'Alès, Nîmes, France
{Guillaume.Grondin, Matthieu.Faure, Christelle.Urtado, Sylvain.Vauttier}@mines-ales.fr

Abstract—In pervasive systems, software applications are dynamically composed from the services provided by the smart devices spread in the local environment. A system must react to changes that occur in the environment and reconfigure applications in order to maintain their operation and assume their missions at its best. This paper advocates the need for a mission description language, which enables to describe applications in a declarative way as abstract service compositions. The system uses mission definitions to calculate a configuration that best executes them with the currently available resources. This optimal configuration is intended to maximize the utility of the system, considering user preferences, available resources, and mission criticality. Contextual adaptations are captured in the mission language as modes and strategies, that respectively describe evolutions of the assigned mission set and alternate ways to execute missions. These mechanisms leverage service component approach, for the dynamic deployment of missions, and agent-orientation, for autonomic configuration management.

Keywords-pervasive system; autonomic computing; context-awareness; service composition language.

I. Introduction

As miniaturization of computer systems increases, calculation and communication-enabled (so-called smart) devices spread around us. These new computers altogether form computer networks that have given birth to a new paradigm called pervasive computing. Pervasive computing leverages the services that are dynamically discovered in the environment to meet user requirements. The dynamicity and openness of the environment have a strong impact on the software that must adapt to these changes. Thus, pervasive systems must provide solutions for several inherent issues [1]:

- Context-awareness. The boundaries and constituents of pervasive systems are not known beforehand. Pervasive software must be able to dynamically discover which resources are available in the environment. Moreover, it must know how to map user requirements with the available resources in order to implement valuable scenarios. Context-awareness is the capacity of pervasive software to sense and build a proper inner representation of varying environments.

- User empowerment. Pervasive systems are inherently user-oriented. Their purpose is to help users leverage the services provided by the smart devices that surround them in their environments. To be as useful and relevant as possible, pervasive systems must provide users with means to define and submit their own service compositions. Indeed, elaborated specific requirements cannot be met by predefined services. Pervasive systems must therefore support the dynamic definition of their missions, which entails dynamic adaptation touser demands.

- Autonomous and dynamic adaptability. As devices can freely join or quit the system anytime, resources and services are volatile. Pervasive systems must support dynamic change management to adapt themselves to open, variable environments. For dependability's sake, service continuity must be transparently maintained thus necessitating the system to autonomously and dynamically react in order to evaluate the situation and change either means used to reach its defined objectives or its objectives themselves..

The remaining sections of this paper describe the pervasive system framework we designed to tackle these issues. Section II sets the technical ground of our proposal, which is a combination of a multi-agent system and a service component architecture. It also briefly introduces a water hazard monitoring system as an application for our framework. Section III introduces the concepts and syntax of Arold, our proposed mission description language. Section IV briefly explains the principles of optimized deployment calculation. Section V discusses related work. Section VI draws a conclusion and perspectives about this work.

II. Technical Ground and Case Study

A. System, Agents, Components and Services

A pervasive system is dynamically composed of a set of smart devices co-located in an environment. Smart devices use their embedded communication capabilities and intelligence (software) to interact and cooperate. In our work (*see* Figure 1), the embedded software of each device is managed in a modular and reconfigurable way, which conforms to a service component approach [2].

Components are reusable, decoupled software modules that encapsulate distinct functionalities. They can be assembled together, so as to produce operational software, thanks to well defined connectable interfaces that explicitly document the interaction capabilities of components. Provided (server) interfaces document functionalities that are implemented by a component and proposed as

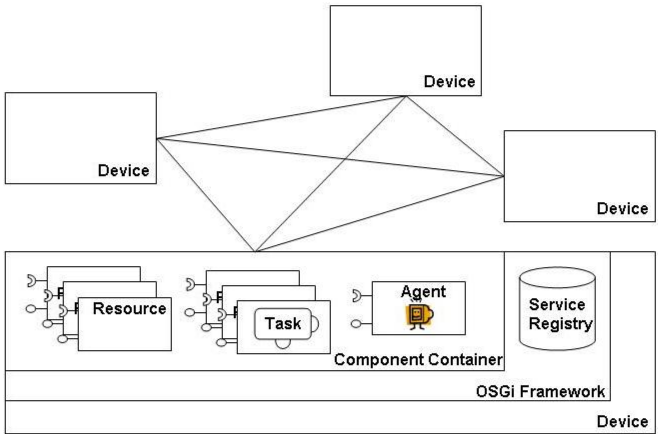


Figure 1. A component-based and autonomic pervasive system

invocable services. Required (client) interfaces document functionalities that must be called by a component as external services it depends on. We use the component model and container facility proposed by the OSGi framework [3] to dynamically administrate the installation, assembling, replacement or removal of components on each device. The OSGi framework also provides a service registry where any instantiated component can publish the services it proposes through its provided interfaces. This enables the dynamic introspection and retrieval of components that are available in a container. Components can thus be managed and handled so as to build and adapt software to the changing contexts in which devices must operate. Three kinds of components are actually deployed on a device: resource components, that provide low-level technical services used to execute concrete actions; task components, that define abilities to manage the execution of higher-level activities, that are intended to become part of mission compositions; and an agent component, that contains a software agent that manages the component-based architecture of the embedded software.

An agent [4] is an active entity that is designed to collect information, reason, make decisions and act autonomously. We use agents' activity to endow devices with self-configuration. Such autonomic capabilities are essential to pervasive systems: manual intervention cannot cope with neither large scale or continuously changing systems. Moreover, an agent is a social entity that is designed to interact with other agents in order to manage complex activities as distributed collaborations. A multi-agent system provides decentralized, peer-to-peer, communication schemes that are naturally suitable for open and dynamic architectures of pervasive systems. Agents senses automatically the presence of other agents thanks to discovery protocols integrated to middlewares. They dynamically create a community, share information about the resources they control and plan the distributed execution of missions.

B. The Hydroguard Pervasive Water Surveillance System

An application of our work is the control of the Hydroguard Pervasive Water Surveillance System[1]. This system is designed to monitor hydrological parameters on rivers and coastal areas in order to detect critical situations such as floods or pollutions. It is composed from on-site devices that use various types of sensors to measure meaningful physical quantities (pressure, temperature, rainfall, pH, etc.). Each site has a specific geographical situation. Its streams and waters have specific characteristics. Devices have to operate in various contexts, for which their software must be specifically configured, thanks to the chosen component-based approach. Moreover, this context is not fixed once and for all. As devices operate outdoor, they may endure bad weather conditions that cause communication losses or device failures. In emergency cases, these changes may be intentional: extra devices may be added to extend the monitored area or to get additional measures; conversely, devices may be moved to a more demanding site or to mitigate failures. Hydroguard is thus a pervasive system that must support dynamic arrival or leaving of devices. Its devices must react to these changes and adapt their behaviors and collaborations to maintain the continuity of their monitoring missions. This is handled by the autonomic dynamic adaptation capabilities provided by the embedded agents. This way, supervision by human operators is not required and the system may stay operational even when remote administration is impossible in exceptional situations.

Besides, as aforementioned, the system is designed to run multiple missions in parallel, in order to monitor multiple hazards. Depending on environmental situations, defined by domain experts in terms of thresholds on specific monitored data, the criticality of missions may evolve from normal (routine) to vigilance and finally to crisis. This may require to change the operation mode of some missions, for instance to timely follow the evolution of an incident, granting them higher priority so that the system concentrates its resources on the more critical and relevant missions first. Combined with the pervasive nature of the system, the presence or the availability of a resource is never guaranteed. Mission definitions must therefore include various admissible ways to achieve them, depending on criticality and resource availability, that agents use to find the best execution configuration, in any changing context. Mission definition is thus a key issue for system adaptation, extending the perimeter of

[1]Partly funded by the French government FUI-AAP8 project.

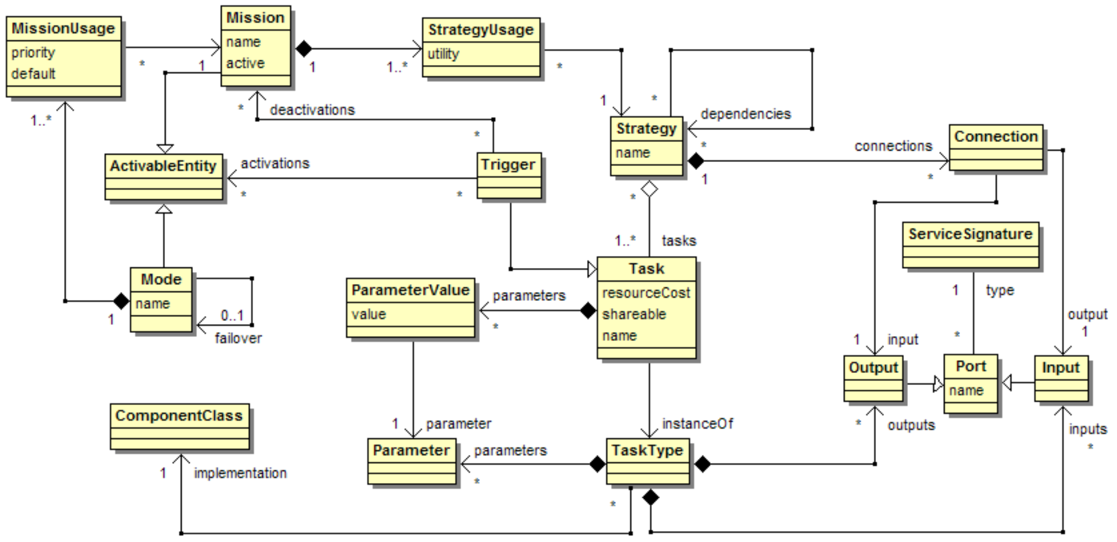


Figure 2. Metamodel of AROLD, a multi-context mission description language

context-awareness [5] to user requirements. We address this issue with the proposal of AROLD, a multi-context mission definition language.

III. AROLD: A MULTI-CONTEXT MISSION DESCRIPTION LANGUAGE

As discussed in the previous sections, pervasive systems have open, dynamic architectures that may evolve at any moment, in unanticipated ways. The invariant part of the system is the set of functional objectives it must achieve to meet the requirements of its users. AROLD is proposed as a mean to capture users' requirements as abstract, declarative mission definitions. Missions definitions are then automatically mapped by the manager agents controlling the pervasive system to available resources so as to find an appropriate configuration to execute the missions in the current context. This way, missions are not defined by components that may disappear when devices leave the system. The main concepts of AROLD, as shown in its metamodel (*see* Figure 2) are *modes*, *missions*, *strategies* and *tasks*.

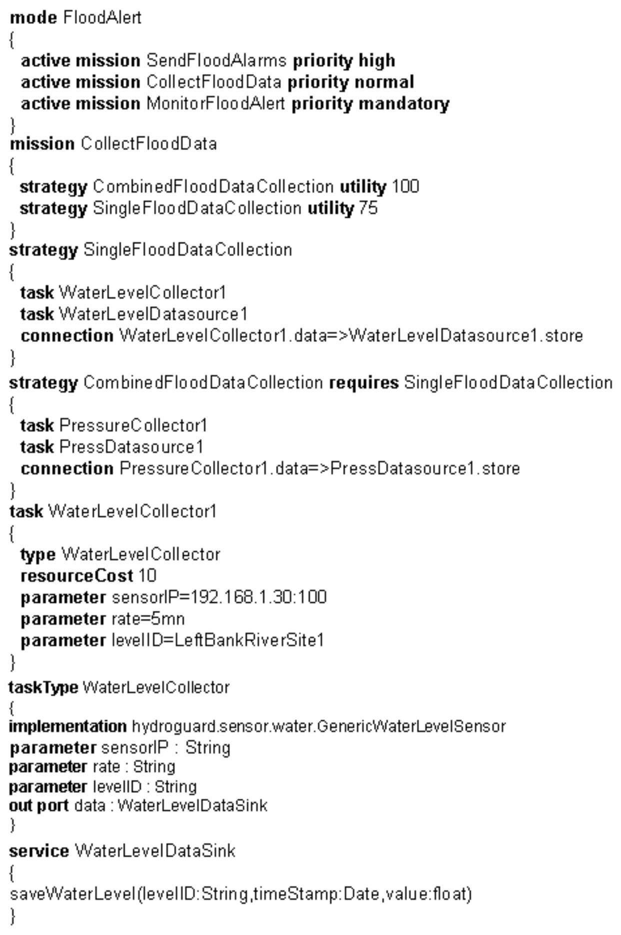
A. Modes and Missions

A *mission* defines a functional objective that is assigned to the system. Being open, one inherent characteristics of pervasive systems is to be composed from various devices and to support versatile missions. In Hydroguard, missions are, among many others, to forecast weather, to monitor flood level, to send flood alerts and to monitor water quality. Though every mission corresponds to some user requirements, all the missions do not have the same criticality and relevance, regarding specific situations of the environment. *Modes* thus model the different situations that the system is expected to manage distinctively. In Hydroguard, *Routine*, *FloodVigilance* and *PollutionCrisis* are mode examples. Modes build a first level of context representation, which defines the specific environmental situations that the system has to manage. Missions build a second level of context representation, which captures user requirements. Each mode explicitly specifies the set of missions that is to be achieved in the current situation (*see* Figure 3). Missions are defined as independent concepts, so they can be reused in as many modes as necessary. Mission criticality is expressed by a priority: *Mandatory*, *High*, *Normal*, *Low*. These priorities are affected by each mode, so that mission criticality may vary to be adapted to the context. Mandatory missions corresponds to critical activities. If any mandatory mission cannot be executed, the system cannot properly manage the current mode. It must signal a failure and enter a failover mode. Monitoring the conditions that determine transitions to other modes is part of the mandatory missions that must be associated to a mode. The system then tries to execute as many missions as possible, depending on available resources, starting by high priority missions and finishing with low priority missions. To enable a more flexible definition of modes, all the missions associated with a mode are not intended to always execute at the same time, but possibly successively, as the situation evolves. To manage this, missions hold a status attribute that specify if they are active or not. Apart from its mandatory missions, which are always active, a mode defines a set of default missions that are active when entering in this mode. The set of active missions then evolves by explicit mission activations or deactivations controlled by the already active missions.

B. *Strategies and Tasks*

A *strategy* defines a concrete implementation of a mission. A mission can be implemented by a set of strategies (*see* Figure 3), that represent alternative ways to achieve the functional objective represented by the mission. Strategies build a third level of context representation, which defines how the execution of missions may vary to enable its adaptation to available resources. Strategies need not be strictly equivalent. On the contrary, the different strategies associated with a mission should represent various tradeoffs between resource requirements and utility. This way, among the strategies that can be currently executed, the system tries to find an optimal set of strategies, that is the set of strategies that enables to execute the largest set of active missions, with the best efficiency. The efficiency of a strategy depends on its cost (resource requirements) but most importantly to its utility, regarding the achievement of the mission. Utility is a score defined for each strategy by a mission, which characterizes the quality of the result that is produced by the execution of the strategy. Utility is contextual. Associated to a mission used in a routine mode, an economic strategy with a poor quality of service would be ranked with a low utility. Associated with a mission used in an emergency mode, it would be anyway useful and ranked with a high utility.

