PATTERNS 2014

The Sixth International Conferences on Pervasive Patterns and Applications

ISBN: 978-1-61208-343-8

May 25 - 29, 2014

Venice, Italy

PATTERNS 2014 Editors

Alexander Mirnig, University of Salzburg, Austria

Jaap Kabbedijk, Utrecht University, Netherlands

PATTERNS 2014

Foreword

The Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014), held between May 25-29, 2014 in Venice, Italy, targeted the application of advanced patterns, at-large. In addition to support for patterns and pattern processing, special categories of patterns covering ubiquity, software, security, communications, discovery and decision were considered. As a special target, the domain-oriented patterns cover a variety of areas, from investing, dietary, forecast, to forensic and emotions. It is believed that patterns play an important role on cognition, automation, and service computation and orchestration areas.

We take here the opportunity to warmly thank all the members of the PATTERNS 2014 Technical Program Committee, as well as all of the reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and efforts to contribute to PATTERNS 2014. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations, and sponsors. We are grateful to the members of the PATTERNS 2014 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that PATTERNS 2014 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the field of pervasive patterns and applications.

We are convinced that the participants found the event useful and communications very open. We hope that Venice, Italy, provided a pleasant environment during the conference and everyone saved some time to enjoy the charm of the city.

PATTERNS 2014 Chairs:

Herwig Manaert, University of Antwerp, Belgium
Fritz Laux, Reutlingen University, Germany
Michal Zemlicka, Charles University - Prague, Czech Republic
Alfred Zimmermann, Reutlingen University, Germany
Richard Laing, Robert Gordon University, UK
Ricardo Sanz, UPM ASlab, Spain
Mauricio Hess-Flores, University of California, Davis, USA
Teemu Kanstren, VTT, Finland
Markus Goldstein, DFKI (German Research Center for Artificial Intelligence GmbH), Germany
Zhenzhen Ye, IBM, Essex Junction, USA
René Reiners, Fraunhofer FIT - Sankt Augustin, Germany
Nguyen Vu Hoàng, Vertigo, France
Alexander Mirnig, University of Salzburg, Austria
Juan C Pelaez, Defense Information Systems Agency, USA
Bastian Roth, University of Bayreuth, Germany
Steve Strauch, IAAS at University of Stuttgart, Germany
Jaap Kabbedijk, Utrecht University, Netherlands

PATTERNS 2014

Committee

Table of Contents

SQL Pattern Design and Development

Huda Al-Shuaily
Department of Information Technology
Higher College of Technology
Muscat, Oman
huda.alshuaily@hct.edu.om

Karen Renaud
School of Computing Science
University of Glasgow
Glasgow, UK
karen.renaud@glasgow.ac.uk

Abstract—**This paper presents the processes involved in the design and development of a set of Structure Query Language (SQL) patterns. The intention is to support novices during SQL acquisition. This process is grounded in the SQL learning model developed from learning theory literature and empirical investigations into SQL acquisition. One of the crucial cross-cutting factors identified during the development of this model was the quality of the instructional material provided to learners to support the acquisition process. Since patterns have been successfully deployed in other areas to support knowledge transfer, we set out to develop SQL patterns to meet this need for effective instructional material. We detail the process by which we identified the required components of SQL patterns. Our patterns were also informed based on observations of how novices and experts solved SQL queries. We conclude by presenting our proposed SQL pattern content and structure.**

Keywords-Pattern; SQL; Expert

I. INTRODUCTION

SQL is one of those languages that seem particularly challenging to master. We previously reported on an investigation which advanced explanations for this apparent difficulty [1] and developed a model of SQL learning shown in Fig.1. Our investigation uncovered four cross-cutting issues that impacted the learning process. One major factor was the quality and suitability of the instructional material to support the learning process. In providing instructional material, two specific aspects are important:

- The structuring of knowledge within the instructional material. Merrill [2] explains that the organization and representation of knowledge impacts learning. Mayer [3] makes the same argument, positing that the way in which a body of knowledge is structured determines how readily it will be grasped by learners.
- When, during the learning process, the afore-mentioned instructional material should be introduced. Learning happens in a predictable and mediated way, with subsequent knowledge and skills building on prior knowledge and understanding [4], [5]. Hence, the sequence in which we present knowledge is important. It should support learning rather than encouraging trial-and-error attempts to produce correct SQL queries without understanding the underlying principles [6].

Patterns are a widely used mechanism for supporting knowledge transfer. We set out to investigate whether patterns could meet the need for optimally-structured instructional material in this context. Schlager and Ogden [7] found that incorporating a cognitive model in the form of expert user and product-independent knowledge into novice instruction enhances learning. They concluded that such a cognitive model framework could potentially help to support knowledge acquisition.

Patterns traditionally structure knowledge in such a way that they can transfer best practice from experts to novices. We cannot assume that SQL patterns can be structured in the exactly the same way as other more well-established patterns, however, so we need to carefully align them with the SQL acquisition process.

We briefly present our previously developed model of SQL learning (Fig. 1), which is the logical place to start when identifying SQL patterns and positioning them within the learning process. This model is grounded in Bloom's taxonomy [4] and validated by studies of how novices learn to write SQL queries. The SQL learning model emerged from the analysis of the educational literature, and was augmented by the analysis of data gathered during qualitative and quantitative studies of SQL acquisition.

This model incorporates the notion of the development of mental models. We demonstrate how mental models are constructed during SQL acquisition. Learners start with the development of individual schemata, moving on towards meaningful structuring of schemata into hierarchies and constructed mental models. Existence of these models suggests that the learner will be able to solve a variety of problems of similar nature, i.e., they have abstracted. An abstraction exists and they should be able to apply core concepts in many contexts: learning has resulted in a heuristic.

The SQL acquisition process is also modeled in the diagram, showing that learners need first to have a basic knowledge of SQL concepts, and an understanding of how to use them. They then have to practice applying these concepts to a variety of problems: analyzing, synthesizing and evaluating. They ought to emerge from this stage with an appreciation of the core principles; with an ability to make judgments about strategies to be deployed. Learners who have progressed to this upper level can be considered to have mastered SQL.

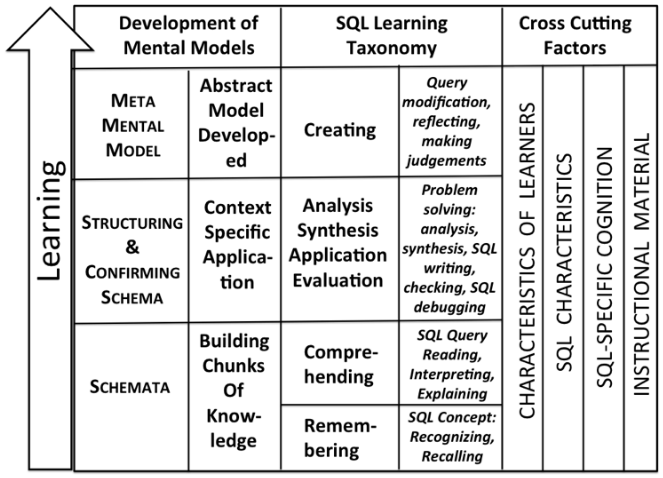

Learning
Development of Mental Models
SQL Learning Taxonomy
Cross Cutting Factors
META MENTAL MODEL
Abstract Model Develop-ed
Creating
Query modification, reflecting, making judgements
STRUCTURING & CONFIRMING SCHEMA
Context Specific Applica-tion
Analysis Synthesis Application Evaluation
Problem solving: analysis, synthesis, SQL writing, checking, SQL debugging
SCHEMATA
Building Chunks Of Know-ledge
Compre-hending
SQL Query Reading, Interpreting, Explaining
Remem-bering
SQL Concept: Recognizing, Recalling
CHARACTERISTICS OF LEARNERS
SQL CHARACTERISTICS
SQL-SPECIFIC COGNITION
INSTRUCTIONAL MATERIAL

Figure 1. A Model of SQL Learning

SQL patterns' identification process is explored in Section II. The processes that involve SQL pattern identification using text mining are explored in Section III. The SQL pattern identification phase using novices' observation is presented in Section IV while expert observation is considered in Section V. Section VI discusses and Section VII concludes.

II. SQL Pattern Identification

To identify our SQL patterns, we commenced by studying other pattern identification methods and procedures. Patterns are not an optimistic or ephemeral collection of ideas. They encapsulate specific tried-and-tested best practice techniques specific to a particular field [8]. Patterns do not state obvious solutions to trivial problems nor do they cover every possible eventuality, but they do capture important "big ideas" [9]. A pattern should explain *how* a problem should be solved and *why* the presented solution is appropriate in a particular context. Alexander [8] points out that patterns may be discovered by identifying a problem and later finding a solution or by seeing a positive set of examples and abstracting a common solution. Coad and Mayfield [10] suggest that patterns are based on a designer's experience of the area.

The SQL pattern development process needed to focus on both the behavioral and the cognitive aspects of SQL acquisition. Understanding learner ability to perform different cognitive tasks such as query formulation, translation, and writing is essential to be able to design this new SQL instructional material.

The SQL patterns we derived emerged from an iterative research process, which involved a review of educational research, uncovering relevant human factors related to SQL usability as well as psychology-related research. The process aimed to accommodate the nature of SQL acquisition. We explored this process by conducting a general overview of the literature about educational theory and cognitive psychology research [4], [5], [11] and instructional design research. The next step narrowed to cover Computer Science (CS) educational research [12] and focused on fostering of problem solving skills.

Having confirmed the possibility of using patterns to support SQL acquisition, we proceeded to identify and define the patterns using text mining followed by observation of novices and experts.

III. Pattern Mining

According to Bruner [15] new instructional methods need first to apply what educators know about how students learn, remember, and use related skills. A text mining procedure was therefore used to extract this information from texts, to discover patterns from existing knowledge repositories, solutions, or designs. This process captures practice that is both good and significant [14].

This identified common knowledge arguably represents the core concepts and practices of SQL query writing. There is a strong precedent for this approach [13].

The mining process was carried out manually based on the dimensions defined in the SQL Learning Model (Fig 1), and is depicted in Fig. 2.

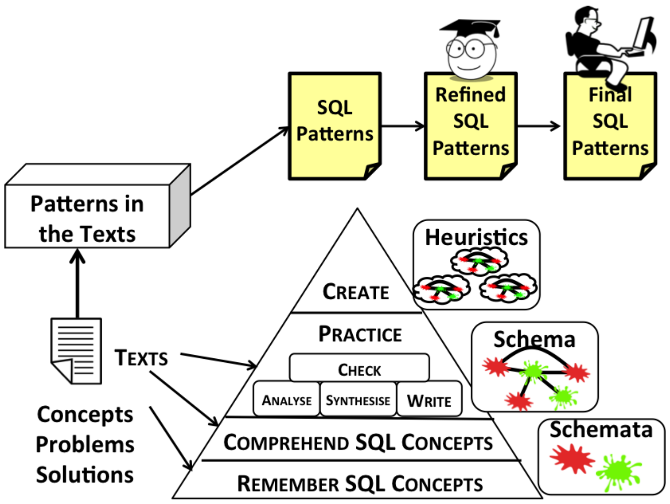


Figure 2. Mining Process

The following steps were followed during the mining process:

1. Identifying Data:
   a. Identify expected SQL knowledge from database texts and categorize the knowledge into the SQL learning model categories.
   b. Identify the declarative or "remembering" SQL concepts. Here, we mined data such as SQL facts or concepts. For example joining, aggregation, sub-query.

2. Identifying Information:
   a. Identify the procedural or "comprehension" SQL concepts, and how they are used in a particular context.
   b. Identify the "practice" skills by considering how concepts should be applied in solving problems. For example, show a context scenario and explain how the relevant syntax and rules are applied. Also illustrate the scenario with appropriate examples which show, step-by-step, how such a concept should be applied.
3. Identifying Knowledge:
   a. Investigate the "creating" activity. For example, find evidence of generic principles being applied in particular contexts.
4. Identify the SQL misconceptions that could potentially be corrected by the patterns.

The mining process was informed by knowledge management research that distinguishes between data, information and knowledge [16].

The process of knowledge extraction and categorization led to an initial set of patterns. The mining process provided a starting point, delivering a static understanding of how SQL core knowledge is presented in textbooks and commonly-used texts. How such SQL concepts are applied in reality, by query writers, could not be gauged without a field investigation. The next step was to observe and analyze novice SQL problem-solving behavior.

IV. OBSERVING NOVICES

Researchers in the field of pattern identification agree on the value of direct observation in arriving at patterns. "*In order to discover patterns which are alive we must always start with observation*" [8] (p.254). Furthermore, [17] points out that "*Patterns are not created or invented; they are identified via an invariant principle*".

Strategy identification, by means of learner observation, helps determine what gaps "best practice" SQL patterns should fill. Cognitive science suggests giving learners a problem and observing everything they do and say while attempting a solution. Unstructured observations were thus conducted over a period of two semesters.

The focus of the observation was on the following particular aspects:

- Remembering:
  - When they remembered the required concepts, were they correct?
- Searching (Not Remembering):
  - How were the required, but forgotten, concepts obtained? For example, did they refer to textbooks or teaching materials, or did they search the Web to find similar problems and related solutions.
- Problem Solving:
  - Were the required concepts identified correctly?
  - Were the gathered concepts correctly matched to the given problem context?
  - Did they search for examples on the Web?
  - Did they try different possible solutions? If so, why was a particular solution selected?
  - How did they react to errors?

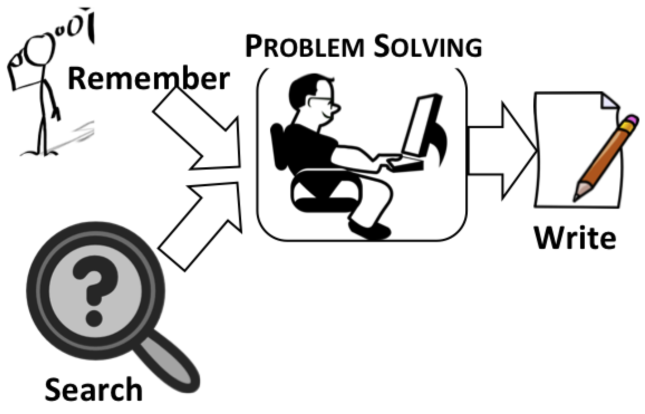


Figure 3. Novice Observation

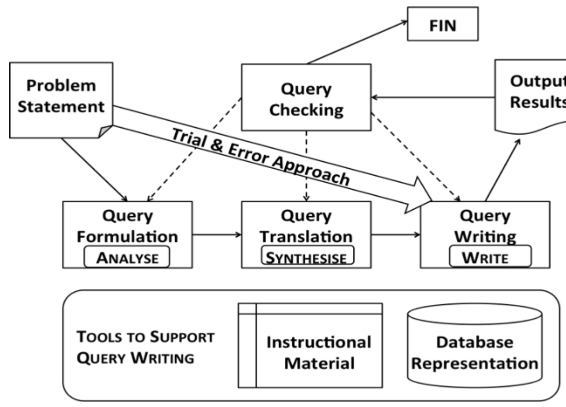
The observation data, based on observation of 63 students (see Table I), revealed that many students lacked problem solving skills. Students often started to write SQL queries without taking the time to consider different approaches. There was no attempt to choose an optimal approach from a number of candidate approaches. They behaved tactically and did not take time to analyze the problem description and to consider what they should do before attempting to write the query. This tendency confirms previous research findings [18]. Students spent the bulk of their time solving syntax and semantic errors and assessing the correctness of the generated results.

TABLE I: STUDENTS' DEMOGRAPHICS INFORMATION

Time	Participants	Participants
2009/10	Students registered for 2nd year course	17
2010/11	Students registered for 2nd year course	21
2010/11	Students registered for third year course	15

Furthermore, novices lacked the ability to sub-divide the problems into sub-problems or to identify the specific knowledge required to solve individual sub-problems [19]. If they *do* divide and conquer, they then have to synthesize identified sub-solutions to design a complete solution to the problem. This, too, seems to be a skill that novices lack (Fig. 4).

Less searching behavior than anticipated was observed and when it did take place it was often unproductive. Students searched for similar problems on the Web or spent some time looking at the lecture notes, trying to understand different concepts. This was often unproductive since they wasted time searching for irrelevant concepts.



Analogy	Sources of analogy
A1→ Formulating the problem "number of loan …"	Schemata
D1→ Dividing the problem into sub problem or sub goals	Advanced knowledge
A2:→ Analysis : Joining two tables "joining the copy with titles"	Basic knowledge
R1→Reasons why *D1* " to get single table"	Basic knowledge
A3→ writing: Write SQL syntax	Basic knowledge
D2→ use subquery	Advanced knowledge
A4→Evaluation: Execute the *A3* without applying *D2* and checking the results	Basic knowledge
D5→to apply Self join for the table	Advanced knowledge
A12→ Writing: Modifying the query	Basic knowledge
A13→Evaluation: Modifying the query with no clear decision	Schemata
A14→Writing Applying D5	Schemata
A15: Analysis of the problem	Schemata
Applying aggregation	Advanced knowledge
A16: Writing: Iteration of changing the query	Advanced knowledge

Figure 4. Problem Solving Stages, and Novice Strategies

Observation of novices was invaluable in understanding how to design supporting instructional material. However, we needed also to understand which particular strategies were deployed by SQL experts since this was the behavior we wanted to guide the novices towards. What emerged from this analysis was the fact that an intervention was required to support students during problem solving, where they apply the basic SQL concepts and principles.

The initial set of SQL patterns were refined based on the observation process.

V. Expert Observation

Professionals have acquired knowledge and skills through study and practice over the years and are termed "expert". Patterns are means of codifying experts' knowledge and expertise to facilitate knowledge transfer. Pattern content must be informed by experts' actual practices. This section presents a description of problem solving strategies deployed by two individual expert SQL query writers. The experts were master students at Glasgow University who had experience using SQL. The tasks they solved are shown in Fig. 5. The aim of this observation was to determine how experts solved these problems, as opposed to novices.

Q1: Give the titles of books that have more than one author.
Q2: Display the names of borrowers who have never returned a book late

Figure 5. Expert Observation task

The cognitive activities performed during problem solving of two tasks were recorded by employing a "talk-aloud" protocol [20].

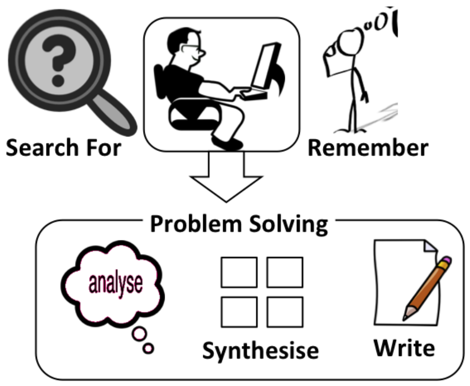


Figure 6. Expert Observation process

The observation process recorded all cognitive activities (see Fig. 7), such as schemata retrieval (Remembering) and Searching (Not Remembered). The collected information was categorized as conceptual (basic building blocks from Fig. 1), schemata (knowledge of how concepts are used), or rule (abstract heuristic knowledge) as recommended by [7], [15], [21].

Experts, after reading the problem description, made an initial decision about the type of technique that had to be applied. They then looked at the provided data model and verbally listed the possible approaches to solving the problem that they could deploy. After mulling it over, they settled on one particular approach and provided reasons for discarding the other options. Both experts used a divide-and-conquer approach and sub-divided the problem: they did not attempt to write the whole query at once. They wrote and tested the commands related to the sub-queries and then synthesized the sub-solutions to arrive at the final complete solution.

The most interesting part of this observation was the fact that the experts applied an implicit pattern matching approach to their assessment of the problem. They clearly tried to match a number of different learned heuristics to the problem before settling on the best approach. One can only assume that they had internalized a number of abstract heuristics which they tried to match to the given problem before settling on a "best-fit" approach.

1-Reading and understanding the problem 2-Search for more information from the Internet "Googling" 3-Problem Solving: a. Analysis ✓ Consulting ER model ✓ Identify the available table holding the required data. ✓ Rereading the problem. b. Synthesis ✓ Deciding which concepts to apply. ✓ Searching for SQL syntax or relevant examples. c. Writing ✓ Start writing the first SQL query in the tool. d. Checking: ✓ Evaluate the result of the first attempt. ✓ Manipulate the query with some justification (fixing the errors). This is done iteratively until they are satisfied. 4-Repeat sub-steps in number 3 until satisfied.	The participant broke the overall problem into a number of sub-problems. He first started by joining Book-Copy and Book-Title tables. At this stage a few actions (A1-A3) and a decision (D1) were performed and other decisions were pending. The participant was happy with his performance at this stage. He then applied another decision, i.e., to use sub-queries The participant then exercised the decision to apply the self-join technique. However, he failed to apply it correctly. The participant then deployed aggregation and was satisfied that he had solved the problem.

Figure 7. The cognitive activities experts deployed

The analysis acted to inform research into the type of approach that ought to be nurtured in novices. The next section discusses how the reported results contributed to the SQL pattern identification.

VI. Discussion

The upper half of Fig. 8 shows how experts solve a task using an analogical approach. The model is based on the ideas of [23], which depicts how scientists think and solve physical problems. The bottom half shows how this model reflects the SQL acquisition process. This model presents the different sources of knowledge and strategies experts deploy.

Observation of expert activities showed that they divided the problem into sub-problems. Then, for each sub problem, different relevant knowledge is applied to arrive at a sub-solution. When experts solved the first part they applied basic knowledge. Then, as the problem requirements required more understanding they applied advanced knowledge which was sometimes obtained by searching. They then applied problem solving strategies such as incremental development, division into sub-queries, consideration of a number of different ways of solving the problem, and choice of the optimal strategy.

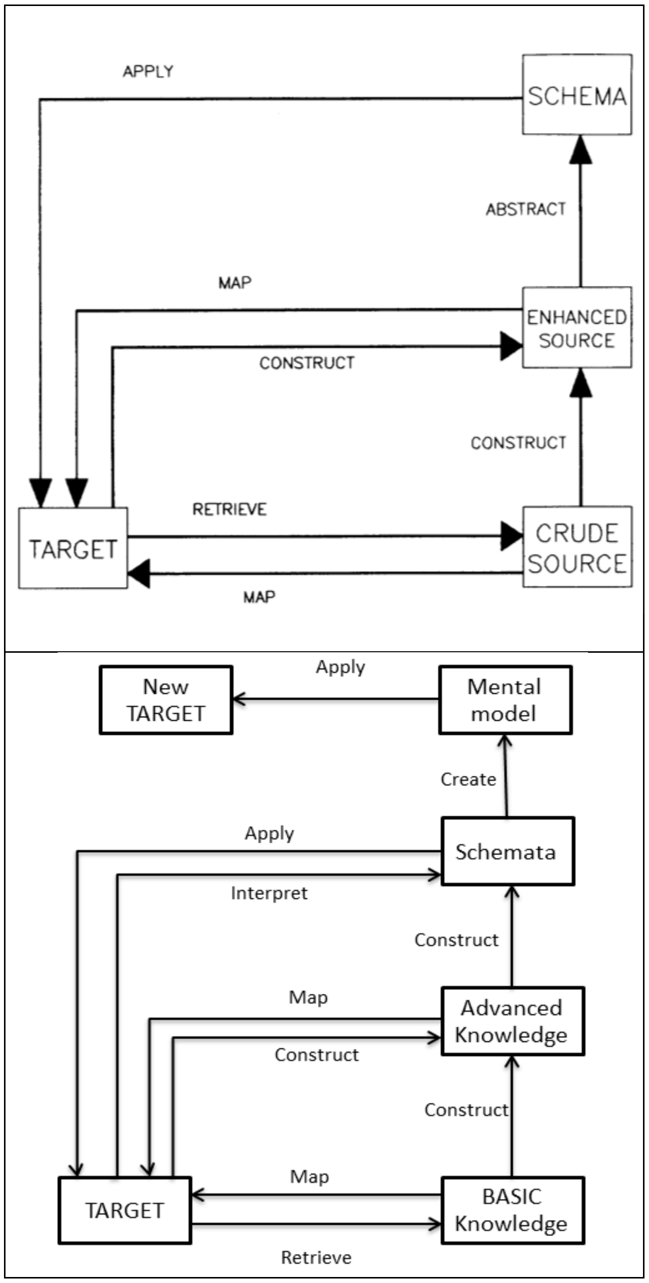


Figure 8. Expert Problem Solving [23](top) and SQL Acquisition on Expert Model (bottom)

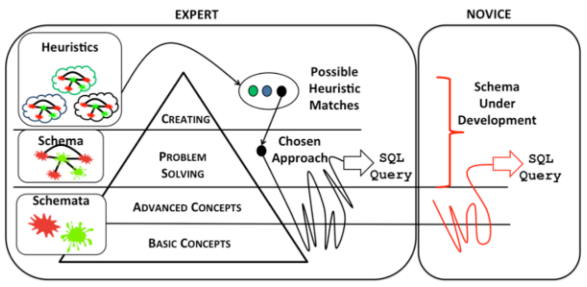


Figure 9. Typical Expert Actions (left) and Novice Actions (right)

Observing experts and novices solving led us to visualize both the expert and novice actions in solving SQL queries (Fig. 9).

Considering how experts approach the task and relating it to novices' actions indicates the nature of the gap between expert and novice. There was no evidence that novices struggled with basic knowledge of SQL syntax. They also knew how the SQL constructs ought to be used. However, novices clearly lacked the knowledge and skills required to solve novel problems. This is related to the "PROBLEM SOLVING" stage as seen in Fig. 9. This, then, is where more effective instruction material needs to assist the process.

This analysis allowed us to determine what type of knowledge and skills are required to solve the SQL problems. We were also able to determine how the information should be presented to learners, i.e., what the optimal sequence of information. The results suggest that:

- Experts start solving the problem by re-formulating the problem statement and determining its context using the data model.
- Expert knowledge is structured, connected and abstract. They have:
  1- Basic knowledge about SQL syntax and semantics "SQL Syntax and Semantics";
  2- Advanced knowledge about the meaning of SQL concepts "SQL Query comprehension" and about how to apply SQL concepts in the given context;
  3- Heuristics:
    - Knowledge about the wisdom of SQL applicability in a certain context "problem-context-solution". This is a high level of knowledge that novice lack as was discussed.
    - Knowledge about the consequences of applying SQL concepts "impact-of-solution". This is a skill of evaluating SQL concepts, which is a high level of knowledge that novices lack.

Observation made it clear that instructional materials, such as notes, did not nudge students towards productive activities or support effective problem solving. To help novices to achieve a measure of SQL expertise we propose that the SQL patterns should include components shown in Table II.

TABLE II. PROPOSED SQL PATTERN CONTENTS

Provide students with data models to help them understand the context of the problem	Schema Formation
The impact of applying the pattern in such a problem context.	Schema Formation
Support for matching a problem to a solution in a simple format such as a checklist	Schema Formation
A section which includes the basic knowledge required to solve the problem.	Schemata
Step-by-step SQL visual examples of the pattern being applied	Schemata
This should be augmented with a step by step plan to train students to deploy effective problem solving strategies, as suggested by (Mayer, 2008).	Encourage Engagement with Analysis and Synthesis Phases during Problem Solving

VII. CONCLUSION AND FUTURE WORK

The successful implementation of instructional materials (SQL patterns) will depend on the pattern writer understanding all the different factors that influence SQL learnability, such as: learner characteristics, SQL language specifications, human cognition and instructional material. The pattern writer must align with established wisdom about human cognition. Our study has provided the guidance to inform SQL pattern content, which should ultimately serve as the link between the task requirement and the generic pattern.

REFERENCES

[1] Al-Shuaily, H., SQL pattern design, development & evaluation of its efficacy. PhD Thesis. University of Glasgow. 2013

[2] Merrill, M.D. Knowledge objects and mental models. in Advanced Learning Technologies, 2000. IWALT 2000. Proceedings. International Workshop on. 2000. pp. 244-246.

[3] Mayer, R.E., Learning and instruction 2003: Prentice Hall.

[4] Bloom, Benjamin Samuel, and David R. Krathwohl. Taxonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain. 1956.

[5] Gorman, M.E., Types of Knowledge and Their Roles in Technology Transfer, in The Journal of Technology Transfer Springer Netherlands. 2002, pp. 219-231.

[6] Al-Shuaily., H. Analyzing the Influence of SQL Teaching and Learning Methods and Approaches. in

10th International Workshop on the Teaching, Learning and Assessment of Databases. UK, London. 2012.

[7] Schlager, M.S. and W.C. Ogden, A cognitive model of database querying: a tool for novice instruction. SIGCHI Bull. 17(4) 1986, pp. 107-113.

[8] Alexander, C., The timeless way of building. 1979, Oxford, UK: Oxford University Press.

[9] Winn, T. and P. Calder, Is This a Pattern? IEEE Softw. 19(1) 2002, pp. 59-66.

[10] Coad, P. and M. Mayfield, Object model patterns: workshop report. SIGPLAN OOPS Mess., 5(4) 1994, pp. 102-104.

[11] Anderson, et al., A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives (Complete edition) 2001. New York: Longman.

[12] Bower, M., A taxonomy of task types in computing. SIGCSE Bull. 40(3) 2008, pp. 281-285.

[13] van Welie, M., K. Mullet, and P. McInerney. Patterns in practice: a workshop for UI designers. in CHI'02 extended abstracts on Human factors in computing systems. 2002. ACM.

[14] Fincher, S. and I. Utting. Pedagogical patterns: their place in the genre. in ACM SIGCSE Bulletin. 2002. ACM.

[15] Bruner, J.S., Toward a theory of instruction. Vol. 59. 1966: Belknap Press.

[16] Tian, J., Y. Nakamori, and A.P. Wierzbicki, Knowledge management and knowledge creation in academia: a study based on surveys in a Japanese research university. Journal of Knowledge Managemen. 13(2) t, 2009, pp. 76-92.

[17] Fincher, S. Patterns for HCI and Cognitive Dimensions: two halves of the same story. in Kuljis, J., Baldwin, L., Scoble, R., Proceedings of the Fourteenth Annual Workshop of the Psychology of Programming Interest Group. 2002.

[18] Ramalingam, V., D. LaBelle, and S. Wiedenbeck, Self-efficacy and mental models in learning to program. SIGCSE Bull 36(3) 2004, pp. 171-175.

[19] Lahtinen, E., K. Ala-Mutka, and H.M. Järvinen. A study of the difficulties of novice programmers. in ACM SIGCSE Bulletin. 2005. ACM.

[20] Dunbar, K., How scientists think: On-line creativity and conceptual change in science. Creative thought: An investigation of conceptual structures and processes. 1997, pp. 461-493.

[21] Ogden, W.C., Implications of a cognitive model of database query: comparison of a natural language, formal language and direct manipulation interface. ACM SIGCHI Bulletin. 18(2) 1986, pp. 51-54.

[22] Reisner, P., Human Factors Studies of Database Query Languages: A Survey and Assessment. ACM Comput. Surv. 13(1) 1981, pp. 13-31.

[23] Nersessian, N.J., How do scientists think? Capturing the dynamics of conceptual change in science. Cognitive models of science 15 1992, pp. 3-44.

Building a General Pattern Framework via Set Theory: Towards a Universal Pattern Approach

Alexander G. Mirnig, Manfred Tscheligi
Christian Doppler Laboratory for "Contextual Interfaces"
HCI & Usability Unit, ICT&S Center, University of Salzburg
Salzburg, Austria
Email: {firstname.lastname}@sbg.ac.at

Abstract— **Patterns have been successfully employed for capturing knowledge about proven solutions to reoccurring problems in several domains. Despite that, there is still little literature available regarding pattern generation or common pattern quality standards across the various domains. We present an attempt for a universal (i.e., domain independent) pattern framework. Via basic set theory, it is possible to describe pattern sets that are composed of several subsets regarding pattern types, quantities, sequence, and other factors. We can thus describe patterns as sets of interrelated elements instead of isolated entities, thus corresponding with the scientific reality of complex problems with multiple relevant factors. The framework can be used to describe existing pattern languages and serve as a basis for new ones, regardless of the domain they are or were created for.**

Keywords-patterns; pattern basics; pattern framework; set theory

I. INTRODUCTION AND MOTIVATION

Patterns have been used as a tool for capturing knowledge about proven solutions to reoccurring problems in many domains. Most prominent among these domains are architecture and software design [1][2][6]. Patterns allow documenting knowledge about methods and practices in a structured and systematic manner. Another major benefit of patterns is that they can serve to "make implicit knowledge explicit" [10], i.e., they can be used to explicitly capture what is normally only acquired via experience after having worked in a certain field or domain for an extended period of time. The information contained in such patterns can then be provided to others (researchers or other interested parties) in a relatively quick and efficient manner. Despite this, there is little general (i.e., domain independent) literature available on patterns and pattern creation.