A strategy is concretely defined by a set of *tasks* that must be instantiated and connected together by the system in order to execute the mission (*see* Figure 3). This part of Arold is thus equivalent to a simple architecture description language [6]. Each task represents a specific service invocation, specified by a set of parameter values. A task (*e.g.*, *WaterLevelCollector1*) corresponds to an instance of a task type (*e.g.*, *WaterLevelCollector*), so that different tasks can be defined corresponding to a common kind of activity. A task type defines the list of parameters (*e.g.*, *sensorIP, rate, levelID*) to be specified in each individual task. It also defines how the task type is implemented, as a reference to a concrete component (*e.g.*, *GenericWaterLevelSensor*) that must be used to instantiate this kind of tasks. Finally, a task type specifies a set of in and out ports (*e.g.*, the *WaterLevelCollector* task type has an out port as it is a data sink). In and out ports define the information (data, control) that a task respectively receives from or sends to others tasks. Task compositions are defined in strategies by sets of connections between in and out ports of their component tasks (*e.g.*, connection between the *WaterLevelDatasource1* task and the *WaterLevelCollector1* task). This can be regarded as similar to workflows. However, task compositions do not hold any explicit control structures in Arold. Composition is thus purely structural and amounts to assemblies, as proposed in SCA [2]. Workflow control is embedded in tasks. When interactions are required, external control management is defined by ports. Ports are typed by service signatures (*e.g.*, the *WaterLevelDataSink* service saves a water level value along with a localization ID and a timestamp). These syntactical specifications are used to check the soundness of connections between ports.



```
mode FloodAlert
{
  active mission SendFloodAlarms priority high
  active mission CollectFloodData priority normal
  active mission MonitorFloodAlert priority mandatory
}
mission CollectFloodData
{
  strategy CombinedFloodDataCollection utility 100
  strategy SingleFloodDataCollection utility 75
}
strategy SingleFloodDataCollection
{
  task WaterLevelCollector1
  task WaterLevelDatasource1
  connection WaterLevelCollector1.data=>WaterLevelDatasource1.store
}
strategy CombinedFloodDataCollection requires SingleFloodDataCollection
{
  task PressureCollector1
  task PressDatasource1
  connection PressureCollector1.data=>PressDatasource1.store
}
task WaterLevelCollector1
{
  type WaterLevelCollector
  resourceCost 10
  parameter sensorIP=192.168.1.30:100
  parameter rate=5mn
  parameter levelID=LeftBankRiverSite1
}
taskType WaterLevelCollector
{
implementation hydroguard.sensor.water.GenericWaterLevelSensor
parameter sensorIP : String
parameter rate : String
parameter levelID : String
out port data : WaterLevelDataSink
}
service WaterLevelDataSink
{
saveWaterLevel(levelID:String,timeStamp:Date,value:float)
}
```

Figure 3. Excerpt of a mission descriptor in Arold textual syntax

Finally, each task is defined by its resource consumption. For the sake of simplicity, it is represented in this paper as a unique attribute (*resourceCost*). In our concrete implementation, resource consumption encompasses CPU, RAM, disk and network usage. These attributes are first used individually to check that the instantiation of a task on a device does not exceed its amount of resources.

IV. Optimized Mission Deployment

Taking into account an operation context modelled in Arold, the role of the manager agents embedded in the devices composing the pervasive system is to determine an optimized deployment of the set of active missions in the current active mode. As aforementioned, this deployment should be optimized so that, taking into account resource limitation, the executed missions are the most critical and the more useful to users. To calculate this optimal deployment is an optimization problem, with a high computational complexity, due to the choice of the missions, strategies and devices where tasks are instantiated. On a formal ground, the optimization of mission deployment is a constraint satisfaction problem analogous to a variant of the knapsack problem, which is a NP-hard problem [7]. This problem consists in filling up a knapsack with items of various volumes and values. The best combination of items must be chosen so that the knapsack the volume of which is fixed contains a maximal total value. In our mission deployment problem, the best combination of tasks must be allocated on the devices of the system, so that the total utility of the corresponding strategies, for the active missions in the current mode, is maximal. Mission deployment is calculated separately for each level of priority, so that resources are used by higher priority mission first.

For the sake of efficiency, we have implemented the resolution of the mission deployment problem as a centralized mechanism. When started, a manager agent queries the service repository of the OSGi platform to retrieve the list of the task components that are installed on the device it controls. It then broadcast a message to request the election of a leader among the group of manager agents of the pervasive system. We have designed an election protocol, derived from the bully algorithm [8], but adapted to the open structure of pervasive systems (the description of which is out of the scope of this paper). Centralization is thus mitigated because it is a dynamic process in which any manager agent can become the leader. When the manager agent receives the address of the elected leader, it sends back to the leader a list of its features, meaning its resource capacity and the list of the task types it can instantiate. It then waits for the leader to send configuration instructions, describing which tasks to instantiate and which connection to set up. Meanwhile, the leader collects the feature informations sent by the other manager agents. Taking into account the missions specified by AROLD descriptors, these informations enable the leader to build a global definition of the optimization problem to be solved. This is a point where centralization becomes an advantage: the more complete is the problem definition, the more optimized the found solution may be.

Many resolution algorithms exist for the knapsack problem. Our application is the dynamic reconfiguration of a pervasive system. The optimization problem must therefore be solved not only timely but also on a device with limited resources. We thus designed an adaptation of the greedy approximation algorithm, chosen for its very low time and space complexity. The strategies associated with active missions are ordered by their efficiency (ratio between their utility and resource consumption). The possible deployment of the most efficient strategies are checked first (most rational choices). Strategies are thus chosen, until every active mission is executed by a strategy or no more strategy can be deployed because of resource unavailability.

Determining where a task should be allocated, between a set of candidate devices, is another source of combinatorial explosion. To solve this issue and preserve the linear complexity of the greedy algorithm, we designed a choice algorithm using an ordered list of devices, similar to the ordered list of strategies. Based on both the set of tasks that appear in the mission descriptors and the list of features transmitted by each manager agent, a probability that a task has to be affected on a device is calculated. A potential load is calculated for each device. A task is allocated on the device that has the lowest potential load: this is here again the most rational choice to avoid resource shortage and to tend towards a balanced load between devices. After each allocation, probabilities and potential loads are updated, to take into account the already known task allocations. When the allocation of a task is impossible, the corresponding strategy is skipped. Deployment resolution carries on with the next more efficient strategy to be tried. This way, deployment is solved with a linear complexity that fits the computation capabilities embedded in devices. Though simple, this algorithm is able to produce on average interesting suboptimal results.

V. Related Work

AROLD deals with service compositions, a much studied topic related to web services choreographies but also pervasive systems [1], [9], [10], [11]. These works focuses on a transparent management of the execution context, to let users declaratively express their required compositions, without any specific knowledge about the environment. The supporting frameworks deal with a wide range of mechanisms: discovering available services, matching available services with user defined compositions, maintaining execution by automatically replacing faulty services, balancing execution load. Goal-oriented systems even go one step further. Users express the results they wish to obtain and the system automatically calculates, by backward-chained inferences, theservice compositions that achieve the specified results [12]. All these systems consider the context from a technical point of view, that can thus become transparent for the user. With AROLD, user expertise and preferences are part of the context definition, modelling variability and utility as alternate strategies. The primary goal of the system becomes to best satisfy all its users: the aforementioned problems are then considered as constraints in a decision problem (mission and strategy selection).

Multi-agent systems are often proposed to manage pervasive systems [13], [14], [15], [16], [17], as they both have open and dynamic architectures. Autonomic adaptation is handled in a distributed way and emerges as the result of peer-to-peer negotiation protocols. On the opposite, we have chosen a centralized mechanism for mission deployment , adapted from the optimized execution plans used in grid-computing [18]. Though very simple, to suit the limited resources of the devices, we consider that it is able to produce faster more optimized results than the convergence of a distributed process.

VI. Conclusion and future work

This paper presented a work in progress. Arold is an original contribution, as a service composition language that enables the contextual adaptations of the missions of a pervasive system to be defined. It takes into account mission criticality, resource availability and user preferences.

Dynamic context redefinition and multiple context definition mitigation still are open issues. Future work will have to detect and manage inconsistencies. Task mobility will also be studied to support the redeployment of running missions and its cost evaluated. Experiments still have to be run to validate the performance hypothesis about our proposed centralized mission deployment scheme as compared to a massively distributed alternative. To do so, we plan to run simulations of large scale distributed arcitectures on a machine cluster.

Ackowledgments

The authors would like to thank the participants of the Hydroguard consortium.

References

[1] Bronsted J., Hansen K. M., and Ingstrup M.: Service Composition Issues in Pervasive Computing. IEEE Pervasive Computing, vol. 325, pp. 311-326 (2011).

[2] Open SOA Collaboration: Service Component Architecture Specification. http://www.osoa.org/, accessed 1-july-2012, (2006).

[3] OSGi Alliance: Osgi service platform Release 4.http://www.osgi.org, accessed 1-july-2012, (2005)

[4] Wooldridge M.: An Introduction to Multi-agent Systems. Wiley (2002).

[5] Bolchini C., Curino C. A., Quintarelli E., Scheiber F.A., and Tanca L: A data-oriented survey of context models. ACM SIGMOD Record, vol. 36(4), pp.19-26 (2007).

[6] Medvidovic N. and Taylor R. N.: A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering, vol. 26(1), pp. 70-93 (2000).

[7] Kellerer H., Pferschy U., and Pisinger D.: Knapsack Problems. Springer (2004).

[8] Mamun Q.E.K., Masum S.M., and Mustafa M.A.R.: Modified bully algorithm for electing coordinator in distributed systems. WEAS Transactions on Computers, vol. 3(4), pp.948-953 (2004).

[9] Ibrahim N. and Le Mouel F.: A Survey on Service Composition Middleware in Pervasive Environments. International Journal of Computer Science Issues, Vol. 1, pp. 1-12 (2009).

[10] Urbieta A., Barrutieta G., Parra J., and Uribarren A.: A survey of dynamic service composition approaches for ambient systems. Proceedings of the Ambi-Sys workshop on Software Organisation and MonIToring of Ambient Systems, ICST, pp. 1-8 (2008).

[11] Bakhouya M. and Gaber J.: Service Composition Approaches for Ubiquitous and Pervasive Computing Environments: A Survey. Agent Systems in Electronic Business, IGI Publishing, pp. 323-350 (2007).

[12] Heider T. and Kirste T.: Multimodal appliance cooperation based on explicit goals : concepts and potentials. Proceedings of the Joint Conference on Smart objects and Ambient Intelligence, ACM, pp. 271-276 (2005).

[13] Malek S., Mikic-rakic M., and Medvidovic N.: A Decentralized Redeployment Algorithm for Improving the Availability of Distributed Systems. Proceedings of the 3rd International Conference on Component Deployment, Springer, pp. 99-114 (2005).

[14] Weyns D., Malek S., and Andersson J.: On Decentralized Self-Adaptation: Lessons from the Trenches and Challenges for the Future. Proceedings of the ICSE workshop on Software Engineering for Adaptive and Self-Managing Systems, ACM, pp. 84-93 (2010).

[15] Bakhouya M. and Gaber J. : Self-organizing Approach for Emergent Multi-agent Structures. Proceedings of the Workshop on Complexity through Development and Self-Organizing Representations, ACM (2006).

[16] Grondin G., Bouraqadi N., and Vercouter. L.: MADCAR, an Abstract Model for Dynamic and Automatic (Re-)Assembling of Component-Based Applications. Proceedings of the 9th International SIGSOFT Symposium on Component-Based Software Engineering, LNCS 4063, Springer, pp. 360-367 (2006).

[17] Hamoui F., Huchard M., Urtado C., and Vauttier S.: SAASHA: a Self-Adaptable Agent System for Home Automation. Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE, pp. 227-230, (2010).

[18] Parashar M. and Pierson J. M.: Pervasive Grids: Challenges and Opportunities. In Handbook of Research on Scalable Computing Technologies, IGI Global, pp. 14-30 (2010).

The Consolidated Enterprise Java Beans Design Pattern for Accelerating Large-Data J2EE Applications

Reinhard Klemm
Collaborative Applications Research Department
Avaya Labs Research
Basking Ridge, New Jersey, U.S.A.
Email: klemm@research.avayalabs.com

***Abstract*— J2EE is a specification of services and interfaces that support the design and implementation of Java server applications. A key concept in J2EE is *Entity Enterprise Java Beans* (EJBs). Their purpose is to persist the state of application objects and to share objects between transactions. Although typically desirable, the persistence in entity EJBs can also incur a heavy performance penalty. In this article, we describe a novel software design pattern aimed at improving the performance of entity EJBs in J2EE applications with large numbers of EJB instances. The pattern maps multiple real-world entities of the same type (e.g., users) to a single *consolidated* entity EJB (*CEJB*), thereby significantly reducing the number of required entity EJB instances. Consequently, CEJBs can increase EJB cache hit rates and database search performance. We present detailed quantitative assessments of performance gains from CEJBs and show that CEJBs can accelerate some common EJB operations in large-data J2EE applications by factors between 2 and 14.**

Keywords-caching; Enterprise Java Beans; object consolidation; software design patterns; software performance

I. Introduction

Enterprise Java Beans (EJBs) [1] take advantage of a wide range of platform services from EJB containers in J2EE application servers. Examples of platform services are data persistence, object caching and pooling, object lifecycle management, database connection pooling, transaction semantics and concurrency control, entity relationship management, security, and clustering. EJB containers obviate the need for redeveloping such generic functionality for each application and thus allow developers to more quickly build complex and robust server-side applications. However, common EJB operations, in particular *entity* EJB operations, such as creating, accessing, modifying, and removing EJBs, tend to execute much more slowly than analogous operations for Java (J2SE) objects (*Plain Old Java Objects* or *POJOs*) that do not implement the functional equivalent of the J2EE platform service [2].

One of the platform services for entity EJBs that can incur a heavy performance penalty is data persistence. Although not mandated by the EJB specification, entity EJBs are typically stored as persistent objects in relational databases and we will assume this type of storage in the remainder of this article. Furthermore, we will concentrate on entity EJBs with container-managed persistence (CMP) rather than bean-managed persistence (BMP). CMP entity EJBs have the advantage of receiving more platform assistance than BMP entity EJBs and are thus usually preferable from a software engineering point of view. They also tend to perform better than BMP entity EJBs because of extensive application-independent performance optimizations that EJB containers incorporate for CMP EJBs [3]. For the sake of simplicity, we will refer to CMP entity EJBs simply as “EJBs”. Note that the mapping from EJBs to database tables and the data transfer between in-memory (cached) EJBs and the database is the responsibility of the J2EE platform and can therefore be only minimally influenced by the EJB developer. Hence, we cannot discuss the impact of the technique presented in this article on structural or operational details of the data persistence layer of the J2EE platform. Instead, we will discuss how our technique changes the characteristics of the EJB layer that is under the control of the EJB developer and show how these changes affect the overall performance of EJB operations.

In the past, a lot of research into improving J2EE application performance has focused on tuning the configuration of EJBs and of the EJB operating environment consisting of J2EE application servers, databases, Web servers, and hardware. In addition, some software engineering methods such as software design patterns and coding guidelines have been developed to address performance issues with J2EE applications. This article presents a novel software design pattern for accelerating J2EE applications that we call *consolidated* EJBs (*CEJBs*). We devised the pattern during a multiyear research project at Avaya Labs Research where we developed a J2EE-based context aware communications middleware called *Mercury*. Mercury operates on a large number of EJB instances that represent enterprise users (hence our *User* EJB examples later in this article). Due to a large frequency of retrieval, query, and update operations on these EJBs, Mercury suffered from slow performance even after tuning J2EE application server and database settings. Thus, we felt compelled to investigate structural changes to Mercury’s J2EE implementation as a remedy for the performance problems and we arrived at the CEJB design pattern.

The remainder of this article is organized as follows. In Section II, we describe some of the related work. Section III presents the CEJB software design pattern and its use in J2EE applications. We describe the details of CEJB allocation, the mapping of entities to CEJBs, the storage of entities within CEJBs, and retrieval of entities from CEJBs. Our presentation focuses on EJBs according to the EJB 2.1 specification. This specification has been supplanted by the EJB 3.1 specification [4] in the meantime. However, the salient ideas of our work remain valid with EJB 3.1. We compare the performance of CEJBs and EJBs in Section IV. A summary and an outline of future work conclude the article in Section V.

II. Related Work

Much research has been devoted to speeding up J2EE applications by tuning EJBs and J2EE application server parameters. Pugh and Spacco [5] and Raghavachari et al. [6] discuss the potentially large performance impact and difficulties of tuning J2EE application servers, connected software systems such as databases, and the underlying hardware. In contrast, CEJBs constitute an application-level technique to attain additional J2EE application speed-ups.

The MTE project [7][8] offers more insight into the relationship between J2EE application server parameters, application structure, and application deployment parameters on the one hand and performance on the other hand. The MTE project underscores the sensitivity of J2EE application performance to application server parameters as well as to the application structure and deployment parameters.

Another large body of research into J2EE application performance has investigated the relationship between J2EE software design patterns and performance. Cecchet et al. [9] study the impact of the internal structure of a J2EE application on its performance. Many examples of J2EE design patterns such as the session façade EJB pattern can be found in [10] and [11], while Cecchet et al. [9] and Rudzki [12] discuss performance implications of selected J2EE design patterns. The CEJB design pattern improves specifically the performance of bean caches and database searches for EJBs. The Aggregate Entity Bean Pattern [13] consolidates logically dependent entities of *different* types into the same EJB while CEJBs consolidate entities of the *same* type into an EJB. Converting EJBs into CEJBs can therefore be automated by a tool whereas the aggregation pattern requires knowledge of the specific application and the logical dependencies of its entities. Aggregation and CEJBs can be synergistically used in the same application to increase overall execution speed.

Leff and Rayfield [14] show the importance of an EJB cache in a J2EE application server for improving application performance. We can find an in-depth study of performance issues with entity EJBs in [3]. The authors point out that caching is one of the greatest benefits of using entity EJBs provided that the bean cache is properly configured and entity EJB transaction settings are optimized.

The CEJB technique complies with the EJB specification and therefore can be applied to any J2EE application on any J2EE application server. Several *J2SE*-based technologies, from Java Data Objects (JDO) to Java Object Serialization (JOS), sacrifice the benefit of J2EE platform services in return for much higher performance than would be possible on a J2EE platform. Jordan [15] provides an extensive comparison of EJB data persistence and several J2SE-based data persistence mechanisms and their relative performance.

Trofin and Murphy [16] present the idea of collecting runtime information in J2EE application servers and to modify EJB containers accordingly to improve performance. CEJBs, on the other hand, do not change EJB containers but improve performance by multiplexing multiple logical entities into one entity as seen by the EJB container.

III. Consolidated EJBs

A. *CEJB Goal and Concept*

CEJBs are intended to narrow the performance gap between EJBs and POJOs in J2EE applications with large numbers of EJBs of the same class. A look at common operations during the life span of an EJB explains some of the performance differences between EJBs and POJOs:

- Creating EJBs entails the addition of rows in a table in the underlying relational database at transaction commit time, whereas POJOs exist in memory.
- Accessing EJBs requires the execution of *finder* methods to locate the EJBs in the bean cache of the J2EE application server or in the database, whereas access to POJOs is accomplished by simply following object references.
- Depending on the selected transaction commit options (pessimistic or optimistic), the execution of business methods on EJBs is either serialized or requires frequent synchronization with the underlying database. Calling POJO methods, on the other hand, simply means accessing objects in the Java heap in memory, possibly with application-specific concurrency control in place.
- Deleting EJBs also removes the corresponding database table rows at commit time. Deleting POJOs affects only the Java heap in memory.

The preceding list identifies the interaction between EJBs and the persistence mechanism as a performance bottleneck for EJBs that POJOs do not suffer from. The persistence mechanism includes the bean cache and the database. One way of decreasing the performance gap between EJBs and POJOs, therefore, is to increase the bean cache hit rate, thereby reducing the database access frequency. In case of bean cache misses and when synchronizing the state of EJBs with the database, we would like to speed up the search for the database table rows that represent EJBs. CEJBs are intended to significantly decrease the number of EJBs in a J2EE application. A smaller number of EJBs translates into higher bean cache hit rates *and* faster EJB access in the database due to a smaller search space in database tables for EJB *finder* operations. In other words, CEJBs reduce the number and execution times of database accesses by increasing the rate of in-memory search operations.

CEJBs are based on a simple idea. Traditionally, when developing EJBs we map each real-world entity in the application domain such as a user to a separate EJB. This approach can result in a large number of EJB instances in the application. With CEJBs, on the other hand, we consolidate multiple entities of the same type into a single “special” EJB. Specifically, we store up to N POJO entities in the same EJB (the CEJB), where N is an priori determined constant. Because N is determined at application design time, the CEJB-internal data structure for storing entities can be an array of size N. Hence, locating an entity within a CEJB can be accomplished through a simple array indexing operation requiring only constant time. The challenge for developing CEJBs is devising an appropriate mapping function $m:K_E \rightarrow K_C \times [0,N-1]$, where K_E is the primary key space of the entities and K_C is the primary key space of the CEJBs. Function *m* maps a given entity primary key *k*, for example a user ID, to a tuple (k_1, k_2) where

- k_1 is an artificial primary key for a CEJB that will store the entity,
- k_2 is the index of the array element inside the CEJB that stores the POJO with primary key *k*.

The mapping function *m* has to ensure that no more than *N* entities are mapped to the same CEJB. On the other hand, *m* also has to attempt to map as many entities to the same CEJB as possible. Otherwise, CEJBs would perform little or no better than EJBs. Moreover, the computation of *m* for a given entity primary key has to be fast.

B. Developing a CEJB

Consider a simple entity represented as an EJB *User* with the J2EE-mandated local home interface, local interface, and bean implementation:

- The local home interface is responsible for creating new *Users* through a method *create(String userID, String firstName, String lastName)* and finding existing ones through method *findByPrimaryKey(String userID).*
- The local interface allows a client to call getter and setter methods for the *firstName* and *lastName* properties of *Users*. It also contains a method *businessMethod(String firstName, String lastName)* with some business logic: the method simply assigns its parameters to the *firstName* and *lastName* properties of a *User*, respectively.
- The bean implementation is the canonical bean implementation of the methods in the local (home) interfaces. For the sake of brevity, we omit showing the (quite trivial) bean implementation here.

In Figures 1-4, we present a CEJB *CUser* that we derived from the *User* EJB. To arrive at *CUser*, we first map the persistent (CMP) fields in *User* to transient *String* arrays *firstNames* and *lastNames* and persistent *String* fields *encodedfirstNames* and *encodedlastNames*. Note that we do not implement *firstNames* and *lastNames* as *persistent* array fields. Instead, we encode *firstNames* and *lastNames* as persistent *Strings* *encodedFirstNames* and *encodedLastNames*, respectively, during *ejbStore* operations. To do so, *ejbStore* creates a #-separated concatenation of all elements of *firstNames* and one of all elements of *lastNames* where # is a special symbol that does not appear in first or last names. This technique allows us to store the first names and last names as VARCHARs in the underlying database and avoid the much less time-efficient storage as VARCHARs *for bit data* that persistent array fields require. During *ejbLoad* operations the *encodedFirstNames* and *encodedLastNames* are being demultiplexed into the transient arrays *firstNames* and *lastNames*, respectively. The *CUserBean* then uses the state of the latter two arrays until the next *ejbLoad* operation refreshes the state of the two arrays from the underlying database.

The *ejbCreate* method in Figure 3 assigns an *objectID* to the appropriate persistent field. We will discuss the choice of the *objectID* later. The method also allocates and initializes the transient *firstNames* and *lastNames* arrays. The size of the arrays is determined by the formal parameter *N*.

In the *CUser* local interface, we add an *index* parameter to all *getter* and *setter* methods and to the *businessMethod*. We also add the lifecycle methods *createUser* and *removeUser*. The *getter* and *setter* methods in *CUserLocal* have to be implemented by *CUserBean* because they are different from the abstract *getter* and *setter* methods in *CUserBean*. The new *getter* and *setter* methods access the indexed slot in the array fields *firstName* and *lastName*. Similarly, we have to change the *businessMethod*, which now accesses the indexed slot in the *firstName* and *lastName* fields rather than the entire EJB state. The *createUser* method first ensures that the indexed slots in the *firstNames* and *lastNames* are empty. If not, this user has been added before and a *DuplicateKeyException* is raised. If the slots are empty, *createUser* will assign the state of the new user to the indexed slots in the arrays. The *removeUser* method ensures that the indexed *firstNames* and *lastNames* slots are not empty, i.e., the referenced user is indeed stored in this *CUser*. If so, *removeUser* deletes the state of this user from the *firstNames* and *lastNames* arrays.

Figure 5 shows a class *ObjectIDMapping* that encapsulates an exemplary mapping function *m* from *User* primary keys (*Strings*) to *CUser* primary keys (*objectIDs*). Figure 6 contains an example of retrieving a *CUser* through an *ObjectIDMapping* and executing the *businessMethod* on the retrieved *CUser*. The only argument for the constructor of an *ObjectIDMapping* is *N*, the maximum number of entities consolidated in a *CUser*. The mapping function *m* is computed in the *setObjectID* method. This method maps a *User* primary key, *objectIDArg*, to the tuple (*objectID*, *index*). The *objectID* is derived from *objectIDArg* by replacing *objectIDArg's* last character *c* (viewed as an integer) with $c - index$. The value of *index* is the result *of c modulo N*, i.e., $c = q \cdot N + index$ where $0 \leq index < N$ and *q* is the integer quotient of *c* and *N*. While the *objectID* identifies the *CUser* in which we store an entity with *objectIDArg* as its primary key, the *index* identifies the slots in the CMP array fields in *CUser* that store the given entity. Although our definition of *m* is somewhat complex, its computation is fast and it maps at most *N* entities to each *CUser*, which is a key requirement for *m*.

C. Design Considerations for CEJBs

By creating a simple façade session bean we can completely hide *CUsers* from the rest of the application and expose only *POJO* entities to clients. With a façade session bean, the two-step process of first retrieving a *CUser* and subsequently accessing a *POJO* entity shown in Figure 6 is reduced to one step. The façade bean is straightforward and therefore we do not show it here. For more complicated

```
public interface CUserLocalHome extends EJBLocalHome {
    CUserLocal create(String objectID, int numElements) throws CreateException;
    CUserLocal findByPrimaryKey(String objectID) throws FinderException;
    CUserLocal getUser(String objectID, int numElements) throws FinderException;
}
```

Figure 1. Local home interface for *CUser*.

```
public interface CUserLocal extends EJBLocalObject {
    void createUser(int index, String firstName, String lastName) throws DuplicateKeyException;
    void removeUser(int index) throws RemoveException;
    String getFirstName(int index);
    void setFirstName(int index, String firstName);
    String getLastName(int index);
    void setLastName(int index, String lastName);
    void businessMethod(int index, String firstName, String lastName);
}
```

Figure 2. Local interface for *CUser*.

```
public abstract class CUserBean implements EntityBean {
    private transient String[] firstNames = null;
    private transient String[] lastNames = null;
    public abstract String getObjectID();
    public abstract void setObjectID(String objectID);
    public abstract String getEncodedFirstNames();
    public abstract void setEncodedFirstNames(String encodedFirstNames);
    public abstract String getEncodedLastNames();
    public abstract void setEncodedLastNames(String encodedLastNames);

    public String ejbCreate(String objectID, int N) throws CreateException {
        setObjectID(objectID);
        firstNames = new String[N];
        lastNames = new String[N];
        for (int index = 0; index < N; index++) {
            firstNames[index]= null;
            lastNames[index] = null;
        }
        return null;
    }

    public void ejbLoad() {
        StringTokenizer encodedFirstNames = new StringTokenizer(getEncodedFirstNames(), “#”),
                        encodedLastNames = new StringTokenizer(getEncodedLastNames(), “#”);
        int numElements = encodedFirstNames.countTokens();
        if (firstNames == null) {
            firstNames = new String[numElements];
            lastNames = new String[numElements];
        }
        for (int index = 0; index < numElements; index++) {
            firstNames[index] = encodedFirstNames.nextToken();
            lastNames[index] = encodedLastNames.nextToken();
        }
    }

    public void ejbStore() {
        StringBuffer encodedNames = new StringBuffer();
        for (int index = 0; index < firstNames.length; index++) {
            encodedNames.append(firstNames[index]);
            encodedNames.append(“#”);
        }
        setEncodedFirstNames(encodedNames.toString());
        encodedNames.setLength(0);
        for (int index = 0; index < lastNames.length; index++) {
            encodedNames.append(lastNames[index]);
            encodedNames.append(“#”);
        }
        setEncodedLastNames(encodedNames.toString());
    }
```

Figure 3. Methods in *CUserBean* relevant to the CEJB discussion, part I.

```
    public void createUser(int index, String firstName, String lastName) throws DuplicateKeyException {
        if (!(firstNames[index] == null && lastNames[index] == null)) throw new DuplicateKeyException("User exists already");

        firstNames[index] = firstName;
        lastNames[index] = lastName;
    }

    public void removeUser(int index) throws RemoveException {
        if (firstNames[index] == null || lastNames[index] == null) throw new RemoveException("User does not exist");
        firstNames[index] = " ";
        lastNames[index] = " ";
    }

    public void businessMethod(int index, String firstName, String lastName) {
        firstNames[index] = firstName;
        lastNames[index] = lastName;
    }

    public void setFirstName(int index, String firstName) {
        firstNames[index] = firstName;
    }

    // other getter/setter methods go here…
}
```

Figure 4. Methods in *CUserBean* relevant to the CEJB discussion, part II.

```
public class ObjectIDMapping {
    private int N,
                index;
    private String objectID;

    public ObjectIDMapping(int N) {
        this.N = N;
        index = -1;
        objectID = null;
    }

    public void setObjectID(String objectIDArg) {
        int lastElementIndex = objectIDArg.length() - 1,
            lastCharacter = objectIDArg.charAt(lastElementIndex);

        index = lastCharacter % N;
        objectID = objectIDArg.substring(0, lastElementIndex) + (lastCharacter - index);
    }

    public int getIndex() {
        return index;
    }

    public String getObjectID() {
        return objectID;
    }
}
```

Figure 5. Class for mapping *User* primary keys to *CUser* primary keys and array index slots.

```
ObjectIDMapping idMapping = new ObjectIDMapping(N);
idMapping.setObjectID("rKlemm");
CUserLocal  cUser = cuserLocalHome.findByPrimaryKey(idMapping.getObjectID());
cUser.businessMethod(idMapping.getIndex(), "Reinhard", "Klemm");
```

Figure 6. Accessing a *CUser* EJB.

entities than *Users*, consolidation through CEJBs requires more effort but is straightforward and could be supported by a tool. Ideally, such a tool would be offered as part of a J2EE development environment and convert EJBs into CEJBs at the request and under the directions of the developer. The tool would also need to support the following scenarios:

- If *User* implements customized *ejbLoad*, *ejbStore*, *ejbActivate*, or *ejbPassivate* methods, these need to be adapted in *CUserBean* to reflect the fact that the state of a *User* is stored across different arrays in the *CUserBean*.
- *Finder* and *select* queries for *User* must be re-implemented for the CEJB because they need to access both a *CUser* and the arrays within a *CUser*.
- If *User* has customized *ejbHome* methods, we need to add functionally equivalent *ejbHome* methods to

CUser. Changes to the original *User ejbHome* methods are only necessary if these methods access the state of a specific *User* EJB after a prior *select* method. In this case, the *CUser ejbHome* methods need to retrieve *POJO* entities instead of *Users*.

- If *User* is part of a container-managed relationship (CMR), consolidation through CEJBs requires removal of the CMRs and manual re-implementation of the CMRs without direct J2EE support.