Having access to a structured collection of implicit and explicit knowledge about research practices is useful when conducting research in any domain. There is no *How to Generate Patterns in 10 Easy Steps* or similar basic literature. This is not an entirely new idea [8], and there has already been a big push in that direction by, e.g., the work of Meszaros and Doble [11] and Winn and Calder [13], which we want to expand and build upon.

Two of the main benefits of patterns are that they facilitate re-application of proven solutions and that they serve to make implicit knowledge explicit. These benefits are of particular importance to researches, who do not already have this knowledge themselves, i.e., it is a way to draw from a vast pool of knowledge. If working with patterns has extensive domain experience as a prerequisite, then those that would need that knowledge the most would benefit the least from it. The final goal of this research is to arrive at a structured but still easy to understand framework that captures the essence of patterns and makes them understandable as well as usable for practitioners and researchers in any domain. The first step is to provide a basic set theoretic analysis that allows to describe patterns and pattern languages in a general manner. This later on serves as a domain independent basis for reflections on how patterns can or should be created and structured.

We argue for a general strand of research on patterns as a means to capture knowledge about research practices. With such a theoretical basis available, practitioners from any domain could have a pool of knowledge to draw from, which would help them create patterns suitable for their needs. This should not mean that a variety in pattern languages and approaches is not desirable. It makes sense to assume that different domain requirements need different pattern approaches. However, the basics of patterns should ideally be similar for everyone and easily accessible, like with general mathematics. A statistician needs and employs different mathematical means than a fruit vendor. But both draw from the same pool of general mathematics as their basis. In our research, we take a step back, look at patterns from a general point of view and describe them via basic set theory [5]. A general analysis of patterns allows us to treat them as separate phenomena, independent of the domains they are created and used in. Set theory is one of the most basic, but at the same time very powerful, mathematical tools available. By using set theory, we can ensure consistency of our framework, while still keeping things basic and relatively easy to understand. An additional benefit of our approach is that it permits the creation of pattern sets across different pattern languages that address a similar purpose. This can facilitate the consolidation of already existing knowledge within the various domains. In this paper, we begin with an overview of existing general literature on patterns in Section II, followed by an outline of the initial proposed set theoretic pattern framework in Section III. In Section IV, we present our planned next steps for further iteration and finalization of the framework and conclude with a few paragraphs on the perceived advantages and possible future challenges of our approach.

II. Related Work

Patterns have been employed in a multitude of application domains [1][6][12] and a good number of extensive pattern collections [3][4][7] have been created in the past. Literature on the pattern generation process itself, sometimes also referred as *pattern mining* [4], is still scarce [9]. Existing literature on pattern generation is mostly focused on specific domains [3][6][8][12]. The work of Gamma et al. [4] can be considered important elementary literature, but it is still centered on software design. Although covering a wide spectrum of software design problems, it is arguably of limited applicability outside of the software engineering domain. The same can be said about other specialized pattern generation guidances [8], which would require adaptation to be employed in other domains (e.g., biology or linguistics).

Meszaros and Doble [11], developed a pattern language for pattern writing, which serves to capture techniques and approaches that have been observed to be particularly effective at addressing certain reoccurring problems. Their *patterns for patterns* were divided into the following five sections: Context-Setting Patterns, Pattern Structuring Patterns, Pattern Naming and Referencing Patterns, Patterns for making Patterns Understandable, Pattern Language Structuring Patterns. Another interesting approach being quite similar in its aims to the one presented in this paper, is the *Pattern Language for Pattern Language Structure* by Winn and Calder [13]. They identified a common trait among pattern languages (i.e., they are symmetry breaking) and built a rough, nonformal general framework for pattern languages in multiple domains. These ideas are similar in concept to what we pursue in our research. The difference is that we want to provide a purely formal framework without or minimal statements regarding its content (such as types or traits). We want to focus on the basics behind patterns and structure these, so that they can be applied as widely as possible, although we intend to incorporate the work of Meszaros and Doble, and Winn and Calder at a later stage (see Section IV).

Another interesting aspect of patterns is that one single pattern is usually not enough to deal with a certain issue. Alexander et al. [2] already expressed this by stating the possibility of making buildings by “stringing together patterns“. The pattern itself, however, does not include the information of which other pattern might be relevant in a particular case. This information is only available once the pattern is part of an actual pattern language. Borchers [3] introduced the notion of high level patterns, which reference lower level patterns to describe solutions to large scale design issues. This hierarchy is expressed via references in the patterns themselves, which is a good way of understanding and describing patterns as interconnected entities. A suitable framework for patterns and pattern languages should ideally be able to capture these relations between patterns.

III. The General Pattern Framework

A. Patterns and Pattern Sets

Before starting to build the framework, we first need to take a look at patterns, pattern languages, and the concepts behind them. Alexander [2] characterized patterns in the following way: “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.” So on a basic level, patterns can be understood as a structured assortment of statements. A pattern language is a complete hierarchy of patterns, ordered by their scope [12]. We will translate these concepts into a basic set theoretic structure by employing regular sets, ordered sets, and subsets, via the following example based on a Contextual User Experience (CUX) pattern structure by Krischkowsky et al. [8] (see Tab.1). Please note that this analysis would work for any pattern language that is or can be structured in a similar way, such as, e.g., the design patterns template laid out by Gamma et al. [6], but we wanted to give a more current and not software-centered example to prove our point.

TABLE I. CUX Pattern Structure [8]

Instructions on Each Pattern Section		
#	***Section Name***	***Instruction on Each Section***
1	Name	*The name of the pattern should shortly describe the suggestions for design by the pattern (2-3 words would be best).*
2	UX Factor	*List the UX factor(s) addressed within your chosen key finding (potential UX factors listed in this section can be e.g., workload, trust, fun/enjoyment, stress...). Please underpin your chosen UX factor(s) with a definition.*
3	Key Finding	*As short as possible - the best would be to describe your key finding (either from an empirical study or findings that are reported in literature) in one sentence.*
.		
.		
.		
8	Key-words	*Describe main topics addressed by the pattern in order to enable structured search.*
9	Sources	*Origin of the pattern (e.g., literature, other pattern, studies or results)*

We now want to generate an actual pattern language set, let us call it CUX Language (and refer to it as *CL* for brevity’s sake), based on the structure outlined in Tab. 1. We can do so by introducing nine subsets (i.e., sets of the set *CL*) CL_1 to CL_9, each subset corresponding to one of the nine categories (from Name to Sources, respectively) described above. To actually generate a pattern for *CL*, we need to assign statements to each of the nine subsets. We do that by assigning a yet undefined set of statements *S* to *CL*, making sure that none of the subsets remains empty. Note that 'statement' in this regard not only refers to full sentences, but also to single words or sequences of words which are not full sentences. We can generate n-number of *CL*-patterns P_1 to P_n this way.

Of course, simply arranging patterns into sets and subsets does not in itself guarantee that any of these patterns are actually useful or reasonable. What this analysis *can* tell us is (a) the pattern language (*CL*) the patterns are generated in, (b) how many statement categories a successful pattern generated in that language must contain, and (c) which statements can be found in which category, i.e., the patterns themselves. So, this elementary analysis has already yielded a powerful starting framework, via which we can express in a domain-independent manner how patterns and pattern languages stand in relation to each other, regardless of domain they were generated in.

B. Descriptors

CL does not yet fully qualify as a pattern language in the actual sense of the word and there are two things that an individual *CL*-pattern does not tell the reader at this stage. These are (a) which *other* patterns might be useful or even necessary for a given purpose, and (b) exactly at which point during a given task or activity and in which order will they be needed. Without knowing these, one can only guess what else they might need upon being presented with only a single pattern or depend on prior experience. It would be undesirable and arguably defeat the purpose of patterns, if extensive meta-knowledge were necessary to be able to use them successfully. This is why we enrich the basic set theoretic framework with specialized descriptor sets, which serve to understand patterns in context with each other. We shall again illustrate this via a simple example: Assume that we have three patterns, P_1 to P_3, which would help us in conducting a user study in the car. P_1 and P_2 are *CL*-patterns to reduce user distraction, whereas P_3 is a pattern about processing the data gained from the study. P_3 was created in a different pattern language, let us call that one *DL*. We can now specify which of these patterns we want or need and in which order by introducing an *ordered set* D. Let us further assume that we want to express that we need only one CL-pattern as well as the DL-pattern and that the DL-pattern will be needed after the CL-pattern. We can express all of this via the following example descriptor set D_1. Please note, that angle brackets ('<' and '>') denote an ordered set, as opposed to an ordinary set, which would be denoted by curly brackets ('{' and '}').

$$D_1: \langle CL^1, DL^1 \rangle \qquad (1)$$

Instead, if we need both CL-patterns, we can express this via the following modification to D_1:

$$D_2: \langle CL^2, DL^1 \rangle \qquad (2)$$

Considering the fact that D_1 in (1) does not tell us which of the two CL-patterns is needed, we could also specify a pattern directly, if not any of them would do:

$$D3: \langle P2, DL1 \rangle \qquad (3)$$

But how do we now specify which of these descriptors (D_1-D_3) is the appropriate one for a given scenario? Patterns are created for a certain purpose, and in most cases that purpose is how to deal with a certain reoccurring problem. We can specify which descriptor fits a certain purpose better than another. To properly express this, we introduce the notion of targets *T* that contain the general purpose of a certain activity (e.g., car user experience). This is different from the problem-field of a pattern, since a given high level pattern could very well reference a lower level problem that addresses a different problem, while both serve the same general purpose. We can now map descriptors to targets, depending on what is needed. In this example, a target that does not require both CL-patterns would be assigned either D_1 or D_3, whereas one that does would be assigned D_2. So, in addition to being able to specifying the relations between patterns in a single pattern language, we are not confined to that single pattern language. This means, that we can also describe hierarchical pattern sets from different domains and pattern languages in the same framework. By adding one additional layer (targets and descriptors) to what was already available before, we have arrived at a highly modular and flexible pattern framework. Fig. 1 provides an overview of the interrelation of pattern languages, patterns, descriptors, and targets.

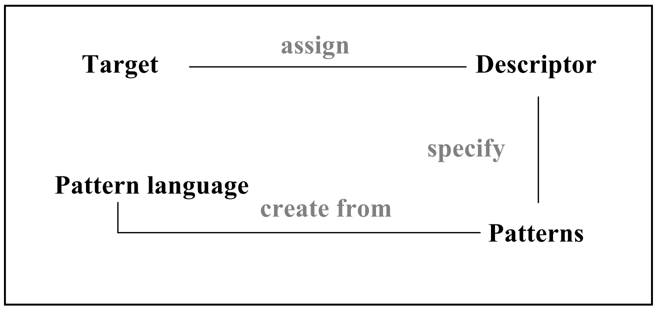


Figure 1. The Pattern Framework – Overview

Currently, one problem of this proposed structure is that any sets of statements can be made into a set and called a pattern language. This is hardly acceptable, of course, and needs to be rectified. We are currently still working on identifying requirements, that any set of statements should fulfill in order to be called a pattern language and what the respective descriptors should look like. Descriptors are ordered sets themselves, so each of their elements have a clearly defined spot in a concrete sequence. This rigid structure also means that it can be difficult to express that particular patterns could be relevant at any point in the sequence or at several fixed points. But it cannot be assumed that all patterns would only ever be relevant at one very specific point. Even if that were the case, it could similarly not be assumed that these specific points were always known. A refinement of the descriptor sets will be in order, to permit more numerous and less cumbersome expression possibilities with regard to sequences.

IV. Next Steps

The framework outlined in this paper is still a work in progress. To arrive at a sufficiently detailed and refined framework, we are currently working on the following:

A. *Refine the Framework: Pattern Languages, Descriptors, Targets*

We intend to pursue a refinement of pattern language requirements similar to Winn and Calder's approach [13] and introduce symmetry breaking (or a similarly suitable property) as a necessary property of descriptors. We will then research the requirements a descriptor and its subsets need to fulfill in order to acquire that property. A more detailed analysis of pattern languages as hierarchical structures and how this translates into concrete descriptor sets is being worked on. Descriptors permit specifying patterns directly, via specifying the language they are part of, or with regard to other additional factors. The type of a pattern could be regarded as one such relevant factor. We will, therefore, incorporate the notion of pattern types into the framework, in particular the pattern types put forward by Meszaros and Doble [11]. We consider these as particularly important in this regard, due to their general nature, but also analyze domain-specific pattern types (e.g., the three types of software design patterns put forward by Gamma et al. [6]) will have to be taken into consideration. In addition, we will provide a more concrete structure for targets, with more detailed information on what a target is and the information it should contain.

B. *Apply the Framework*

Once the framework has been completed, we will demonstrate the suitability of the framework by actually generating and describing sample pattern sets from two very different domains (e.g., User Experience (UX) research and Neuroscience).

V. Conclusion

The great advantage of the approach described in this paper is that patterns are separate from descriptors, which are themselves separate from the targets. This means that patterns can be generated as usual, descriptors generated and assigned on an as-needed basis. For the pattern user, this means that they do not have to scour vast databases of patterns for those they might need. All they need is to have a look at the descriptor(s) that is/are assigned to the target they have in mind. Thus, existing pattern databases can be expanded with descriptors, which help make them more usable and reduce the amount of domain experience and previous knowledge required in order to employ patterns successfully.

But even more importantly, descriptors can specify patterns with regard to certain properties, such as pattern language, context, etc. Descriptors functions similarly to references are contained in the patterns themselves (as suggested by Borchers [3]), but enable additional or alternative references to other patterns at any time, since they are not actual parts of a pattern. This means that descriptors can be used to describe virtually any pattern set, regardless of which domain(s) its patterns came from or when the pattern was created. Not only is it possible to capture the hierarchical order of existing pattern languages via descriptors, but also reference patterns from other languages that might fit a certain purpose. This means that the framework is not tied to a single pattern language or even a single domain and permits references to patterns from multiple pattern languages. We therefore consider it a suitable basis for domain independent pattern research.

Acknowledgment

We gratefully acknowledge the financial support by the Austrian Federal Ministry of Economy, Family and Youth and the National Foundation for Research, Technology and Development (Christian Doppler Laboratory for „Contextual Interfaces").

References

[1] C. Alexander, The Timeless Way of Building, New York: Oxford University Press, 1979.

[2] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, A Pattern Language: Towns, Buildings, Construction, Oxford: University Press, 1979.

[3] J. Borchers, A pattern approach to interaction design, New York: John Wiley & Sons, 2001.

[4] A. Dearden and J. Finlay, "Pattern Languages in HCI: A Critical Review," HCI, Volume 21, January 2006, pp. 49-102.

[5] K. Devlin, The Joy of Sets: fundamentals of contemporary set theory, 2nd ed., Springer, 1993.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: elements of reusable object-oriented software, Boston: Addison-Wesley Professional, 1995.

[7] S. Günther and T. Cleenewerck, "Design principles for internal domain-specific languages: a pattern catalog illustrated by Ruby," Proc. 17th Conf. on Pattern Languages of Programs (PLOP '10). ACM, New York, NY, USA, , Article 3 , pp. 1-35, DOI=10.1145/2493288.2493291, retrieved: April, 2014.

[8] A. Krischkowsky, D. Wurhofer, N. Perterer, and M. Tscheligi, "Developing Patterns Step-by-Step: A Pattern Generation Guidance for HCI Researchers," Proc. PATTERNS 2013, The Fifth International Conferences on Pervasive Patterns and Applications, ThinkMind Digital Library, Valencia, Spain, May 2013, pp. 66–72.

[9] D. Martin, T. Rodden, M. Rouncefield, I.Sommerville, and S. Viller, "Finding Patterns in the Fieldwork," Proc. Seventh European Conf. on Computer-Supported Cooperative Work, Bonn, Germany, September 2001, pp. 39-58.

[10] D. May and P. Taylor, "Knowledge management with patterns," Commun. ACM 46, 7, July 2003, pp. 94-99, DOI=10.1145/792704.792705, retrieved: April, 2014.

[11] G. Meszaros and J. Doble, "A pattern language for pattern writing," Pattern languages of program design 3, Robert C. Martin, Dirk Riehle, and Frank Buschmann (Eds.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, November 1997, pp. 529-574.

[12] J. Tidwell, "Designing Interfaces : Patterns for Effective Interaction Design," O'Reilly Media, Inc., 2005.

[13] T. Winn and P. Calder, "A pattern language for pattern language structure," Proc. 2002 Conf. on Pattern Languages of Programs - Volume 13 (CRPIT '02), James Noble (Ed.), Vol. 13. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, June 2003, pp. 45-58.

From Pattern Languages to Solution Implementations

Michael Falkenthal, Johanna Barzen, Uwe Breitenbücher, Christoph Fehling, Frank Leymann
Institute of Architecture of Application Systems
University of Stuttgart
Stuttgart, Germany
{falkenthal, barzen, breitenbuecher, fehling, leymann}@iaas.uni-stuttgart.de

Abstract—**Patterns are a well-known and often used concept in the domain of computer science. They document proven solutions to recurring problems in a specific context and in a generic way. So patterns are applicable in a multiplicity of specific use cases. However, since the concept of patterns aims at generalization and abstraction of solution knowledge, it is difficult to apply solutions provided by patterns to specific use cases, as the required knowledge about refinement and the manual effort that has to be spent is immense. Therefore, we introduce the concept of Solution Implementations, which are directly associated to patterns to efficiently support elaboration of concrete pattern implementations. We show how Solution Implementations can be aggregated to solve problems that require the application of multiple patterns at once. We validate the presented approach in the domain of cloud application architecture and cloud application management and show the feasibility of our approach with a prototype.**

Keywords-pattern; pattern languages; pattern-based solution; pattern application; cloud computing patterns

I. Introduction

Pattern and pattern languages are a well-established concept in different application areas in computer science and information technology (IT). Originally introduced to the domain of architecture [2], the concept of patterns recently got more and more popular in different domains such as education [18], design engineering [16], cloud application architecture [23] or costumes [17]. Patterns are used to document proven solutions to recurring problems in a specific context. However, since the concept of patterns aims at generalization and abstraction, it is often difficult to apply the captured abstracted knowledge to a concrete problem. This can require immense manual effort and domain-specific knowledge to refine the abstract, conceptual, and high-level solution description of a pattern to an individual use case. These following examples show that this problem occurs in several domains due to the abstraction of solution knowledge into patterns. For example, if a PHP: Hypertext Preprocessor (PHP) developer uses the Gang of Four patterns of Gamma et al. [19], he is faced with the problem that he has to translate the general solution concepts of the patterns to his concrete context, i.e., he has to implement solutions based on a given programming paradigm predefined by PHP. An enterprise architect who has to integrate complex legacy systems may use the enterprise application architecture patterns of Fowler [20] or the enterprise integration patterns of Hohpe and Wolf [12] to gain insight to proven solutions of his problems; but, these are still generic solutions and he has to find proper implementations for the systems to integrate. This can lead to huge efforts since besides paradigms of used programming languages he has also to consider many constraints given by the running systems and technologies. A teacher who uses the learning patterns of Iba and Miyake [18] has to adapt them to match his prevailing school system with all the teaching methods. To give a final example, a costume designer could use the patterns of Schumm et al. [17] to find clothing conventions for a cowboy in a western film but he still has to come up with a specific solution for his current film.

While patterns in general describe proven generic solutions at a conceptual level, the examples above show that it is still time consuming to work out concrete solutions of those generic solutions.

To overcome this problem, we suggest that patterns should be linked to the (i) original concrete solutions from which they have been deduced (if available) and (ii) to individual new concrete implementations of the abstractly described solution. This enables users that want to apply a certain pattern to take already existing implementations for their use cases, which eases applying patterns and reduces the required manual effort significantly.

The remainder of this paper is structured as follows: we clarify the difference between the common concept of pattern solutions and concrete solutions in Section II. In Section III, we discuss related work and the lack of directly usable concrete solutions in state of the art pattern research. We show how to keep patterns linked to concrete solution knowledge and how to select them to establish concrete solution building blocks, which can be aggregated in Section IV. In Section V, we give an example of how to apply the introduced concepts in the domains of cloud application architecture and cloud application management and verify the feasibility of the presented approach by means of an implemented prototype in Section VI. We conclude this paper with an outline of future work in Section VII.

II. Motivation

Patterns document proven solution knowledge mainly in natural text to support human readers of a pattern. Patterns are often organized into pattern languages, i.e., they may be connected to each other. Pattern languages provide a common *template* for documenting all contained patterns.

This template typically defines different items to be documented such as “Problem”, “Context”, “Solution”, and “Known Uses”. The problem and context section describe the problem to be solved in an abstract manner where the solution describes the general characteristics of the solution – all only conceptually, in an abstract way. Thus, the general solution is refined for individual problem manifestations and use cases resulting in different concrete solutions every time the pattern is applied. The known uses section is the only place where concrete solutions from which the pattern has been abstracted are described. But these are commonly not extended as the pattern is applied nor do they guide pattern readers during the creation of their own solutions.

Therefore, due to the abstract nature of patterns and generalized issues, most pattern languages only contain some concrete solutions a pattern was derived from in the known uses section. This leads to the problem that the user of the pattern has to design and implement a specific solution based on his individual and concrete use case, i.e., a solution has to be implemented based on the user’s circumstances considering the given pattern. However, many patterns are applied several times to similar use cases. Thus, the effort has to be spent every time for tasks that were already executed multiple times. For example, the Model-View-Controller (MVC) Design Pattern is an often used pattern in the domain of software design. This pattern was, therefore, implemented for many applications in many programming languages from scratch, as patterns typically provide no directly usable concrete solutions for use cases in a concrete context. Patterns are not linked with a growing list of solutions that can be used as basis to apply them to individual use cases rapidly: each time a pattern should be applied, it has to be refined manually to the current use case. The provided sections such as “Known Uses” and “Examples”, which are part of the pattern structure in most pattern languages, therefore, support the reader in creating new solutions only partially [10][12][13]: they provide only partial solution refinements or solution templates as written text but not directly applicable implementations that can be used without additional effort. The major reasons for this problem are, that neither the concrete solution is documented in a way that enables reusing it efficiently nor it is obvious how to aggregate existing solutions if multiple patterns are applied together. Thus, the reader of a pattern is faced with the problem of creation and design to elaborate a proper solution based on a given pattern each time when it has to be applied – which results in time-consuming efforts that decrease the efficiency of using patterns.

As of today, patterns are typically created by small groups of experts. By abstracting the problems and solutions into patterns relying on their expertise, these experts determine the content of the patterns. This traditional way of pattern identification created the two issues already seen: first, the patterns are not verifiable because the concrete solutions they have been abstracted from are not traceable (“pattern provenance”) and second the patterns document abstracted knowledge, therefore manual effort and specific knowledge is needed to apply them to concrete problems.

Another problem occurs if multiple patterns have to be combined to create a concrete solution. Pattern languages tackle the problem of aggregating patterns to solve overall problems. As shown by Zdun [9], this can be supported by defining relationships between patterns within a pattern language, which assure that connected patterns match together semantically, i.e., that they are composable regarding their solutions. This means that patterns can be used as composable building blocks to create overall solutions. Once patterns are composed to create overall solutions the problem arises that concrete solutions have to be feasible in the context of concrete problem situations. Referring to the former mentioned example of a PHP developer, the overall concrete solution, consisting of the concrete solutions of the composed patterns, has to be elaborated that it complies with the constraints defined by the programming language PHP. So, the complexity of creating concrete solutions from composed patterns increases with the number of aggregated solutions, since integration efforts add to the efforts of elaborating each individual solution. Thus, to summarize the discussion above, we need a means to improve the required refinement from a pattern’s abstract solution description to directly applicable concrete solutions and their composition.

III. Related Work

Patterns are human readable artifacts, which combine problem knowledge with generic solution knowledge. The template documenting a pattern contains solution sections presenting solution knowledge as ordinary text [2][19][13]. This kind of solution representation contains the general principle and core of a solution in an abstract way. Common solution sections of patterns do not reflect concrete solution instances of the pattern. They just act like manuals to support a reader at implementing a solution proper for his issues.

Iterative pattern formulation approaches as shown by Reiners et al. [6] and Falkenthal et al. [5] can enable that concrete solution knowledge is used to formulate patterns. Patterns are not just final artifacts but are formulated based on initial ideas in an iterative process to finally reach the status of a pattern. Nevertheless, in these approaches concrete solution knowledge only supports the formulation process of patterns but is not stored explicitly to get reused when a pattern is applied.

Porter et al. [15] have shown that selecting patterns from a pattern language is a question of temporal ordering of the selected patterns. They show that combinations and aggregations of patterns rely on the order in which the patterns have to be applied. This leads to so called pattern sequences which are partially ordered sets of patterns reflecting the temporal order of pattern application. This approach focuses on combinability of patterns, but not on the combinability of concrete solutions.

Many pattern collections and pattern languages are stored in digital pattern repositories such as presented by

Fehling [4], van Heesch [7] and Reiners [3]. Although these repositories support readers in navigating through the patterns they do not link concrete solutions to the patterns. Therefore, readers have to manually recreate concrete solutions each time when they want to apply a pattern.

Zdun [9] shows that pattern languages can be represented as graphs with weighted edges. Patterns are the nodes of the graph and edges are relationships between the patterns. The weights of the edges represent the semantics of the relationships as well as the effects of a pattern on the resulting context of a pattern. These effects are called goals and reflect the influence of a pattern on the quality attributes of software architectures. While this approach helps to select proper pattern sequences from a pattern language it does not enable to find concrete solutions and connect them together.

Demirköprü [8] shows that Hoare logic can be applied to patterns and pattern languages such that patterns are getting enriched by preconditions and postconditions. By considering this conditions, pattern sequences can be connected into aggregates respectively compositions of patterns where preconditions of the first pattern of the sequence are the preconditions of the aggregate and postconditions of the last pattern in the sequence are accordingly the postconditions of the aggregate. This approach also only tackles aggregation of patterns without considering concrete solutions.

Fehling et al. [31][33] show that their structure of cloud computing patterns can be extended to annotate patterns with additional implementation artifacts. Those artifacts can represent instantiations of a pattern on a concrete cloud platform. Considering those annotations, developers can be guided through configurations of runtime environments. Although patterns can be annotated with concrete implementation artifacts, this approach is only described in the domain of cloud computing and does not introduce a means to ease pattern usage and refinement in general.

IV. Solution Implementations: Building Blocks for Applying and Aggregating Concrete Solutions from Patterns

In the section above, we summarized the state of the art and identify that (i) concrete solutions are not connected to patterns and that (ii) there are no approaches dealing with the aggregation of concrete solutions if multiple patterns have to be applied together. Even though there are approaches to derive patterns from concrete solution knowledge iteratively [5][6], concrete solutions are not stored altogether with the actual patterns nor are they linked to them. Concrete solutions, thus, cannot be retrieved from patterns without the need to work them out manually over and over again for the same kind of use cases. Therefore, we propose an approach that (i) defines concrete, implemented solution knowledge as reusable building blocks, (ii) that links these concrete solutions to patterns, and (iii) enables the composition of concrete solutions.

A. *Solution Implementations*

We argue that concrete solutions are lost during the pattern writing process since patterns capture general core solution principles in a technology and implementation agnostic way. In addition, applications of patterns to form new concrete solutions are not documented in a way that enables reusing the knowledge of refinement. As a result, the details of the concrete solutions are abstracted away and must be worked out again when a pattern has to be applied to similar use cases. Thus, the benefits of patterns in the form of abstractions lead to effort when using them due to the missing information of concrete realizations. We suggest keeping concrete solutions linked to patterns in order to ease pattern application and enable implementing new concrete solutions for similar use cases based on existing, already refined, knowledge. These linked solutions can be, for example, (i) the concrete solutions which were considered initially to abstract the knowledge into a pattern, (ii) later applications of the pattern to build new concrete solutions, or (iii) concrete solutions that were explicitly developed to ease applying the pattern.

Concrete solutions, which we call *Solution Implementations (SI)*, are building blocks of concrete solution knowledge. Therefore, Solution Implementations describe concrete solution knowledge that can be reused directly. For example, in the domain of software development, Solution Implementations provide code, which can be used directly in the development of an own application. For example, a PHP developer faced with the problem to implement the Gang of Four Pattern MVC [19] in an application can reuse a Solution Implementation of the MVC pattern written in PHP code. Especially, patterns may provide multiple different Solution Implementations – each optimized for a special context and requirements. So, there could be a specific MVC Solution Implementation for PHP4 and another for PHP5, each one considering the programming concepts of the specific PHP version. Another Solution Implementation could provide a concrete solution of the MVC pattern implemented in Java. So, in this case also a Java developer could reuse a concrete MVC solution to save implementation efforts.

By connecting Solution Implementations to patterns, users do not have to redesign and recreate each solution every time a pattern is applied. The introduced Solution Implementations provide a powerful means to capture existing fine-grained knowledge linked to the abstract knowledge provided by patterns. So, users can look at the connected Solution Implementations once a pattern is selected and reuse them directly. To distinguish between pattern's abstract solutions and Solution Implementations, we point out that the solution section of patterns describes the core solution principles in text format and the Solution Implementations represent the real solution objects – which may be in different formats (often depending on the problem domain), e.g., executable code in software development or real clothes in the domain of costumes. Thus, while patterns are documented commonly in natural text, their Solution Implementations depend mainly on the domain of the pattern language and can occur in various forms. Since many specific Solution Implementations can be linked to a pattern, we need a means to select proper Solution Implementations of the pattern to be applied.

B. *Selection of Solution Implementations from Patterns*

Once a user selects a pattern, he is faced with the problem to decide which Solution Implementation solves his problem in his context properly. To enable selecting proper Solution Implementations of a pattern we introduce *Selection Criteria (sc)*, which determine when to use a certain Solution Implementation. The concept of keeping Solution Implementations linked to the corresponding pattern and supporting the selection of a proper Solution Implementation is shown in Figure 1. Selection Criteria are added to relations between Solution Implementations and patterns. Selection Criteria may be human readable or software interpretable descriptions of when to select a Solution Implementation. They provide a means to guide the selection using additional meta-information not present in the Solution Implementation itself.

To exemplify the concept, we give an example of Solution Implementations from the domain of architecture. In this domain addressed by Christopher Alexander [1][2], a Solution Implementation would be, e.g., a real entrance of a building or a specific room layout of a real floor, which are described in detail and linked to the corresponding pattern [1][2]. To find the most appropriate Solution Implementation for a particular use case, Selection Criteria such as the cost of the architectural Solution Implementation or the choice of used material can be considered. For example, two Solution Implementations for the pattern mentioned above that deals with room layouts might differ in the historical style they are built or by the functional purpose like living, industrial or office, etc. Thus, based on such criteria, the refinement of a pattern's abstract solution can be configured by specifying desired requirements and constraints.

To summarize the concept of Solution Implementations it has to be pointed out that solutions in the domain of patterns are abstract descriptions that are agnostic to

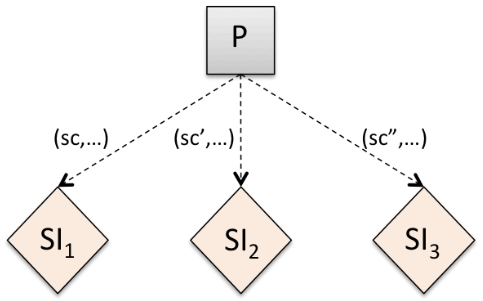


Figure 1. Solution Implementations (SI) connected to a pattern (P) are selectable under consideration of defined Selection Criteria (sc).

concrete implementations and written in ordinary text to support readers. In contrast to this abstract description, we grasp Solution Implementations as fine-grained artifacts, which provide concrete implementation information for particular use cases of a pattern. Solution Implementations are linked to patterns where Selection Criteria are added to the relation between the pattern and the Solution Implementation to guide pattern users during the selection of Solution Implementations.