The mapping function *m* has a strong impact on the performance of CEJBs and therefore needs to be defined carefully for the given application. The mapping function delivers its best performance if primary keys that occur in the application are *clustered*. Clustering here means that for every primary key *k* in the application there is a set of roughly *N* primary keys for other entities in the application that are similar enough to *k* to be mapped to the same *objectID* by *m*. The challenge is therefore to analyze the actual key space of the entities that are to be consolidated in a given application and to then define an efficient and effective mapping function based on this analysis.

IV. Performance Evaluation

A. Methodology

We compared the performance of *Users* and *CUsers* in a J2EE test application. It uses the mapping function *m* in Figure 5 because this function clusters the primary keys that we chose for the entities in the test application - lexicographically consecutive strings - to facilitate the generation of a large number of user entities. The test application executes a sequence of operations either on *Users* (*EJB mode*) or *CUsers* (*CEJB mode*). In EJB mode, the application executes the following sequence of steps:

1. Create *n User* EJBs.
2. Find *User* EJB with randomly selected primary key and read its state through *getter* operations. Repeat *n* times.
3. Find *User* EJB with randomly selected primary key and execute *businessMethod* on it, thus changing the EJB state. Repeat *n* times.
4. Delete all *User* EJBs through EJB *remove* operations.

Between any two consecutive steps, the test application creates 20000 unrelated EJBs in order to introduce as much disturbance as possible in the application server bean cache and in the connection to the underlying database. During our performance testing, however, it turned out that these cache disturbance operations had a negligible effect on the performance *differences* between the CEJB and EJB modes.

In CEJB mode, the application performs the same steps on *CUsers* instead of *Users*. Also, in step 4 in CEJB mode, the application sequentially deletes all entities in each CEJB but not the CEJB itself. We varied the maximum number *N* of entities per CEJB, from *2* to *250* in consecutive runs of the test application. The performance of the test application peaked around *N=20*. We only present the performance results for *N=20*.

We configured the test application with two different transaction settings in two different experiments: in *long transaction mode*, each step of the test application is executed in one long-lived transaction. In *short transaction mode*, the application commits every data change as soon as it occurs, i.e., after each entity creation, change, or deletion. Here, the application performs a large number of short-lived transactions. In successive runs of the test application, *n* iterated over the set {*1000*, *10000*, *50000*}. After each run, we restarted the database server and the application server and deleted all database rows created by the application.

We deployed the test application on an IBM WebSphere 5.1.1.6 J2EE application server with default bean cache and performance settings. The hardware is a dual Xeon 2.4 GHz server running Microsoft Windows 2000 Server. An IBM DB2 8.1.9 database provides the data storage. All EJBs use the WebSphere default commit option C.

B. Performance Analysis

Figures 7-12 display the results of our performance testing with the test application in long and short transaction modes for the three different values of *n*. The speedup in the figures is defined as the time for an EJB operation divided by the time for the equivalent CEJB operation. Speedup values greater than 1 indicate results where CEJBs outperform EJBs, values of less than 1 indicate EJBs performing better than CEJBs. In long transaction mode, CEJBs significantly outperformed EJBs. For *n=50000*, for example, creating users with CEJBs was more than twice as fast as with EJBs, finding and reading users was more than 5 times faster, finding and changing users was more than 7 times faster, and deleting users with CEJBs was more than 14 times faster.

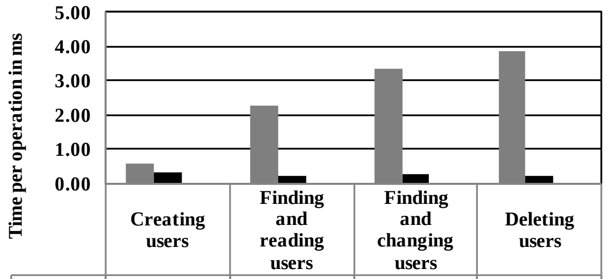
Because the mapping function *m* in our test application clusters the primary keys of the user entities, the CEJBs consolidate almost the maximum possible number of entities (20 per our definition of *N*). Hence, the number of CEJBs necessary to store all user entities in the test application is about *1/20th* that of the number of EJBs in EJB mode, which translates into much improved application server caching behavior and accelerated database search times. Once a CEJB has been retrieved, extracting the desired entity from the CEJB is a simple and fast array indexing operation. However, if the chosen mapping function *m* for a given application does not achieve the cluster property, CEJBs may lose some of their performance advantage over EJBs.

In CEJB mode, entity deletion does not force the deletion of EJBs in the application server or the database. Instead, entity deletion in CEJBs is accomplished through the removal of entities *inside* EJBs. Not surprisingly therefore, deleting users in CEJB mode is much faster than in EJB mode where an EJB needs to be removed in the application server bean cache and the underlying database.

In short transaction mode, our performance testing showed a very different outcome. Here, CEJBs only offer performance advantages over EJBs for finding and reading users operations. CEJBs are about as fast as EJBs during finding and changing of users and during deletion of users but much slower in creating users. In short transaction mode, transaction commits after EJB state changes dominate the execution time of the test application and void many performance advantages due to consolidation. Hence, J2EE applications that eagerly commit every EJB state change will experience a significant speed-up as a result of consolidation only if the EJB read to write ratio is very high.

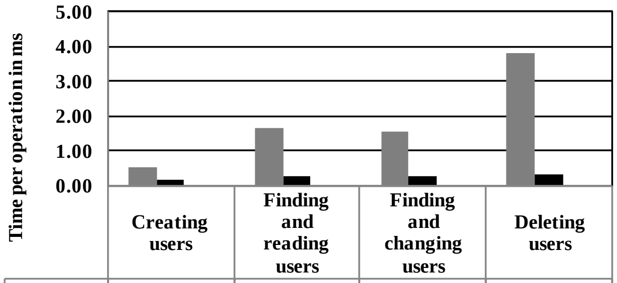
In conclusion, CEJBs provide strong performance advantages over EJBs in a J2EE application if (1) the application contains a large number of EJBs, (2) it accesses EJBs either in long-lived transactions or in short-lived transaction with a large EJB read to write ratio, and (3) if a mapping function *m* can be found for the EJB key space that exhibits the cluster property.

Our test application is designed to execute a large number of common EJB operations in a repeatable fashion. As such, the test application is somewhat artificial. It does not involve human interactions and arbitrary timing delays due to human input. The pattern of EJB operations is highly regular and maximizes EJB accesses, whereas other J2EE applications may have irregular EJB accesses and also contain computationally or I/O-intensive tasks. Our *User* EJBs are simple while EJBs in common J2EE applications can be more complex and linked to each other. However, we believe that our test application realistically captures the performance *differences* between EJBs and CEJBs in a large class of J2EE applications that are characterized by large numbers of entities, a high frequency of EJB accesses with a large degree of regularity (e.g., certain data mining applications such as our Mercury system), and a predictable and regular primary key space for the entities.



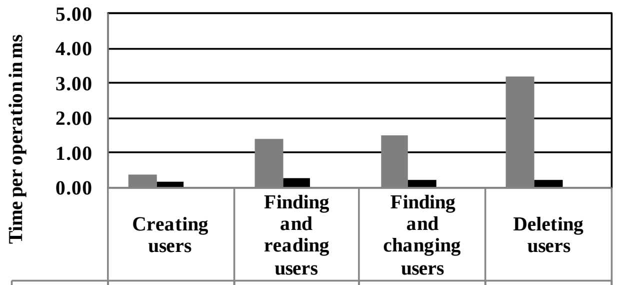
	Creating users	Finding and reading users	Finding and changing users	Deleting users
■ EJBs	0.55	2.25	3.38	3.88
■ CEJBs	0.30	0.23	0.28	0.20
Speedup	1.84	9.62	12.01	19.00

Figure 7. Test application performance: long transaction mode, n=1000.



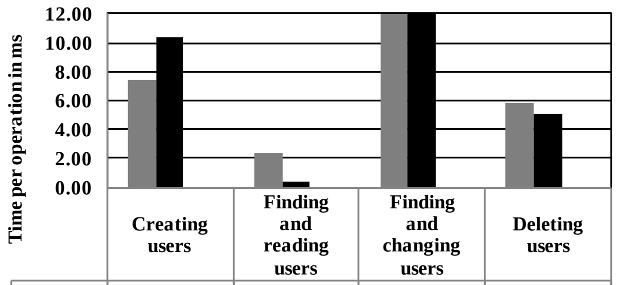
	Creating users	Finding and reading users	Finding and changing users	Deleting users
■ EJBs	0.50	1.67	1.56	3.79
■ CEJBs	0.16	0.27	0.27	0.32
Speedups	3.02	6.14	5.72	11.84

Figure 8. Test application performance: long transaction mode, n=10000.



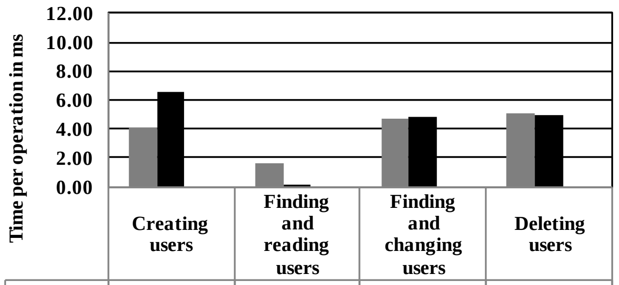
	Creating users	Finding and reading users	Finding and changing users	Deleting users
■ EJBs	0.39	1.38	1.48	3.21
■ CEJBs	0.17	0.24	0.20	0.23
Speedup	2.24	5.75	7.58	14.28

Figure 9. Test application performance: long transaction mode, n=50000.



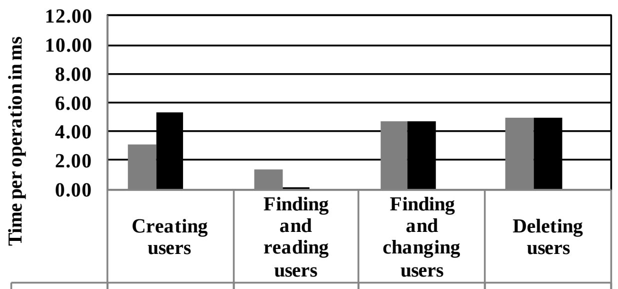
	Creating users	Finding and reading users	Finding and changing users	Deleting users
■ EJBs	7.41	2.31	12.05	5.81
■ CEJBs	10.34	0.36	12.41	5.08
Speedup	0.72	6.44	0.97	1.14

Figure 10. Test application performance: short transaction mode, n=1000.



	Creating users	Finding and reading users	Finding and changing users	Deleting users
■ EJBs	4.14	1.66	4.72	5.06
■ CEJBs	6.52	0.18	4.84	4.94
Speedups	0.64	9.05	0.98	1.02

Figure 11. Test application performance: short transaction mode, n=10000.



	Creating users	Finding and reading users	Finding and changing users	Deleting users
▨ **EJBs**	**3.08**	**1.34**	**4.69**	**5.01**
■ **CEJBs**	**5.35**	**0.18**	**4.75**	**4.96**
Speedup	**0.58**	**7.34**	**0.99**	**1.01**

Figure 12. Test application performance: short transaction mode, n=50000.

V. CONCLUSION AND FUTURE WORK

We presented a J2EE software design pattern that consolidates multiple entities in J2EE applications into special-purpose entity EJBs that we call consolidated EJBs (CEJBs). Consolidation increases the locality of data access in J2EE applications, thus making bean caching in the application server more effective and decreasing search times for entity EJBs in the underlying database. In J2EE applications with large numbers of EJBs, CEJBs can therefore greatly increase the overall application performance. Using a test application we showed that CEJBs can outperform traditional EJBs by a wide margin for common EJB operations. For example, the CEJB equivalent of an EJB *findByPrimaryKey* operation is more than five times faster in one of our experiments, and the execution of a data-modifying business method on an EJB is more than seven times faster in CEJBs. CEJBs conform to the EJB specification and can therefore be used in any J2EE application on any J2EE application server.

We have three future research goals for CEJBs. First, we would like to modify CEJBs in such a way that applications with short-lived transactions and a small ratio of EJB read to EJB write operations perform better than our current solution. Secondly, we intend to investigate mapping functions for CEJBs that (1) perform well if the primary key space for EJBs is irregular or unpredictable, and (2) that can be automatically defined without requiring complex developer decisions. Thirdly, we would like to address a currently open question for our CEJB design pattern, which is how to adjust CEJBs so that they are beneficial in most J2EE applications and thus could ultimately become a standard way of implementing entities in J2EE applications.

REFERENCES

[1] Oracle Inc., “Enterprise JavaBeans Specification 2.1,” retrieved September 28, 2012, from http://bit.ly/Ovip59.

[2] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W, Zwaenepoel, “Performance Comparison of Middleware Architectures for Generating Dynamic Web Content,” Lecture Notes in Computer Science, Vol. 2672, Jan. 2003, pp. 242-261.

[3] S. Kounev and A. Buchmann, “Improving Data Access of J2EE Applications by Exploiting Asynchronous Messaging and Caching Services,” Proc. 28th International Conference on Very Large Databases (VLDB), Aug. 2002, retrieved September 28, 2012, from http://bit.ly/QgduUf.

[4] Oracle Inc., “Enterprise JavaBeans Specification 3.1,” retrieved September 28, 2012, from http://bit.ly/SlMyPN.

[5] S. Pugh and J. Spacco, “RUBiS Revisited: Why J2EE Benchmarking is Hard,” Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Oct. 2004, pp. 204-205.

[6] M. Raghavachari, D. Reiner, and R. Johnson, “The Deployer’s Problem: Configuring Application Servers for Performance and Reliability,” Proc. 25th International Conference on Software Engineering ICSE ’03, May 2003, pp. 484-489.

[7] S. Ran, P. Brebner, and I. Gorton, “The Rigorous Evaluation of Enterprise Java Bean Technology,” Proc. 15th International Conference on Information Networking (ICOIN), IEEE Computer Society, Jan. 2001, pp. 93-100.

[8] S. Ran, D. Palmer, P. Brebner, S. Chen, I. Gorton, J. Gosper, L. Hu, A. Liu, and P. Tran, “J2EE Technology Performance Evaluation Methodology,” Proc. International Conference on the Move to Meaningful Internet Systems 2002, pp. 13-16.

[9] E. Cecchet, J. Marguerite, and W. Zwaenepoel, “Performance and Scalability of EJB Applications,” Proc. 17th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Application (OOPSLA), Nov. 2002, retrieved September 28, 2012, from http://bit.ly/Qge98e.

[10] D. Alur, J. Crupi, and D. Malks, “Core J2EE Patterns,” Prentice Hall/Sun Microsystems Press, Jun. 2001.

[11] F. Marinescu, “EJB Design Patterns: Advanced Patterns, Processes, and Idioms,” John Wiley & Sons Inc., Mar. 2002.

[12] J. Rudzki, “How Design Patterns Affect Application Performance – A Case of a Multi-Tier J2EE Application,” Lecture Notes in Computer Science, No. 3409, Springer-Verlag, 2005, pp. 12-23.

[13] C. Larman, “The Aggregate Entity Bean Pattern,” Software Development Magazine, Apr. 2000, retrieved September 28, 2012, from http://bit.ly/PgBoxe.

[14] A. Leff and J. T. Rayfield, “Improving Application Throughput with Enterprise JavaBeans Caching,” Proc. 23rd International Conference on Distributed Computing Systems (ICDCS), May 2003, pp. 244-251.

[15] M. Jordan, “A Comparative Study of Persistence Mechanisms for the Java Platform,” Sun Microsystems Technical Report TR-2004-136, Sep. 2004, retrieved September 28, 2012, from http://bit.ly/U3GGPf.

[16] J. Trofin and J. Murphy, “A Self-Optimizing Container Design for Enterprise Java Beans Applications,” 8th International Workshop on Component Oriented Programming (WCOP), Jul. 2003, retrieved September 28, 2012, from http://bit.ly/O4biAD.

Explicit Use of Working-set Correlation for Load-balancing in Clustered Web Servers

Stoyan Garbatov and João Cachopo
INESC-ID Lisboa / Instituto Superior Técnico, Technical University of Lisbon
stoyangarbatov@gmail.com and joao.cachopo@ist.utl.pt

***Abstract*— This work presents a new load balancing policy for clustered web server systems. With this policy, every node in the cluster is responsible for dealing with a particular subset of request types. All requests are partitioned into disjoint groups, according to the domain data contained in their working sets. The composition of the working sets is established by using automatic data access pattern analysis and prediction techniques. Latent Dirichlet Allocation is employed as the partitioning algorithm to maximize the similarity (correlation) between the working sets of the requests placed in the same group. The working-set correlation is used explicitly with the aim of improving the data locality of all functionality that is to be executed on a given node, leading to more efficient use of computational resources and, ultimately, to increased performance. The work was validated on the TPC-W benchmark.**

Keywords-clustered web servers; load balance; locality awareness; Latent Dirichlet Allocation; scalability; performance.

I. Introduction

Scalability is a crucial property for many web systems. A system is defined as scalable if it is possible to change it (e.g., by adding more hardware) when the volume of requests increases, so that it maintains the same performance (in terms of throughput, response time, etc.). It takes some careful engineering to achieve good scalability. The solutions usually consist in designing the system in a particular way so that whenever some of the supporting (software or hardware) resources are upgraded, the system will be able to deal transparently with growing workloads without compromising its performance.

In the following, we use a classification of the existing approaches that is similar to the one presented by Cardellini et al. [5]. The simplest alternative for improving the performance of a web system is to upgrade the machine that is running the server to a machine with better specifications (e.g., processing units, disks, etc), which is referred to as hardware scale-up [7]. Unfortunately, this is a rather short-term and not very cost-effective approach because the increase of the workload, often driven by the increase of clients, far outpaces the hardware performance growth that is viable to be achieved for a single machine.

Many researchers concentrated their efforts on improving the performance of the server at the software level (software scale-up). Among these are improving the server's operating system [13, 14], developing more efficient web servers [16], and designing alternative request scheduling policies [6, 3]. Unfortunately, similarly to its hardware counterpart, improving the software performance of a single server is not a long-term solution to the web scalability issue.

Another class of approaches considers distributed systems composed of multiple servers. The main objective of this type of solutions is to spread the workload of the incoming requests among the existing server nodes, attempting to maximize the usefulness of the available resources. The policy for distributing the requests is usually performed by a single component called the load balancer. Any web system solution that achieves scalability and better overall performance by means of multiple server nodes is referred to as a scale-out approach [7].

These can be further refined into global and local scale-out solutions. The global scale-out approaches are characterized by having server nodes placed in geographically distinct locations, whilst a local scale-out has all nodes in a single network.

Our work falls into the category of the local scale-out approaches referred to as cluster-based web systems. These approaches are characterized by having a single entry point (the load-balancer), which is the only visible component from the client point-of-view. This is done for transparency purposes, so that the clients do not need to be aware of the potential multiplicity of entities that are effectively processing their requests, thereby avoiding all issues that such knowledge would entail.

The great majority of research in this field is concerned with the distribution of the workload in a uniform fashion among all server nodes. However, it has also been unequivocally demonstrated in [15], [1], and [8] that it is possible to achieve significant performance gains if the routing algorithm attempts to improve the data locality of server nodes when requests are being processed. In fact, it is well-known that to have good performance, a system should exploit and maximize its data locality.

In this work, we do not attempt to distribute the workload uniformly among server nodes. There are plenty of already existing solutions that can be combined into the solution that we propose here to give it uniform workload distribution properties. Instead, in this work we concentrate on the issue of improving the efficiency and performance of clustered dynamic content web systems, by developing a request routing algorithm that explicitly takes into account the existing correlation between the working sets of incoming requests, distributing them in a way that tries to maximize the data locality of operations performed at server nodes.

With the solution proposed here, every server node processes only a particular subset of request types: The requests are partitioned into disjoint groups, based on the contents of their working sets.

We identify the composition of the working set of each request type with the help of automatic data access pattern analysis and prediction routines. Then, we use the Latent Dirichlet Allocation partitioning algorithm to maximize the correlation between the working sets of all requests placed in a particular group. This is done to improve the data locality of all operations that are to be executed on a given server node, leading to a more efficient use of system resources and improved performance.

The contributions of this work are twofold. First, the explicit use of the correlation between request working sets is new and there is no previously existing work that makes request distribution decisions in a similar manner. Second, the system that we propose is entirely autonomous and self-sufficient in its operation,: from the analysis of the composition of the requests' working sets, up to the distribution of the incoming requests among operating server nodes, there is no point where human intervention is necessary in the decision making process.

The article is organized as follows. Section II discusses related work. Section III describes our new proposal in detail. Section IV presents the benchmark that we used to evaluate the new proposal and discusses the results obtained. Finally, Section V presents some concluding remarks.

II. Related Work

Scalability is an essential property for web systems that are expected to deal with a large volume of traffic and extensive variations in the number of clients. Thus, it is not surprising the sheer volume of research that has been performed in this domain. Given the scope of our work, in this section we limit our discussion of related work to research regarding request distribution for clustered web systems. Cardellini et al. [5] performed a comprehensive study of locally clustered systems, whereas Amza et al. [2] evaluated transparent scaling approaches tailored for dynamic content systems.

Pai et al. [15] introduced the concept of locality-aware request distribution (LARD). The main idea behind this concept is that for a clustered web server system to have good scalability and operate efficiently, the load-balancing policy should take into account the content associated with incoming requests and redirect them so that the data locality of the server node responsible for processing them is improved. They use a hash function to partition the functionality provided by a given system (under the form of the types of requests that are available). The load-balancer uses this information to redirect requests so that every available server node is responsible for processing requests that belong only to a certain partition. This approach decreases the working-sets of all nodes to a portion of the system working-set, allowing for improved data locality and scalability. Pai et al. [15] performed simulations and validation on a working prototype, demonstrating that, through a locality-aware request distribution approach, it is possible to achieve significantly better scalability and performance than policies that do only uniform load distribution among nodes.

The main issue that can be pointed out in the work of Pai et al. [15] is that the partitioning of request types among server nodes is performed by a hash function. The fact that the partitioning mechanism is oblivious to the domain data accessed when processing requests does not give any guarantees as to the similarity of the types of requests redirected to a given node, in terms of the data necessary for their execution. Whereas such an approach may still lead to smaller-than-system working-sets at most nodes, it is far from optimal, because requests whose working-sets do not intersect can be placed in the same partition, degrading the overall locality of data. To maximize data locality, the working-sets of all requests placed in the same partition should overlap as much as possible, so that when a new request arrives at a node, it is more likely that most (if not all) of the data necessary for its successful completion is already available. This cannot be achieved without explicitly accounting for the relation between the functionality and the data needed for its execution.

Zhang et al. [19] present a detailed simulation study of AdaptLoad, which is a self-adjusting load-balancing policy that takes into account observed workload variations to tune its control parameters. With AdaptLoad, server nodes are responsible for dealing with requests that have similar sizes (e.g. processing times), seeking to minimize the overall job slowdown by separating the execution of differently sized tasks. The authors evaluate their approach against previously existing load-balancing solutions and demonstrate positive results, in particular when target systems are subject to highly dynamic load conditions.

The work of Amza et al. [1] presents a novel lazy replication technique, intended for scaling database back-ends of dynamic content site applications operating on top of computer clusters. This approach is referred to as conflict-aware scheduling and provides throughput scaling and one-copy serializability. This technique exploits the fact that, in the context of database clusters, there is a scheduler responsible for processing all incoming requests. By making use of information regarding the domain data accessed within transactions, Amza et al. [1] developed a conflict-aware scheduler that provides one-copy serializability, as well as reducing the rate at which conflicts occur. This is achieved by guiding incoming requests to nodes based on the data access patterns that are expected to occur during the execution of the associated transactions. Yet, there is a drawback of this approach that severely impairs its practicality: Programmers are responsible for providing information (under the form of a manually added tip) at the beginning of each transaction about which domain data is going to be accessed during its execution. Instead, we claim that the system should be capable of performing an automatic identification of the data access patterns that take place during transactions.

The work of Elnikety et al. [8] introduced a memory-aware load balancing method for dispatching transactions to replicas in systems employing replicated databases. The algorithm uses information about the data manipulated in transactional contexts with the goal of assigning transactions to replicas so as to guarantee that all necessary data for their execution is in memory, thereby reducing disk I/O. For guiding the load balancing technique, the authors developed an auxiliary approach for estimating the volume and type of data manipulated during transactions. An additional contribution of their work is an optimization designated update filtering for decreasing the overheads due to the propagation of updates between replicas.

Zhong et al. [20] performed a study of the improvements that can be achieved by placing data when taking into account correlation information. They proposed a polynomial-complexity algorithm for calculating object placement that achieves a close to optimal solution with regards to minimizing communication costs. Further optimizations of the algorithm are indicated, by concentrating upon a small set of higher-importance objects. The approach is evaluated and demonstrated to produce significant reductions of communication overheads.

Given the existing related work described so far, we may draw some conclusions. Request distribution policies emphasizing data locality appear to offer much better scalability and performance gains, when compared against approaches concentrating only on load distribution. Uniform load distribution is still important and should be used to complement locality aware policies, but should not be the main goal. Despite all the good work done in the area of locality-aware load distribution, existing approaches have (at least) one of the following shortcomings:

- In their search for improved data locality, they do not take into account explicitly the data access patterns performed during execution of requests, either because they lack a proper analysis of data usage or because they expect that locality emerges “naturally” when requests are distributed among server nodes without taking into account the data manipulated in their contexts.
- The analysis of which data access patterns occur is performed manually.
- The clustering of requests/functionality among server nodes is performed manually.

The current state-of-the-art in the area of load distribution has ample room for improvements, which we explore with the solution proposed in this paper.

III. System Description

Our system is composed of three main modules – a data access pattern analysis module, an optimal clustering module, and a request distribution module.

The first module is responsible for identifying the composition of the working-sets associated with all types of requests that are executed by the system under consideration. This is achieved by analyzing and predicting the behavior of the target application in terms of the data access patterns that occur throughout the application’s execution contexts (such as methods and services). The analysis and prediction can be performed by means of one of three alternative stochastic model implementations, namely: Bayesian Updating [12], Markov Chains [10], and Criticality Analysis [9] (each of these references contains a thorough description and discussion of the implementation, functionality, and properties of the models). All three implementations provide highly accurate results that characterize the contents of the working set of any unit of functionality present in an application. The process of collecting behavioral data as input for this analysis is performed in an online fashion and incurs an average of 5% to 8% overhead in comparison with the original version of the target application performance. The relatively low overhead makes it feasible to gather this information even while the application is operating normally. It should be noted that all modifications necessary for the acquisition of the behavioral data are performed in a completely automated manner by the system presented here. The composition of the working-sets, as acquired by the first module, is then supplied as input to the optimal clustering module.

The second module, which has been described in more detail in [11], is responsible for identifying the optimal clustering of the target application's functionality (which, in this particular case, is characterized by the request types that the application provides) and domain data (working-set composition), based on the data access pattern behavior observed at runtime.

The algorithm used to perform the partitioning of application request types is the Latent Dirichlet Allocation (LDA) [4], which corresponds to the current state-of-the-art in multivariate clustering algorithms. By providing the composition of the working-sets associated with application request types as input to the LDA, the algorithm is capable of grouping the requests into several clusters. The partitioning is performed so that all members of a given cluster have the strongest possible (positive) correlation among themselves. For the current work, cluster elements are application request types, which are characterized by their working-sets. This leads to LDA generating as output clusters of request types, whose working-sets are very similar, because of their positive correlation.

However, there are several control parameters (among which is the number of clusters into which the elements should be grouped) that need to be provided as input to the clustering algorithm, and, thus, the only guarantee that LDA provides is that the cluster output composition maximizes the correlation between cluster elements, but only for the particular set of control parameters given. Unfortunately, there is no way to know, a priori, what are the control values that would lead to the best possible results. So, to identify the set of control parameters that leads to the highest quality results, we use the Silhouette technique [17]. Intuitively, good clusters have the property that cluster elements are (conceptually) close to each other and far from the elements of other clusters. The Silhouette technique captures this notion and provides an indicator value of how good a particular clustering is.

To find the optimal values of the control parameters for the LDA, our system calculates the average Silhouette values from several executions of the LDA algorithm for different configurations of the control parameters, within their valid range of values. Once the Silhouette coefficients are available for all the evaluated scenarios, the values of the control parameters that produced the highest coefficient correspond to the optimal input scenario leading to the best possible clustering. The cluster results obtained after this step are provided as input to the request distribution module.

The request distribution module, which is where the new distribution policy is enforced, was implemented as a software switch, corresponding to Layer 7 (application) of the OSI protocol stack.

The switch acts as the sole entry point through which all incoming requests pass, before being distributed to a particular server node. It is the only visible component of the system, from the client point of view, making the multiplicity of application server nodes transparent to the clients.

Whenever a request arrives at the switch, it is examined, determining its type. The output provided by the optimal clustering module is consulted to determine the group to which this particular request belongs. Once this has been established, the request is forwarded to a server node that is responsible for processing requests of that particular group. After the request has been processed, the result is returned to the switch, and then forwarded back to the client.

As long as the target application supports having multiple instances of itself operating in parallel, without compromising application consistency, there is no need to make any modifications to the application for it to benefit from the request distribution functionality provided by the switch. The server nodes themselves are not aware of the presence of the switch, acting as if they were receiving their requests directly from the client.

The benefits of making sure that every server node deals only with request types that belong to a single group are the following. The effective working-set at server nodes should be significantly lower than the overall system working-set, not only because just a subset of all system functionality will be processed there, but also because that functionality was specifically selected with the purpose of maximizing the similarity of the composition of the working sets. This leads to more efficient use of computational resources (e.g. smaller memory footprints) and to better data locality, resulting ultimately in improved overall performance.

It can be argued that having all incoming requests and outgoing results going through the switch may be a performance and scalability bottleneck. However, we made this design decision to keep an implementation aspect of the proposed solution as simple as possible, because it is not in the request hand-off protocol that the main contribution of this work resides. Nevertheless, the switch implementation supports replication. So, multiple switches may be arranged in a hierarchical structure for the purpose of providing added performance or availability, at the cost of some added latency. This particular aspect of the work shall not be discussed any further.

IV. Results

To validate the effectiveness of our approach, we used the TPC-W benchmark [18]. The TPC-W benchmark specifies an e-commerce workload that simulates the activities of a retail store website, where emulated users can browse and order products from the website.

The main evaluation metric used in our experiments is the number of web interactions per second (WIPS) that can be sustained by the system under test. The benchmark execution is characterized by a series of input parameters. The first of these indicates the type of workload, which varies the percentage of read and write operations that is to be simulated by the emulated browser (EB) clients. Three types of workload are considered here: Type1, with 95% read and 5% write operations; Type2, with 80% read and 20% write operations; and Type3, with 50% read and 50% write operations.

The analysis of the system was performed with the benchmark executing in Type2 mode. The same profiling results (from Type2) were employed for all 3 workload configurations (Type1, Type2, and Type3) in the performance testing phase.