C. *Aggregation of Solution Implementations*

The concepts of Solution Implementations and Selection Criteria enable to reuse concrete solutions, which are linked to patterns. But most often problems have to be solved by combining multiple patterns. Therefore, we also need a means to combine Solution Implementations of patterns to solve an overall problem altogether. For this purpose, Solution Implementations connected to patterns can have additional interrelations with other Solution Implementations of other patterns affecting their composability. For example, Solution Implementations in the domain of software development are possibly implemented in different programming languages. Therefore, there may exist various Solution Implementations for one pattern in different programming languages, remembering the above example of the PHP and Java Solution Implementations of the MVC pattern. To be combined, both Solution Implementations often have to be implemented in the same programming language.

This leads to the research question "How to compose Solution Implementations selected from multiple patterns into a composed Solution Implementation?"

Patterns are often stored and organized in digital pattern repositories. These repositories, such as presented by Fehling [4], van Heesch [7] and Reiners [3], support users in searching for relevant patterns and navigating through the whole collection of patterns, respectively a pattern language formed by the relations between patterns. To support navigation through pattern languages, these relations can be formulated at the level of patterns indicating that some patterns can be "combined" into working composite solutions, some patterns are "alternatives", some patterns can only be "applied in the context of" other patterns etc. Zdun [9] has shown that pattern languages can be formalized to enable automated navigation through pattern languages based upon semantic and quality goal constraints reflecting a pattern's effect once it is applied. This also enables combining multiple patterns based on the defined semantics. The approach supports the reader of a pattern language to select proper pattern sequences for solving complex problems that require the application of multiple patterns at once. But, once there are Solution Implementations linked to patterns this leads to the requirement to not only compose patterns but also their concrete Solution Implementations into overall solutions.

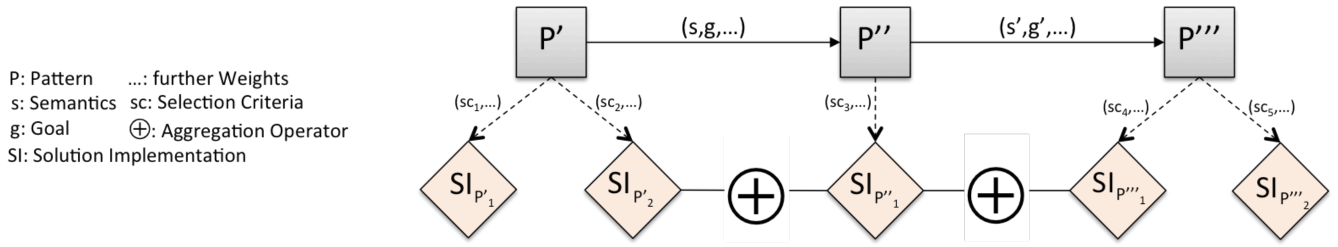


Figure 2. Aggregating Solution Implementations (SI) along the sequence of selected patterns (P).

We extend the approach of Zdun to solve the problem of selecting appropriate patterns to also select and aggregate appropriate Solution Implementations along the selected sequence of patterns.

To assure that Solution Implementations are building blocks composable with each other, we introduce the concept of an *Aggregation Operator,* as depicted in Figure 2. The Aggregation Operator is the connector between several Solution Implementations. Solution Implementations can just be aggregated if a proper Aggregation Operator implements the necessary adaptations to get two Solution Implementations to work together. Adaptions may be necessary to assure that Solution Implementations match together based on their preconditions and postconditions. Preconditions and postconditions are functional and technical dependencies, which have to be fulfilled for Solution Implementations. In Figure 2., the three patterns P', $\mathrm{P}''$and P''' show a sequence of patterns, which can be selected through the approach of Zdun considering semantics (s) of the relations, goals (g) of the patterns and further weights. Solution Implementations are linked with the patterns and can be selected according to the Selection Criteria introduced in the section above. Furthermore, there are two Solution Implementations associated with pattern P' but only Solution Implementation $\mathrm{SI}_{\mathrm{P}'_2}$ can be aggregated with Solution Implementation $\mathrm{SI}_{\mathrm{P}''_1}$ of the succeeding pattern P'' due to the Aggregation Operator between those two Solution Implementations. There is no Aggregation Operator implemented for $\mathrm{SI}_{\mathrm{P}'_1}$, so that it cannot be aggregated with $\mathrm{SI}_{\mathrm{P}''_1}$, but, nevertheless, it is a working concrete solution of P'. So, in the scenario depicted in Figure 2 an Aggregation Operator has to be available to aggregate $\mathrm{SI}_{\mathrm{P}'_1}$ and $\mathrm{SI}_{\mathrm{P}''_1}$.

In general, Aggregation Operators have to be available to compose Solution Implementations for complex problems requiring the application of multiple patterns. Solution Implementations aggregated with such an operator are concrete implementations of the aggregation of the selected patterns. Aggregated Solution Implementations are, therefore, concrete building blocks solving problems addressed by a pattern language.

Aggregation Operators depend on the connected Solution Implementations, i.e., they are context-dependent due to the context of the Solution Implementations. In contrast to the context section of a pattern, which is used together with the problem section to describe the circumstances when a pattern can be applied, the Solution Implementations' context is more specific in terms of the concrete solution. For example, if an Aggregation Operator shall connect two Solution Implementations consisting of concrete PHP code, the operator itself could also be concrete PHP code wrapping functionality from both Solution Implementations. If the Solution Implementations to aggregate are Java class files, e.g., an Aggregation Operator could resolve their dependencies on other class files or libraries and load all dependencies. Afterwards it could configure the components to properly work together and execute them in a Java runtime. Thus, an Aggregation Operator composes and adapts multiple Solution Implementations considering their contexts. Another example on how the Aggregation Operators can be used in very different domains is an example of the domain of costumes in films. When dressing the characters of a western movie usually the sheriff costume pattern and the outlaw costume pattern need to be applied. But there are numerous Solution Implementations of these patterns in terms of concrete sheriff and outlaw costumes, e.g., for different historical time periods. To make sure the costumes of the sheriff and outlaw match together, an Aggregation Operator*,* for example, can ensure that certain Solution Implementations originate from the same time period or the same country and can be used together in one movie. Further the Aggregation Operator adapts Solution Implementations to suit to the settings of a scene in a film, i.e., by adapting the color of the costumes. Thus, the costumes' Solution Implementations are aggregated to solve a problem in combination. Those examples show that Solution Implementations of patterns from different domains have to be aggregated using specific Aggregation Operators. Since different pattern languages deal with different contexts, they can contain different Aggregation Operators to compose Solution Implementations.

V. VALIDATION

To validate the proposed concept of Solution Implementations, this section explains the application of

Solution Implementations in the domains of cloud application architecture and cloud management.

A. Deriving Solution Implementations in the Domain of Cloud Application Architecture

To explain the concept of Solution Implementations in the domain of cloud computing patterns, the example depicted in Figure 3 shows the three patterns *stateless component*, *stateful component*, and *elastic load balancer* from the pattern language and pattern catalogue of Fehling et al. [10][31]. The stateless component and stateful component patterns describe how an application component can handle state information. They both differentiate between session state – the state with the user interaction within the application and application state – the data handled by the application, for example, customer addresses etc. While the stateful component pattern describes how this state can be handled by the component itself and possibly be replicated among multiple component instances, the stateless component pattern describes how state information is kept externally of the component implementation to be provided with each user request or to be handled in other data storage offerings. The elastic load balancer pattern describes how application components can be scaled out: their performance is increased or decreased through addition or removal of component instances, respectively. Decisions on how many component instances are required are made by monitoring the amount of synchronous requests to the managed application components. The elastic load balancer pattern is related to both of the other depicted patterns as it conceptually describes how to scale out stateful components and stateless components: while stateless components can be added and removed rather easily, internal state may have to be extracted from stateful components upon removal or synchronized with new instances upon addition.

As depicted in Figure 3, the stateless component and stateful component pattern both provide Solution Implementations, which implement these patterns for Java web applications packaged in the web archive (WAR) format that are hosted on Amazon Elastic Beanstalk [21] which is part of Amazon Web Services (AWS) [30]. The elastic load balancer has three Solution Implementations implementing the described management functionality for stateful components and stateless components for WAR-based applications on Amazon Elastic Beanstalk and Microsoft Azure [22]. The Selection Criteria "WAR is deployed on Microsoft Azure" respectively "WAR is deployed on Elastic Beanstalk" support the user to choose the proper Solution Implementation. For example, if SI_2 is selected the user knows that this results in a concrete load balancer in the form of a deployed WAR file on Elastic Beanstalk. Since a load balancer scales components, it needs concrete instances of either stateless component or stateful component to work with. Thus, the user can select a proper Solution Implementation for the components based on his concrete requirements considering the Selection Criteria of the relations between the patterns stateless component and stateful component and their Solution Implementations. To assure that Solution Implementations are composable, i.e., that they properly work together, they refine and enrich the pattern relationships to formulate preconditions respectively postconditions on the Solution Implementation layer. The preconditions and postconditions of the elastic load balancer Solution Implementations, therefore, capture which related pattern – stateless component or stateful component – they expect to be implemented by managed components. Furthermore, they capture the supported deployment package – WAR in this example – and runtime environment for which they have been developed: $SI_{3.1}$ of stateless component has the postcondition "WAR on Elastic

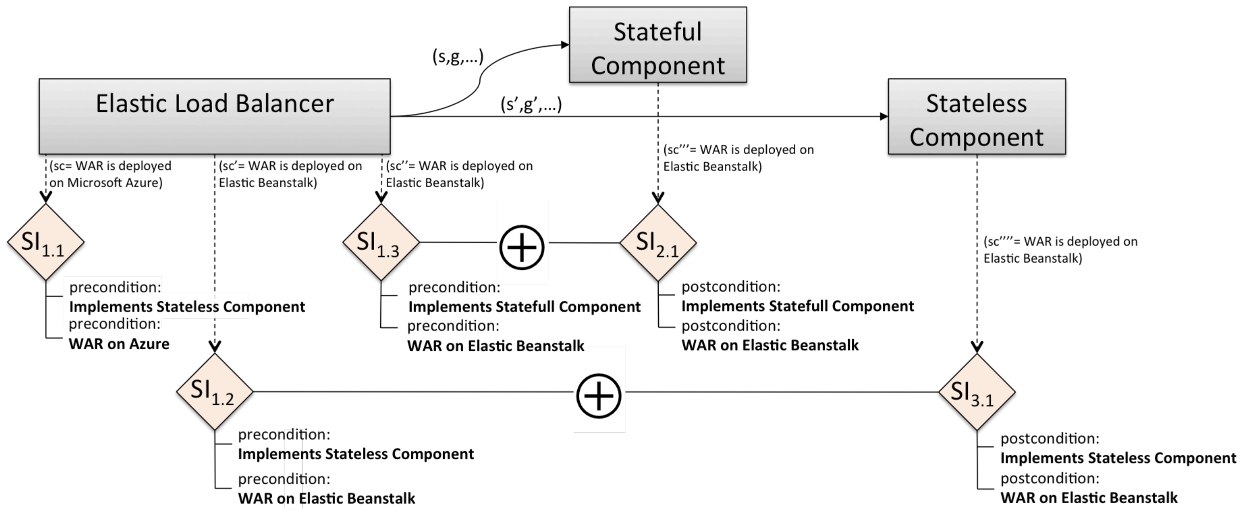

Stateful Component
Elastic Load Balancer
Stateless Component
(s,g,...)
(s',g',...)
(sc= WAR is deployed on Microsoft Azure)
(sc'= WAR is deployed on Elastic Beanstalk)
(sc''= WAR is deployed on Elastic Beanstalk)
(sc'''= WAR is deployed on Elastic Beanstalk)
(sc''''= WAR is deployed on Elastic Beanstalk)
SI1.1
SI1.3
SI2.1
SI1.2
SI3.1
precondition: Implements Stateless Component
precondition: WAR on Azure
precondition: Implements Statefull Component
precondition: WAR on Elastic Beanstalk
postcondition: Implements Statefull Component
postcondition: WAR on Elastic Beanstalk
precondition: Implements Stateless Component
precondition: WAR on Elastic Beanstalk
postcondition: Implements Stateless Component
postcondition: WAR on Elastic Beanstalk

Figure 3. Solution Implementations in the domain of cloud application architecture linked to patterns and aggregated by Aggregation Operators.

Beanstalk" while $SI_{1.2}$ of elastic load balancer is enriched with the precondition "WAR on Elastic Beanstalk" and $SI_{1.1}$ with "WAR on Azure". The previously introduced Aggregation Operator interprets these dependencies and, for example, composes $SI_{3.1}$ and $SI_{1.2}$. During this task, the configuration parameters of the solutions are adjusted by the operator, i.e., the elastic load balancer is configured with the address of the stateless component to be managed. As some of this information may only become known after the deployment of a component, the configuration may also be handled during the deployment.

Following, this example is concretely demonstrated by an AWS Cloud Formation template [28] generated by the Aggregation Operator in Listing 1. An AWS Cloud Formation template is a configuration file, readable and processable by the AWS Cloud to automatically provision and configure cloud resources. For the sake of simplicity the depicted template in Listing 1 shows only the relevant parts of the template, which are adapted by the Aggregation Operator. To run the example scenario on AWS, three parts are needed within the AWS Cloud Formation template to reflect the aggregation of $SI_{3.1}$ and $SI_{1.2}$: (i) an elastic load balancer (MyLB), which is able to scale components, (ii) a launch configuration (MyCfg), which provides configuration parameters about an Amazon Machine Image (AMI) containing the implementation of stateless component as well as a runtime to execute the component in the form of an AWS Elastic Compute Cloud (EC2) [32] instance and, (iii) an autoscaling group (MyAutoscalingGroup) to define scaling parameters used by the elastic load balancer and the wiring of the elastic load balancer and the launch configuration.

```
"MyLB" : {
    "Type" : "AWS::ElasticLoadBalancing::LoadBalancer",
    "Properties" : {
        "Listeners" : [ {
            "LoadBalancerPort" : "80",
            "InstancePort" : "80",
            "Protocol" : "HTTP"
        } ],
    }
},
"MyCfg" : {
    "Type" : "AWS::AutoScaling::LaunchConfiguration",
    "Properties" : {
        "ImageId" : { "ami-statelessComponent" },
        "InstanceType" : { "m1.large" },
    }
},
"MyAutoscalingGroup" : {
    "Type" : "AWS::AutoScaling::AutoScalingGroup",
    "Properties" : {
        …
        "LaunchConfigurationName" : { "Ref" : "MyCfg"},
        "LoadBalancerNames" : [ { "Ref" : "MyLB" } ]
        …
    }
}
```

Listing 1. Extract from AWS Cloud Formation template produced by an Aggregation Operator to aggregate configuration snippets to aggregate elastic load balancer and stateless component.

MyLB defines an AWS elastic load balancer for scaling Hypertext Transfer Protocol (HTTP) requests on port 80. Further, MyCfg defines the AMI ami-statelessComponent in the property ImageId, which is used for provisioning new instances by an elastic load balancer. The autoscaling group MyAutoscalingGroup wires the elastic load balancer and the stateless component instances by means of referencing the properties LoadBalancerNames and LaunchConfigurationName to MyLB and MyCfg, respectively. Since all the mentioned properties are in charge of enabling an elastic load balancer instance to automatically scale and load balance instances of components contained in an AMI, an Aggregation Operator can dynamically adapt those properties based on the selected Solution Implementations to be aggregated. So, presuming that ami-statelessComponent contains an implementation of $SI_{3.1}$, an Aggregation Operator can aggregate $SI_{3.1}$ and $SI_{1.2}$ by adapting the mentioned properties and, therefore, provides an executable configuration template for AWS Cloud Formation. The same principles can be applied to aggregate $SI_{1.3}$ and $SI_{2.1}$ because of their matching preconditions and postconditions. By adapting the ImageId of the LaunchConfiguration to an AMI, which runs an AWS EC2 instance with a deployed stateful component, the Aggregation Operator can aggregate $SI_{1.3}$ and $SI_{2.1}$.

Further, $SI_{1.1}$ has precondition "WAR on Azure" and is, therefore, incompatible with $SI_{2.1}$ and $SI_{3.1}$, i.e., $SI_{1.1}$ cannot be combined with these Solution Implementations due to their preconditions and postconditions. The selection of a Solution Implementation, therefore, may restrict the number of matching Solution Implementations of the succeeding pattern since postconditions of the first Solution Implementation have to match with preconditions of the second. This way, the space of concrete solutions is reduced based on the resulting constraints of a selected Solution Implementation. To elaborate a solution to a overall problem described by a sequence of patterns exactly one Solution Implementation has to be selected for each pattern in the sequence considering its selection criteria to match non-functional requirements, as well as postconditions of the former Solution Implementation.

B. Deriving Solution Implementations in the Domain of Cloud Application Management

In this section, we show how the presented approach can be applied in the domain of cloud application management. Therefore, we describe how applying *management patterns* introduced in [10][29] to cloud applications can be supported

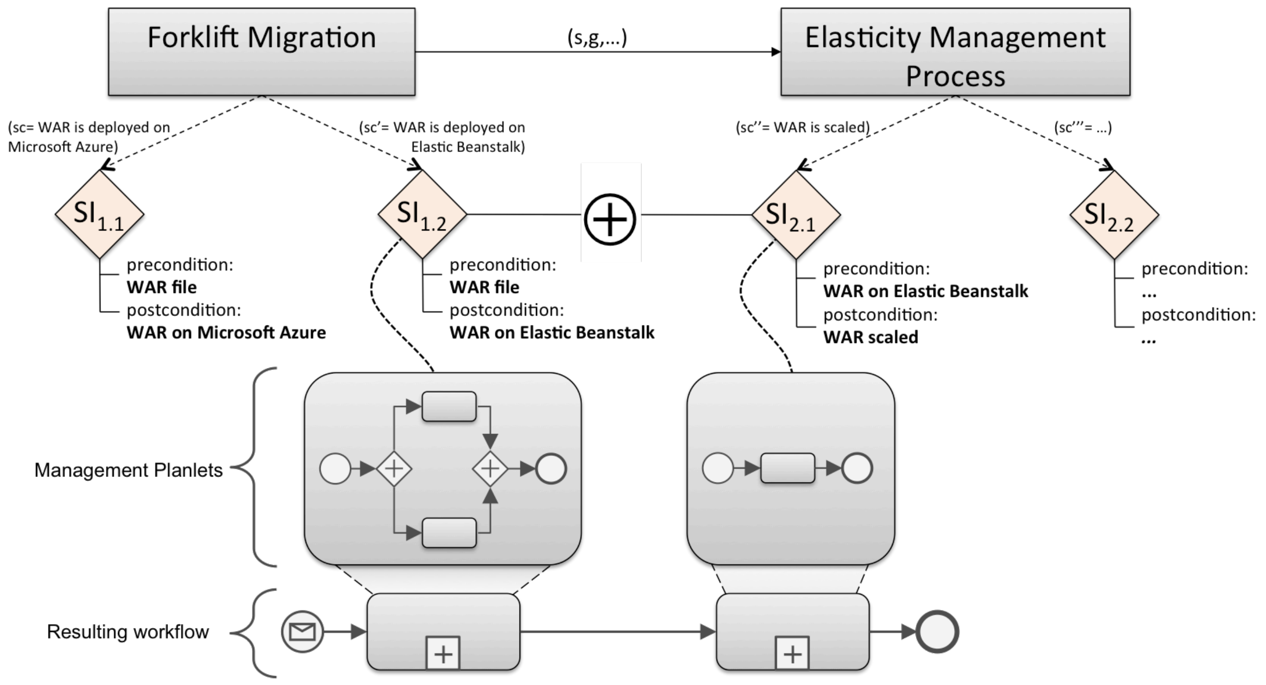

Forklift Migration
(s,g,...)
Elasticity Management Process
(sc= WAR is deployed on Microsoft Azure)
(sc'= WAR is deployed on Elastic Beanstalk)
(sc''= WAR is scaled)
(sc'''= ...)
SI1.1
SI1.2
SI2.1
SI2.2
precondition: WAR file postcondition: WAR on Microsoft Azure
precondition: WAR file postcondition: WAR on Elastic Beanstalk
precondition: WAR on Elastic Beanstalk postcondition: WAR scaled
precondition: ... postcondition: ...
Management Planlets
Resulting workflow

Figure 4. Management Planlets are Solution Implementations in the domain of cloud management linked to patterns and aggregated by an Aggregation Operator.

by reusing and aggregating predefined Solution Implementations in the form of executable management workflows.

In the domain of cloud application management, applying the concept of patterns is quite difficult as the refinement of a pattern's abstract solution to an executable management workflow for a certain use case is a complex challenge: (i) mapping abstract conceptual solutions to concrete technologies, (ii) handling the technical complexity of integrating different heterogeneous management APIs of different providers and technologies, (iii) ensuring non-functional cloud properties, (iv) and the mainly remote execution of management tasks lead to immense technical complexity and effort when refining a pattern in this domain. The presented approach of Solution Implementations enables to provide completely refined solutions in the form of executable *management workflows* that already consider all these aspects. Thus, if they are linked with the corresponding pattern, they can be selected and executed directly without further adaptations. This reduces the (i) required management knowledge and (ii) manual effort to apply a management pattern significantly. To apply the concept of Solution Implementations to this domain, two issues must be considered: (i) selection and (ii) aggregation of Solution Implementations in the form of management workflows.

To tackle these issues, we employ the concept of *management planlets*, which was introduced in our former research on cloud application management automation [24]. Management planlets are *generic management building blocks* in the form of workflows that implement management tasks such as installing a web server, updating an operating system, or creating a database backup. Each planlet exposes its functionality through a formal specification of its *effects* on components, i.e., its *postconditions*, and defines optional *preconditions* that must be fulfilled to execute the planlet. Therefore, each specific precondition of a planlet must be fulfilled by postconditions of other planlets. Thus, planlets can be combined to implement a more sophisticated management task, such as scaling an application. If two or more planlets are combined, the result is a *composite management planlet (CMP)*, which can be recursively combined with other planlets again: the CMP inherits all postconditions of the orchestrated planlets and exposes all their preconditions, which are not fulfilled already by the other employed planlets. Thus, management planlets provide a recursive aggregation model to implement management workflows. Based on these characteristics, management planlets are ideally suited to implement management patterns in the form of concrete Solution Implementations. We create Solution Implementations, which implement a pattern's refinement for a certain use case by orchestrating several management planlets to an overall composite management planlet that implements the required functionality in a modular fashion as depicted in Figure 4.

As stated above, selection and aggregation of Solution Implementations must be considered, the latter if multiple patterns are applied together. For example, Figure 4 shows two management patterns: (i) *forklift migration* [29] – application functionality is migrated with allowing some downtime and (ii) *elasticity management process* [10] – application functionality is scaled based on experienced workload. Both patterns are linked to two Solution

Implementations each in the form of composite management planlets. The forklift migration pattern provides two Solution Implementations: one migrates a Java-based web application (packaged as WAR file) to Microsoft Azure [22], another to Amazon Elastic Beanstalk [21]. Thus, if the user selects this pattern and chooses the Selection Criteria defining that a WAR application shall be migrated to Elastic Beanstalk, $SI_{1.2}$ is selected. Whether this Solution Implementation is applicable at all depends on the context: if the application to be migrated is a WAR application, then the Solution Implementation is appropriate. Equally to this pattern, the elasticity management process pattern shown in Figure 4 provides two Solution Implementations: one provides executable workflow logic for scaling a WAR application on Elastic Beanstalk ($SI_{2.1}$). Thus, if these two patterns are applied together, the selection of $SI_{1.2}$ restricts the possible Solution Implementations of the second pattern, as only $SI_{2.1}$ is applicable (its preconditions match the postconditions of $SI_{1.2}$). As a result, the selection of appropriate Solution Implementations can be reduced to the problem of (i) matching Selection Criteria to postconditions of Solution Implementations and (ii) matching preconditions and postconditions of different Solution Implementations to be combined.

After Solution Implementations of different patterns have been selected, the second issue of aggregation has to be tackled to combine multiple Solution Implementations in the form of workflows into an overall management workflow that incorporates all functionalities. Therefore, we implement a single Aggregation Operator for this pattern language as described in the following: to combine multiple Solution Implementations, the operator integrates the corresponding workflows as subworkflows [27]. The control flow, which defines the order of the Solution Implementations, i.e., the subworkflows, is determined based on the patterns' solution path depicted in Figure 2. So in general, if a pattern is applied before another pattern, also their corresponding Solution Implementations are applied in this order.

VI. SOLUTION IMPLEMENTATIONS PROTOTYPE

To prove the approach's technical feasibility, we implemented a prototype. That consists of two integrated components: (i) a pattern repository and (ii) a workflow generator. The pattern repository aims to capture patterns and their cross-references in a domain-independent way to support working with patterns. Based on semantic wiki-technology [11] it enables capturing, management and search of patterns. To adapt to different pattern domains, the pattern format is freely configurable. The pattern repository already contains various patterns from different domains like cloud computing patterns, data patterns and costume patterns to demonstrate the genericity of our approach. The cross-references between the patterns enable an easy navigation through the pattern languages. Links like "apply after" or "combined with" supports to connect the patterns to result in a pattern language. The pattern repository does not only contain the patterns and their cross-references but can be connected to a second repository containing the solution implementations of these patterns. Also, based on semantic wiki-technology we implemented a Solution Implementation repository for the domain of costume patterns [14]. Here, for example, the concrete costumes of a sheriff occurring in a film can be understood as the Solution Implementation of a sheriff costume pattern. By connecting the pattern to a Solution Implementation as a concrete solution of the abstracted solution of the pattern the application of the pattern in a certain context is facilitated.

The combination of several concrete Solution Implementations has been prototyped for the domain of cloud management patterns. A workflow generator has been built that is used to combine different management planlets to an overall workflow implementing a solution to a problem that requires the use of multiple patterns. The input for this generator is a partial order of (composite) management planlets, i.e., Solution Implementations that have to be orchestrated into an executable workflow. This partial order is determined by the relations of combined patterns: if one pattern is applied after another pattern, also their Solution Implementations, i.e., management planlets, have to be executed in this order. The workflow generator creates BPEL-workflows while management planlets are also implemented using BPEL. As BPEL is a standardized workflow language, the resulting management plans are portable across different engines and cloud environments supporting BPEL as workflow language, which is in line with TOSCA [25][26].

VII. CONCLUSION AND FUTURE WORK

In this paper, we introduced the concept of Solution Implementations as concrete instances of a pattern's solution. We showed how patterns and pattern languages can be enriched by Solution Implementations and how this approach can be integrated into a pattern repository. To derive concrete solutions for problems that require the application of several patterns we proposed a mechanism to compose these solutions from concrete solutions of the required patterns by means of operators. We concretized the general concept of Solution Implementations in the domain of cloud management by introducing management planlets as examples for Solution Implementations. We verified the approach by means of a prototype of an integrated pattern repository and workflow generator.

Currently, we extend the implemented repository for solution knowledge in the domain of costume design to capture Solution Implementations more efficiently. This repository integrates patterns and linked Solution Implementations in this domain and we are going to enlarge the amount of costume Solution Implementations. We are also going to tackle the limitation of the presented approach to not only work on solution implementation sequences but also on aggregations of concrete solution instances not ordered temporally due to pattern sequences. Since Solution Implementations are composed by Aggregation Operators we are going to enhance our pattern repositories to also store and manage the Aggregation Operators. Finally, we will investigate Aggregation Operators in domains, besides the above mentioned to formulate a general theory of Solution Implementations and Aggregation Operators.

References

[1] C. Alexander, "The timeless way of building," Oxford University Press, 1979.

[2] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, "A pattern language: towns, buildings, constructions," Oxford University Press, 1977.

[3] R. Reiners, Bridge Pattern Library, http://bridge-pattern-library.fit.fraunhofer.de/pattern-library/, last accessed on 2014.01.30.

[4] C. Fehling, F. Leymann, R. Mietzner, and W. Schupeck, "A collection of patterns for cloud types, cloud service models, and cloud-based application architectures," http://www.cloudcomputingpatterns.org, last accessed on 2014.01.30, University of Stuttgart, Report 2011/05, Mai 2011.

[5] M. Falkenthal, D. Jugel, A. Zimmermann, R. Reiners, W. Reimann, and M. Pretz, "Maturity assessments of service-oriented enterprise architectures with iterative pattern refinement," Lecture Notes in Informatics - Informatik 2012, September 2012, pp. 1095–1101.

[6] R. Reiners, "A pattern evolution process – from ideas to patterns," Lecture Notes in Informatics – Informatiktage 2012, March 2012, pp. 115–118.

[7] U. van Heesch, Open Pattern Repository, http://www.patternrepository.com, last accessed on 2014.01.30.

[8] M. Demirköprü, "A new cloud data pattern language to support the migration of the data layer to the cloud," in German "Eine neue Cloud-Data-Pattern-Sprache zur Unterstützung der Migration der Datenschicht in die Cloud," University of Stuttgart, diploma thesis no. 3474, 2013.

[9] U. Zdun, "Systematic pattern selection using pattern language grammars and design space analysis," Software: Practice and Experience, vol. 37, 2007, pp. 983–1016.

[10] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, "Cloud computing patterns," Springer, 2014.

[11] N. Fürst, "Semantic wiki for capturing design patterns," in German "Semantisches Wiki zur Erfassung von Design-Patterns," University of Stuttgart, diploma thesis no. 3527, 2013.

[12] G. Hohpe and B. Wolf, "Enterprise integration patterns: designing, building, and deploying," Addison-Wesley, 2004.

[13] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, "Pattern-oriented software architecture volume 1: a system of patterns," Wiley, 1996.

[14] D. Kaupp, "Application of semantic wikis for solution documentation and pattern identification," in German "Verwendung von semantischen Wikis zur Lösungsdokumentation und Musteridentifikation," University of Stuttgart, diploma thesis no. 3406, 2013.

[15] R. Porter, J. O. Coplien, and T. Winn, "Sequences as a basis for pattern language composition," in Science of Computer Programming, Special issue on new software composition concepts, vol. 56, April 2005, pp. 231–249.

[16] F. Salustri, "Using pattern languages in design engineering," Proceedings of the International Conference on Engineering Design, August 2005, pp. 248–362.

[17] D. Schumm, J. Barzen, F. Leymann, and L. Ellrich, "A pattern language for costumes in films," Proceedings of the 17th European Conference on Pattern Languages of Programs (EuroPLoP), July 2012, pp. C4-1–C4-30.

[18] T. Iba, T. Miyake, "Learning patterns: a pattern language for creative learners II," Proceedings of the 1st Asian Conference on Pattern Languages of Programs (AsianPLoP 2010), March 2010, pp. I-41 – I-58.

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: elements of reusable object-oriented software," Addison-Wesley, 1995.

[20] M. Fowler, "Patterns of enterprise application architecture," Addison-Wesley, 2003.

[21] Amazon, Elastic Beanstalk, http://www.amazon.com/elasticbeanstalk, last accessed on 2014.01.30.

[22] Microsoft, Microsoft Azure, http://www.windowsazure.com, last accessed on 2014.01.30.

[23] C. Fehling, F. Leymann, R. Retter, D. Schumm, and W. Schupeck, "An architectural pattern language of cloud-based applications", Proceedings of the 18th Conference on Pattern Languages of Programs (PLoP), October 2011, pp. A-20–A-30.

[24] U. Breitenbücher, T. Binz, O. Kopp, and F. Leymann, "Pattern-based runtime management of composite cloud applications", Proceedings of the 3rd International Conference on Cloud Computing and Service Science (CLOSER), May 2013, pp. 475–482.

[25] OASIS, Topology and Orchestration Specification for Cloud Applications Version 1.0, http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html, last accessed on 2014.01.30.

[26] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "TOSCA: portable automated deployment and management of cloud applications," in Advanced Webservices, A. Bouguettaya, Q. Z. Sheng, F. Daniel, Eds., Springer, 2014, pp. 527–549.

[27] O. Kopp, H. Eberle, and F. Leymann, "The subprocess spectrum", Proceedings of the 3rd Business Process and Services Computing Conference (BPSC), September 2010, pp. 267–279.

[28] Amazon, AWS Cloud Formation, http://aws.amazon.com/cloudformation/, last accessed on 2014.01.30.

[29] C. Fehling, F. Leymann, S. T. Ruehl, M. Rudek, and S. Verclas "Service migration patterns – decision support and best practices for the migration of existing service-based applications to cloud environments," Proceedings of the IEEE International Conference on Service Oriented Computing and Applications (SOCA), December 2013, in press.

[30] Amazon, Amazon Web Services, http://aws.amazon.com, last accessed on 2014.01.30.

[31] C. Fehling, F. Leymann, J. Rütschlin, D. Schumm, "Pattern-based development and management of cloud applications," Future Internet, vol. 4, 2012, pp. 110–141.

[32] Amazon, AWS EC2, http://aws.amazon.com/de/ec2/, last accessed on 2014.04.10.

[33] C. Fehling, F. Leymann, R. Retter, D. Schumm, W. Schupeck, "An architectural pattern language of cloud-based applications," Proceesings of the 18th Conference on Pattern Languages of Programs (PLoP), Oct. 2011, pp. A-20 – A-21

Reconfiguration Patterns for Goal-Oriented Monitoring Adaptation

Antoine Toueir, Julien Broisin, Michelle Sibilla
IRIT, Université Paul Sabatier
Toulouse, France
Email: {toueir,broisin,sibilla}@irit.fr

***Abstract*—This paper argues that autonomic systems need to make their distributed monitoring adaptive in order to improve their "comprehensive" resulting quality; that means both the Quality of Service (QoS), and the Quality of Information (QoI). In a previous work, we proposed a methodology to design monitoring adaptation based on high level objectives related to the management of quality requirements. One of the advantages of adopting a methodological approach is that monitoring reconfiguration will be conducted through a consistent adaptation logic. However, eliciting the appropriate quality goals remains an area to be investigated. In this paper, we tackle this issue by proposing some monitoring adaptation patterns falling into reconfiguration *dimensions*. Those patterns aim at facilitating the adaptation design of monitoring behavior of the whole set of distributed monitoring modules part of autonomic systems. The utility of those patterns is illustrated through a case-study dealing with monitoring adaptation based on high level quality objectives.**

***Keywords*–Quality requirements; adaptive monitoring; autonomic systems; goal-oriented adaptation.**

I. Introduction

Autonomic systems that are implemented by virtue of their four characteristics self-configuration, self-optimization, self-healing and self-protection, are serving the ultimate objective of making them self-managed to achieve high level objectives [1]. These objectives are strongly related to the quality level provided by autonomic systems. When large and complex systems are targeted, the self-management characteristic (self-*) is a key issue to deal with. Basically, self-management is thought as the auto-adaptation capability that brings the system to reach an absolute or preferable state. Concretely, the four self-* characteristics are realized by implementing the *Monitoring, Analyzing, Planning, Executing - Knowledge* (MAPE-K) loop modules. This implementation is either embedded within a resource, or distributed over several resources.

In MAPE-K loop, the monitoring module plays a crucial role, since wrong decisions might be taken by the analyzing & planning modules. Therefore, autonomic systems need to ensure the quality of information (*e.g.,* correctness, freshness, timeliness, accuracy, etc.) exposed by the distributed monitoring modules. Moreover, within autonomic systems, monitoring is usually QoS-oriented. Thus, the services implemented by the functional system must respect the required QoS level that is determined through the *Service Level Agreements* (SLAs) that have been agreed with clients. Since the management system (managing the functional system) could provide the possibility to renegotiate or modify the QoS specification afterward, the monitoring system has to adapt its behavior according to these new requirements and constraints.

To summarize, the monitoring of autonomic systems has to be capable of configuring the underlying mechanisms carrying the monitoring functions (*e.g.,* measuring, gathering, calculating, evaluating, filtering, etc.) starting from QoS specification, as well as reconfiguring those mechanisms based on quality requirements.

Most of the time, reconfiguration is held through ad-hoc logic (proposing solutions for particular scenarios dealing with specific issues). But, this approach is not suitable for reuse in other scenarios, and it also does not satisfy high level objectives extended over the whole scale of the autonomic system. To overcome these issues, we adopted a Requirements Engineering methodology to design monitoring adaptation; it starts from high level goals, and ends up with the configuration of monitoring mechanisms [2].

Right now, the key question is: how to identify goals representing the "starting point" for deriving monitoring (re)configuration? In other words, reconfiguration questions such as: why to delay launching some monitoring mechanisms? Why to substitute remote agents? How to aggregate alarms? What determines the monitoring of this set of metrics and not another one? Why to exchange metrics among distributed management entities? This paper deals with these questions by proposing monitoring adaptation patterns that assist human administrators in designing meaningful adaptations and thus increase the overall quality of the autonomic systems.

The work presented here relies on both a 3-layered adaptive monitoring framework [3][4][5] and our goal-oriented adaptation methodology [2]. We pursue this research by focusing on adaptation patterns dedicated to the identification of high level goals, together with their refinement. The paper is organized as follows: the next section gives an overview of the studied monitoring framework; the monitoring adaptation patterns are discussed in Section III; and then applied to a case-study in Section IV; before concluding, Section V enumerates other monitoring adaptation approaches and points out their weaknesses.

II. The Studied Adaptive Monitoring Framework

Our approach is based on a 3-layered framework [3][4][5] illustrated in Figure 1, and defines three fundamental capabilities required to control monitoring: being **configurable**, **adaptable** and **governable**.

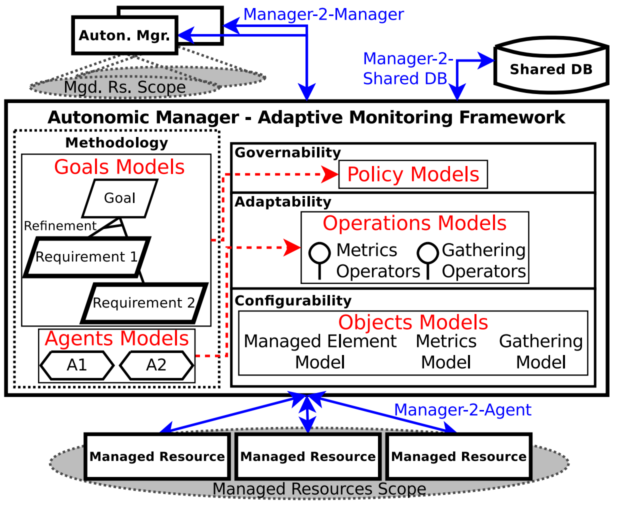


Figure 1: Adaptation Methodology & Monitoring Framework

The **configurability layer** relies on the Distributed Management Task Force (DMTF) Common Information Model (CIM) standard. In addition to the managed resources, this low level layer aims at representing the metrics [6] and the gathering mechanisms [3] that are required to monitor both the QoS provided by the functional system and the QoI of the monitoring system itself; this layer deals with both mechanisms. The **adaptability layer** provides an interface encapsulating operations to be applied on the lower layer models. Thus, the behavior of the underlying monitoring mechanisms will be reconfigured during runtime by invoking these operations. Finally, the **governability layer** is the top level layer representing the "intelligence" of the monitoring adaptation. To express the quality requirements, it uses Event/Condition/Action (ECA) policies to describe *when* and *how* adaptation should take place, that is, in which contexts those operations of the adaptability layer should be invoked.

We exploit the Requirements Engineering (RE) to propose monitoring adaptation methodology, and to build configurability and adaptability models [2]. RE iterates activities of "*eliciting, evaluating, documenting, consolidating and changing the objectives, functionalities, assumptions qualities and constraints that the system-to-be should meet based on the opportunities and capabilities provided by new technologies*" [7]. Keep All Objectives Satisfied (KAOS) is adopted as RE goal-oriented method, due to its formal assertion layer that proves correctness and completeness of goals [8]. In KAOS, the system-to-be is divided into various models. *Goals Model* determines the objectives to be realized through that system (*e.g.,* minimizing monitoring cost). *Agents Model* comprises the agents (*e.g.,* human automated component) responsible for realizing the refined elicited goals. Notice that the term *Agents* in networks and systems management represents entities responding to management requests coming from other management entities called *Managers*; therefore, the term *Agent* in RE has a different meaning. *Operations Model* deals with the internal operations to be carried by agents (*e.g.,* updating polling period). *Object Model* identifies the system-to-be objects (*e.g.,* entities, agent, relationships).

Therefore, based on KAOS, our methodology identifies the high level quality objectives the monitoring framework carries on. By iterating a refinement process, we finally identify what it is called *leaf goals* or *requirements* (see Figure 1). Once the leaf goals are determined, both policies (to be inserted into the governability layer) and agents (invoking operations of the adaptability layer) will be recognized. Thus, monitoring system adaptation is automatically handled. However, human administrators have to manually identify the leaf goals according to the high level objectives they want to reach. To facilitate this task, we conducted an investigation about the monitoring aspects that could be subject to adaptation. As a result, we have identified various leaf goals belonging to four *dimensions* (*i.e.,* Spatial, Metric, Temporal, Exchange) [2]. Hereafter, in the next section, we propose monitoring adaptation patterns falling into those dimensions.

TABLE I: Patterns Refining Achieve Goals ($P \Rightarrow \Diamond Q$)

Pattern	Subgoal 1	Subgoal 2	Subgoal 3
Milestone	$P \Rightarrow \Diamond R$	$R \Rightarrow \Diamond Q$	
Case	$P \wedge P1 \Rightarrow \Diamond Q1$	$P \wedge P2 \Rightarrow \Diamond Q2$	$\Box(P1 \vee P2)$ $Q1 \vee Q2 \Rightarrow Q$
Guard	$P \wedge \neg R \Rightarrow \Diamond R$	$P \wedge R \Rightarrow \Diamond Q$	$P \Rightarrow PWQ$

III. Monitoring Adaptation Patterns

With regard to the refinement process, besides the basic AND/OR-decompositions, we rely on some predetermined correct and complete refinement patterns proved mathematically [9]. Those patterns refine *Achieve* goals of the form $P \Rightarrow \Diamond Q$ (see Table I), and are written in *Linear Temporal Logic* (LTL) classical operators where $\Diamond$, $\Box$ and $\mathcal{W}$ mean *some time in the future*, *always in the future*, and *always in the future unless*, respectively. Starting from a given goal (P), *milestone pattern* identifies one (various) intermediate goal(s) (R, [...]) that must be reached orderly before reaching the ultimate one (Q). Rather, *case pattern* identifies the set of different and complete cases ($P1, P2$) for reaching final goals ($Q1, Q2$) that OR-decompose the ultimate goal (Q). Finally, the *guard pattern* requires the recognition of a specific condition (R) before achieving the ultimate goal (Q).

In order to clarify the exploitation contexts, pattern goals and requirements, as well as some application situations, our pattern structure encompasses: context, pattern refinement, and examples. Notice that we are focusing on adaptation actions taken at the autonomic manager side only. Thus, investigating adaptations at the agent side is out of scope. In addition, the patterns are refined using KAOS graphical language [7].

A. Exchange Dimension Pattern

Context. Relying on IBM blueprint reference architecture [10], autonomic systems could distribute self-management (MAPE) loops over multiple collaborating autonomic managers. Each of them is responsible for managing a particular scope of managed resources. Patterns belonging to this dimension are useful to overcome metrics gathering/delivering problems. Those problems could manifest either on metrics values, reliability of communication between information sources & destinations, or even on their trustworthiness.

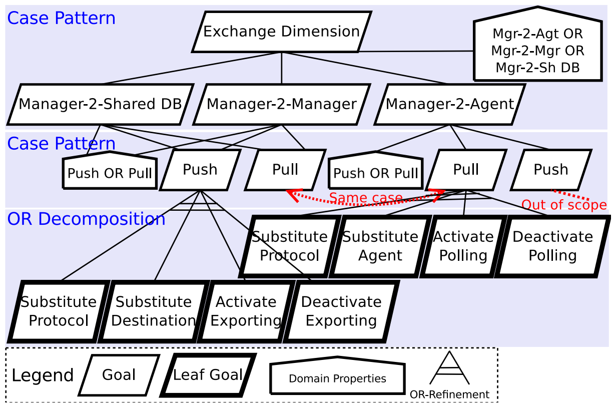


Figure 2: Exchange Dimension Pattern

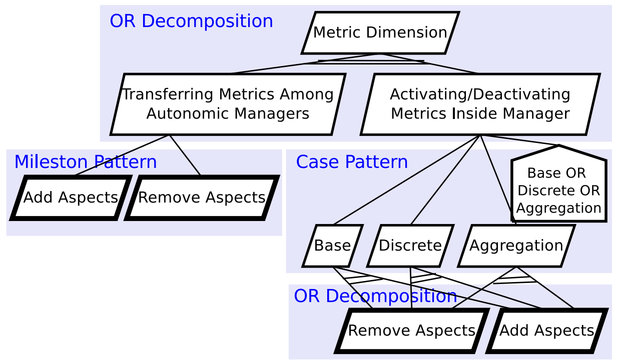


Figure 3: Metric Dimension Pattern

Pattern Refinement. Communications inside autonomic system could be classified according to the entities involved in information exchange (*i.e.,* managers, agents, shared databases). Therefore, we identify three communication classes: Manager-2-Agent, Manager-2-Manager, and Manager-2-Shared Database (see Figure 1). Besides identifying communication classes, we need to deal with *pull & push* communication modes. In *pull*, the entity needing information solicits the one possessing it, which responds with the queried information; where in *push*, the entity possessing the information reports it to other entities. By taking into consideration push and pull modes, along with previous communications classes, we use *case pattern* for the first two refinement levels to cover all possible cases (see Figure 2).

Based on the triplet ⟨ Information Source, Communication Protocol, Information Destination ⟩, the Manager-2-Agent pull mode will be OR-decomposed into *Substitute Agent* and *Substitute Protocol* leaf goals. Rather, *Substitute Protocol* and *Substitute Destination* OR-decompose both Manager-2-(Manager/Shared DB) push mode. Besides, *Activate/Deactivate Polling & Exporting* leaf goals are elicited to launch and stop polling & exporting.

Notice that in both Manager-2-(Manager/Shared DB) pull mode communications, the manager responding to requests is considered as agent (because it is the information source); therefore, this case becomes identical to Manager-2-Agent pull mode. Moreover, adaptation actions related to Manager-2-Agent push mode are not treated because they need to be held at the agent side.

Examples. This pattern is suitable for the following cases: (1) Increasing accuracy or precision of pulled/pushed metrics values, by replacing information source. (2) Querying more available agents, or blocking fake agents trying to integrate the distributed management system. (3) Securing the communication between information sources and destinations. (4) Modifying information destination when changing the topology of collaborating autonomic managers.

B. Metric Dimension Pattern

Context. The main idea behind building autonomic systems is to delegate decisions, that human administrators are used to make, to autonomic systems themselves. Thus, to be able to make ”wise” decisions, the monitoring system needs to instrument specific metrics that could be activated/deactivated according to the management needs during runtime. Patterns belonging to this dimension are useful to control the trade-off between constructing more knowledge and monitoring the information that is necessary for management.

Pattern Refinement. Metric instrumentation must be thought at the whole management system level. In other words, a given autonomic manager could activate/deactivate instrumentation of particular metrics, but when deactivating metrics on that manager, it doesn’t mean necessarily that those metrics are ”abandoned”, because they could be transferred to other collaborating autonomic manager on which they are activated. These two cases are OR-decomposing the first refinement level (see Figure 3).

Regarding metrics manipulation inside an autonomic manager, the second refinement level uses *case pattern* to cover metric classes. Our research exploits both CIM Metric Model classifying metrics into *Base, Discrete & Aggregation*, as well as our mathematical extension [6] classifying base metrics into *Resource, Measurable & Mathematical*. Each of these classes is OR-decomposed using *Add Aspects* and *Remove Aspects* leaf goals. On the other hand, the transfer of metrics among autonomic managers could be refined through *milestone pattern*, when metrics are activated on the collaborating manager (*Add Aspects* in Figure 3, as Subgoal 1 in Table I) first, and then removed from the delegating one (*Remove Aspects*, as Subgoal 2).

It is worth noting that previously mentioned aspects are representing ”metric definitions”, rather than ”metric values”. The former encompasses attributes related to the nature of metric (*e.g.,* data type, unit), where the latter attributes describe the instrumented values and their relevant contexts. For further information, the reader is referred to the DMTF Base Metric Profile [11].

Examples. This pattern can be applied in the following cases: (1) Performing troubleshooting, or applying root cause analysis algorithms, because they require the instrumentation of additional metrics. (2) Modifying the hierarchical topology of the management system by instrumenting aggregated metrics to be exported to other managers or shared DBs. (3) ”Engineering” the distribution of monitored metrics among autonomic managers.

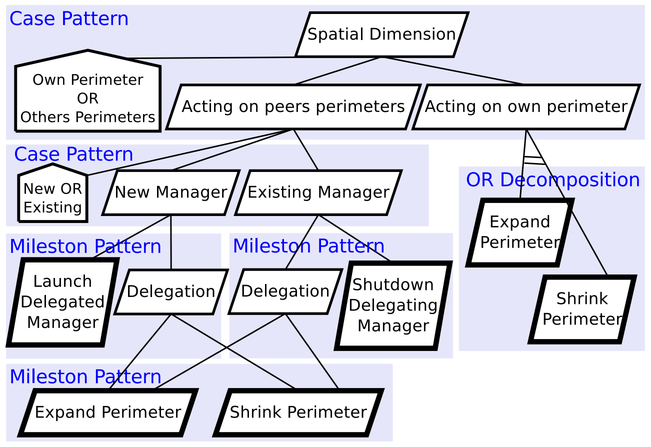


Figure 4: Spatial Dimension Pattern

C. Spatial Dimension Pattern

Context. As mentioned earlier, in an autonomic system, each manager is responsible for managing a set of managed resources. In many cases, the number of users consuming the autonomic system services may oscillate rapidly, or even become quite important in term of size. Thus, managed resources are subject to be joined/withdrawn during the runtime. Patterns belonging to this dimension are useful to react in regard with important changes concerning the scope of managed resources.

Pattern Refinement. Management of autonomic systems is orchestrated by the collaboration of multiple autonomic managers, each of which can act on its own perimeter, as well as the perimeters of its collaborating peers. Thus, the first refinement level uses *case pattern* to cover these two cases (see Figure 4).

In fact, acting on its own perimeter is OR-decomposed using *Expand* and *Shrink Monitoring Perimeter* leaf goals. Rather, acting on others perimeters is refined using *case pattern* into deploying a new manager, or soliciting an existing one. First, the case of deploying a new manager is refined using *milestone pattern* into launching manager (*Launch Delegated Manager* in Figure 4, as Subgoal 1 in Table I), and then, delegating perimeter (*Delegation*, as Subgoal 2). In turn, the delegation goal is also refined though *milestone pattern* into joining delegated perimeter on the delegated manager (*Expand Perimeter*, as Subgoal 1), and then, deleting this perimeter from the delegating manager (*Shrink Perimeter*, as Subgoal 2).

In the second case, where acting is held on an existing manager, the refinement is done twice, first using *milestone pattern*, into delegating the whole perimeter to the delegated manager (*Delegation*, as Subgoal 1), and then shutting down the delegating one (*Shutdown Delegating Manager*, as Subgoal 2).

Examples. This pattern is suitable for the following cases: (1) Load balancing of monitoring among autonomic managers. (2) Supporting scalability of the autonomic systems. (3) Minimizing the overall monitoring charge in terms of dedicated monitoring entities.

D. Temporal Dimension Pattern

Context. Temporal aspects are decisive factors in adapting monitoring behavior. Notice that previous patterns are explained without time considerations, but in fact, they imply some temporal aspects. Patterns belonging to this dimension are useful either to overcome both temporal violations and scheduling problems, or to tune the analysis over the instrumented metrics.

Pattern Refinement. Regarding information exchange, once again, we use *case pattern* to represent the same cases identified in *Exchange* dimension. Obviously, dealing with information exchange temporal aspects means that the exchange is done iteratively and not once. Thus, Manager-2-Agent case is OR-decomposed into periodic poll, and both Manager-2-(Manager/Shared DB) cases are OR-decomposed into periodic export (see Figure 5). Note that Manager-2-Agent push mode and Manager-2-(Manager/Shared DB) pull mode are not mentioned for the reasons explained in Section III-A.

We distinguish two levels of temporal granularity: the fine-grained level deals with an individual polling (exporting), whereas the coarse-grained level addresses a collective polling (exporting). Based on this distinction, we identify six leaf goals OR-decomposing periodic poll (export), namely: *Update Polling (Exporting) Period* to update the frequency of a given polling (exporting), *Align Polling (Exporting)* to launch a set of synchronized parallel pollings (exported metrics) at the same time, and *Misalign Polling (Exporting)* to launch pollings (exported metrics) according to a given/adjustable offset.

Regarding metrics calculation, we identify the case of modifying the temporal interval covered by the metric value. However, the validity of a metric value that is not instantaneous (*e.g.,* throughput) is equal to the temporal interval through which that value was measured. Therefore, *case pattern* is used twice to cover all possible metric classes previously mentioned. We refine only the measurable, mathematical & aggregation metrics, because time has a sense in their calculation, but not the other metrics. Thus, at the fourth refinement level, we OR-decompose measurable & mathematical metrics using *Update Time Scope Interval*. Rather, *Update Time Series Interval* OR-decomposes aggregation metrics.

Examples. This pattern is suitable for the following cases: (1) Controlling (*e.g.,* relaxing, stressing) the monitoring load on autonomic managers, network paths among autonomic managers and shared DBs, as well as remote agents. (2) Tuning temporal parameters of metrics analysis.

Notice that all previous patterns are subject to be updated and enriched, in order to integrate new monitoring adaptation actions. For instance, we can address temporal aspects of *alarms filtering* by delaying delivery of redundant alarms [12], as well as *alarms correlation* by modifying the stream interval time during which *Complex Event Processing* engines (*e.g.,* Esper & Drools) perform correlations. OR-decomposition into *Update Waiting Time* & *Update Window Time* could be used for these two cases respectively.

IV. Case-Study

Context. Our scenario takes place in a cloud data center hosting a large number of virtual machines (VMs), and pro-

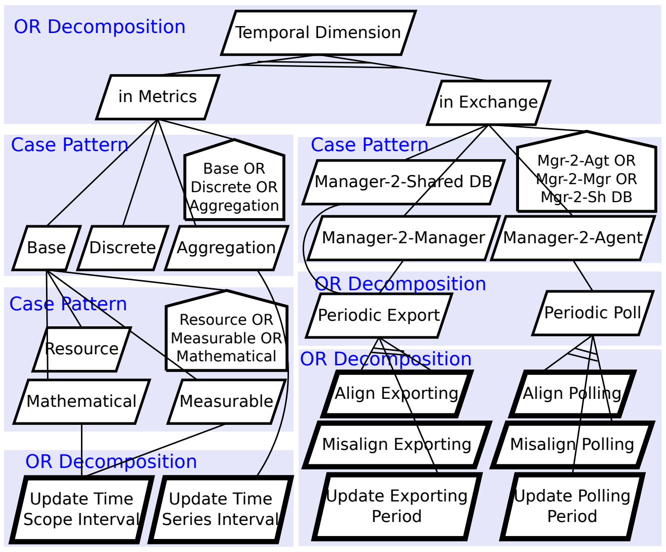


Figure 5: Temporal Dimension Pattern

viding its clients with a continuous monitoring of the enforced SLAs metrics. Each VM integrates an agent providing predetermined metrics reflecting VM healthiness. In most large scale systems, distributed agents periodically push their metrics; in our case, agents push those metrics every 10 seconds to specific pre-configured autonomic managers. To facilitate the case-study, we assume that our studied SLA template encapsulates the same metrics pushed by agents. Besides, this SLA template distinguishes two time-slots: metrics are to be refreshed at the client side with a freshness falling into the range of 3-6 seconds during the first time-slot, and a range of 30-40 seconds for the second one. The SLAs metrics values are instrumented and delivered automatically through polling and exporting, respectively. Once a new SLA is enforced, the autonomic managers use pull mode to collect VMs metrics with the lowest freshness value (3 seconds).

Objectives. Human administrators identify two high level goals to be satisfied during the monitoring system runtime: *Respect Metrics Freshness* makes sure that SLAs are monitored appropriately, and *Minimize Monitoring Cost* aims at limiting the resources dedicated to monitoring as much as possible.

Patterns. We can exploit several patterns to deal with the first objective. During the first time-slot, we use the *temporal pattern* to relax polling & exporting by updating their periods (*Update Polling & Exporting Period* in Figure 6) with respect to the highest freshness range (6 seconds). If delivering freshness violates the highest freshness, that would be a result of overloading manager [2], thus we apply the *spatial pattern* as a second alternative, and consequently, a new autonomic manager will be deployed to assist the overloaded one (*Launch Delegated Manager, Expand Perimeter & Shrink Perimeter*). As a third alternative, and in case that the overloaded autonomic manager monitors non-SLAs metrics (*e.g.,* physical servers healthiness), the *metric pattern* could be applied to transfer them to other manager, in order to relax the first one (*Add & Remove Aspects*). Since the second time-slot freshness (30-40 seconds) is greater than agents push period (10 seconds), there is no need to poll metrics, nor to export all received metrics. Rather, we apply the *temporal pattern* to update the exporting period from 3-4 to 30-40 seconds (*Update Exporting Period*). This adaptation necessitates applying another one belonging to *exchange pattern* to stop the pollings that are launched during the first time-slot (*Deactivate Polling*).

The second objective is refined using *spatial pattern* in order to shutdown recently deployed managers, during the first time-slot. Thus, an underloaded manager delegates its whole perimeter to another one, and shutdowns itself (*Expand Perimeter, Shrink Perimeter & Shutdown Delegating Manager*). During the second time-slot, autonomic managers already deliver to clients around one-third of the metrics pushed by agents, thus no adaption actions are to be taken in regard with minimizing monitoring resources.

Autonomic managers would be able to adapt their monitoring, if they recognize adaptation stimuli. Therefore, we exploit *guard pattern* to apply adaptation actions (*Adaptation* in Figure 6, as Subgoal 2 in Table I) as response to specific stimulus (*Guard*, as Subgoal 1), while maintaining the current monitoring behavior unless adaptation takes place (*Unless*, as Subgoal 3).

V. Related Work

In this section, we try to align our approach of adapting monitoring using goal-oriented dimensional patterns with other existing trends focusing on monitoring of QoS in autonomic systems [13][14][15][16][17][18].

In order to manage QoS in autonomic systems, the latter applies adaptation actions. In many cases, for instance [13], this adaptation doesn't concern the monitoring system itself, but precisely, is applied on the managed system services and infrastructure (*i.e.,* reconfiguring resource allocation). Certainly, this adaptation will result in increased quality, but this way, the knowledge of the management system won't exceed a "maximum ceiling" and management will be limited in terms of treating new situations.

Monitoring more metrics or managed resources is addressed in [14][16][18] either to deal with the managed scope changes, or to operate a "minimal" monitoring that is able to be extended in case of SLA violations, or even to adapt monitoring to meet SLA modifications. Indeed, it is important to scale up/down monitored metrics and resources. But it isn't clear whether this capability could be applied in other scenarios for other objectives, if so, how that could be feasible.

Runtime deployment of monitoring resources (*i.e.,* managers, probes) is discussed in [14][15][17] either to integrate monitoring into the SLA management life-cycle of large scale systems, or to replace failed managers, or even to monitor some metrics concerning particular paths or segments. But here also, besides the undeniable gains of deploying monitoring resources during runtime, we don't see how the system administrators can orchestrate the monitoring adaptation (*i.e.,* planning & executing) of the distributed monitoring among several collaborating managers.

Inspired from the autonomic computing reference architecture proposed in [10], patterns regarding the distribution of the MAPE loop modules were proposed in [19][20]. Those patterns are useful in terms of design reuse as well as clarifying

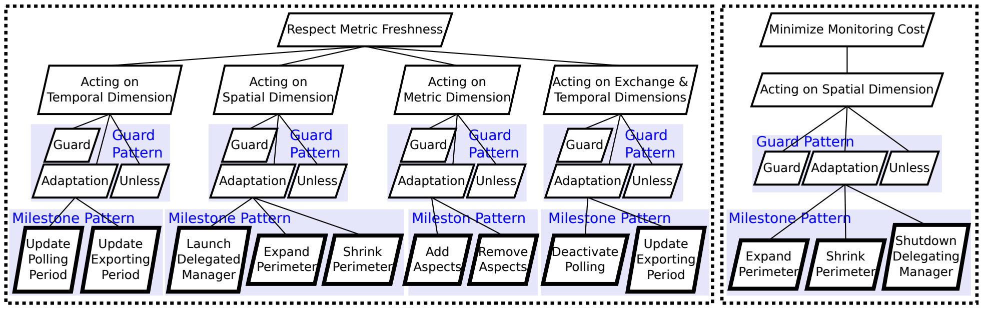


Figure 6: *Respect Metrics Freshness & Minimize Monitoring Cost* Refinement

the application contexts and benefits, but they target mainly the deployment of the monitoring modules rather than the monitoring behavior itself. In addition, they don't treat the monitoring adaptation in regard with quality requirements.

VI. Conclusion & Perspectives

We proposed a goal-oriented approach for designing self-managed monitoring in autonomic systems. This approach assists human administrators to adapt the monitoring system behavior regarding quality requirements. It means that monitoring is configured starting from quality specification (*e.g.,* SLA), and reconfigured based on adaptation patterns, that are exploited to achieve high level quality objectives. We designed four monitoring adaptation patterns according to dimensions that represent various aspects on which adaptation actions can apply to. Thus, each dimension represents a "starting point" reflection to elicit monitoring goals that are refined till reaching leaf goals.

About perspectives, we are currently investigating how monitoring adaptations could influence the stability at the autonomic system whole scale, in case of applying many overlapped adaptation leaf goals over several autonomic managers. In addition, the agent side adaptations need to be investigated, and orchestrated with those applied at the autonomic manager side.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," Computer, vol. 36, no. 1, Jan. 2003, pp. 41–50.

[2] A. Toueir, J. Broisin, and M. Sibilla, "A goal-oriented approach for adaptive sla monitoring: a cloud provider case study," in LATINCLOUD 2013, Maceió, Brazil, December 2013.

[3] A. Moui, T. Desprats, E. Lavinal, and M. Sibilla, "A cim-based framework to manage monitoring adaptability," in Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm), 2012, pp. 261–265.

[4] A. Moui, T. Desprats, E. Lavinal, and M. Sibilla, "Information models for managing monitoring adaptation enforcement," in International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE), Nice, 22/07/2012-27/07/2012, 2012, pp. 44–50.

[5] A. Moui, T. Desprats, E. Lavinal, and M. Sibilla, "Managing polling adaptability in a cim/wbem infrastructure," in 2010 4th International DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM), 2010, pp. 1–6.

[6] A. Toueir, J. Broisin, and M. Sibilla, "Toward configurable performance monitoring: Introduction to mathematical support for metric representation and instrumentation of the cim metric model," in 2011 5th International DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM), 2011, pp. 1–6.

[7] A. Van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, 2009.

[8] A. Van Lamsweerde, "Requirements engineering in the year 00: A research perspective," in Proceedings of the 22Nd International Conference on Software Engineering, ser. ICSE '00, 2000, pp. 5–19.

[9] R. Darimont and A. Van Lamsweerde, "Formal refinement patterns for goal-driven requirements elaboration," in ACM SIGSOFT Software Engineering Notes, vol. 21, no. 6. ACM, 1996, pp. 179–190.