The remaining input parameters for the benchmark were as follows: the number of EBs was fixed at 10; we used a ramp-up time of 300 seconds; the measurement was performed for 1200 seconds after the ramp-up time; the ramp-down time was 120 seconds; the number of book items in the database varied between 1k, 10k, and 100k; and the think time was set to 0, ensuring that the EBs do not wait before making a new request. All results were obtained as the average of 4 independent executions of the benchmark, with the same configurations. The EBs, the request distribution switch, and the benchmark servers were always running on the same physical machine.

All of the performance measurements were made with the benchmark running on a machine equipped with 2x Intel Xeon E5520 (a total of 8 physical cores with hyper-threading running at 2.26 GHz) and 24 GB of RAM. Its operating system was Ubuntu 10.04.3, and the JVM used was Java (TM) SE Runtime Environment (build 1.6.0 22-b04), Java HotSpot (TM) 64-Bit ServerVM (build 17.1-b03, mixed mode). The benchmark server nodes, as well as the request distribution switch, were run on top of instances of Apache Tomcat 6.0.24, with the options set to "-server -Xms64m -Xmx${heapSize}m -Xshare:off -XX: +UseConcMarkSweep GC -XX:+AggressiveOpts".

The performance results achieved when running the TPC-W benchmark with three different request distribution policies, and with three and four server nodes, shall be presented and discussed next.

The first policy corresponds to the new policy developed with this work. The second policy corresponds to an idealized locality-aware request distribution (LARD), where each server node is responsible for processing a subset of request types, and there are no intersections among the sets of request types assigned to different server nodes. This policy is idealized because it assumes prior knowledge of the composition of the workload, in terms of the relative proportions of incoming request types, as well as the average time that requests of a given type take to be processed. The policy attempts to achieve the most uniform load distribution possible, whilst keeping every server node dedicated to processing a fixed subset of request types. The third policy employs a classic (unweighted) round-robin approach for distributing incoming requests among existing server nodes.

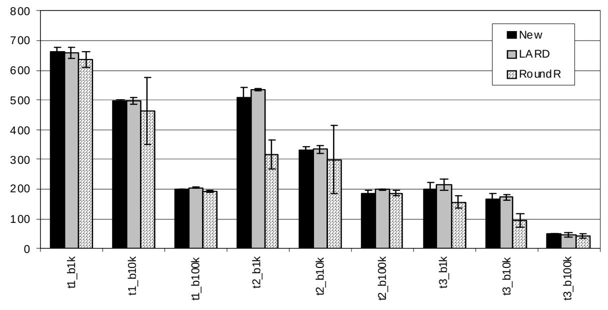


Figure 1. WIPs, 4 server nodes with 512MB heaps

The objective here is to demonstrate that it is possible to achieve improved throughput and efficiency in resource utilization by employing a request distribution policy that is explicitly aware of data locality, by taking into account the composition of the working-sets associated with request types. Given that, at this point of the work, there is no attempt to do a uniform load distribution, and that the conditions under which the system evaluation is performed (all server nodes share the computational resources of the same physical machine) do not allow for an objective and accurate evaluation of the load distribution among server nodes, no attempt shall be made at doing so. The evaluation metrics that we use here are only the overall system throughput and the efficiency in memory usage measured by the sizes of the effectively used heaps by the server nodes. A particular emphasis is given to the execution of the target application in sub-optimal memory availability conditions. These are expressed by running the server nodes with five different configurations for the JVM maximum heap size (Xmx): 512MB, 640MB, 768MB, 896MB, and 4096MB.

The throughput achieved by the three request distribution policies when the system is operating with 4 server nodes (where each is allowed to use up to 512MB of heap) can be seen in Figure 1. Each group of bars corresponds to a particular benchmark configuration, where t1, t2, and t3 indicate the type of workload, and b1k, b10k, and b100k indicate the size of the database used. By analyzing the results, it is possible to observe that the round-robin distribution is consistently outperformed by the other two policies, which display a rather similar throughput, across most configurations.

TABLE I. THROUGHPUT DIFFERENCE (%), 4 SERVER NODES

	512MB		640MB		768MB		896MB		4096MB	
	N/L	N/R	N/L	N/R	N/L	N/R	N/L	N/R	N/L	N/R
t1_b1k	0.6	4.2	-4.2	1.8	-7.1	5.1	-2.3	6.3	-1.4	2.9
t1_b10k	-0.1	6.8	-1.7	-4.7	-6.1	-4.5	-3.9	-1.8	-1.9	-5.1
t1_b100k	-2.3	3.8	0.0	3.7	-1.6	4.5	-2.6	3.2	-3.1	2.4
t2_b1k	-4.4	61.5	1.6	27.5	-1.9	18.5	9.3	19.8	1.6	19.1
t2_b10k	-1.3	10.1	0.5	3.8	-0.3	-8.5	0.0	-4.4	1.1	-5.5
t2_b100k	-7.4	-0.5	-3.4	3.8	0.2	6.4	-3.2	3.1	-1.8	0.2
t3_b1k	-6.7	28.8	-9.5	29.1	-9.9	31.1	-13.8	47.5	-10.8	26.2
t3_b10k	-3.1	76.7	8.4	47.2	2.6	48.3	-11.3	30.2	-9.7	26.1
t3_b100k	5.3	18.9	9.6	-42.6	16.2	-13.6	-6.1	-8.4	-8.4	-6.0
average	-2.2	23.4	0.1	7.7	-0.9	9.7	-3.8	10.6	-3.8	6.7

A thorough comparison of the performance gains achieved when using the New policy against LARD and Round-Robin, can be seen in Table I for 4 server nodes, and in Table II for 3 server nodes. The columns with "N/L" in the header contain throughput difference of New against LARD, calculated as $((T_{New} - T_{LARD})/T_{LARD}) \times 100$, % , whereas "N/R" is the comparative data of New versus Round-Robin.

TABLE II. THROUGHPUT DIFFERENCE (%), 3 SERVER NODES

	512MB		640MB		768MB		896MB		4096MB	
	N/L	N/R	N/L	N/R	N/L	N/R	N/L	N/R	N/L	N/R
t1_b1k	-2.07	5.13	3.95	28.89	-6.75	1.81	-2.65	2.46	-5.66	-4.10
t1_b10k	-1.06	-7.56	1.23	-7.91	6.18	-0.67	-10.11	-9.59	0.95	-5.99
t1_b100k	-2.53	3.37	-4.38	0.20	-3.08	0.97	-2.89	2.33	-1.30	3.95
t2_b1k	-7.40	42.26	-5.59	36.83	-1.50	11.53	-9.35	14.44	-6.14	11.39
t2_b10k	-6.39	-11.89	3.09	-3.89	-3.72	-8.78	4.17	-1.88	-7.76	-11.78
t2_b100k	-7.16	0.90	-3.03	1.78	-3.24	-0.47	-1.77	2.98	-3.56	2.03
t3_b1k	12.56	56.01	-5.07	40.55	14.21	74.91	3.32	63.02	0.46	46.36
t3_b10k	1.28	82.07	1.77	97.21	2.58	46.64	9.13	58.50	-11.55	35.50
t3_b100k	35.42	36.60	17.71	-8.35	-3.64	-31.28	7.65	-10.83	-6.64	-8.42
average	2.52	22.99	1.08	20.59	0.11	10.52	-0.28	13.49	-4.58	7.66

The New policy outperforms Round-Robin in 33 out of 45 (73%) configurations for 4 server nodes with an average of 11.62% better throughput, and in 29 out of 45 (64%) for 3 server nodes, with an average of 15.05%. The throughput achieved by using Round-Robin is the most inconsistent, across all configurations. This can be confirmed by observing Table III, where the uncertainty, expressed in terms of the covariance of the measurements, is displayed. The lower the value of the covariance of a given measurement, the higher the confidence in it - low covariance implies that the quantity being measured is unlikely to display values far from the mean.

TABLE III. MEASUREMENT UNCERTAINTY, 4 SERVER NODES

	512MB			640MB			768MB			896MB			4096MB		
	N	L	R	N	L	R	N	L	R	N	L	R	N	L	R
t1_b1k	0.03	0.04	0.053	0.00	0.08	0.17	0.07	0.03	0.05	0.01	0.04	0.02	0.01	0.02	0.04
t1_b10k	0.04	0.04	0.012	0.04	0.03	0.01	0.02	0.10	0.06	0.03	0.02	0.02	0.04	0.04	0.02
t1_b100k	0.01	0.02	0.009	0.00	0.03	0.01	0.01	0.01	0.02	0.00	0.01	0.01	0.01	0.02	0.01
t2_b1k	0.03	0.02	0.539	0.04	0.08	0.24	0.04	0.04	0.01	0.02	0.03	0.09	0.03	0.04	0.06
t2_b10k	0.03	0.04	0.051	0.01	0.04	0.07	0.03	0.07	0.02	0.05	0.06	0.02	0.03	0.05	0.03
t2_b100k	0.02	0.05	0.006	0.00	0.02	0.02	0.02	0.03	0.03	0.03	0.03	0.02	0.00	0.02	0.01
t3_b1k	0.20	0.07	0.058	0.11	0.02	0.06	0.13	0.09	0.11	0.09	0.14	0.16	0.21	0.08	0.10
t3_b10k	0.13	0.04	0.24	0.11	0.09	0.47	0.02	0.16	0.14	0.12	0.15	0.11	0.11	0.09	0.13
t3_b100k	0.21	0.03	0.237	0.02	0.09	0.32	0.06	0.17	0.15	0.06	0.08	0.20	0.08	0.12	0.13

The LARD policy outperforms the New approach in most configurations, by a small margin. On the average, the New policy provides 2.12% less throughput than LARD, for 4 server nodes, and 0.23% less throughput, for 3 nodes. Given that this particular LARD implementation is an idealized approach that makes use of perfect knowledge, a priori, about the target's system behavior, it is quite positive that the performance offered by the newly developed policy is so similar to an approach that would not be possible to achieve in practice.

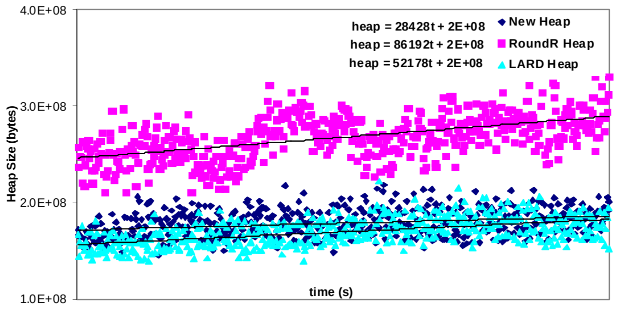


Figure 2. Heap size, t3, 10k books, 4 nodes, 640MB

Figure 2 contains the effective heap size usage achieved by the three policies, for a particular configuration of the benchmark, when 4 nodes are running with a maximum of

640MB of heap. The average heap size of New is 5.3% higher than LARD and 33.3% lower than Round-Robin. To identify the trends in memory usage we show also the result of a linear regression over the data collected. The gradients indicate that the heap growth rate of the New policy equals 0.33 of the Round-Robin and 0.54 of the LARD, resulting in the overall lowest growth in terms of heap.

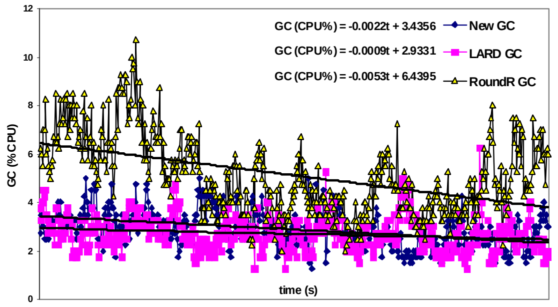


Figure 3. Garbage Collection (%CPU), t3, 10k books, 4 nodes, 640MB

Figure 3 shows the percentage of CPU spent performing garbage collection, for that same configuration of the benchmark. The average values are 2.9% for New, 2.7% for LARD and 5.1% for Round-Robin.

V. CONCLUSIONS

This work presented a new load balancing policy for clustered web server systems that seeks to maximize data locality by explicitly accounting for the correlation between the composition of the working sets of requests. That policy was applied to the TPC-W benchmark and evaluated against two alternative request distribution strategies. The newly developed approach provided significant performance gains under the form of increased throughput and improved efficiency in terms of memory usage, when compared against the alternative solutions.

ACKNOWLEDGMENT

This work was partially supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and by the Specific Targeted Research Project (STReP) Cloud-TM, which is co-financed by the European Commission through the contract no. 257784. The first author has been funded by the Portuguese FCT under contract SFRH/BD/64379/2009.

REFERENCES

[1] Amza, C., Cox, A. L. and Zwaenepoel, W., 2003, Conflict-aware scheduling for dynamic content applications, Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems, USENIX Association, Vol. 4, pp. 6-20.

[2] Amza, C., Cox, A. L. and Zwaenepoel, W., 2005, A comparative evaluation of transparent scaling techniques for dynamic content servers, Proceedings of the 21st International Conference on Data Engineering (ICDE 2005), IEEE, pp. 230-241.

[3] Bansal, N. and Harchol-Balter, M., 2001, Analysis of SRPT scheduling: Investigating unfairness, ACM.

[4] Blei, D., Ng, A. and Jordan, M., 2003, Latent dirichlet allocation, Journal of Machine Learning Research, 3 pp. 993-1022,

[5] Cardellini, V., Casalicchio, E., Colajanni, M. and Yu, P., 2002, The state of the art in locally distributed Web-server systems, ACM Computing Surveys (CSUR), 34 (2), pp. 263-311,

[6] Crovella, M., Frangioso, R. and Harchol-Balter, M., 1999, Connection scheduling in web servers, Proceedings of the 2nd conference on USENIX Symposium on Internet Technologies and Systems - Volume 2, Boulder, Colorado, USENIX Association, pp. 22-22.

[7] Devlin, B., Gray, J., Laing, B. and Spix, G., 1999, Scalability Terminology: Farms, Clones, Partitions, Packs, RACS and RAPS, http://www.scientificcommons.org/21762514,

[8] Elnikety, S., Dropsho, S. and Zwaenepoel, W., 2007, Tashkent+: Memory-aware load balancing and update filtering in replicated databases, ACM SIGOPS Operating Systems Review, 41 (3), pp. 399-412,

[9] Garbatov, S. and Cachopo, J., 2010, Importance Analysis for Predicting Data Access Behaviour in Object-Oriented Applications, Journal of Computer Science and Technologies, 14 (1), pp. 37-43, IEEE.

[10] Garbatov, S. and Cachopo, J., 2010, Predicting Data Access Patterns in Object-Oriented Applications Based on Markov Chains, Proceedings of the Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, pp. 465-470.

[11] Garbatov, S. and Cachopo, J., 2011, Optimal Functionality and Domain Data Clustering based on Latent Dirichlet Allocation, Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA 2011), Barcelona, Spain, ThinkMind, pp. 245-250.

[12] Garbatov, S., Cachopo, J. and Pereira, J., 2009, Data Access Pattern Analysis based on Bayesian Updating, Proceedings of the First Symposium of Informatics (INForum 2009), Lisbon, Paper 23.

[13] Hu, Y., Nanda, A. and Yang, Q., 1999, Measurement, analysis and performance improvement of the Apache web server, Proceedings of the Performance, Computing and Communications Conference, IEEE, pp. 261-267.

[14] Nahum, E., Barzilai, T. and Kandlur, D. D., 2002, Performance issues in WWW servers, IEEE/ACM Transactions on Networking, 10 (1), pp. 2-11,

[15] Pai, V., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W. and Nahum, E., 1998, Locality-aware request distribution in cluster-based network servers, Proceedings of the eighth international conference on Architectural support for programming languages and operating systems, San Jose, California, United States, ACM, pp. 205-216.

[16] Pai, V., Druschel, P. and Zwaenepoel, W., 1999, An Efficient and Portable Web Server, Proceedings of the 1999 USENIX Annual Technical Conference.