[10] IBM Corp., "An architectural blueprint for autonomic computing," IBM White Paper, June 2005.

[11] A. Merkin, "Base metrics profile," December 2009, document Number: DSP1053.

[12] A. Clemm, Network Management Fundamentals. Cisco Press, 2006, ch. 5, pp. 138–141.

[13] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A self-adaptive hierarchical monitoring mechanism for clouds," Journal of Systems and Software, vol. 85, no. 5, 2012, pp. 1029–1041.

[14] D. Roxburgh, D. Spaven, and C. Gallen, "Monitoring as an sla-oriented consumable service for saas assurance: A prototype," in 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2011, pp. 925–939.

[15] P. Thongtra and F. Aagesen, "An adaptable capability monitoring system," in 2010 Sixth International Conference on Networking and Services (ICNS), 2010, pp. 73–80.

[16] M. Munawar, T. Reidemeister, M. Jiang, A. George, and P. Ward, "Adaptive monitoring with dynamic differential tracing-based diagnosis," in Managing Large-Scale Service Deployment, ser. Lecture Notes in Computer Science, F. Turck, W. Kellerer, and G. Kormentzas, Eds. Springer Berlin Heidelberg, 2008, vol. 5273, pp. 162–175.

[17] J. Nobre, L. Granville, A. Clemm, and A. Prieto, "Decentralized detection of sla violations using p2p technology," in Proceedings of the 8th International Conference on Network and Service Management, 2012, pp. 100–107.

[18] P. Grefen, K. Aberer, H. Ludwig, and Y. Hoffner, "Crossflow: Cross-organizational workflow management in dynamic virtual enterprises," International Journal of Computer Systems Science & Engineering, vol. 15, 2000, pp. 277–290.

[19] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. Göschka, "On patterns for decentralized control in self-adaptive systems," in Software Engineering for Self-Adaptive Systems II, R. Lemos, H. Giese, H. Müller, and M. Shaw, Eds. Springer Berlin Heidelberg, 2013, vol. 7475, pp. 76–107.

[20] A. J. Ramirez and B. H. C. Cheng, "Design patterns for developing dynamically adaptive systems," in Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, ser. SEAMS '10, 2010, pp. 49–58.

Dynamic Incremental Fuzzy C-Means Clustering

Bryant Aaron, Dan E. Tamir
Department of Computer Science,
Texas State University,
San Marcos, Texas, USA
{ba1127, dt19}@txstate.edu

Naphtali D. Rishe and Abraham Kandel
School of Computer Science
Florida International University
Miami, Florida, USA
rishen@fiu.edu, abekandel@yahoo.com

***Abstract*-Researchers have observed that multistage clustering can accelerate convergence and improve clustering quality. Two-stage and two-phase fuzzy C-means (FCM) algorithms have been reported. In this paper, we demonstrate that the FCM clustering algorithm can be improved by the use of static and dynamic single-pass incremental FCM procedures.**

Keywords-Clustering; Fuzzy C-Means Clustering; Incremental Clustering; Dynamic Clustering; Data-mining.

I. INTRODUCTION

The FCM algorithm provides a soft (fuzzy) assignment of patterns to clusters [1]. The assignment is represented by a partition matrix. The algorithm starts with either seeding the FCM with an initial partition matrix or through initial cluster centers and attempts to improve the partition matrix according to a given quality criterion.

Traditionally, in each of the FCM iterations, the algorithm is applied to the entire data set represented as a vector residing in the processor memory and representing a multi-dimensional set of measurements. Recently, however, the FCM algorithm, as well as numerous other clustering and data-mining algorithms, such as K-means, ISODATA, Kohonen neural networks (KNN), expectation maximization, and simulated annealing [1]-[6], have been exposed to a relatively new challenge referred to as "big-data." Often, the enormous amount of data available online cannot fit processors' physical memory. In fact, often the data does not even fit secondary memory. Given that input/output operations are generally the most taxing computer operations, working on the entire data set in every FCM iteration requires numerous consecutive reads of massive amounts of data. A scenario that might challenge the traditional approach to FCM clustering occurs when portions of the data are generated or become available dynamically and it is not practical to wait for the entire data set to be available.

This brings the need for incremental clustering into the forefront. Incremental clustering is also referred to as a single-pass clustering, whereas the traditional clustering is referred to as multi-pass clustering [7]-[11]. The idea is to cluster a manageable portion of the data (a data block) and maintain results for the next manageable block until exhausting the data. Under this approach, each block is processed by the algorithm a limited number of times, potentially only once. Ideally, a block, along with the preliminary number of initial centers selected should be as large as possible, occupying as much of the available internal processor memory. One might question the validity of "visiting" every data element for a limited number of iterations in a specific order as opposed to the traditional approach which considers every piece of data in each iteration.

A related approach is the multi-resolution or multistage clustering. Researchers observed that a multistage-based training procedure can accelerate the convergence and improve the quality of the training as well as the quality of the classification/decision phases of many of the clustering algorithms [5][12][13]. For example, our previous research reports show that the pyramid K-means clustering algorithm, the pyramid FCM, and multi-resolution KNN yield two-to-four times convergence speedup [13]. Both the multistage clustering and the incremental clustering apply an approach of sampling the data. In the multistage clustering, data is sampled with replacement, whereas in incremental clustering, due to the cost of replacement, the data is sampled without replacement. In both cases the validity of the sampling has to be addressed.

This paper describes a new approach for incremental FCM clustering. In difference from the pyramid FCM approach, the sampling is done without replacement. Furthermore, the sampling size is fixed. On the other hand, two measures are applied to the data in order to overcome the fact that each data block is processed only one time. First, the algorithm starts with a relatively large number of clusters and scales the number down in the last stage. This is referred to as a two-phase procedure. Hence, in the intermediate stages (first phase) each block might affect different cluster centers. A second and innovative version of the algorithm enables a dynamic number of clusters. Again, the algorithm starts with a relatively large number of clusters; however, each transaction on a block might change (increase or decrease) the number of clusters. In both cases the algorithms work on "chunks" of data referred to as blocks. This paper presents several experiments with multi-pass and static/dynamic single-pass versions of the FCM and empirically evaluates the validity of the static and dynamic incremental clustering approach.

The main contributions of this paper are: 1) a new approach, described in Section III.F, for incremental clustering, where the number of clusters is relatively high and is followed by clustering the resultant centers, is presented — this approach, increases the validity of incremental clustering, and 2) a second approach, described in Section III.G, where initially the number of clusters is relatively high and the number of clusters dynamically changes throughout execution, provides better incremental clustering quality/validity, and can be used to resolve the issue of identifying the right number of cluster centers.

A literature review performed shows numerous papers on incremental clustering. To the best of our knowledge there are no reports on research that applies the operations listed in this paper to the FCM algorithm.

The rest of the paper is organized in the following way. Section II reviews related research. Section III provides details of several single-pass and multi-pass variants of FCM clustering and lists metrics used to assess the quality of clustering. Section IV describes a set of experiments conducted to assess the performance and validity of the incremental clustering algorithms described in Section III, and Section V concludes with findings and proposes further research.

II. Review of Related Research

Clustering is a widely-used data classification method applied in numerous research fields, including image segmentation, vector quantization (VQ), data mining, and data compression [14]-[20]. K-means is one of the most commonly used clustering algorithms; a variant of the K-means algorithm — the Linde, Buzo, and Gray (LBG) VQ algorithm with unknown probability distribution of the sources — is utilized in many applications [1][16]. The LBG algorithm has been intensively researched. Some of these research results relevant to K-means and FCM are reviewed next.

Lloyd proposes an iterative optimization method for quantizer design, it assumes that the distribution of the data is unknown and attempts to identify the optimal quantizer [21]. This approach is equivalent to 1-means (that is, K-means with $k = 1$). While Lloyd's method yields optimal minimum mean square error (MMSE) for the design of one dimensional quantizer, its extension to multi-dimensional data quantizer (i.e., VQ) with unknown distributions is not guaranteed to yield optimal results [21]. Consequently, K-means with $k > 1$ is not guaranteed to reach a global optimum.

The LBG method for VQ with unknown underlying distribution generalizes Lloyd's iterative method and sets a VQ design procedure that is based on K-means [16]. The LBG VQ procedure is currently the most commonly used/researched VQ approach. Garey has shown that the LBG VQ converges in a finite number of iterations, yet it is NP-complete [22]. Thus, finding the global minimum solution or proving that a given solution is optimal is an intractable problem. Another problem with K-means is that the number of clusters (k) is fixed and has to be set in advance of executing the algorithm. ISODATA is a generalization of K-means which allows splitting, merging, and eliminating clusters dynamically [23][24]. This might lead to better clustering (better local optimum) and eliminate the need to set k in advance. ISODATA, however, is computationally expensive and is not guaranteed to converge [2].

Several clustering algorithms and combinatorial optimization techniques, such as genetic algorithms and simulated annealing, have been devised in order to enforce the clustering algorithm out of local minima [4][25][27]. These schemes, however, require long convergence time, especially for large clustering problems. FCM and fuzzy ISODATA generalize the crisp K-means and ISODATA. The FCM clustering algorithm is of special interest since it is more likely to converge to a global optimum than many other clustering algorithms, including K-means. This is due to the fact that the cluster assignment is “soft” [28][29]. On the other hand, the FCM attempt to “skip” local optima may bear the price of numerous soft iterations and can cause an increase in computation time. FCM is used in many applications of pattern recognition, clustering, classification, compression, and quantization including signal and image processing applications such as speech coding, speech recognition, edge detection, image segmentation, and color-map generation [28]-[35]. Thus, improving the convergence time of the FCM is of special importance.

Multistage processing is a well-known procedure used for reducing the computational time of several applications, specifically, image processing procedures. This method uses a sequence of reduced resolution versions of the data to execute an image processing task. Results of execution at a low resolution stage are used to initialize the next, higher resolution stage. For example, Coleman proposes an algorithm for image segmentation using K-means clustering [14]. Hsiao has applied Coleman's technique for texture segmentation [36]. He used a $\frac{1}{16}$-sample of the image to identify k. Huang and Zhu have applied the Coleman algorithm to DCT based segmentation and color separation respectively [37][38]. Like Hsiao, they used $\frac{1}{16}$-of the image-pixels to set up the parameters of the final clustering algorithm, where the final clustering is performed on the entire image. They found that the final cluster-centers obtained in the training-stage are very close to the final cluster centers obtained from clustering the entire image. This lends itself to a two-stage K-means procedure that uses one low resolution sample to initiate the parameters of the actual clustering. Pyramid processing is a generalization of the two-stage approach where the resolution of samples is growing exponentially and each execution stage doubles the number of samples.

Additional applications of multistage architectures are reported in the literature [27][31][39]. Rosenfeld surveys the area and proposes methods for producing the multistage snapshots of an image [40]. Kasif shows that multistage linking is a special case of ISODATA [41] and Tilton uses multistage for clustering remote sensing data [42]. Tamir introduces a pyramid multistage method to non-supervised training in the context of K-means and neural networks. He has shown that the pyramid approach significantly accelerates the convergence of these procedures [12][13].

Several papers deal with accelerating the convergence of FCM [39][45][46]. Altman has implemented a two-stage FCM algorithm [47]. The first stage operates on a random sample of the data and the second stage uses the cluster centers obtained in the first stage to cluster the entire set.

Cheng improves the method proposed by Altman and has investigated a two-phase approach [31]. The first phase implements a linear multistage algorithm which operates on small random slices of the data. Each slice contains $\Delta\%$ of the data. The algorithm finds the cluster centers of the first slice (say S_1), then uses these centers as initial centers for clustering a sample that contains the first slice and an additional slice (S_2) obtained through random sampling. After running the multistage phase for n stages, the final centers for the combination of slices $\{S_1, S_2, \ldots S_n\}$ which contain $n\Delta\%$ of the entire data are obtained. Next, in the second phase, these centers are used to cluster the entire data.

Other approaches for improving the convergence rate of clustering include data reduction techniques and data sampling using hypothesis testing [48][49].

A related research effort deals with clustering of very large data sets that are too big to fit into the available memory. One approach to this problem is to use incremental algorithms [10][11][27]. Several of these algorithms load a slice of the data, where the size of a slice is constrained by the available memory, and cluster this slice [7][9]. Results of clustering current slices (e.g., centers, partition matrices, dispersion, etc.) are used in the process of clustering upcoming slices. Hore has proposed a slice based single-pass FCM algorithm for large data sets [39]. The proposed method lumps data that has been clustered in previous slices into a set of weighted points and uses the weighted points along with fresh slices to commence with the clustering of the entire set in one path [39]. Another approach for clustering large data sets is to sample, rather than slice, the data [49].

Instead, in this report, the incremental approach we use has two phases. In the first phase, a very large set of clusters is used. Practically, we are trying to fit as many elements in a block and as many clusters per block as possible in the memory. In the second phase, after processing all the blocks, a process of clustering the centers obtained from the last block is applied.

It is interesting to note that K-means, FCM, Neural Networks (e.g., KNN), and many other iterative optimization algorithms have two main modes of operation, the batch mode and the parallel-update mode. For example, in the batch mode execution of FCM, each iteration considers every pattern individually and the centers are updated with respect to every pattern considered. The parallel-update mode, which is less computationally expensive and is the predominantly used mode in most current applications, assigns all the patterns to the relevant clusters and then updates the centers. In this context, the slice approach which is used for large data sets can be considered as a hybrid of batch and parallel update.

This brings the issue of parallel processing of clustering algorithms. Several ways to partition and distribute the clustering task have been considered [19][39][42][50]-[52]. One possible way is to assign a set of samples or a slice of data to each processor and eventually merge the cluster centers obtained from each processor into one set of centers. We plan to address this problem as a future research subject.

III. Fuzzy C-means Clustering Variants

In this section, we present several variants of FCM clustering.

A. The Classical Fuzzy C-Means Clustering Algorithm

The FCM algorithm is a generalization of the crisp K-means clustering. Actually, the generalization is quite intuitive. In the K-means algorithm, set membership is crisp. Hence, each pattern belongs to exactly one cluster. In the FCM, set membership is fuzzy and each pattern belongs to each cluster with some degree of membership. The following formalizes this notation.

Let $X = \{x_1, x_2, \ldots, x_m\}$, where $x_i \in R^n$, be a set of m, n-dimensional vectors representing the data to be clustered into c clusters $S = \{S_1, S_2, \ldots, S_c\}$ with cluster centers $\Omega = \{\omega_1, \omega_2, \ldots, \omega_c\}$. Under the FCM, each element x_j belongs to every cluster S_i with some degree of membership u_{ij}. Hence, the matrix $U = [u_{ij}]$, referred to as the partition matrix, represents the fuzzy cluster assignment of each vector x_j to each cluster S_i. The goal of FCM is to identify a partition matrix U, such that U optimizes a given objective function. A commonly used FCM objective function is defined to be:

$$J_q = \sum_{i}^{c} \sum_{j}^{m} {u_{ij}}^{q-1} \cdot \|x_j - \omega_i\| \qquad (1)$$

where $q > 1$ is the weighting exponent. In this research, q is set to the most commonly used and proposed value of 2 [28].

The most common measures for FCM clustering quality are: 1) the value of the objective function, 2) the partition coefficients, 3) the classification entropy, 4) measures of deviation of the partition matrix from a matrix obtained with uniformly distributed data, and 5) measures of induced fuzziness [24][28][44]. It should be noted that some of the quality criteria are derived from distortion measures. Hence, in this case, the goal is to minimize distortion, and high quality means low distortion. In other words, the quality can be considered as the inverse of distortion. Measures 1 through 5 assume that the end result of the clustering is soft. Nevertheless, in many cases, it is desirable to obtain “hard clustering” assignment to be used for VQ, image segmentation, or other classification applications. In these cases, two additional quality criteria can be considered: 6) the rate distortion function, and 7) the dispersion matrix [2][44]. Of all these measures, 1, 6, and 7 are most commonly used. Specifically, for metric 1, the functional J_q can be interpreted as a generalized distortion measure, which is the weighted sum of the squared distances from all the points in the cluster domain to their assigned cluster center. The weights are the fuzzy membership values [28][29]. Hence, this metric is proportional to the inverse of the quality of FCM. Lower distortion denotes higher quality. Metrics 6 and 7 are further elaborated in the next section.

In general, the rate distortion function is used when the FCM is utilized for quantization. In this case, after convergence, the matrix $U = [u_{ij}]$ is defuzzified; e.g., by using a nearest neighbor assignment. The compression rate of FCM is fixed by the selection of c. Hence, the rate distortion quality-measure boils down to the MMSE, given by:

$$D = \frac{1}{m}\sum_{i=1}^{c}\sum_{x_j \in \omega_i}\|x_j - \omega_i\| \tag{2}$$

Again, lower distortion denotes higher quality. When the clustering is used for classification, a quality criteria that measure the density of cluster as well as the relative distance between clusters can be used to estimate the recognition accuracy. In this case, a dispersion measure can be used. To elaborate, let $S = \{S_1, S_2, \dots, S_c\}$ be the set of clusters obtained through "hard clustering," and let $\Omega = \{\omega_1, \omega_2, \dots, \omega_c\}$ be the set of the corresponding cluster centers, then W_i, the within dispersion matrix of the cluster S_i, is defined to be the covariance matrix of the set of elements that belong to S_i. The within dispersion matrix of S, (W) is a given function of the entire set of the within dispersion matrices of the individual clusters. For example, the elements of W can be the averages of the compatible elements of W_i for $1 \le i \le c$. The between dispersion matrix of S, (B), is the covariance matrix of Ω. The quality of the clustering can be expressed as a function of the within dispersion matrix W and the between dispersion matrix B. A commonly used dispersion function is [44]:

$$D = tr(W)/tr(B) \tag{3}$$

where $tr(M)$ is the trace of the matrix M.

B. The Fuzzy C-Means Algorithm

The FCM consists of two main phases: setting/updating the membership of vectors in clusters and setting/updating cluster centers. Some variants of FCM start with a set of centers which induces a partition matrix [28][29]. In this case, seeding the algorithm relates to the initial selection of centers. Other variants initialize a partition matrix which induces initial centers [24]. Hence, seeding these FCM variants amounts to initializing the partition matrix. The two approaches are virtually equivalent that choosing one over the other is just a matter of convenience related to the format of data and the form of the application. We are using the second approach, where the seeding relates to selecting the initial partition matrix. Hence, in the seeding step, the membership matrix is initialized. In the next iterations, the cluster centers are calculated and the partition matrix is updated. Finally, the value of the objective function for the current classification is calculated. The algorithm terminates when a limit on the number of iterations is reached or a "short circuit condition" is met. A commonly used termination condition halts the algorithm when the derivative of the distortion function is small. Because the c-means algorithm is sensitive to the seeding method, a variety of procedures have been proposed for selecting seed points [26][52]. The following paragraphs include a formal definition of the algorithm and presents a pseudo-code.

Given a set of vectors $X = \{x_1, x_2, \dots, x_m\}$, where $x_i \in R^n$ and an initial partition matrix $U^{(0)}$, the FCM is an iterative algorithm for partitioning a set of vectors into c clusters $S = \{S_1, S_2, \dots, S_c\}$, with cluster centers $\Omega = \{\omega_1, \omega_2, \dots, \omega_c\}$. In iteration l the algorithm uses the cluster centers $\Omega^{(l)} = \{\omega_1^{(l)}, \omega_2^{(l)}, \dots, \omega_c^{(l)}\}$ induced by the partition matrix $U^{(l)}$ to re-partition the data set and obtain a new partition matrix $U^{(l+1)}$. Cluster centers at iteration l are computed according to:

$$\omega_i^{(l)} = \left(\sum_{j=1}^{m}\left(u_{ij}^{(l)}\right)^{q-1} \cdot x_j\right) \Big/ \left(\sum_{j=1}^{m}\left(u_{ij}^{(l)}\right)^{q-1}\right) \tag{4}$$

The matrix $U^{(l+1)} = \left[u_{ij}^{(l+1)}\right]$ is calculated according to:

$$U^{(l+1)} = \left[u_{ij}^{(l+1)}\right] = \sum_{l=1}^{c}\left(\frac{\left\|x_j - \omega_i^{(l)}\right\|}{\left\|x_j - \omega_l^{(l)}\right\|}\right)^{-\frac{2}{q-1}} \tag{5}$$

The process of center induction, data partition, and matrix update continues until a given termination condition, which relates to an optimization criteria or limit on the number of iterations, is met. The following is a commonly used criterion [16]:

$$\left|\frac{J_q^{(l-1)} - J_q^{(l)}}{J_a^{(l-1)}}\right| < \varepsilon \tag{6}$$

Fig. 1 is a pseudo code of the algorithm.

```
1. Parameters:
   a. $X = \{x_1, x_2, \dots, x_m\}$, $(x_j \in R^n)$ - a set of vectors
   b. $m$ - the number of vectors
   c. $c$ - the number of partitions
   d. $q$ - a weighting exponent $(q > 1)$
   e. $U^{(k)} = \left[u_{ij}^{(k)}\right]$ - the partition matrix at iteration $k$
   f. $\Omega^{(k)} = \{\omega_1^{(k)}, \omega_2^{(k)}, \dots, \omega_c^{(k)}\}$ - the set of clustering centers at iteration $k$
   g. $N$ - the maximum number of iterations
   h. $J_q^{(k)}$ - the objective-function's value at iteration $k$
2. Set $k = 0$, choose an initial partition matrix $U^{(0)}$
3. In iteration $k \ge 0$ let $\Omega^{(k)} = \{\omega_1^{(k)}, \omega_2^{(k)}, \dots, \omega_c^{(k)}\}$ be the induced clustering centers computed by equation 4.
   a. Set $U^{(k+1)} = \left[u_{ij}^{(k+1)}\right]$ according to equation 5.
   b. Compute $J^{(k+1)}$ according to equation 1.
   c. Set $k = k + 1$.
4. Stop if $k = N$; or if $k > 1$, and equation 6 holds for a small $\varepsilon$ such as $\varepsilon = 10^{-6}$. Otherwise, go to (2).
```

Figure 1. Algorithm for baseline FCM

The idea behind the multistage methods reported in the next section is that an estimate of the partition matrix and the location of the cluster centers can be obtained by clustering a sample of the data. There is a trade-off that relates to the sample size. A small sample is expected to produce a fast yet less reliable estimation of the cluster centers. This leads to a multistage approach, which involves several stages of sampling (with replacement) of the data and estimating the membership matrix for the next stage. The size of the first sample should be as small as possible. On the other hand, it should be statistically significant [44]. Each of the stages includes more objects from the data and sets the initial partition matrix of stage I according to the final partition matrix of stage $I-1$.

C. *The LBG Termination Criterion*

The main difference between the LBG variant of the FCM algorithm and the classical FCM algorithm is the termination condition. The LBG algorithm stops when an approximation for the derivative of the MSE given by $\frac{\left|D^{(m-1)}-D^{(m)}\right|}{D^{(m-1)}}$ is smaller than a threshold, $\left(\frac{\left|D^{(m-1)}-D^{(m)}\right|}{D^{(m-1)}}<\varepsilon\right)$.

D. *The Sequential Fuzzy C-Means Algorithm*

FCM can be executed in one of two basic modes; batch mode and online mode. The batch mode updates the partition matrix after considering the entire set of data. The online mode is also referred to as the sequential mode. In an epoch l of a sequential mode, for each pattern, the partition matrix $U^{(l)}$ and cluster centers $\Omega^{(l)}$ are found. The cluster centers $\Omega^{(l)}$ are used to find the distortion value and weighted distortion value. The partition matrix $U^{(l)}$ are used in the next epoch of the sequential mode.

E. *The Block Sequential Fuzzy C-Means Algorithm*

The block sequential mode is a compromise between the stringent computational requirements of the sequential FCM and the need to operate on data online. In this case, the clustering occurs on accumulated blocks of data. Each block is going through l epochs of FCM where the final centers of block i are used as the initial centers for block $i+1$. In many cases $l=1$. In this sense, the algorithm resembles other multistage clustering such as the pyramid FCM. The block sequential algorithm might be utilized in an iterative fashion, where each of the iterations performs l epochs of FCM on a single block of data elements at a time.

Note that all the clustering algorithms described so far assume that (at some point) the entire data set is available. Moreover, generally, due to the iterative fashion of execution, these algorithms access the same elements more than once (in different iterations). When the data is very large and cannot fit the memory of the processor, a different approach, referred to as single-pass, has to be adapted. Under the single-pass (incremental) approach, each block/data-element is accessed only one time and then removed from internal memory to provide space for new elements. This is described next.

F. *The Incremental Fuzzy C-Means Algorithm*

The incremental FCM algorithm presented in this paper is similar to the block sequential algorithm with the exception that each block is accessed only one time. Each block is going through a set of l epochs of FCM where the final centers of block i are used as the initial centers for block $i+1$.

The fact that each block is "touched" just one time might raise a question about the validity of the results. The results might be valid if the data elements of blocks share similar features. For example, the data elements are drawn from the same probability distribution function or the same fuzzy membership function. Alternatively, validity might be attained if the features of data elements vary "slowly" between blocks. We use two methods to improve the validity of results. First, we use a two-phase incremental algorithm. In the first phase, a very large set of clusters is used. Practically, we are trying to fit as many elements in a block and as many clusters per block as possible in the memory. In the next phase, after processing all the blocks, a process of clustering the centers obtained from the last block is applied. The second measure for increasing validity is using a relatively large number of clusters and at the same time allowing the number of clusters to change dynamically. This is described in the next section.

G. *The Dynamic Incremental Fuzzy C-Means Algorithm*

The dynamic incremental FCM algorithm presented in this paper is similar to the incremental FCM algorithm. The difference is that the number of clusters is allowed to change.

Several operations can change the number of clusters. First, following the ISODATA algorithm principles, clusters with too few elements might be eliminated, clusters that are too close to each other might be merged, and clusters with large dispersion might be split [13][25][27]. The criteria for merge and split might be related to the within and between dispersion of the clusters [1]-[3][43]. Other methods for changing the number of clusters might include incrementing/decrementing the number of clusters (without split/merge) based on a criterion such as a threshold on the distortion. We have implemented the threshold approach.

Each block is going through a set of l epochs of FCM where the final centers of block i are used as the initial centers for block $i+1$. Following the application of FCM on a block, a decision concerning the effective number of clusters is made and the number might be incremented or decremented based on a predetermined quality criteria threshold. We place an upper bound and a lower bound on the number of clusters, where the lower bound ensures that we still have enough clusters to maintain validity and enable the two-phase approach described above. Again, we are trying to fit as many elements in a block and a large number of clusters per block in the memory and apply a two-phase approach where the centers from the last iteration are clustered and provide the final set of clusters.

IV. EXPERIMENTS AND RESULTS

A. *Experimental Setup*

In order to compare and contrast the performance and validity of the FCM variants presented, we have implemented these algorithms numerous times on different data sets, using different parameters. Three sets of data of data are used for the experiments performed. The first set (1) consists of the red, green, and blue (RGB) components of relatively small (512×512 pixels) color images. These images e.g., Lena and Baboon are used by many other researchers and the results of algorithms that use these images are published in numerous papers and books [2][6]. The images of the first set are subject to color quantization. The second set (2) is composed of the RGB components of relatively large aerial photography images ($9K \times 9K$ pixels or 81 million pixels) color this set is used for more aggressive color quantization. The third set (3) includes 20 million elements of six-dimensional synthetic data points with known centers and known distribution. Hence, data sets (2) and (3) are relatively large. It should be noted that the $9K \times 9K$ pixel images are the largest images that fit the memory of our current hardware/software configuration. This is important since we use the entire image for running non-incremental clustering in order to assess the results of the incremental clustering and this is the maximal size that can be used for non-incremental clustering. Three types of output data/results are collected: 1) records of convergences (distortion per iteration), 2) execution time, and 3) clustering quality (inverse of distortion at the final iteration).

The experiments are divided into two classes; multi-pass and single-pass. In the multi-pass experiments, we compared the performance of classical FCM, LBG based FCM, sequential FCM, and block sequential FCM. Given the constraints of these algorithms, they have been applied to a manageable data set (i.e., a data set with a medium number of elements). In the single-pass experiments, we tested the incremental and the dynamic incremental approaches with the same data used for the multi-pass algorithms and with a “huge” set of synthetic data that is not suitable for multi-pass processing. Nevertheless, to verify the results with the large data set, we ran the LBG variant of the FCM algorithm on that data using a “powerful” multicore computer. The computer worked on the data for several hours. For the dynamic incremental algorithm, we used an approach where the number of clusters is incremented/decremented by 3 based on a threshold on distortion and the number of clusters during the current epoch.

1) Color Quantization

The problem of color quantization can be stated in the following way: given an image with N different colors, choose $K \ll N$ colors such that the resulting K-color image is the least distorted version of the original image [1]. Color quantization can be implemented by applying the FCM clustering procedure to the image-pixels where each pixel represents a vector in some color representation system. For example, the clustering can be performed on the three-dimensional vectors formed by the RGB color components of each pixel in the image [1]. After clustering, each three-dimensional vector (pixel) is represented by the cluster-number to which the vector belongs and the cluster centers are stored in a color-map. The K-value image, along with the color-map, is a compressed representation of the N-colors original image. The compressed image can be used to reconstruct the original three-dimensional data set by replacing each cluster-number by the centroid associated with the cluster. In the case of 8-bit per color component and $K = 16$, the original 24-bit per pixel image is represented by a 4-bit per pixel image, along with a small color map. Hence, about six-fold compression is achieved. In this set of experiments, a block processed by the single-pass algorithm consists of an image row. The sequential algorithm is applied to every pixel of a scaled down version of the images, while the rest of the multi-pass algorithms operate on the entire set of the pixels of the original images. The experimental results are scaled to represent the distortion for the entire image. The static incremental algorithm starts with 192 clusters per block. Following the processing of the last block, the 192 centers are clustered into 16 centers and the distortion for the entire image with these centers is measured. The dynamic incremental algorithm starts with 192 clusters per block and allows fluctuations in this number. Following the processing of the last block, the centers are clustered into 16 centers and the distortion for the entire image with these centers is measured.

2) Synthetic Data

A set of 20 random cluster centers with a total of $20{,}000{,}000$ six-dimensional vectors is generated. The vectors within a cluster are distributed according to a 2-D normal distribution with standard deviation of 0.05 around the center. For the single-pass experiments, the data is divided into 500 blocks of 40,000 elements per block. Other parameters are identical to the ones used for the color map quantization experiments.

B. Experimental Results

Fig. 2 shows the distortion per iteration of the multi-pass algorithms executed on the image Lena, which is a 512×512 RGB image. Fig. 2a shows the distortion results of the LBG variant the final distortion is 23.6 db. The block sequential algorithm distortion per iteration is presented in Fig. 2b. The distortion results converge to the value of 23.8 db. This is equivalent to the results of running the other single pass algorithms for several iterations. Visually, all the versions of clustering executed in this research produced about the same reconstructed image.

Fig. 3a and Fig. 3b show the results of running the single-pass incremental version on the image Lena with single row per block. Both the static and dynamic incremental algorithms converge to a distortion value of about 25 db. This is slightly higher than the multi pass algorithms. But, it is expected as the single pass algorithm “visits” every block only one time. The dynamic incremental algorithm has slightly lower distortion than the incremental algorithm. We have repeated these experiments numerous times, with different seed values using nine different images and obtained similar results.

Fig. 4a and Fig. 4b show the results of running the single-pass incremental version on the image “Neighborhood” with single row per block. Both the static This is a much larger image, and it is subject to more aggressive quantization which ends up in a monochromatic image. In this case, the distortion obtained through the LBG variant on the entire image is 20.4 db. The incremental algorithm and the dynamic incremental algorithms produce a result of 23.2 db. and 24 db. Respectively. The degradation is justified by the fact that the $9K \times 9K$ image is the largest image that fits our software/hardware configuration. Hence, a larger image cannot undergo a multi pass procedure.

Fig. 5 shows the original and reconstructed versions of “Neighborhood.” Figs. 6 and 7 show the same experiments with the image “Park.” The results are similar with a distortion of 17.5 db., 19.5 db., and 20 db., for the LBG, incremental and dynamic incremental variants respectively.

Fig. 8a and Fig. 8b show the results of incremental clustering and dynamic incremental clustering with a large amount of synthetic data points (20,000,000 points) in a six-dimensional space). Fig. 8a shows the results of running the static incremental FCM on the synthetic data. Due to the fact that the data is drawn from a fixed distribution, the results of distortion per block are quite stable and the execution ends up at a distortion value of 0.38. The execution of the dynamic version of incremental FCM is depicted in Fig. 8b. The distortion per block is better than results obtained for the static case and stabilizes at 0.11.

V. RESULT EVALUATION

The results of the experiments reported and additional experiments performed show the utility of using a two-phase single-pass incremental FCM algorithm, where the first phase uses a large number of centers and the second phase clusters the centers obtained in the first phase into a desired size of clusters. Moreover, the dynamic clustering approach allows the number of centers in the first phase to vary and, in the case of very large data sets, outperforms the static incremental approach.

VI. CONCLUSION AND FUTURE WORK

This paper has reviewed static and dynamic single-pass and multi-pass variants of the FCM. A novel two-phase static single-pass algorithm as well as a dynamic two-phase single-pass algorithm have been presented and are showing high utility. Future research will concentrate on additional methods for dynamic change in the number of clusters in both steps of dynamic incremental FCM. In addition, we plan to initiate research on equivalent approaches in the KNN. We also plan to investigate parallel incremental algorithms. Finally, we have recently received access to an aerial photography data set where the resolution of each image is $10^6 \times 10^6$ pixels (1 tera pixels) and we plan to explore the algorithms with this data set. Additionally, we have access to a vector data set related to the aerial photography. This set contains 170 million records of varying size with an average size of 40 entries per record. Moreover, this set contains a mixture of categorical and numerical data. We plan to use this set as well for further exploration of our algorithms.

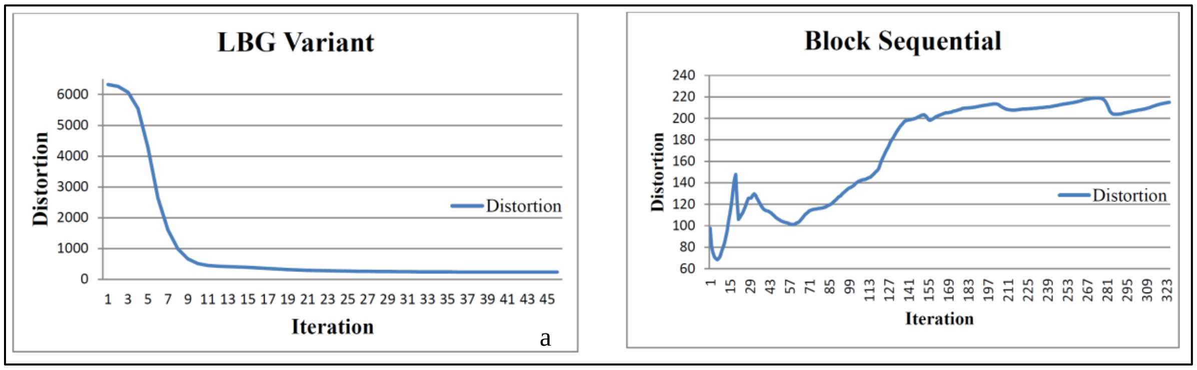


Figure 2. Distortion per iteration for the multi-pass algorithms with the Image Lena

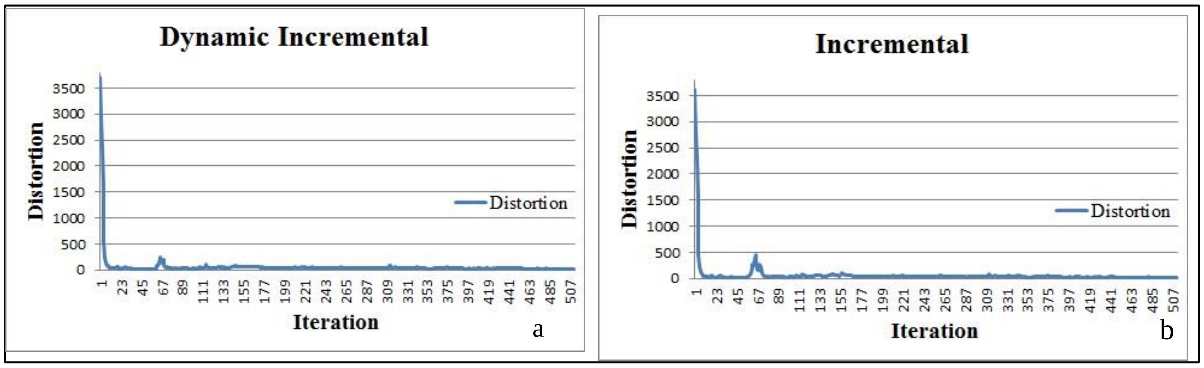


Figure 3. Incremental and Dynamic Incremental Clustering of the Image Lena

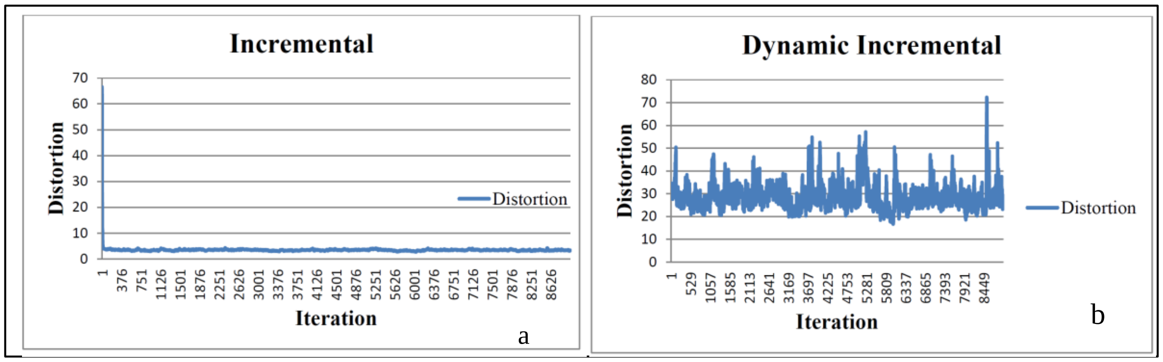


Figure 4. Incremental and Dynamic Incremental Clustering of the Image “Neighborhood”

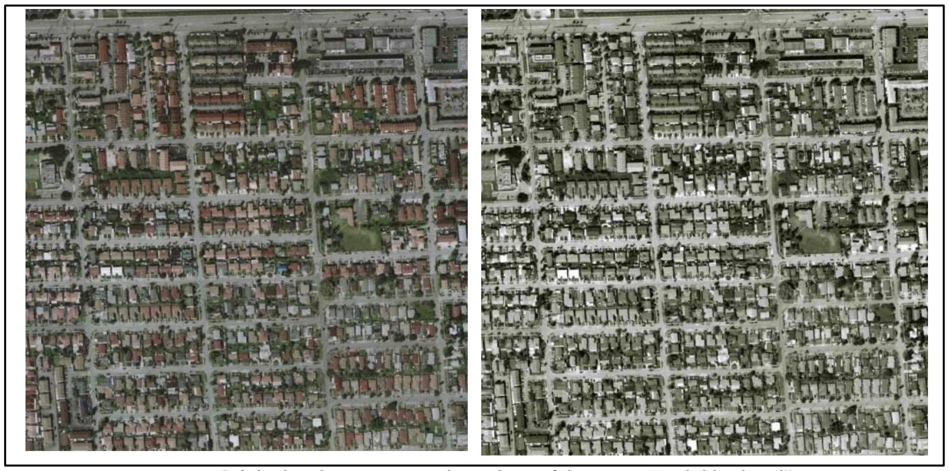


Figure 5. Original and Reconstructed Versions of the Image “Neighborhood”

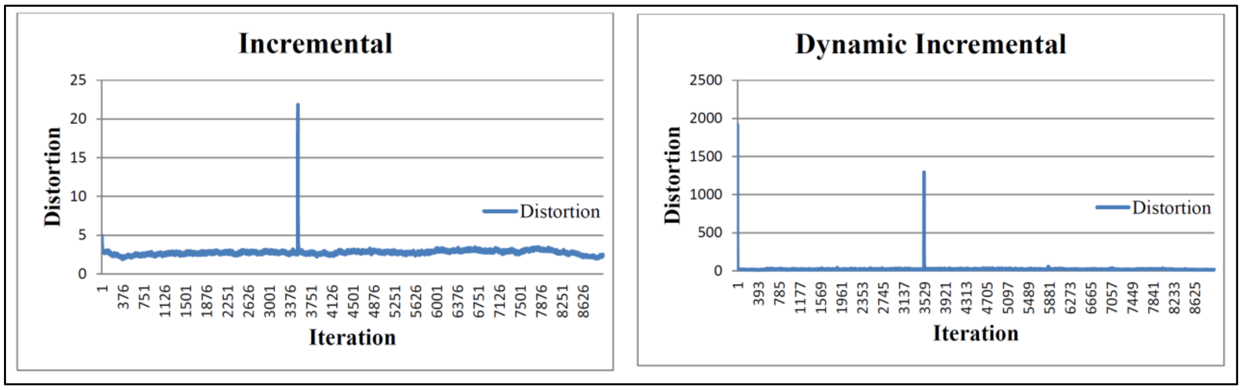


Figure 6. Incremental and Dynamic Incremental Clustering of the Image “Park”

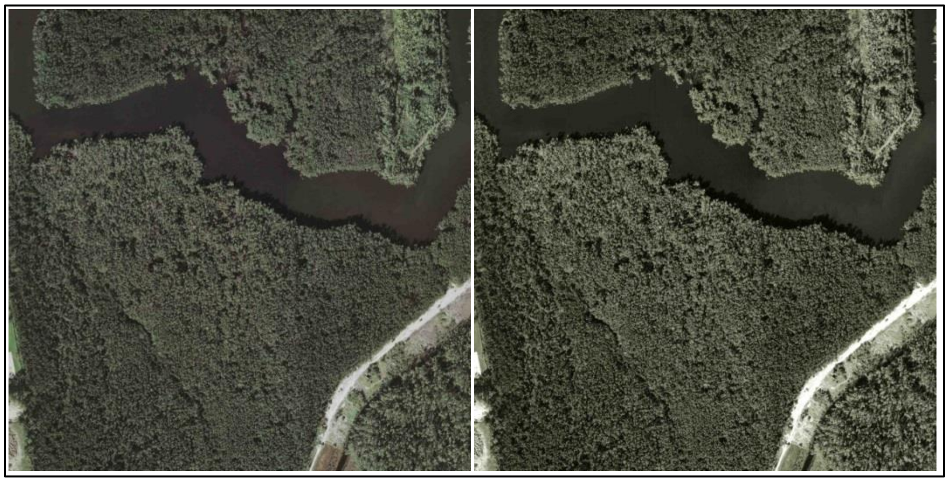


Figure 7. Original and Reconstructed Versions of the Image “Park”

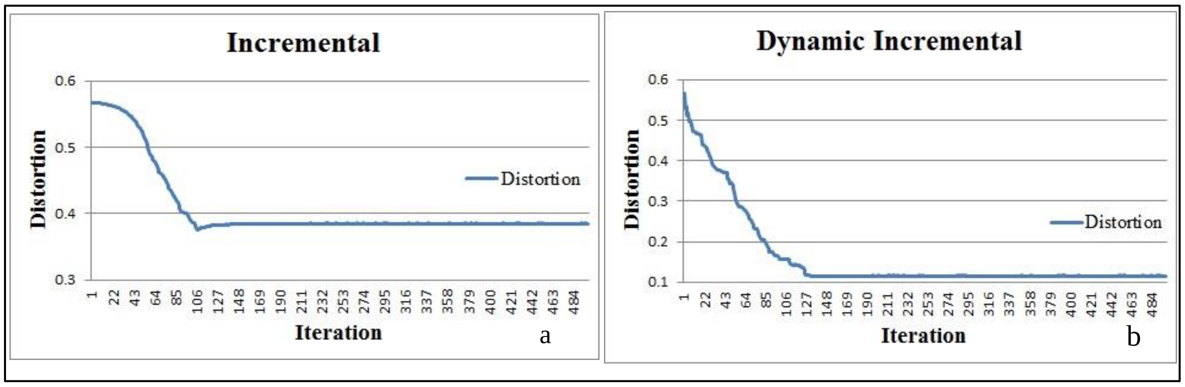


Figure 8. Incremental and Dynamic Incremental Clustering of Synthetic Data

ACKNOWLEDGMENT

This material is based in part upon work supported by the National Science Foundation under Grant Nos. I/UCRC IIP-1338922, AIR IIP-1237818, SBIR IIP-1330943, III-Large IIS-1213026, MRI CNS-0821345, MRI CNS-1126619, CREST HRD-0833093, I/UCRC IIP-0829576, MRI CNS-0959985, and FRP IIP-1230661.

REFERENCES

[1] D. E. Tamir and A. Kandel, “The Pyramid Fuzzy C-means Algorithm,” International Journal of Computational Intelligence in Control, vol. 2 iss: 2, Dec. 2010, pp. 270-302.

[2] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in Proc. 5th Berkeley Symposium on Mathematical Statistics and Probablity, vol. 1, University of California Press, 1967, pp. 281-297.

[3] J. T. Tou and R. C. Gonzalez, Pattern Recognition Principles. London: Addison-Wesley, 1974.

[4] P. Berkhin, ”Survey of Clustering Data Mining Techniques,” Technical Report, Accrue Software, San Jose, CA, 2002.

[5] A. A. El-Gamal, “Using Simulated Annealing to Design Good Codes,” IEEE Transactions on Information Theory, vol. 33 iss: 1, Jan. 1987, pp. 116-123.

[6] T. Kohonen, (1991) Self-organizing maps: optimization approaches. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, (eds), Artificial Neural Networks. Proceedings of ICANN'91, International Conference on Artificial Neural Networks, vol. II, 1991, pp. 981-990, North-Holland, Amsterdam.

[7] P. S. Bradley, U. M. Fayyad, and C. A. Reina, ”Scaling Clustering Algorithms to Large Databases,” in Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD98), R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro (eds), AAAI Press, Menlo Park, CA, 1998, pp. 9-15.

[8] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, “Incremental clustering and dynamic information retrieval,” in STOC '97 Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, ACM, New York, NY, USA, May 1997, pp. 626-635.

[9] F. Farnstrom, J. Lewis, and C. Elkan, “Scalability for clustering algorithms revisited,” in ACM SIGKDD Explorations Newsletter, ACM, New York, NY, USA, vol. 2 iss: 1, June 2000, pp. 51-57.

[10] C. Gupta and R. Grossman, “GenIc: A Single Pass Generalized Incremental Algorithm for Clustering,” in the Fourth (SIAM) International Conference on Data Mining (SDM04), 2004, pp. 147-153.

[11] J. Lin, M. Vlachos, E. Keogh, “Iterative Incremental Clustering of Time Series,” Advances in Database Technology - EDBT 2004, vol. 2992, March 2004, pp. 106-122.

[12] D. E. Tamir, C. Park, and W. Yoo, “The validity of pyramid K-means clustering,” in Proc. SPIE 6700 Mathematics of Data/Image Pattern Recognition, Compression, Coding, and Encryption X, with Applications, 67000D, Sept. 2007, pp. 658-675, doi:10.1117/12.735436.

[13] D. E. Tamir, Cluster Validity of Multi Resolution Competitive Neural Networks. International Conference on Artificial Intelligence, 2007, pp. 303-312.

[14] G. B. Coleman and H. C. Andrews, “Image segmentation by clustering,” in Proceedings of the IEEE, vol. 67 iss: 5, May 1979, pp. 773-785.

[15] W. H. Equitz, “A new vector quantization clustering algorithm,” in Acoustics, Speech and Signal Processing, IEEE Transactions, vol. 37 iss: 10, Oct. 1989, pp. 1568-1575.

[16] Y. Linde, A. Buzo, and R. Gray, “An Algorithm for Vector Quantizer Design,” in IEEE Transactions on Communications, vol. 28 iss: 1, Jan. 1980, pp. 84-95.

[17] H. V. Lung and J.-M. Kim, “A generalized spatial fuzzy C-means algorithm for medical image segmentation,” in Proceedings of the 18th International Conference on Fuzzy Systems, Jeju Island, Korea, Aug. 2009, pp. 409-414. ISSN: 1098-7584, doi:10.1109/FUZZY.2009.5276878.

[18] S. Das and A. Konar, “Automatic image pixel clustering with an improved differential evolution,” Applied Soft Computing, vol. 9 iss: 1, Jan. 2009, pp. 226-236, doi:10.1016/j.asoc.2007.12.008.

[19] X. Xu, J. Jager, and H.-P. Kriegel, “A Fast Parallel Clustering Algorithm for Large Spatial Databases,” Data Mining and Knowledge Discovery, vol. 3 iss: 3, Sept. 1999, pp. 263-290.

[20] P. Heckbert, “Color image quantization for frame buffer display,” ACM Transactions on Computer Graphics, vol. 16 iss: 3, July 1982, pp. 297-307.

[21] S. Lloyd, “Least squares quantization in PCM,” IEEE Transactions on Information Theory, vol. 28 iss: 2, Mar. 1982, pp. 129-137.

[22] M. Garey, D. Johnson, and H. Witsenhausen, “The complexity of the generalized Lloyd-Max Problem (Corresp.),” IEEE Transactions on Information Theory, vol. 28 iss: 2, Mar. 1982, pp. 255-256.

[23] G. H. Ball and J. D. Hall, “ISODATA, A Novel Method of Data Analysis and Pattern Classification,” Technical Report, SRI International, Stanford 1965.

[24] J. C. Bezdek, “A Convergence Theorem for the Fuzzy ISODATA Clustering Algorithms,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-2 iss: 1, Jan. 1980, pp. 1-8.

[25] D. Cheng, R. Kannan, S. Vempala, and G. Wang, “A divide-and-merge methodology for clustering,” ACM Transactions on Database Systems (TODS), vol. 31 iss: 4, Dec. 2006, pp. 1499-1525.

[26] C. Alippi and R. Cucchiara, “Cluster partitioning in image analysis classification: a genetic algorithm approach,” in Proceedings of the Computer Systems and Software Engineering, CompEuro ’92, May 1992, pp. 139-144, doi: 10.1109/CMPEUR.1992.218520.

[27] S. Young, I. Arel, T. P. Karnowski, D. Rose, "A Fast and Stable Incremental Clustering Algorithm," 2010 Seventh International Conference on Information Technology: New Generations (ITNG), April 2010, pp.204-209.

[28] J. C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithm, New York, USA: Plenum Press, 1981.

[29] A. Kandel, Fuzzy Mathematical Techniques with Applications, New York: Addison Wesley, 1987.

[30] A. M. Bensaid, L. O. Hall, J. C. Bezdek, and L. P. Clarke, “Partially supervised clustering for image segmentation,” Pattern Recognition, vol. 29 iss: 5, May 1996, pp. 859-871.

[31] T. W. Cheng, D. B. Goldgof, and L. O. Hall, “Fast fuzzy clustering,” Fuzzy Sets and Systems, vol. 93 iss: 1, Jan. 1998, pp. 49-56.

[32] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler, Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition, New York: Wiley, 1999.

[33] S. K. Pal and D. K. D. Majumder, Fuzzy Mathematical Approach to Pattern Recognition, New York: Wiley, 1986.

[34] L. Ma and R. C. Stauton, “A modified fuzzy C-means image segmentation algorithm for use with uneven illumination patterns,” Pattern Recognition, vol. 40 iss: 11, Nov. 2007, pp. 3005-3011.

[35] R. R. Yager, S. Ovchinnikov, R. M. Tong, and H. T. Nguyen, Fuzzy Sets and Applications: Selected Papers by L. A. Zadeh. New York: Wiley, 1987.

[36] J. Y. Hsiao and A. Sawchuk, “Unsupervised textured image segmentation using feature smoothing probabilistic relaxation techniques,” Computer Vision, Graphics, and Image Processing, vol. 48 iss: 1, Oct. 1989, pp. 1-21.

[37] J. F. Huang, “Image Segmentation Using Discrete Cosine Transform and K-Means Clustering Algorithm,” in Computer Science. 1991, Florida Institute of Technology: Melbourne, Florida.

[38] Y. Zhu, “Image Segmentation for Color Separation,” in Computer Science. 1991, Florida Institute of Technology: Melbourne, Florida.

[39] P. Hore, L. W. Hall, and D. B. Goldgof, “Speedup of Fuzzy Clustering Through Stream Processing on Graphics Processing Units,” in IEEE International Conference on Fuzzy Systems, London 2007.

[40] A. Rosenfeld, “Pyramids: Multi-resolution Image Analysis,” 3rd Scandinavian Conference on Image Analysis, June 1983, pp. 23-28.

[41] S. Kasif and A. Rosenfeld, “Pyramid linking is a special case of ISODATA,” IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-13 iss: 1, Jan.-Feb. 1983, pp. 84-85.

[42] J. C. Tilton, “Multi-resolution Spatially Constrained Clustering of Remotely Sensed Data on the Massively Parallel Processor,” IGARSS Symposium, 1984, pp. 661-666.

[43] E. Lughofer, “Dynamic Evolving Cluster Models Using On-line Split-and-Merge Operations,” in Proceedings of the 2011 10th International Conference on Machine Learning and Applications and Workshops (ICMLA), vol. 2, Dec. 2011, pp. 20-26.

[44] A. K. Jain and R. C. Dubes, Algorithms for Clustering Data. Englewood Cliffs: Prentice Hall, 1988.

[45] R. L. Cannon, J. V. Dave, and J. C. Bezdek, “Efficient Implementation of the Fuzzy C-Means Clustering Algorithms,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8 iss: 2, March 1986, pp. 248-255.

[46] J. F. Kolen and T. Hutcheson, “Reducing the time complexity of the fuzzy C-means algorithm,” IEEE Transactions on Fuzzy Systems, vol. 10 iss: 2, Apr. 2002, pp. 263-267.

[47] D. Altman, “Efficient fuzzy clustering of multi-spectral images,” in IEEE 1999 International, IGARSS ’99 Proceedings, Geoscience and Remote Sensing Symposium, vol. 3, Jun.-Jul. 1999, pp. 1594-1596.

[48] S. Eschrich, J. Ke, L. O. Hall, and D. B. Goldgof, “Fast accurate fuzzy clustering through data reduction,” IEEE Transactions on Fuzzy Systems, vol. 11 iss: 2, Apr. 2003, pp. 262-270.

[49] N. R. Pal and J. C. Bezdek, “Complexity reduction for "large image" processing,” IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 32 iss: 5, Oct. 2002, pp. 598-611.

[50] S. Rahimi, M. Zargham, A. Thakre, and D. Chhillar, “A parallel Fuzzy C-Mean algorithm for image segmentation,” IEEE Annual Meeting of the Fuzzy Information (Processing NAFIPS ’04), vol. 1, June 2004, pp. 234-237.

[51] A. Petrosino and M. Verde, “P-AFLC: a parallel scalable fuzzy clustering algorithm,” in Proceedings of the 17th International Conference on Pattern Recognition (ICPR 2004), vol. 1, Aug. 2004, pp. 809-812, ISSN: 1051-4651, doi: 10.1109/ICPR.2004.1334340.

[52] D. T. Anderson, R. H. Luke, and J. M. Keller, “Speedup of Fuzzy Clustering Through Stream Processing on Graphics Processing Units,” IEEE Transactions on Fuzzy Systems, vol. 16 iss: 4, Aug. 2008, pp. 1101-1106.

SPEM: A Software Pattern Evaluation Method

J. Kabbedijk, R. van Donselaar, S. Jansen
Department of Information and Computing Sciences
Utrecht University, The Netherlands
{J.Kabbedijk, R.VanDonselaar, Slinger.Jansen}@uu.nl

***Abstract*—Software architecture makes extensive use of many software patterns. The decision on which pattern to select is complex and architects struggle to make well-advised choices. Decisions are often solely made on the experience of one architect, lacking quantitative results to support the decision outcome. There is a need for a more structured evaluation of patterns, supporting adequate decision making. This paper proposes a Software Pattern Evaluation Method (SPEM), that enables the quantification of different pattern attributes by using structured focus groups. The method is formed using a design science approach in which an initial method was created using expert interviews, which was later refined using several evaluation sessions. SPEM helps software producing companies in structuring their decision making and selecting the most appropriate patterns. Also, SPEM helps in enriching pattern documentation by providing a way to add quantitative information to pattern descriptions.**

Keywords*—*architectural patterns. quality attributes. software architecture. decision making. pattern evaluation.

I. Introduction

Modern software architecture heavily relies on the use of many different software patterns, often used complementary to each other in order to solve complex architectural problems. Software architecture provides guidelines and tools for high level system design in which architects select best fitting patterns to be used within the software product [1]. Many different patterns and tactics exist, leading to a complicated trade-off analysis between different solutions and causing the evaluation and selection of the appropriate software patterns to be a complex task [2]. This complexity means architects need to have in-depth understanding of the project characteristics and requirements combined with extensive experience in software development.

The information needed for appropriate pattern selection is seldom available to all architects in a centralized or standardized way. Architectural decisions are frequently made based on experience and personal assessment of one person, instead of using the knowledge of many [3]. Allowing software architects to use all information efficiently saves time when selecting fitting software patterns and leads to better and more adequate decision making. For this to be possible, a method has to be created enabling the evaluation and documentation of crucial attributes of a software pattern [4]. This structured evaluation will allow architects and decision makers to compare different solutions and select the best matching pattern. Patterns, however, are a high-level solution that can be used different scenarios, making it impossible to use one specific implementation of the pattern to evaluate the entire pattern. Because specific implementations are unusable, the relevant pattern attributes can not be directly measured in a quantitative way.

Pattern evaluation adds retrospect and the knowledge of many experts to existing pattern documentation. This study also relates to software architecture as it solves a problem found in the software pattern selection process. Software pattern evaluation helps when performing pattern oriented software architecture in cases where alternative patterns to solve the same problem and only a single pattern can be selected. This is an important factor to take into account, because it means that rather than selecting individual patterns, an architect will want to select an architectural style, and thus select a large set of patterns that fit this style. This area of software architecture has developed, which resulted in a large amount of documented patterns and allows for comparing architectural styles [5].

Although it seems that comparing individual patterns is less relevant for software architecture, an architectural style is selected at the early stages of software design and cannot easily be changed after the development has started. This creates a problem because while the software is being developed, the requirements for the project or the environment will change. Therefore it is necessary to extend the architecture or at times alter the existing architecture. At this point it becomes relevant to compare individual patterns in order to select the pattern that fits the project requirements. This is an ongoing process that happens throughout software development and relies on the experience of software architects and developers. Current documentation of software patterns is lacking a way to compare them with each other. But, if multiple patterns tackle the same problem, how does an architect decide which one to use? This is tacit knowledge of experienced architects and developers, leading to the following problem statement:*“There is no formal way to express the quality of one pattern over another”*.

This paper presents the Software Pattern Evaluation Method (SPEM). Using SPEM, software producing companies are supported in pattern selection decision making and are able to quantitatively compare different patterns. SPEM enables them to get an overview of specific pattern characteristics in a timely manner. Also, SPEM can be used to generate a publicly available pattern related body of knowledge, helping research and practitioners in architectural research and decision making. This paper first gives an overview of research related to pattern comparison in Section II. The design science approach used in the research is described in Section III, after which SPEM is presented in Section IV. The pattern evolution, including the initial method creation (Section V-A) and method evaluation (Section V-B), showing the changes during the method creation process can be found in Section V. To conclude, the applications of SPEM are discussed (Section VI), followed by a conclusion (Section VII).

II. Related Work

Software Patterns — As software development was maturing in the 1980s, the need arose to share common solutions to recurring problems. This process started out by developers communicating to their colleagues how they solved a recurring development issue. The communication was informal and there were no clear rules for documentation. In later years, software patterns have become an essential part of software development as a way to capture and communicate knowledge. Software patterns are solutions to a recurring problem in a particular context [6], [7]. When properly documented these solutions are a valuable asset for communication with and among practitioners [8]. Usage of software patterns allows for time and cost reduction in software development projects, making them an important tool for software design and development. Although software patterns started out as a way to communicate solutions among developers, they have become a crucial part of software architecture [9] as well. A pattern selected by a developer, however, does not take into account the entire architecture and how it combines with existing patterns. This problem is solved by selecting patterns at the architectural level.

Architecture Evaluation — Evaluation is commonly used in software architecture in order to increase quality and decrease cost [10]. Many evaluation methods for software architecture have been developed and compared in recent years [11]. The evaluation should be performed as early as possible in order to prevent large scale changes in later stages of development. Software architecture evaluation is linked to the development requirements and desired quality attributes. Therefore, it is not a general evaluation of software architecture, nor an evaluation of a specific implementation. The evaluation should be an indication of whether the proposed architecture is a good fit for the project. Pattern comparison and evaluation has been done before [12] in a quantitative manner, but has focussed on the *implementation* of different patterns and lacks the evaluation of the *idea* the pattern describes.

III. Research Approach

This section presents the research questions answered in this paper and the design science approach used to construct SPEM. The main research question (MRQ) answered in this paper is:

> ***MRQ:*** *How can software patterns be evaluated in a manner that is objective and allows for comparison?*

The aim of this study is to aid software architects in the decision making process of selecting software patterns. This can only be useful when the evaluation method yields objective results. Since a software pattern can not be objectively measured in any way, the opinions of multiple software architects are used in the form of scores. A quantitative study also allows for easy comparison between alternative patterns. For the purpose of answering this research question, multiple sub-questions are constructed:

> ***SQ1:*** *Which attributes are relevant in pattern evaluation?*

Rationale: Patterns can possess many attributes that give important information on usefulness and quality. For example, how the pattern effects performance or maintainability can both be attributes of a pattern.

Attributes are used in software architecture to evaluate the quality of certain aspects of the architecture. We apply the same principles for evaluation of software patterns. The first step is to create a list of attributes by looking at related literature. This list is then reduced by performing expert interviews. This tells us which of the listed attributes are important to software architects when evaluating a pattern. A validation of the reduced list of attributes is performed by interviewing a second expert. The result of these interviews is a validated list of attributes relevant in the software pattern evaluation process.

> ***SQ2:*** *How can attributes relevant in pattern evaluation be quantified in a manner that allows for comparison?*

Rationale: Typical documentation on software patterns is qualitative in nature. Although this might be suited for documentation on patterns it does not allow for comparison. For this reason, the different attributes relevant for pattern evaluation need to be quantified. A structured method of quantification that is used for evaluations would allow for patterns to be compared on attribute level.

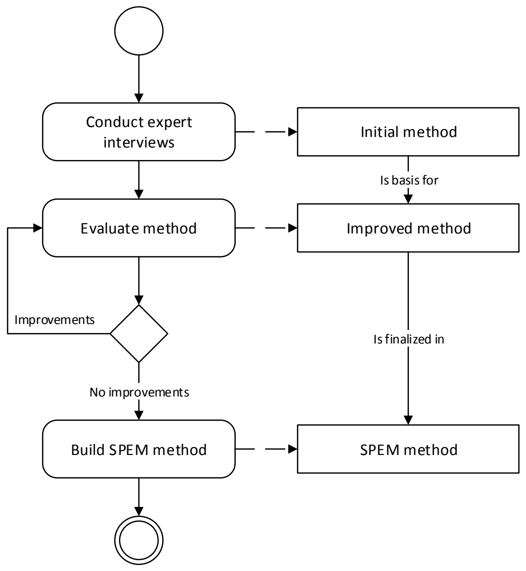


Fig. 1. Design Science Research Method

To answer this question we first look at comparable methods of quantification within the domain of software engineering. From these methods the specific characteristics are deduced. An example of these characteristics can be the ability to assign a negative value to an attribute. A method for quantification is constructed based on the list of characteristics. The method is evaluated by using it in a focus group session after which is can be incrementally improved.