[17] Rousseeuw, P. J., 1987, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, Journal of computational and applied mathematics, 20 pp. 53-65,

[18] Smith, W. TPC-W: Benchmarking An Ecommerce Solution. Intel Corporation, 2000.

[19] Zhang, Q., Riska, A., Sun, W., Smirni, E. and Ciardo, G., 2005, Workload-aware load balancing for clustered web servers, IEEE Transactions on Parallel and Distributed Systems, 16 (3), pp. 219-233,

[20] Zhong, M., Shen, K. and Seiferas, J., 2008, Correlation-Aware Object Placement for Multi-Object Operations, Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems, IEEE Computer Society, pp. 512-521.

Evaluating Performance of Distributed Systems With MapReduce and Network Traffic Analysis

Thiago Vieira, Paulo Soares, Marco Machado
Rodrigo Assad, Vinicius Garcia
Federal University of Pernambuco (UFPE) – Recife, Pernambuco, Brazil
{tpbv,pfas,masm,rea,vcg}@cin.ufpe.br

***Abstract*—Testing, monitoring and evaluation of distributed systems at runtime is a difficult effort, due the dynamicity of the environment, the large amount of data exchanged between the nodes and the difficulty of reproduce an error for debugging. Application traffic analysis is a method to evaluate distributed systems, but the ability to analyze large amount of data is a challenge. This paper proposes and evaluates the use of MapReduce programming model to deep packet inspection the application traffic of distributed systems, evaluating the effectiveness and the processing capacity of the MapReduce programming model for deep packet inspection of a JXTA distributed storage application, in order to measure performance indicators.**

***Keywords*-Measurement of Distributed Systems; MapReduce; Network Traffic Analysis; Deep Packet Inspection.**

I. Introduction

With the growing of cloud computing usage and the use of distributed systems to provide infrastructure and platform as a service, the monitoring and performance analysis of distributed systems became more necessary [1]. In distributed systems development, the maintenance and administration, the detection of error causes and the analysis and the reproduction of an error are challenges and motivates efforts to the development of less intrusive mechanisms for debugging and monitoring distributed applications at runtime [2]. Network traffic analysis is one option to evaluate distributed systems performance [3], although there are limitations on capacity to process large amount of network packet in short time [3][4] and on scalability to be able to process network traffic over variations of throughput and resource demand. Simulators [5], emulators or testbeds [4][6] are also used for evaluate distributed systems, but these presents lacks for reproduce the real behaviour of a distributed system and its relation within a complex environment, such as a cloud computing environment [4][6].

An approach to process large amount of network traffic was proposed by [7]. The proposal consists in the use of MapReduce [8] programming model to parse network packets to a binary format, that can be used as input for Map and Reduce functions to process network packet flow. This proposal shows that MapReduce improves the computation time and provides fault tolerance to packet flow analysis, but the use of MapReduce for deep packet inspection (DPI) and for evaluate distributed applications was not analyzed.

Because of the need to evaluate the real behaviour of distributed systems at runtime, in a less intrusive way, and the need of a scalable and fault tolerant approach to process large amount of data, using commodity hardware, we propose the use of MapReduce programming model to implement a passive DPI for distributed applications. In this paper we evaluate the effectiveness of MapReduce to a DPI algorithm and its processing capacity to measure a JXTA-based application in order to extract performance indicators at runtime.

The remainder of this paper is organized as follows. Section 2 describes the related works. Section 3 presents the proposed solution. Section 4 describes the experiments performed, Section 5 presents the results and the Section 6 concludes the paper and describes future works.

II. Related Works

To provide infrastructure and platform as a service in a cloud computing model, it is necessary to have available a scalable and fault-tolerant infrastructure. Thus, the use of distributed systems to obtain this requirements has been widely used [9][10][11]. The Google Inc. developed a distributed storage system [10][11] to use on its applications, which processes large amount of data and needs high availability, scalability and feseability. Amazon has its services based on Dynamo [9], which is a peer-to-peer storage system to provide high availability and eventual consistency to data storage. Other distributed technology widely used by cloud computing providers is Hadoop [12], which is an implementation of MapReduce to process data intensive tasks over a cluster.

The evaluation of distributed applications is a challenge, due the cost of monitoring distributed systems and the lacks on performance measurement at runtime of large scale distributed applications. To reproduce the behaviour of a complex system in a test environment it is necessary to know each relevant parameter of the system, and recreate them in the test environment [6]. These needs are more evident in cases where faults occur only when the system is over a high load [4], which difficult the environment reproduction and debugging.

Gupta *et al* [6] presents a methodology and framework for large scale tests, able to obtain resources configurations and scale near of a large scale system, through the use of emulated scalable network, multiplexed virtual machines and resource dilatation. Gupta *et al* also shows accuracy and capacity of increase the scale and the realism on network tests, although it do not obtain the same precision of an evaluation of a real system at runtime.

Large scale tests can be executed in environments near to real, using testbeds such as PlanetLab [13], which is a widely used distributed environment composed by real or virtual machines, geographically distributed over the world, through real network resources. Although a testbed uses real resources, to achieve precision on real time and data intensive simulations, it is need to know and reproduce each relevant parameter of the system, and recreate them in the simulation environment, including impacts caused by external systems.

Loiseau *et al* [4] argues the necessity of network traffic evaluation with more granularity to complement simulations and measurements, and to better understanding the network traffic evolution and behaviour of each kind of network traffic. To apply this kind of evaluation, it is necessary devices and approaches to capture and process large amount of data, such as Metroflux [4] and DPI solutions [14], but it is still necessary scalable solutions to handle large amount of network traffic over different demands and throughput.

MapReduce [8] is a programming model and a framework for processing large data sets trough data-intensive distributed computing, providing fault tolerance and high scalability to big data processing. MapReduce became an important programming model for large scale parallel systems, with applications on indexing, data mining, machine learning, and scientific simulation [15]. Although many tasks are expressible in MapReduce [8], it is necessary to know if DPI problems can be expressed in MapReduce programming model.

Lee *et al* [16] proposes a flow analysis method using MapReduce programming model, where the network traffic is captured, converted to text, and used as input to Map functions. This work shows the improvement achieved in computation time in comparison with flow-tools, which is a flow analyses tool widely used to capture, filter and report flow traffic. The conversion time from network traffic to text may represent a relevant additional time, but it is not clear if Lee *et al* (2010) considered this conversion time on its evaluation and comparison with flow-tools.

Lee *et al* [7] presents a Hadoop-based packet trace processing tool to process large amount of binary network traffic. A new input type to Hadoop was developed, the PcapInputFormat, which encapsulates the complexity of processing a captured pcap trace and to extract the packets using the libpcap [17] packet capture library. Lee *et al* (2011) compares its proposal with CoralReef, which is a network traffic analysis tool that also relies of libpcap, and shows speedup on completion time to processing packet traces with more than 100GB. The evaluation was limited to process packet flow and extract indicators from IP, TCP and UDP, not considering deep packet inspection, which needs reassembly two or more packets to extract information. Additionally, the PcapInputFormat rely on a timestamp based heuristic for finding the first record from each block, using sliding-window, which can be a limitation on accuracy, if compared with the accuracy obtained by Tcpdump [18].

JXTA provides support to the development of peer-to-peer applications, through the specification of protocols, services and communications layers. Halepovic and Deters [19] proposed a performance model, describing important indicators to evaluate throughput, scalability, services and the JXTA behaviour in different versions. Also, Halepovic and Deters highlights the performance and scalability limitations of JXTA, which can be improved by configurations, source code modifications or by new JXTA's versions. Halepovic and Deters [20] analizes the JXTA performance in order to show the increasing cost or latency with higher workload and with concurrent requests, and they suggests more evaluations about scalability of large group of peers in direct communication. Halepovic *et al* [21] cites network traffic analysis as an approach to performance evaluation of JXTA-based applications, but do not adopt it, due the lack on JXTA traffic characterization. Although there is a performance models and evaluations of JXTA, there are not evaluations for the current versions and neither mechanisms to evaluate JXTA applications at runtime, being necessary a solution to measure the performance of JXTA-based applications and provides information to evaluate its behaviour over different circumstances.

III. The Solution

To evaluate JXTA distributed applications through network traffic analysis it is necessary to capture and analyse the content of JXTA messages split into network packets, and be able to process large amount of network traffic in acceptable time. To achieve this, we propose the use of MapReduce, implemented by Apache Hadoop, to process JXTA network traffic, extract performance indicators over different scenarios, and provide an efficient and scalable solution for DPI using commodity hardware. The architecture of our solution is composed by four main component: the *Sniffer*, that captures, splits and stores network packets into Hadoop Distributed File System (HDFS); the *Manager*, that orchestrates the collected data, the job executions and stores the results generated; the *jnetpcap-jxta* [22], that converts network packets into JXTA messages; and the *JXTAAnalyzer*, that implements Map and Reduce functions to extract performance indicators.

Figure 1 shows the overview of the proposed architecture to capture the network traffic, through the *Sniffer*, split them into files, and store this files into HDFS. The *Sniffer* must be connected to the network where the target nodes are connected, and be able to stablish communication with the others nodes that composes the HDFS cluster. Figure 2 shows the architecture proposed to process network traffic through MapReduce functions, which are implemented by *JXTAAnalyzer* and deployed at each node of the Hadoop cluster, and managed by the *Manager*.

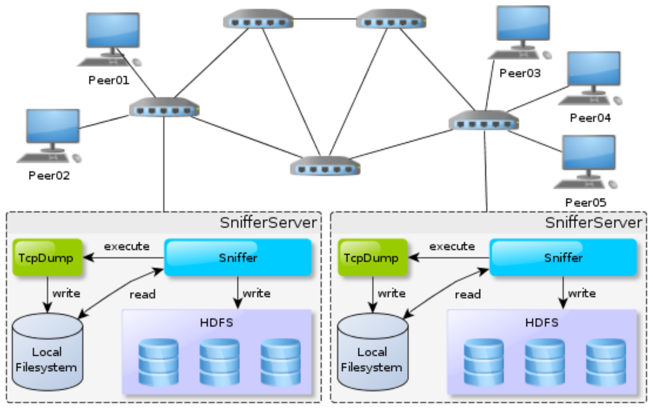


Figure 1. Shows the overview of the architecture proposed to capture and store the network traffic

Initially, the network traffic must be captured, split and stored into HDFS. The packets are captured using Tcpdump, a widely used libpcap network traffic capture tool, and are split into files with 64 MB of size, which is the default block size of the HDFS, although this block size may be configured to different values. Files that are greater than the HDFS block size are split into blocks with size equal or smaller than the block size, and are spread among machines in the cluster. As the libpcap, used by Tcpdump, stores the network packets in binary files known as pcap files, it is necessary to avoid split this files or to provide to Hadoop an algorithm to split pcap files. Due the file split demands additional computing time and increases the complexity of the system, we adopted the split of pcap files into default HDFS block size, using the split functionality provided by Tcpdump. Thus, the network traffic is captured by Tcpdump, split and stored into local file system of the *Sniffer*, and periodically transferred to HDFS, which is responsible to replicate the files into the machines of the cluster.

In the MapReduce programming model, the input data is split into blocks and into records, which are used as input for Map tasks. We adopt the use of whole files, with size defined by the HDFS block size, as input for each Map tasks, in order to extract information of more than one packet, differently of Lee *et al* [7] approach, where each Map task receives only one packet as input. With our approach it is possible reassembly TCP packets, JXTA messages and other protocols that has its content divided into many packets to be transferred over TCP.

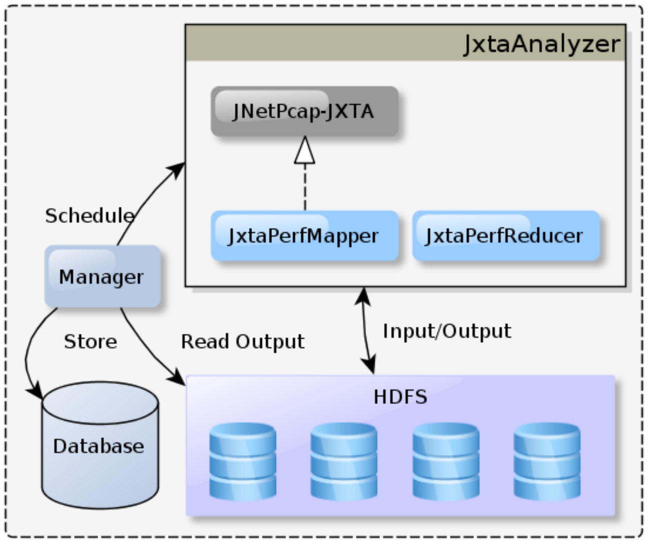


Figure 2. Shows the architecture proposed to process the network traffic

Once the pcap files has been stored into HDFS, an agent called *Manager* is responsible for selecting the files to be processed, to schedule the Map and Reduce tasks, and store the generated results into a database. Each Map function receives as input a path of a pcap file stored into HDFS, the path received for each Map task is defined by the data locality control of the Hadoop, which delegates each task to nodes that have a local replica of the data, or to nodes placed in the same rack of a replica. Then, the file is opened and each network packet is processed to extract the performance indicators and to generate as output a *SortedMapWritable* object, with a sorted collection of values for each performance indicator evaluated, which will be summarized by Reduce functions.

One TCP packet can transport one or more JXTA message, due to window size of the buffer used by JXTA Socket to send and receive messages. Because of this, it is necessary to evaluate the full content of each TCP segment to identify all messages, instead of evaluate only the message header or signature, as is commonly done in DPI techniques and by widely used traffic analysis tools, such as Wireshark [23], which is unable to recognize all JXTA messages, because its approach do not identify when two or more messages are transported into a TCP packet. Moreover, if a message is greater than the size of the PDU in the TCP, the message is split into some TCP segments. To handle these problems, we developed a reassembly algorithm to recognize, sort and reassembly TCP segments into JXTA messages, which is described at Algorithm 1.

For all TCP packet of a pcap file, is verified if it is a JXTA message or if it is part of a JXTA message that was

Algorithm 1 JxtaPerfMapper

```
for all $tcpPacket$ do
    if $isJxta$ OR $isWaitingForPendings$ then
        $parsePacket(tcpPacket)$
    end if
end for

function PARSEPACKET($tcpPacket$)
    $parseMessage$
    if $isMessageParsed$ then
        $upddateSavedFlows$
        if $hasRemain$ then
            $parsePacket(remainPacket)$
        end if
    else
        $savePendingMessage$
        $lookForMoreMessages$
    end if
end function
```

not fully parsed and is waiting for its complement, then a parse attempt is made, using *Jnetpcap-jxta*. As a TCP packet may to contain one or more JXTA message, if a message is fully parsed, then it is done another parse attempt with the content not used by the previous parse. If the content is a JXTA message and the parse is not successful, then its TCP content is stored, with its TCP flow identification as a key, and all next TCP packets that match with the flow identification will be sorted and used to try to mount a new JXTA message, until the parser is entirely successful. With these characteristics, to inspect JXTA messages it is required more effort than others cases of deep packet inspection, where the analysis is based on inspection of message header or protocol signature.

To evaluate JXTA messages captured as binary network packet, we developed a parser called *jnetpcap-jxta*, which converts libpcap network packet into Java JXTA messages. *Jnetpcap-jxta* is written in Java and provides methods to convert byte arrays into JXTA messages, using an extension of JXTA default library for Java, known as JXSE. With this, we are able to parse all kind of messages defined by JXTA specification. *Jnetpcap-jxta* relies on JNetPcap library to support the instantiation and inspection of libpcap packets, JNetPcap was adopted due the performance to iterate over packets, the large quantity of functionalities provided to handle packet traces and due the recent update activities for this library.