A design science approach is used, which is depicted in Figure 1. An initial method is created based on an earlier exploratory study [13], extended by expert interviews. The method is evaluated in multiple cycles in which the method was put to practice in a real-world setting. Three subsequent sessions are organized in which both professional software architects and software architecture students used the method. Information system master students can be used as test subjects instead of professional software developers [14]. All sessions were recorded and an evaluation form is filled in by all participants after each session. A revised method was constructed after each session, based on the feedback, which is used as input for the next session. After three sessions no significant changes were needed any more, leading to the creation of the final method (i.e. SPEM).

IV. SPEM - Software Pattern Evaluation Method

SPEM has been constructed to evaluate software patterns in a manner which allows for comparison. There are two distinct roles:

Evaluator — Leads the evaluation process by introducing concepts and directing discussions. Is responsible for timekeeping, collecting all deliverables and noting scores.

Participant — A software architect or developer who uses his knowledge to assign scores to attributes, enters discussion, shares arguments and tries to reach consensus.

The evaluation data is gathered during a focus group session. These sessions vary in duration from one to two hours. Four to twelve participants can partake in the evaluation, excluding the evaluator. The basis of the evaluation are attributes, categorized in both quality attributes and pattern attributes. Quality attributes are used to measure the impact the pattern has on software quality and are based on ISO/IEC 25010 [15]. The following quality attributes (excluding sub-attributes) are used in SPEM:

- **Performance efficiency —** Degree to which the software product provides appropriate performance, relative to the amount of resources used, under stated conditions.
- **Compatibility —** The ability of multiple software components to exchange information or to perform their required functions while sharing the same environment.
- **Usability —** Degree to which the software product can be understood, learned, used and attractive to the user, when used under specified conditions.
- **Reliability —** Degree to which the software product can maintain a specified level of performance when used under specified conditions.
- **Security —** The protection of system items from accidental or malicious access, use, modification, destruction, or disclosure.
- **Maintainability —** Degree to which the software product can be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.
- **Portability —** Degree to which the software product can be transferred from one environment to another.

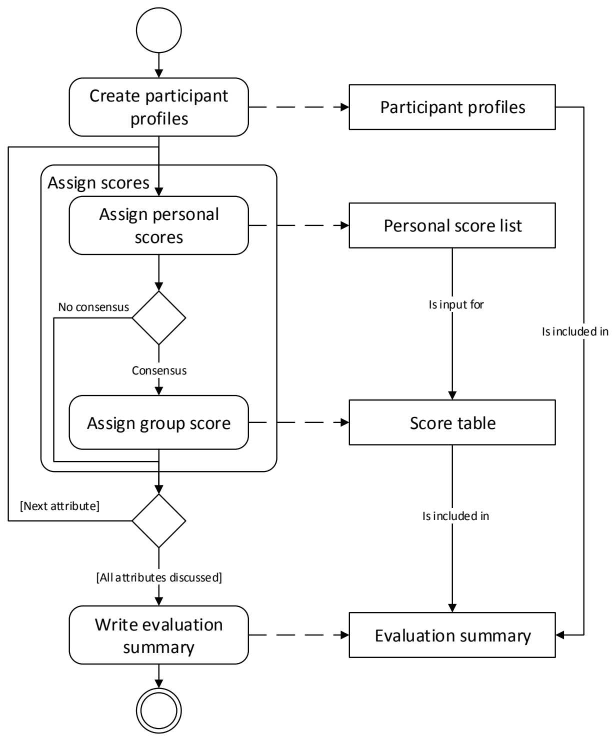
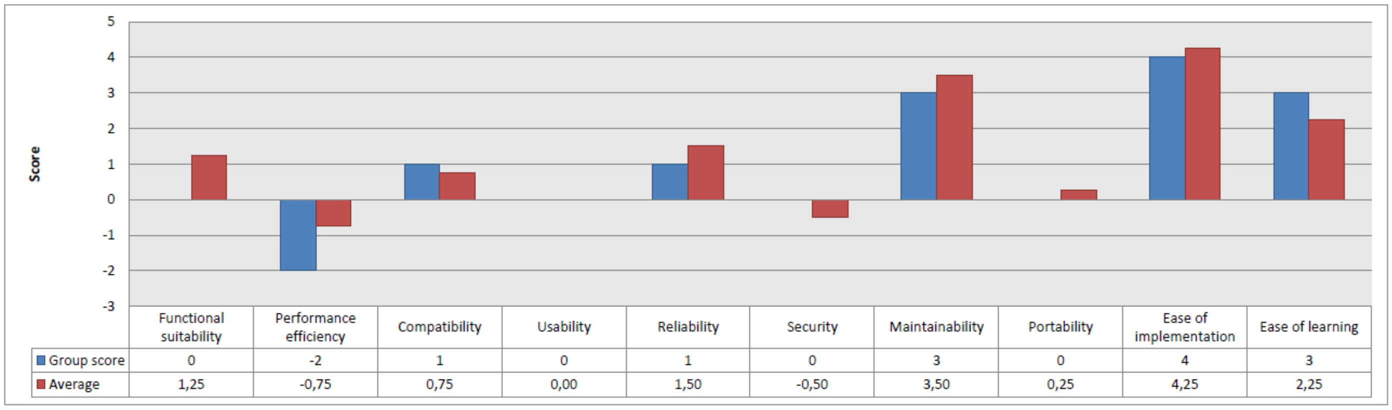


Fig. 2. SPEM: Software Pattern Evaluation Method

Pattern attributes are characteristics of the pattern itself, used to measure its learnability or ease of implementation. The goal of the evaluation is to assign a score to each attribute by all participants. The score is a relative measure based on the experience of the participant, ranging from -3 to $+3$. The score is a generalization of the software pattern, not based on a specific implementation. Experience using the pattern in a variety of situations is expressed by the score. Therefore the difference in experience among all participants is a key factor in the evaluation, which is compensated in a group score. A group score is assigned to each attribute (excluding sub-attributes) and expresses a score after a round of discussion. The discussion of each attribute allows the participants to share their knowledge with each other. The goal of the discussion is to reach consensus, meaning that after knowledge has been shared between participants with different amounts of experience, one score is assigned on which all participants agree. The result is quantitative data in the form of scores based on personal experience and the knowledge of a group, visualized in an evaluation summary (see Figure 3).

SPEM consists of four activities and four deliverables, as shown in Figure 2. The first activity focuses on creating participant profiles [16]. These profiles are forms containing fields for the participant's name, job description and years of experience. Additionally, there are input fields for the pattern name and experience with the pattern. Provided with the participant profile is a personal score list containing a list of quality attributes, sub-attributes and pattern attributes. For



	Functional suitability	Performance efficiency	Compatibility	Usability	Reliability	Security	Maintainability	Portability	Ease of implementation	Ease of learning
🟦 Group score	0	-2	1	0	1	0	3	0	4	3
🟥 Average	1,25	-0,75	0,75	0,00	1,50	-0,50	3,50	0,25	4,25	2,25

Fig. 3. SPEM evaluation summary (observer pattern)

each item on this list there is a possibility to give a personal score. The evaluator introduces the method to the participants by explaining each deliverable and the focus group session protocol. In the protocol, all activities and by who they are performed are listed and described. Thereafter, the evaluator asks the participants to fill out the participant profile.

In the second process, personal scores are assigned to an attribute. During the evaluation the scores are recorded in the personal score list. After the evaluation the personal scores are entered in the score table. The score table contains rows with all attributes used in the evaluation and columns containing all personal scores, average scores, standard deviations and group scores. The evaluator introduces an attribute by giving a short description. The participants are then asked to assign a score to the attribute and all corresponding sub-attributes.

In the next phase, a group score is assigned to an attribute and noted in the score table. The group score is a score which is produced by gaining consensus. This means all participants partake in a discussion. The focus of the discussion is to exchange arguments on the score of an attribute. If consensus is reached among all participants, the resulting group score is assigned and noted on the score table. If consensus is not reached, the group score is not assigned and no score will be noted in the score table. The evaluator initiates a discussion on the current attribute by asking a single participant's score and motivation for the score. Other participants are free to respond and exchange views, directed by the evaluator. If the discussion ends or if no time is left, the evaluator asks the participants if they have reached consensus. When consensus is reached, the group score is recorded in the score table.

When all attributes have been evaluated, an evaluation summary is created. The evaluation summary is a combination of all participant profiles and a filled out score table. Additionally a new form is added containing the name of the evaluator, date and threats to validity. This gives the evaluator the opportunity to note any occurrences that are not expressed in the main deliverables. This process is performed by the evaluator at the end of the focus group session and concludes the evaluation.

V. Method Evolution

This section discusses how the initial method evolved and shows the explicit changes made to the method based on the expert evaluation sessions.

A. Initial Method Construction

Expert interviews formed the basis of the initial version of the SPEM method. Two software architects from different companies cooperated to share their views on software pattern evaluation. Understanding which attributes are relevant in pattern evaluation and how they could be quantified was the goal of the interview. During the interview a list of quality attributes derived from ISO/IEC 9126 [17] and ISO/IEC 25010 [15] was discussed, the latter being preferred by the interviewees. Although both interviews had different results on the importance of each individual attribute of the standard, none could be excluded. Ease of learning and ease of implementation are both attributes describing characteristics of software patterns. Both these attributes should be included in software pattern evaluation as they play an important part in software pattern selection.

Scenarios are often used in software architecture evaluation, but do not fit pattern evaluation. The fact that patterns are evaluated without a specific implementation in mind makes the use of scenarios irrelevant. A software architect should interpret the results of pattern evaluation by relating it to their own project. When attributes are quantified using a score, it should be possible to assign a negative value. Patterns can affect software quality in a negative way or have negative characteristics, which a score should be able to express. The range of the scores should be between a five and ten point scale. At larger ranges it would be difficult for an architect to assign an accurate score.

When multiple architects perform a pattern evaluation, they are likely to have varying degrees of experience. Experience is key in understanding software patterns and their effect on software quality. It is important to assign a score to an attribute which takes into account the varying degrees of experience software architects have. This should be done using discussion and consensus. In a discussion, those who have more experience can share their knowledge with those who have less experience. Together working towards consensus can improve the level of knowledge of the participants and consequently improve the score. Software pattern evaluation should be performed with at least one architect who has experience using the pattern which is being evaluated. This restriction makes sure the evaluation yields a valuable result.

Based on these interview results a method was constructed incorporating the following:

- All attributes and sub-attributes from ISO/IEC 25010 [15].
- Two additional attributes; *ease of implementation* and *ease of learning*.
- Scoring ranging from -5 to $+5$.
- Discussion after each attribute.
- The goal of trying to reach consensus on each attribute.

B. Method Evaluation

Using a design science approach the initial method was evaluated and improved several iterations. A total of three focus group sessions were hosted to evaluate the method. In these sessions the method was carried out by evaluating a software pattern. Participants were asked to fill out an evaluation form at the end of the focus group session. The feedback gathered in the evaluation forms and experiences from hosting the sessions were the basis for each new iteration of the SPEM method.

During the first focus group session, four software architects participated, each having over nine years of experience in software development. During this session the observer pattern was evaluated using the initial version of SPEM. The first attribute, functional suitability and corresponding sub-attributes raised many questions. It was not possible to assign a score, as the attribute demanded a specific context. Not having a description for sub-attributes was confusing and diverted discussions to the definitions of certain sub-attributes. Based on the evaluation session, the following improvements were incorporated in the method:

- **Removal of attribute ‘Functional suitability’ —** This attribute, including its sub-attributes turned out to be irrelevant based on the focus group session. Functional suitability can only be assessed by looking at specific implementations.
- **Including a description for all sub-attributes —** A description of each attribute, including all sub-attributes was needed. This way different interpretations of attributes can be precluded.

The second focus group session was performed with twelve participating master students. The students have an information systems background and were all enrolled in the Software Architecture course, which prepared the students for the focus group session. The Access Point pattern was evaluated and participants were free to discuss attributes without any intervention from the evaluator. This resulted in lengthy discussions making the session take longer than anticipated. Discussions should be halted by the evaluator after a certain period of time based on the time that is available.

Assigning scores to sub-attributes and discussing them was time consuming. Sub-attributes needed a less prominent role in the method. It was not always possible for participants to assign a score to an attribute. Therefore it should be possible to have an explicit option stating that no score is assigned, instead of leaving it empty which might imply a neutral score. The introduction of the pattern was unclear, leading to discussion and debate. How scores should be assigned and what they represent raised questions during the session. The evaluator should focus more on explaining the meaning of scores and the difference between quality attributes and pattern attributes.

Based on the second evaluation session, the following improvements were incorporated in the method:

- **Sub-attributes removed —** Because discussion on sub-attributes took too long, they were removed from the method.
- **Added an option to give an attribute no score —** An explicit way was added for participants to indicate they do not want to give a score to a certain attribute.
- **More focus on pattern introduction —** The pattern needs to be thoroughly explained to prevent discussions.
- **More focus on explaining what the scores represent —** Scores represent the impact the evaluated pattern has on software quality or characteristics of the pattern itself. This distinction needs to be clear in order to properly assign scores.
- **Stronger role of the evaluator —** The evaluator needs to direct the discussions. Apart from initiating discussions, they should also be halted. Time keeping is the responsibility of the evaluator.

The third focus group session was performed with different students from the same group as the second focus group session. Although sub-attributes did not receive a score, they were referred to in discussions to better understand an attribute. Therefore it is important to include the sub-attributes in the method. Discussions for each sub-attribute would increase the time to complete an evaluation substantially. A personal score should be assigned to a sub-attribute while discussions and consequently group scores, should be left out.

Based on the third evaluation session, the following improvements were incorporated in the method:

- **Sub-attributes added —** Can help the understanding of attributes and provide more detail to the data.
- **Sub-attribute discussions removed —** Gives the sub-attributes a less prominent role in the method and focusses more on attributes.
- **Descriptions for each attribute / sub-attribute added to participant profile —** Allows the participants to read descriptions of attributes independent of the evaluator.

After the three sessions, the final method (i.e. SPEM) was created.

VI. SPEM Implementation

SPEM is created to evaluate software patterns in general, without a specific implementation in mind. This enables the option for comparison of software patterns, because each pattern has been evaluated as an abstract solution. It prevents unbalanced comparison between patterns based on different implementations. There is a trade-off between easy to compare generic evaluation and implementation specific evaluation. An implementation specific evaluation provides more accurate data, but it can only be compared to evaluated patterns based on the same implementation. A generic evaluation might not be as accurate, but ensures all evaluated patterns can be compared. SPEM can be used for implementation specific evaluation with few adjustments. It requires the evaluator to explain that the scores should be assigned with an implementation in mind. There needs to be an input field describing the implementation on the score table. With these adjustments an evaluation session would be identical to SPEM and allows for use of all processes and deliverables used in SPEM.

This study provides knowledge on software pattern evaluation by introducing a method to evaluate software patterns. The data SPEM evaluations provide further expands the body of knowledge on patterns. It adds retrospect to the existing software pattern documentation and provides insight on the impact patterns have on software quality. A collection of SPEM evaluation results provides valuable knowledge on the understanding of software patterns and software quality. A knowledge base would enable the disclosure of SPEM evaluation results and would allow results to be combined and compared. From an industrial perspective, a SPEM knowledge base would enable software architects to share their knowledge on software patterns. It would make knowledge available to aid in software pattern selection, leading to better decision making and overall software quality. It is through sharing knowledge that software pattern selection can reach a higher level of maturity, allowing for a structured way of comparing software patterns.

SPEM uses discussion and consensus to obtain quantitative evaluation data. This method of quantification was introduced to cope with different experience levels among participants. It has imposed a constraint on the method of data gathering used in SPEM. As discussions require interaction between participants, all participants need to be able to communicate with each other at the same time. Therefore SPEM is used in focus group sessions, limiting the number of participants. A trade-off exists between a more accurate score based on consensus with a small number of participants and a less accurate, but more reliable score with a large number of participants.

VII. Conclusion

SPEM is an objective software pattern evaluation method which can be used to compare patterns. It is used to evaluate relevant attributes of patterns based on the experience of software architects. SPEM provides quantitative data on attributes in the form of scores. The data can be interpreted and visualized to allow for software pattern comparison. This answers the main research question (MRQ).

The question "Which attributes are relevant in pattern evaluation?" (SQ1) is answered with a list of attributes, consisting of quality attributes and pattern attributes. The quality attributes are based on ISO/IEC 25010 and modified for pattern evaluation, resulting in the following set of attributes: *a*) Performance efficiency, *b*) Compatibility, *c*) Usability, *d*) Reliability, *e*) Security, *f*) Maintainability, and *g*) Portability. These attributes can be quantified in a manner that allows for comparison (SQ2) by rating the different attributes by experts in a focus group setting. It requires that personal scores ranging from -3 to +3 are assigned to all attributes and sub-attributes. A group score is assigned to all attributes after a discussion and reaching consensus. All scores are noted in the score table. A future step is to perform SPEM sessions to gather data and produce results allowing for software pattern comparisons. Gathering the output of evaluation sessions and adding them to the knowledge base can help to further validate the method and produces valuable knowledge for both academic and industrial purposes.

Acknowledgment

The authors would like to thank Leo van Houwelingen from Exact Software and Raimond Brookman from Info Support for their valuable input. We also want to thank the architect team from Info Support for their cooperation. This research is funded by the NWO 'Product-as-a-Service' project.

References

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, 2/E.* Pearson Education India, 1998.

[2] A. Jansen, J. Van Der Ven, P. Avgeriou, and D. K. Hammer, "Tool support for architectural decisions," in *The Working IEEE/IFIP Conference on Software Architecture (WICSA'07).* IEEE, 2007, pp. 4–4.

[3] M. A. Babar and I. Gorton, "A tool for managing software architecture knowledge," in *Proceedings of ICSE Workshop on Sharing and Reusing Architectural Knowledge (SHARK).* IEEE, 2007, pp. 11–17.

[4] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture," *Software, IEEE*, vol. 22, no. 2, pp. 19–27, 2005.

[5] G. Booch, "On creating a handbook of software architecture," in *Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, vol. 16, no. 20, 2005, pp. 8–8.

[6] F. Buschmann, *Pattern oriented software architecture: a system of patterns.* Ashish Raut, 1999.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, 1995.

[8] K. Beck, R. Crocker, G. Meszaros, J. Vlissides, J. O. Coplien, L. Dominick, and F. Paulisch, "Industrial experience with design patterns," in *Proceedings of the 18th international conference on Software engineering.* IEEE Computer Society, 1996, pp. 103–114.

[9] F. Buschmann, K. Henney, and D. C. Schmidt, "Past, present, and future trends in software patterns," *IEEE Software*, vol. 24, no. 4, pp. 31–37, 2007.

[10] G. Abowd, L. Bass, P. Clements, R. Kazman, and L. Northrop, "Recommended best industrial practice for software architecture evaluation," DTIC Document, Tech. Rep., 1997.

[11] M. A. Babar, L. Zhu, and R. Jeffery, "A framework for classifying and comparing software architecture evaluation methods," in *Proc. of the Australian Software Engineering Conference.* IEEE, 2004, pp. 309–318.

[12] M. Hills, P. Klint, T. Van Der Storm, and J. Vinju, "A case of visitor versus interpreter pattern," in *Objects, Models, Components, Patterns.* Springer, 2011, pp. 228–243.

[13] J. Kabbedijk, M. Galster, and S. Jansen, "Focus group report: Evaluating the consequences of applying architectural patterns," in *Proc. of the European conference on Pattern Languages of Programs (EuroPLoP)*, 2012.

[14] M. Höst, B. Regnell, and C. Wohlin, "Using students as subjectsa comparative study of students and professionals in lead-time impact assessment," *Emp. Softw. Engineering*, vol. 5, no. 3, pp. 201–214, 2000.

[15] ISO/IEC, *ISO/IEC 25010. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*, 2010.

[16] R. Donselaar and J. Kabbedijk, [Accessed: 2014-04-08]. [Online]. Available: http://www.staff.science.uu.nl/ kabbe101/PATTERNS2014

[17] ISO/IEC, *ISO/IEC 9126. Software engineering – Product quality*, 2001.

Generating Java EE 6 Application Layers and Components in JBoss Environment

Gábor Antal, Ádám Zoltán Végh, Vilmos Bilicki
Department of Software Engineering
University of Szeged
Szeged, Hungary
{antalg, azvegh, bilickiv}@inf.u-szeged.hu

***Abstract*—Nowadays, prototype-based development models are very important in software development projects, because of the tight deadlines and the need for fast development. Prototypes can be very efficient in the analysis, validation and clarification of the requirements during the development iterations. Model-driven development and code generation are frequently used techniques to support the fast development of application prototypes. There are many recurring tasks in the iterations of a prototype development process, which can be performed much faster using high level models and the automatic generation of application component implementations. The goal of this paper is to review model-driven development and code generation methods and tools, which can be used to create Java Enterprise Edition 6 applications. Besides, this paper introduces a new code generation tool, which uses the JBoss Forge framework for generating application layers and components based on the entities of the application.**

Keywords- prototype development; model-driven engineering; code generation.

I. INTRODUCTION

There are many software development projects at present where it is very hard to collect all the requirements at the beginning of the project. The customers can be uncertain or they cannot visualize the requirements in detail and precisely. Therefore, the classical waterfall development model does not work in these projects. The requirements specification and the system plan need more phases to validate and clarify, which need active communication with the customer. Therefore, prototype-based development models can be useful in these cases [1]. Using these models it is possible to show a functional application prototype to the customer to validate and clarify the requirements. But very fast prototype implementation is needed for effective prototype-based development processes. For this purpose, model-driven development and code generation methods are used to speed up the implementation. However, it is important to examine and use software design patterns to ensure the maintainability of the application.

The goal of Model-Driven Development (MDD) [2] is to simplify the system design process and increase development productivity by reusing standard software models and design patterns. The developer defines the components of the system with their relations, the structures and relations of the data to be managed, and the use cases and behaviors of the system using high level models. Based on these high level representations the model-driven development tools can automatically produce lower level system models, and in the end of the process they can generate implementations in specific programming language. Transitions between the different model levels and the generation of the implementation from the models are performed by predefined automatic transformations. The model-driven development process supports the regeneration of the implementation after modifying the system models.

A very important tool of model-driven development is code generation, which is used in the implementation phase [3]. The goal of code generation is to automatically create a runnable implementation of the system in a specific programming language based on the initial high level system models. Using this technique the production of prototypes can be accelerated significantly. The usage of different design patterns can cause a lot of recurring manual implementation tasks, which can be supported with code generation effectively. The generated source code is supported by expert knowledge, so the maintainability of the code can be higher compared to the manual implementation.

However, the existing code generation solutions have some main disadvantages and deficiencies, which make them unable to use effectively for generating Java EE 6 [4] applications. Our goal is to introduce a new code generation toolkit, which eliminates these drawbacks and generates application prototypes based on Java EE design patterns.

This paper provides insights on some methods, which can be used for effective development of Java EE (Enterprise Edition) 6 application prototypes. Section II examines some related work in the area of code generation and shows the disadvantages of the existing generator toolkits. Section III presents the JBoss Forge [5] framework, which can generate Java EE 6 application prototypes with its Forge Scaffold [6] extension. Section IV describes some practical methodologies for code generation using some tools from the JBoss Forge framework. Section V introduces a Forge-based code generation toolkit, which eliminates the defects of the Forge Scaffold and provides more application components to generate. Section VI presents a case study about the usage of our generator toolkit in a telemedicine project. Section VII concludes the paper and discusses some interesting problems that are targeted to be further studied and improved in the future.

II. RELATED WORK

During model-driven development processes the high level system models are often created using Unified Modeling Language (UML) [7], mostly based on class-, sequence-, activity- and state diagrams. The programming language of the implementation is mostly Java, C# or C++. The GenCode toolkit [8] generates Java classes from UML class diagrams, which include fields, getter and setter methods, default constructors, but it can generate only stubs for non-trivial methods. Other UML-based model-driven development tools, e.g., Modelio [9], ObjectIF [10], IBM Rational Rose [11] and Rhapsody [12], have the same drawback. Some toolkits introduced in [13][14][15] papers can generate non-trivial methods based on sequence- and activity diagrams. However, these tools need these diagrams for each non-trivial method and they cannot use generic diagram templates for generating pattern-based applications.

Some MDD tools define their own modeling languages, e.g., Acceleo [16] and ActifSource [17], but they do not have any metamodels for Java EE code generation.

A part of the source code can be also a model. A great example for this statement is the entity layer implementation of an application, which defines the data structures managed by the system. The entity implementations can be used to generate Create-Read-Update-Delete (CRUD) user interfaces for the application. Entity-based code generation tools are, e.g., JBoss Forge, seam-gen [18], MyEclipse for Spring [19], and OpenXava [20]. In the case of these toolkits, the generated user interfaces are in some cases incomplete, inaccurate, or difficult to use, e.g., inheritance, association between entities.

III. THE JBOSS FORGE FRAMEWORK

One of the most important entity-based code generation toolkits is the JBoss Forge framework, which is developed by the JBoss Community. The main goal of this tool is to manage the lifecycle and configuration of Maven projects [21] and to support the automatic generation of some application components with some simple commands. Forge has a command line interface, which includes most of the important Linux file manager commands. These commands are extended to read and modify the structure of Java source files. Besides, there are commands implemented for creating and configuring projects, and creating Java Persistence Application Programming Interface (JPA) entities. The Forge commands can be run batched in a script to make the use of the framework easier.

There are many available and downloadable plug-ins for Forge, extending its functionality. It is also possible to develop new plug-ins to support new code generation capabilities. A very useful plug-in included by the framework is Forge Scaffold, which can be used to generate basic Java EE 6 prototypes with CRUD web user interfaces based on the JPA entities of the application. The Scaffold plug-in can be extended with several scaffold providers, which can be used to specify the UI component library for generating user interface layer, e.g., standard Java Server Faces (JSF), RichFaces, Primefaces. Unfortunately, the Forge Scaffold plug-in has some defects.

- If the entity layer contains inheritance between entities, the user interfaces of the child class do not contain the fields of the super class.
- If a field has a @ElementCollection annotation, the edit page does not contain any component for selecting/adding elements to the field.
- In case of associations between entities, the edit page contains a dropdown menu for selecting/adding elements with the return value of the toString() method of the related entity instances. It is not possible to select a field as label.
- List pages do not provide ordering and filtering by multiple criterions.
- The generated code is difficult to maintain. The business logic is generated into one Bean class per entity. There is no separate Data Access Object (DAO) layer.

IV. CODE GENERATION METHODOLOGIES

The Forge Scaffold plug-in uses the combination of two main methodologies for generating source code: template-based and procedural code generation. The tools for these code generation methods are provided by the JBoss Forge framework, so new code generator plug-ins can be developed using these tools.

The concept of template-based code generation methodology is to produce source code based on predefined templates, which will be rendered depending on the generation context. These templates consist of two types of parts: static parts, which will be generated into every source code instances; and dynamic parts, which will be replaced depending on the different inputs of the generation process. For supporting template-based code generation, JBoss Community provides the Seam Render tool, which is also used by Forge Scaffold plug-in [22]. The syntax of Seam Render code templates is quite simple. The static code parts should be specified the same way as they should appear in the generated code instances. The dynamic code parts should be indicated by expressions with named objects given in the form @{indicatorExpr}. The template-based code generation process consists of three phases:

1. Compiling the template: the specified template is processed and the dynamic template parts are collected with their object names and positions.
2. Defining the rendering context: every object names in indicator expressions are mapped to a Java object (mostly a String) for replacing every occurrence of the proper object name with the value of the object.
3. Rendering the template with the given context.

This method can be efficiently used in cases when the source code to generate contains small dynamic parts, which can be rendered by simple replacements, e.g., DAO classes, Beans for business logic. But, there can also be cases when the source code to generate contains "highly dynamic" parts, which can be rendered using very complex replacements (e.g., user interfaces) or when a given existing source code file must be modified in a later phase of the generation process. In these cases the procedural code generation methodology can be useful.

The procedural code generation aims the production of code elements using libraries for parsing and representing the elements of the specified language. For generating Java source code procedurally, Forge provides the Forge Java Parser library, which can parse Java source code to its own high level representation, create new code elements (classes, interfaces, fields, methods, etc.) or modify existing elements in the source code [23]. For generating XHTML (eXtensible Hypertext Markup Language) elements of the web user interface, the Metawidget framework can be useful, which contains the high level representation of standard XHTML tags, and JSF components [24]. Using these tools, new generator components can be developed to produce highly dynamic code parts and extend existing source code with new elements.

V. FORGE-BASED CODE GENERATION TOOLKIT

For eliminating the defects of Forge Scaffold and extending its functionality with the generation of other application components, we aimed the development of a new code generation toolkit based on the JBoss Forge framework. We analyzed some of our software development projects to find application layers and components, which contain a large amount of repetitive work that can be replaced with code generation methods to accelerate the development process. We found nine general areas that can be effectively supported with code generation. We examined these areas and their design patterns, collected the knowledge and experience we have in relation to the development of these components and implemented a code generation toolkit for supporting the effective production of them. In the following subsections we discuss these areas in detail.

A. *Entity and Validation Management*

The entity generator of the Forge framework is a very useful tool but it has some defects:

- It is not possible to generate inheritance between entities.
- It cannot generate enum types with specific values.
- It is not possible to generate custom constructors, hashCode, equals and toString methods based on specific fields.

We implemented these capabilities to extend the functionality of the entity generator. Besides, Forge cannot generate validation annotations to the entity fields, so we extended our toolkit with the possibility to add annotations of Bean Validation API (Application Programming Interface) to the entities. The generated entities use the Composite Entity design pattern to implement associations between the entities.

B. *DAO Layer*

The DAO layer of the application is responsible for the operations, which can be made on the entities of the system: inserting, updating, deleting, querying. Using JPA-based EntityManagers it is easy to generalize the DAO layer based on the Data Access Object design pattern. We created a generic DAO superclass for providing simple EntityManager operations and the entity-dependent DAO child classes extend this superclass to provide queries and entity-specific operations. Therefore, only the child classes should be generated depending on the entity based on a DAO template. We used Hibernate as JPA provider in our generator toolkit.

C. *Business Logic Layer*

The business logic layer of an application is responsible for establishing connection between the user interface and the entities (using the DAO layer), and managing complex business processes and transactions. There are two main components in general, which are used by CRUD user interfaces:

- Data Models: provides the listing of specific type of entities with paging, ordering and filtering. They use the Session Façade and Value List Handler design patterns.
- Entity Action Beans: provides editing, viewing and deleting of a selected entity instance. They use the Session Façade design pattern.

These business logic components are also easy to generalize using generic superclasses. Only the entity-specific Data Models and Entity Action Beans should be generated depending on the entity based on code templates. We used Contexts and Dependency Injection (CDI) for context management and dependency injection, and Enterprise JavaBeans (EJB) 3 for transaction management in our generator toolkit.

D. *CRUD User Interfaces*

The CRUD user interfaces provide simple listing, creating, editing, viewing and deleting capabilities for the application user. Simple UI components (text fields, labels, dropdown menus, etc.) can be inserted using JSF 2 components. Complex UI components (list items, fields with labels, complex selector components, etc.) can be efficiently generalized using JSF 2 composite components, which can simplify the generation of complex user interfaces. These components use the Composite View design pattern. We also eliminated the defects of Forge Scaffold in our user interface generator. The generation of user interface components is performed by procedural code generation and the result is inserted into predefined XHTML page templates.

An example list user interface generated by our toolkit is shown on Figure 1. The layout and content of list items are generated using the structure of the entity classes.