As showed by Figure 2, the *JXTAAnalyzer* is composed by Map and Reduce methods, *JxtaPerfMapper* and *JxtaPerfReducer*, to extract performance indicators from JXTA Socket communication layer, which is a communication mechanism that implements a reliable message exchange and obtains the better throughput between the communication layers provided by the JXTA.

Each message of a JXTA Socket is part of a Pipe that represents a connection established between the sender and receiver. In a JXTA Socket communication, two Pipes are established, one from sender to receiver and other from receiver to sender, which transports content messages and acknowledges messages, respectively. To evaluate and extract performance indicators from JXTA Socket, the messages must be sorted, grouped and linked with its respectives Pipes of content and acknowledge. The content transmitted into a JXTA Socket is split into byte array blocks and stored into a reliability message, that is sent to the destination and expects to receive an acknowledge message of its arrival. Each block that was sent or that was delivered, is queued by JXTA until the system is ready to process them. The time between the message delivery and the acknowledge be sent back is called round-trip time (RTT), this may vary according to the system load and may be used to evaluate if the system overloaded.

The *JxtaPerfMapper* and *JxtaPerfReducer* evaluates the RTT of each content block transmitted over a JXTA Socket, and extracts information about the number of connection requests and message arrivals. Each Map function evaluates the packet trace to mount JXTA messages, Pipes and Sockets. The parsed JXTA messages are sorted by its sequence number and grouped by its Pipe identification, to compose the Pipes of a JXTA Socket. As soon as the messages are sorted and grouped, then the RTT is obtained, its value is associated with the respective key and written as an output of the Map function. Each Reduce function receives as input a key and a collection of values, which are respectively the indicator and its values, then generates individual files for each indicator.

The implemented Map and Reduce methods can be extended to address others performance indicators, such as throughput or number of retransmissions, for this each indicator must be represented by a unique key and the collected values must be associated with its respective key. Moreover, others Map and Reduce methods can be developed to analyse others protocols and application traffics.

IV. Experiments

As the two main goals of this work are to evaluate the effectiveness and the processing capacity of MapReduce to DPI in order to measure distributed applications, we performed two set of experiments for different size of data input and number of nodes in a Hadoop cluster, in order to evaluate the MapReduce scalability and completion time to DPI. We used as input a network traffic data captured from a JXTA Socket communication between a server and some clients from a distributed backup system. Two data set was captured, with size of 16 GB and 34 GB, split into 35 and 79 files, respectively. The first experiment set processes 16 GB of data and varies the number of Hadoop slave nodes between 3 and 10, the second experiment set processes 34

Nodes	3	4	6	8	10
Time	322.50	245.92	172.83	151.45	126.87
Std. Deviation	4.91	4.63	2.76	6.60	9.97
MB/s	50.80	66.62	94.80	108.18	129.14
(MB/s)/node	16.93	16.66	15.80	13.52	12.91

Table I
COMPLETION TIME TO PROCESS 16 GB SPLIT INTO 35 FILES

Nodes	4	8	12	16	19
Time	464.46	260.48	188.93	166.99	134.22
Std. Deviation	4.18	5.52	6.23	3.81	5.77
MB/s	74.96	133.66	184.28	208.49	259.40
(MB/s)/node	18.74	16.71	15.36	13.03	13.65

Table II
COMPLETION TIME TO PROCESS 34 GB SPLIT INTO 79 FILES

GB of data and varies the number of Hadoop slave nodes between 4 and 19.

For each experiment set, we measure the completion time and the capacity to process pcap files, in a Hadoop cluster with different number of nodes. The experiments were executed 30 times for each number of nodes, then was measured the mean completion time and the standard deviation of the value measured. All experiments were performed at Amazon EC2, with a environment composed by slave nodes with Ubuntu Server 11.10, kernel 3.0.0-16, with 2 virtual cores composed by 2.5 EC2 Compute Units, 1.7 GB of RAM memory and 350 GB of hard disks, and one master node with Ubuntu Server 11.10, with 1 virtual core composed by 1 EC2 Compute Unit, 1.7 GB of RAM memory and 160 GB of hard disks.

The Hadoop cluster was composed by one master node and many slave nodes running Hadoop library version 0.20.203 with default configuration, using 64MB as block size and with the data replicated 3 times over the HDFS.

The network traffic used as input data was captured from a JXTA Socket Server receiving Socket requisitions and transferring data from 5 concurrent clients, which sends data to be stored at the server, with JXTA message content size between 64KB and 256KB. The network traffic was captured and processed, as described previously, in order to extract round-trip time, number of requisitions and the number of data sent to the server per time.

V. RESULTS

The results show the completion time to deep packet inspection of JXTA network traffic using MapReduce, with different input size and number of nodes in a Hadoop cluster. The Tables 1 and 2 present respectively the results of the experiment to process 16 GB and 34 GB of network traffic, showing the number of Hadoop nodes used for each experiment, the mean completion time in seconds, its standard deviation, the processing capacity achieved and the relative processing capacity per node in the cluster.

The completion time decreases with the increment of number of nodes in the cluster, but not in a linear function. This conclusion is clearer when observed that the relative processing capacity per node decreases with the addition of nodes in the cluster. With the growing of the number of nodes in the cluster, increases the cost to manage the cluster, the data replication, the allocation of tasks to available nodes and the management of failures, also is increased the cost with merging and sorting of the data processed by each Map task. In small clusters, the probability of a node to have a replica of the data received as input, is greater than in large clusters. In large clusters there are more options of nodes to delegate a task, but the number of data replication limits these options to the number of nodes with a replica of the data, this limitation increases the cost to schedule tasks and distribute tasks in the cluster.

In our experiment, was achieved a mean processing capacity of 259.40 MB per second, in a cluster with 19 slave nodes, processing 34 GB. For a cluster with 4 nodes was achieved a mean processing capacity of 66.62 MB/s and 74.96 MB/s to process respectively 16 GB and 34 GB of network traffic data, which indicates that the processing capacity may vary as a function of the amount of data processed and the number of files used as input data.

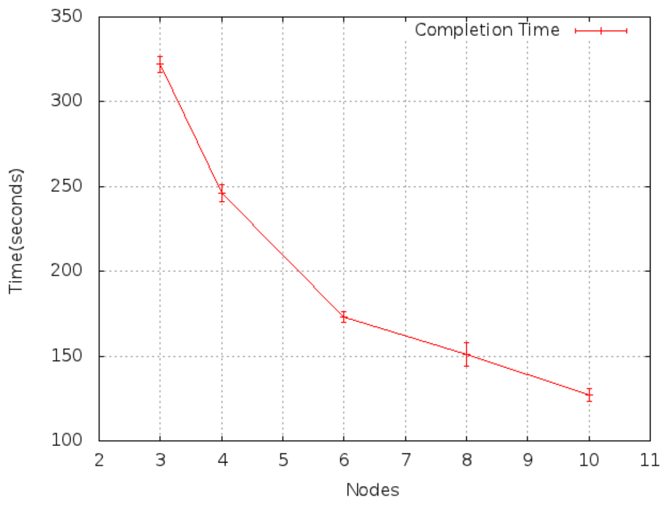


Figure 3. Scalability to process 16 GB

Figures 3 and 4 illustrate how the addition of nodes to the Hadoop cluster reduces the mean completion time and how is the scalability of processing capacity achieved to processing 16 GB and 34 GB of network traffic data. In both graphics, the behaviour of the scalability is similar, with more significant scalability gains, through addition of nodes, in small clusters, and less significant gains with the growing of the number of nodes in the cluster, which indicates the importance of evaluating the relation between costs and benefits to addition of nodes.

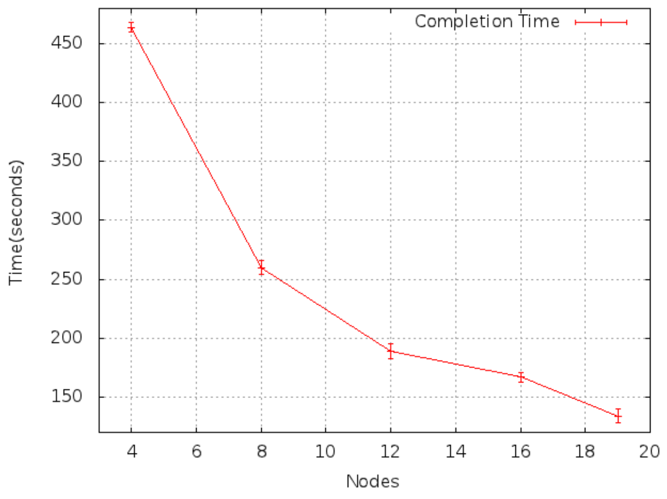


Figure 4. Scalability to process 34 GB

In the JXTA network traffic analized, was possible to use the MapReduce programming model to extract indicators of the number of connection request, number of data receiving and the round-trip time. With these data is possible to evaluate the behaviour of the system, extract information about its performance and provide a better understanding of the network traffic behaviour of a JXTA-based application. Figure 5 shows the network traffic behaviour of a JXTA Socket server receiving connection request and data from 5 concurrent clients. In this figure it is possible to observe the behaviour of the Java just-in-time compiler [24] optimizing the bytecode through its convertion into an equivalent sequence of the native code of the underlying machine, in the time when the rount-trip time increases and is normalized after the optimization of the bytecode.

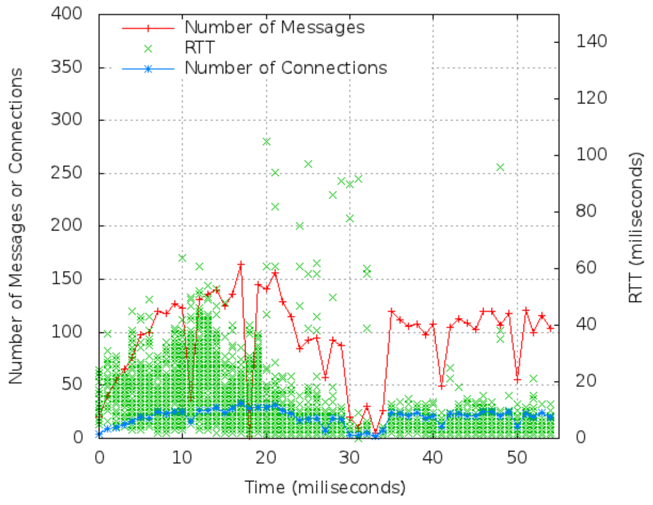


Figure 5. JXTA Socket trace analysis

VI. Conclusion and Future Works

To evaluate the network behaviour of distributed systems at runtime, in a less intrusive way, with high processing capacity, scalability and fault tolerance, it is necessary approaches and tools to support the development, management and monitoring of distributed applications. To address this, we proposed the use of MapReduce programming model to deep packet inspection of distributed application network traffic, and we performed experiments in order to show the effectiveness and efficiency of our proposal.

We showed that MapReduce programming model can express algorithms for DPI, as the Algorithm 1, implemented to extract indicators from a JXTA network traffic, with indicators shown in Figure 5. We applied the MapReduce to DPI, using a network trace split into files with 64 MB, to avoid the cost of split the network trace into packets and also to be able to reassembly two or more packets to mount JXTA messages from packets.

We analized the processing capacity and scalability achieved for different number of nodes in a Hadoop cluster, with different size of network traffic data, showing the processing capacity and scalability achieved, and the influence of the number of nodes and the data input size in the capacity processing of network traffic. We showed that using Hadoop as a MapReduce implementation, it is possible to use commodity hardware, or cloud computing services, to deep packet inspection of large amount of network traffic.

With our proposal, also it is possible to measure and evaluate, at runtime and in a less intrusive way, the network traffic behaviour of distributed applications with intensive network traffic generation, making possible the use of this captured information to reproduce the behaviour of the system in a simulation environment.

In future work, we will evaluate and characterize the behaviour followed by the scalability of MapReduce to DPI, evaluating the optimal size of input data for large and small clusters. Optimizations in the Hadoop environment, HDFS and JNetPcap library can be investigated to improve the approach to load pcap files, making the JNetPcap able to read files from HDFS and avoiding another copy of the data. Other future work is to investigate improvements or propose a Hadoop scheduler to cases where the input data can not be split and a full file is required as input to Map tasks. We will also perform evaluations of MapReduce to deep packet inspection of others protocols and distributed applications.

VII. Acknowledgements

This research was supported by the National Institute of Science and Technology for Software Engineering (INES - www.ines.org.br), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," Tech. Rep., 2009.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, Apr. 2010.

[3] A. Callado, C. Kamienski, G. Szabo, B. Gero, J. Kelner, S. Fernandes, and D. Sadok, "A survey on internet traffic identification," *Communications Surveys Tutorials, IEEE*, vol. 11, no. 3, pp. 37 –52, quarter 2009.

[4] P. Loiseau, P. Goncalves, R. Guillier, M. Imbert, Y. Kodama, and P.-B. Primet, "Metroflux: A high performance system for analysing flow at very fine-grain," in *Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops, 2009. TridentCom 2009. 5th International Conference on*, april 2009, pp. 1 –9.

[5] D. Paul, "Jxta-sim2: A simulator for the core jxta protocols," Master's thesis, University of Dublin, Ireland, 2010.

[6] D. Gupta, K. V. Vishwanath, M. McNett, A. Vahdat, K. Yocum, A. Snoeren, and G. M. Voelker, "Diecast: Testing distributed systems with an accurate scale model," *ACM Trans. Comput. Syst.*, vol. 29, pp. 4:1–4:48, May 2011.

[7] Y. Lee, W. Kang, and Y. Lee, "A hadoop-based packet trace processing tool," in *Proceedings of the Third international conference on Traffic monitoring and analysis*, ser. TMA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 51–63.

[8] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, Jan. 2008.

[9] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 205–220, Oct. 2007.

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 29–43, Oct. 2003.

[11] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, pp. 4:1–4:26, June 2008.

[12] "Hadoop," http://hadoop.apache.org/, [retrieved: september, 2012].

[13] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 3–12, July 2003.

[14] S. Fernandes, R. Antonello, T. Lacerda, A. Santos, D. Sadok, and T. Westholm, "Slimming down deep packet inspection systems," in *INFOCOM Workshops 2009, IEEE*, april 2009, pp. 1 –6.

[15] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 29–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855741.1855744

[16] Y. Lee, W. Kang, and H. Son, "An internet traffic analysis method with mapreduce," in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, april 2010, pp. 357 –361.

[17] V. Jacobson, C. Leres, and S. McCanne, "libpcap," http://www.tcpdump.org/, 1994.

[18] "Tcpdump," http://www.tcpdump.org/, [retrieved: september, 2012].

[19] E. Halepovic and R. Deters, "The jxta performance model and evaluation," *Future Gener. Comput. Syst.*, vol. 21, pp. 377–390, March 2005.

[20] E. Halepovic, R. Deters, and B. Traversat, "Jxta messaging: Analysis of feature-performance tradeoffs and implications for system design," in *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, R. Meersman and Z. Tari, Eds. Springer Berlin / Heidelberg, 2005, vol. 3761, pp. 1097–1114.

[21] E. Halepovic, "Performance evaluation and benchmarking of the jxta peer-to-peer platform," 2004.

[22] T. Vieira, "jnetpcap-jxta," http://github.com/tpbvieira/jnetpcap-jxta, [retrieved: september, 2012].

[23] "Wireshark," http://www.wireshark.org/, [retrieved: september, 2012].

[24] T. Suganuma, T. Ogasawara, M. Takeuchi, T. Yasue, M. Kawahito, K. Ishizaki, H. Komatsu, and T. Nakatani, "Overview of the ibm java just-in-time compiler," *IBM Systems Journal*, vol. 39, no. 1, pp. 175 –193, 2000.