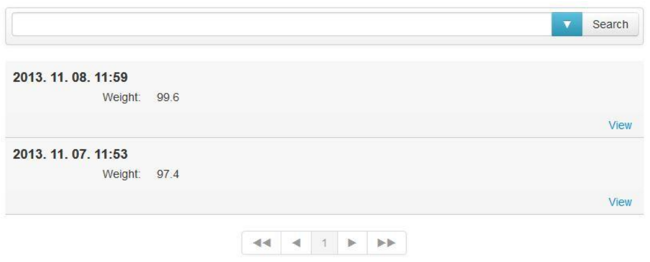


Figure 1. List user interface generated by our toolkit

E. Authentication, Authorization, Auditing

The system needs authentication and authorization modules to provide appropriate data security. Users need to be authenticated before using the system and it is very important to check the permissions of the actual user before data access or modification. These modules can be the same in most of the applications. We use the PicketLink [25] security and identity management framework in our generated security module, which provides a general entity layer pattern and JPA-based solutions for identity and role management. In our security module, we also implemented PicketLink-based registration, password change and password reminder components, which can be automatically generated for the given application.

The logging of the operations made by application users is also a very important data security aspect, which is called data auditing. Our code generation toolkit can produce an auditing module for the application, which is capable to log every data operations with the name of the performer user and the actual timestamp. The auditing module uses Hibernate Envers [26] in a combination with our auditing solutions.

F. REST Interfaces

Mobile client applications often connect with a server application for data querying or uploading. Therefore, both the client and the server application need interfaces to communicate with each other. The most frequently used methodology of client-server communication is the Representational State Transfer (REST) architecture. For transferring data between the client and the server, the JavaScript Object Notation (JSON) and eXtensible Markup Language (XML) formats are often used. Our toolkit can generate common source files for REST-based communication (both for client and server) and REST service stubs for server application based on DAO methods. Data objects transferred between server and clients are implemented using the Data Transfer Object (DTO) design pattern. The generated service implementation includes DTO classes based on entities and converters between entities and DTO classes. Our solution uses the RESTEasy library for REST implementation and Jackson library for JSON implementation.

G. Data Visualization on Charts

In server applications there can be a lot of numerical data (e.g., measurement data, quantities), which should be visualized on charts to analyze their changes. Our toolkit is capable to generate chart visualization user interfaces for specified numerical data over the changes of a specified entity field (e.g., measurement date). A generated example chart user interface is shown on Figure 2. We use the Flot [27] JavaScript-based library for visualizing charts, FlotJF [28] library for representing Flot objects in Java, and RESTEasy for the REST services, which provides data for charts.

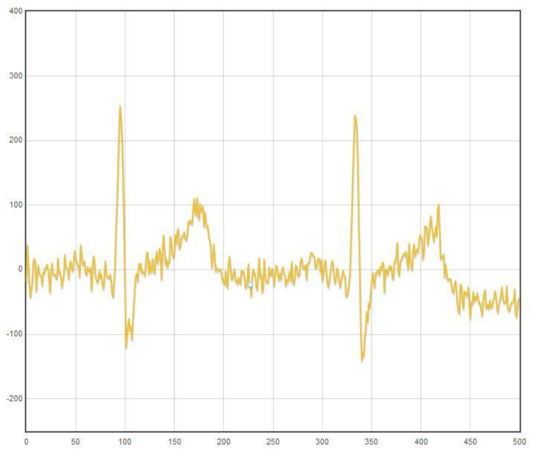


Figure 2. Chart user interface generated by our toolkit

H. Demo Data Generators

During the development the system needs a lot of demo data for testing. Demo data can also make presentation of the application to the customer easier. Therefore, demo data generator components are needed in the system, which can be very time-consuming to implement in case of large domain models. Our toolkit can generate demo data generators for the selected entities in the application. The constraints of entity fields are also taken into consideration during the generation. It is also possible to configure the number of entities to be generated before the production of data generators.

I. Dashboards

Many applications may need special, customizable user interfaces, which can show only the most important information for users, with simple and clear visualization elements. Our code generator toolkit can produce dashboard user interfaces with customizable layout (tiles, which can be moved or hidden) and content (charts, gauges, switches, status indicators, etc.). A generated example dashboard is shown on Figure 3. The dashboard is created using Gridster jQuery-based library [29].

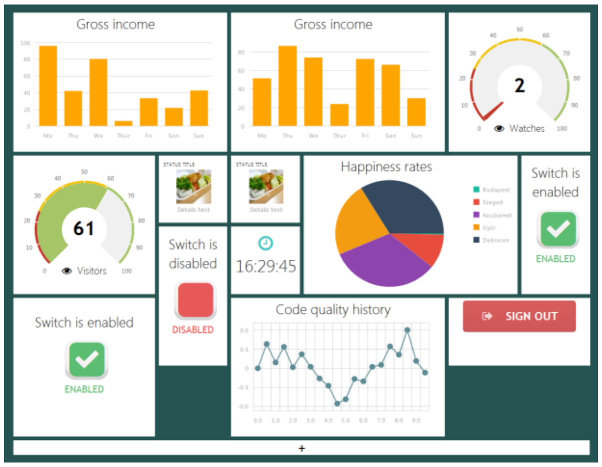


Figure 3. Dashboard user interface generated by our toolkit

VI. Case Study: Development of a Telemedicine Application

We used our code generation toolkit successfully in a telemedicine project where we had to develop a data collector application for supporting the long distance monitoring of patients after heart surgery. The task of server application was the collection, storage, and the visualization of the incoming data from different sensors, e.g., weight scales, ECG sensors, blood pressure monitors.

When planning the mentioned application we analysed the components to be developed and the results showed that a significant part of the development can be supported with code generating methods. Relying on this, the prototype development of the patient monitoring system with our code generating tool was started. The following components of the application were generated:

- DAO (Data Access Object) and business level layer.
- Listing, editing, visualizing user interfaces.
- Charts for the visualization of ECG measurements and for the representation of the temporal changes of blood pressure and weight.
- Data uploading REST service stubs on the server side and REST client stubs in the mobile client application.

Above this, only the refinement of user interfaces and the implementation of REST interfaces were required. Furthermore, for integrating sensors, Android mobile prototypes were developed, whose task was to receive and process the data sent by a single channel ECG sensor, a blood pressure monitor, and a weight scale.

We measured the time requirements at this application with our productivity measurement plug-in for Eclipse IDE, and compared them with the time needed by developing this application manually [30][31]. The measurement results can be examined in Table I. t_m means the manual implementation time (in person days), and t_{cg} means the implementation time with code generation support (in person days). The application layers needed about 80% less time in average to implement with code generation support, compared to the manual development.

TABLE I. COMPARISON OF DEVELOPMENT TIME IN MANUAL- AND CODE GENERATION SUPPORTED DEVELOPMENT

Layer	t_m	t_{cg}	t_{cg}/t_m (%)	Total time spent (%)
Domain model	0.1	0.02	20	5
DAO layer	3	1	33	10
CRUD user interfaces	10	0.2	2	50
Authentication, authorization, auditing	20	0.2	1	15
Data visualization	3	1	33	10
System integration interfaces	3	1	33	10

VII. Conclusion and Future Work

This paper reviewed model-driven development and code generation methods and tools for producing Java EE 6 applications, and introduced a code generation toolkit based on the JBoss Forge framework, which can generate many application components based on Java EE design patterns to accelerate the prototype development process. We also used this toolkit in the development of a telemedicine application and measured the efficiency of the code generation supported development compared to the manual development. The measurement results show that the development process with code generation needed significantly less time than the manual development.

In the future, we would like to continue the development of our generator toolkit and extend its functionality. We identified the following goals, which we would like to reach in the further development of the toolkit.

- Now, the generator works based on Forge 1.4.3. We would like to upgrade our toolkit to Forge 2.
- During the test phase of the development, unit tests provide the correctness of small, low level system units. The development of correctly functioning system units can be more effective with implementing unit tests for given units. We would like to extend our toolkit with the capability to generate unit tests for selected classes.
- An entity-based code generator toolkit can be easier to use if it can process UML class diagrams, which contain the entities of the application. These UML diagrams should be extended with some additional properties, such as JPA and Bean Validation annotations for entity classes and fields. Using these diagrams the Java implementation of entities can be automatically generated. We would like to develop an extension for our code generator toolkit to process extended UML class diagrams of entities as the input of application prototype generation.
- The main goal of our toolkit is to generate web-based server application components. In the future we would like to identify some areas in Android mobile application development, which can be supported with code generation and implement a generator toolkit for Android applications.
- Many entity fields can have special meanings that imply special criterions during the demo data generation and special validation processes. For example, an entity field with String type can be a first name of a person, an address or a country name. This semantic aspect should be taken into consideration during the data generation and validation of values in entity fields. Our goal is to provide an opportunity to add semantic annotations to entity fields, which are related to ontology definitions of special data structures, and to generate special data generators and validators for the annotated fields.

ACKNOWLEDGMENT

The publication is supported by the European Union and co-funded by the European Social Fund.

Project title: "Telemedicine-focused research activities on the field of Mathematics, Informatics and Medical sciences"

Project number: TÁMOP-4.2.2.A-11/1/KONV-2012-0073

REFERENCES

[1] M. F. Smith, Software prototyping: adoption, practice, and management. McGraw-Hill, 1991.

[2] B. Sami, Model-Driven Software Development. Springer, 2005.

[3] J. Herrington, Code Generation in Action. Manning Publications Co., 2003.

[4] http://docs.oracle.com/javaee/, [retrieved: 2014.04.11].

[5] http://forge.jboss.org/, [retrieved: 2014.04.11].

[6] http://forge.jboss.org/docs/important_plugins/ui-scaffolding.html, [retrieved: 2014.04.11].

[7] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Guide, Second Edition, Addison-Wesley Professional, 2004.

[8] A. G. Parada, E. Siegert, and L. B. de Brisolara, "GenCode: A tool for generation of Java code from UML class models", 26th South Symposium on Microelectronics (SIM 2011), Novo Hamburgo, Brazil, pp. 173-176.

[9] http://www.modeliosoft.com/, [retrieved: 2014.04.11].

[10] http://www.microtool.de/objectif/en/, [retrieved: 2014.04.11].

[11] http://www-03.ibm.com/software/products/us/en/ratirosefami/, [retrieved: 2014.04.11].

[12] http://www-03.ibm.com/software/products/us/en/ratirhapfami/, [retrieved: 2014.04.11].

[13] A. G. Parada, E. Siegert, and L. B. de Brisolara, "Generating Java code from UML Class and Sequence Diagrams", Computing System Engineering (SBESC), 2011 Brazilian Symposium, Florianopolis, Brazil, pp. 99-101.

[14] M. Usman and A. Nadeem, "Automatic Generation of Java Code from UML Diagrams using UJECTOR", in International Journal of Software Engineering and Its Applications vol. 3, no. 2, 2009, pp. 21-38.

[15] E. B. Oma, B. Brahim, and G. Taoufiq, "Automatic code generation by model transformation from sequence diagram of system's internal behavior", in International Journal of Computer and Information Technology, vol. 1, issue 2, 2012, pp. 129-146.

[16] http://www.eclipse.org/acceleo/, [retrieved: 2014.04.11].

[17] http://www.actifsource.com/, [retrieved: 2014.04.11].

[18] http://docs.jboss.org/seam/2.3.1.Final/reference/html/gettingstarted.html, [retrieved: 2014.04.11].

[19] http://www.myeclipseide.com/me4s/, [retrieved: 2014.04.11].

[20] http://openxava.org/home, [retrieved: 2014.04.11].

[21] http://maven.apache.org/, [retrieved: 2014.04.11].

[22] https://github.com/seam/render, [retrieved: 2014.04.11].

[23] https://github.com/forge/java-parser, [retrieved: 2014.04.11].

[24] http://metawidget.org/, [retrieved: 2014.04.11].

[25] http://picketlink.org/, [retrieved: 2014.04.11].

[26] http://envers.jboss.org/, [retrieved: 2014.04.11].

[27] http://www.flotcharts.org/, [retrieved: 2014.04.11].

[28] https://github.com/dunse/FlotJF, [retrieved: 2014.04.11].

[29] http://gridster.net/, [retrieved: 2014.04.11].

[30] G. Kakuja-Toth, A. Z. Vegh, A. Beszedes, and T. Gyimothy, "Adding process metrics to enhance modification complexity prediction", Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension (ICPC 2011). Kingston (ON), pp. 201-204.

[31] G. Kakuja-Toth, A. Z. Vegh, A. Beszedes, L. Schrettner, T. Gergely, and T. Gyimothy, "Adjusting effort estimation using micro-productivity profiles", 12th Symposium on Programming Languages and Software Tools. SPLST'11. Tallinn, Estonia, 5-7 October 2011, pp. 207-218.

Refinement Patterns for an Incremental Construction of Class Diagrams

Boulbaba Ben Ammar* and Mohamed Tahar Bhiri+
Faculty of Sciences of Sfax, Sfax University, Sfax, Tunisia
*Boulbaba.Benammar@fss.rnu.tn
+Tahar_Bhiri@yahoo.fr

***Abstract*—Specifying complex systems is a difficult task, which cannot be done in one step. In the framework of formal methods, refinement is a key feature to incrementally develop more and more detailed models, preserving correctness in each step. Our objective is an incremental development, using the technique of refinement with proof for UML specifications. Indeed, UML suffers from two major weaknesses, namely, it is not based on a simple and rigorous mathematical foundation and it does not support the concept of refinement with proof of correction. To achieve this, we advocate a development framework combining the semi-formal features of UML/OCL and the formal one from B method. We chose the B formal language in order to benefit from existing work done on coupling between UML and B. In addition, we propose and formalize in B the refinement patterns that promote incremental development with proof of UML/OCL class diagrams. We illustrate our purpose by the description of some development steps of an access control system.**

Keywords-UML; OCL; refinement pattern; class diagram

I. INTRODUCTION

Refinement is a process to transform an abstract specification into a concrete one [17]. It aims to develop systems incrementally, which are correct by construction [18]. Refinement is defined in a rigorous way in various formal languages, such as B [18], Event-B [17], Communicating Sequential Processes (CSP) [8], Z and Object-Z [14]. The Unified Modeling Language (UML) [19] is an object-oriented modeling language widely used. It is a de-facto standard, allowing graphical visualization of models facilitating communication inter-actors. But, it does not support the concept of refinement. It has a dependency relationship stereotyped «refine» to connect a client (or refined concrete element) to a provider (abstract element). This relationship is subject to several interpretations [15] and does not provide methodological assistance related to how to refine existing UML models. In addition, UML does not allow the verification of a refinement relationship between two models. In a formal language, such as B, Event-B, CSP, Z and Object-Z, although the refinement relationship is well-defined and well-supported (generation of proof obligations, interactive prover, model checker and animator), an experienced designer finds more or less important difficulties in identifying the different levels of abstraction (an “optimal” refinement strategy) for carrying out the process of refinement. Solutions based on the concept of pattern --like the design patterns in Object-Oriented (OO) applications-- to guide the designer during the refinement process begin to appear covering both the horizontal refinement for application areas as reactive systems [17][26][27] and the vertical refinement under B. The horizontal refinement [17] consists in introducing new details in an existing model. Each introduction of details to a model leads to a new model, which must be coherent with the previous one. If it is the case, we said that the second model refines the first one. Horizontal refinement aims at the progressive acquisition, by successive refinement, of a coherent model from abstract specifications, leading to the abstract formal specification of future software or system.

The vertical refinement [17] consists in going from an abstract model to a more concrete one, for example by reducing the non-determinism. The concrete model is a realization of the abstract model. For example, the B Automatic Refinement Tool (BART) [1] tool associated with B offers refinement rules that can be used in the final stages of vertical refinement phase of a formal process development. B and Event-B do not distinguish between horizontal and vertical refinement. In fact, both refinements use two types of refinement allowed by B, i.e., data refinement and algorithms or control refinement [18].

In the following, we briefly present in Section 2 the existing approach for construction of class diagrams. The proposed approach is presented in Section 3. In Section 4, we present our catalog of refinement patterns used in an incremental specification development process. In Section 5, we illustrate the use of the proposed approach using an access control case study. Section 6 concludes this paper and proposes perspectives.

II. RELATED WORK

UML is a graphical modeling language reference, offering an important range of diagrams. Class diagram, which can express the static aspects of a system, are one of the most used diagrams. At the "heart" of the object modeling, it shows the classes and interfaces of a system and the various relationships between them. Approaches to construction and verification of class diagrams have been highlighted in several studies [4][7][10][13][20][21].

The decomposition unit of object-oriented systems is the concept of class. A coarse characterization of classes makes a distinction between the analysis classes belonging to the space of problems (external world in modeling course) and design classes and implementation belonging to the space of solutions.

Methodological works supporting the identification of the useful and relevant classes were completed. A method known as “Underline the names in the document of the requirements” is proposed in [13]. The results of the application of this method are very sensitive to the used style. This can lead designers to omit useful classes while introducing classes, which are not justified.

Class, Responsibility, Collaboration (CRC) cards [20] are paper cards on which the designers evoke the potential classes according to their responsibilities and in the way in which they communicate. This technique promotes interaction within teams but its contribution to the identification of quality classes is uncertain.

Meyer [7] provides the general heuristics for classes discovery, based on the theory of Abstract Data Types (ADT). He defines different uses of inheritance justifying their uses.

The Business Object Notation (BON) method [21] introduces useful advices to identify the classes. It proposes two structural concepts (cluster and class), two strong conceptual relations (customer and inheritance) and a simple language of assertions expressing the semantic properties attached to the modeled elements.

Many analysts and designers use only the class diagrams. Others use development process allowing scheduling of several types of UML diagrams. Some development process with UML adopts an approach based on use case diagrams in order to draw class diagrams [28].

The design classes represent the architectural abstractions, which facilitate the production of elegant and extensible software structures. Design patterns, including those of Gang of Four (GoF) [10] promote the identification of these classes.

Semi-formal graphical notations (such as UML) are generally intuitive, but do not allow rigorous reasoning. On contrary, formal notations (such as B) provide mathematical proofs, but are not easy to understand. Several studies coupling between semi-formal and formal notations exist. Among these works, we studied profitably those linking UML to B [16]. Works of coupling between UML and B go to the combination of UML and B in a new language named UML-B [9].

By using the technique of refinement, the approach described by Ben Ammar et al. [4] allows of a UML/OCL class diagram showing all the formal properties of the future system. The obtained class diagram, containing the analysis classes, represents a coherent abstract model of the future system. Such a model can be concretized (identification of design and implementation classes) by applying the technique of refinement.

III. APPROACH TO DESIGN A SYSTEM IN A STEP-WISE MANNER

Our approach consists on the proposal of a catalogue of refinement patterns (see Section IV) for incremental development of UML class diagrams. These patterns are built to solve recurrent problems in the development of the static part of an OO application, such as: introduction of an intermediate class [3], reification of an attribute, an enrichment of an association, decomposition of an aggregate and the introduction of a new entity. These patterns are characterized by a precise framework composed of six parts showing the fundamental aspects of a refinement pattern. These parts are Intention, Motivation, Solution, Verification, Example and See also. In addition, the proposed refinement patterns are formalized into B specifications, using systematic rules of translation of UML into B [11][16]. This helps to identify precisely the conditions of applicability, the evolution of a UML class diagram and correctness of the refinement relationship. Such B formalization can be reused with advantage when instantiating these patterns by the designer. Thus, in a joint development UML/B, the designer selects and applies a refinement pattern on his abstract specification. Then, he obtains a new specification, which includes new properties related to the application of the refinement pattern. Verification of the correctness of the refinement relation between two specifications is entrusted to Atelier B tool [30].

In the following, we detail a new approach used for development of UML class diagrams guided by refinement patterns. Such development process allows establishing a UML class diagrams, which models the key concepts of the application and has properties considered to be coherent covering the constraints of the application resulting from its specifications. The process advocated has four steps: Rewriting of requirements, Refinement strategy, Abstract specification, and Refinement steps.

A. Rewriting of requirements

Currently, the specifications are often of poor quality. Abrial [27] criticize these specifications to be directed too towards a solution and to present mechanisms of realization to the detriment of the explicitness of the properties of the system to be conceived. We recommend to rewrite the specifications in order to put forward the properties of the future system and to facilitate the development of a suitable strategy of refinement. For that purpose, we use the recommendations of Abrial [17][27] for distinguishing the functional safety and liveness properties.

B. Refinement strategy

Rewriting of requirements facilitates the development of an adequate strategy of refinement. But, this does not guarantee obtaining an “optimal” strategy of refinement. Work making it possible to compare alternative strategies of refinement for a given scope of application, in case of the reactive systems, starts to appear [17][27].

C. Abstract specification

This stage aims to establish an abstract UML/OCL model described by a class diagram based on the refinement strategy previously defined. The UML/OCL class diagram product is translated into B in order to formally verify its coherence.

D. Refinement steps

The refinement process involves several steps. Each refinement step takes as inputs three parameters: the class diagram of level i, the proposed catalog of refinement patterns and the properties resulting from the specifications to be taken into account and produce as output the class diagram of level i+1 (see Figure 1).

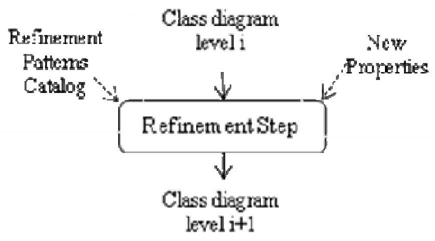


Figure 1. Step of refinement

The consideration of the properties, which guide the process of refinement, can be realized by applying refinement patterns. The formal verification of the correctness of the refinement step is entrusted to the AtelierB tool through the translation of two class diagrams in two B levels. The gluing invariant in B models making a link between these two levels (abstract and refined) can be established by reusing the B formalization of proposed refinement patterns. The refinement process terminates when all the explicit properties in the specification are taken into account in accordance with the adopted refinement strategy. Thus, ultimate UML/OCL class diagram obtained models the key concepts of the system to achieve. In addition, it contains the essential properties deemed formally consistent.

IV. Refinement Patterns

Unlike architecture patterns [12], analysis patterns [22] and design patterns [10] a refinement pattern, has a dynamic character. Applied to a model of level i, a refinement pattern produces a model of level i+1. Recently, refinement patterns begin to appear for formalisms, such as Event-B [26], KAOS [2] and B [1]. In [3], we offer refinement patterns to solve recurring problems in incremental development of the static part of an OO application using UML/OCL. A refinement pattern has two parts: Specification (1) and Refinement (2). A specification describes the UML/OCL class diagram of level i. A refinement describes the UML/OCL class diagram of level i+1 produced by applying the corresponding refinement pattern on the model of level i. The proposed refinement patterns, presented later, are described in the same framework including six parts: Intention, Motivation, Solution, Verification, Example and See Also.

In the following, we detail the patterns only by the Intention, Motivation, Solution and See also.

A. *Pattern 1: Class_Helper*

1) Intention

It allows introducing a class Class_Helper between two classes considered important with respect to the refinement step considered. The direct relationship between the two major classes is refined by a path connecting these two classes through the intermediate class introduced.

2) Motivation

UML class diagram consists of four types of inter-class relationships: generalization (or inheritance), association, aggregation and dependence. In an incremental OO modeling, it is advantageous to start with abstract inter-class relationships. This subsequently facilitates the introduction of details via intermediate classes to refine these abstract relations.

3) Solution

(1)

(2)

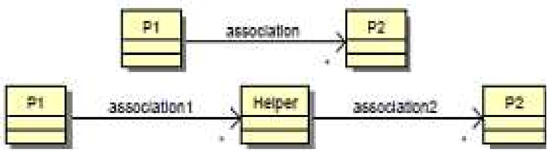


Figure 2. Class_Helper specification

4) See also

Class_Helper [3] the pattern depends on the nature of the relationship between two important classes P1 and P2: generalization, association, aggregation, composition and dependency. In addition, the intermediate class introduced Helper can be connected to P1 and P2 using the same kind of relationship or two relations of different nature. Class_Helper the pattern can be applied in reverse order of the concrete to the abstract. This process of abstraction - as opposed to refinement - can be profitably used in an activity of reverse engineering.

B. *Pattern 2: Class_Attribute*

1) Intention

When a class has an attribute modeling a concept considered interesting and with well-defined operations, this pattern allows to reify this attribute in a new class called Class_Attribute. An aggregation relationship is introduced between the enclosing class --aggregate-- and the class reifying the concerned attribute --component--.

2) Motivation

For reasons of simplification, at a high level of abstraction, a concept can be modeled as an attribute. Then, according to details from the specifications, the same concept can be retained as a class. This is justified by the identification of well-defined operations applicable on this concept by analyzing the introduced details. The type of the attribute is rather discrete: integer, enumerated or alphanumeric.

3) Solution

(1)

(2)

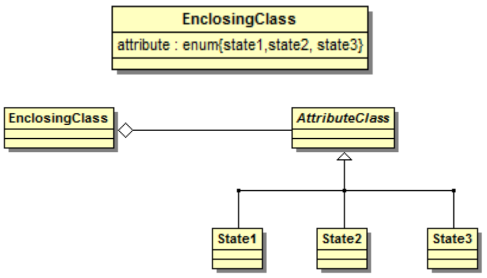


Figure 3. Class_Attribute specification

4) See also

The idea of reification of an attribute may be used with advantage in an activity of restructuring (or refactoring) of an existing OO models. Moreover, in [5], we proposed a refactoring schema based on the reification of an attribute: introduction of the concept of delegation.

C. Pattern 3: Class_Decomposition

1) Intention

It allows detailing the responsibilities of an original class by introducing new classes. The original class and the resulting classes are connected by relations of generalization (inheritance). The number of the resulting classes is at least equal to one. This pattern favors a top-down modeling. The generalization covers mainly the following two cases [7]:

- Heritage subtype: You are an external model system in which a class of objects (external) can be decomposed into disjoint sub categories. We urge that the parent, A, be deferred so that it describes a set of objects not fully specified. The heir B can be effective or delayed.
- Restriction inheritance: Inheritance of restriction applies if the instances of B are among the instances of A, those that satisfy a constraint expressed in the invariant B and absent from the invariant A. A and B should both be deferred or both effective.

2) Motivation

In a top-down modeling approach, it is advantageous to start with a minimum number of classes. Sometimes we think that factoring operations can cause problems with implementation.

3) Solution

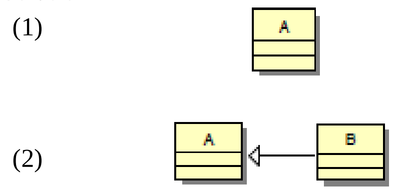


Figure 4. Class_Decomposition specification

4) See also

The pattern Class_Decomposition introduces the idea of a decomposition of a class via inheritance relationship. Both UML relationships: aggregation and composition can be used for the decomposition of an aggregate entity modeled by UML class.

D. Pattern 4: Class_NewEntity

1) Intention

It allows the introduction of UML class, which models a separate entity. The introduced class is related to other classes from abstract level through association relationships.

2) Motivation

In an incremental OO modeling, it is advantageous to start with a minimum number of entities called very abstract entities. This further promotes the introduction of details through less abstract or concrete entities, called (e.g., equipment) to go from the abstract world to the concrete world.

3) Solution

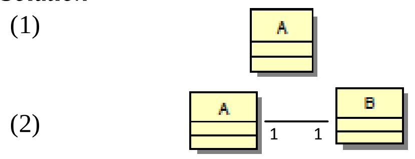


Figure 5. Class_NewEntity specification

4) See also

On the form, the two patterns Class_Helper and Class_NewEntity produce similar effects. But onthe content, they differ. In fact, they have two different gluing invariants. In addition, the pattern Class_NewEntity is oriented towards the horizontal refinement (specification stage) encouraging the construction step-by-step of a business model of the application, while the pattern Class_Helper is oriented towards the vertical refinement (design stage) promoting the gradual construction of a conceptual model of the application.

E. Pattern 5: Refinement_Operation

1) Intention

This pattern provides a passage from an abstract specification of operation into a more concrete one. It is inspired by formal development practices used in B method.

2) Motivation

B method allows several types of refinement: data refinement, control refinement and algorithmic refinement. In control refinement, the following facts are observed:

- the operation to be refined retains the same signature,
- its precondition can be strengthened,
- the nondeterministic behavior, described by substitutions, can be reduced.

In UML framework, we can describe the control refinement using Object Constraint Language (OCL) notations for presenting both abstract and concrete specification of operation to be refined.

3) Solution

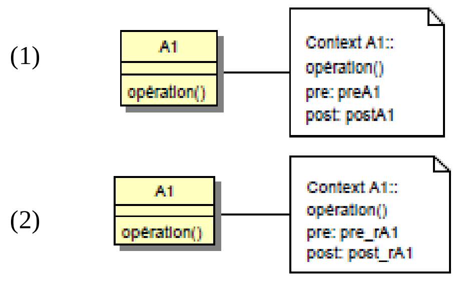


Figure 6. Refinement_operation specification

4) See also

The pattern Refinement_Operation introduced the idea of control refinement. In the same way, we can define a pattern of data refinement. This allows the introduction of concrete variables (data). In this case, a gluing invariant, which links abstract and concrete variables, should be explained. Both control and data refinement are not mutually exclusive; they can be operated in the same refinement step. It is obvious that the pattern Refinement_Operation can be applied combining these two types of refinement.

F. Pattern 6: Class_Abstraction

1) Intention

The pattern Class_Abstraction introduced software qualities, such as efficiency, reusability and scalability in a software development guided by successive refinements. Thus, it allows factoring common properties - attributes, operations and relationships - of some classes within a founding class.

2) Motivation

In the development process, each entity is modeled by a class. But often, classes that are, in fact, variations of the same concept are encountered. Several classes of a class diagram have common characteristics. It is said that these independent classes can be derived from a common ancestor.

The idea is to improve the modeling, a better representation, facilitate data storage and thus avoiding redundant features. For that, we can factor these common features between the different classes into a new founding class.

3) Solution

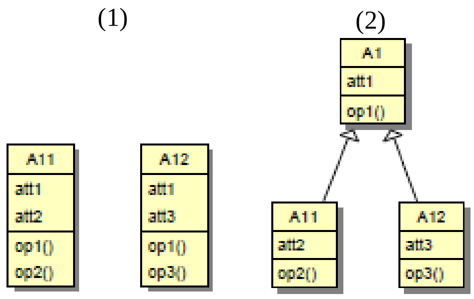


Figure 7. Class_Abstraction specification

4) See also

The pattern of change introduced by this pattern can be used with advantage in the process of refactoring to do to improve the structure (or quality) of an existing OO software. A refinement process with evidence rather favors obtaining a correct by construction software. The pattern Class_Abstraction advocates for the inclusion of other software qualities, such as efficiency, scalability from the initial phases of a development process guided by successive refinements. Besides, the risk of overlooking the efficiency quality in a process of refinement with proof is mentioned in [24].

G. Pattern 7: Class_Association

1) Intention

This pattern can increase the power of an association by considering both as an association relationship and an association class. This can be justified by the emergence of the specific details of the association relationship. Indeed, such details may be attached to extremities of the association. An association class can only exist if the association relationship exists.

2) Motivation

Sometimes, an association must own properties. These properties cannot be attached to the extremities of this association.

3) Solution

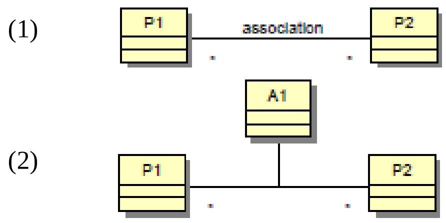


Figure 8. Class_Association specification

4) See also

The specification part of Class_Helper pattern is identical to the pattern Class_Association. However, their refinement parts are different.

V. Example

Our objective is to develop a system to control the access of person to the various buildings of a workplace, inspired by [17]. In [17], this application is modeled in Event-B. In this work, we provide a joint development in UML / OCL and B of this application by using the proposed refinement patterns. Proof tools and animation associated with B are used to perform automated verification of UML /OCL graphical models. Control is carried out from the authorizations assigned to the concerned persons. An authorization allows a person, under the control of the system, to enter in some buildings and not in others. The authorizations are permanent, i.e., they cannot be modified during the operation of the system. When a person is inside a building, his exit must also be controlled so that it is possible to know, at any moment, who are in a given building. A person can move from a building to another only if these two buildings are interconnected. The communication between the buildings is done through one-way doors. Each door has an origin building and a destination building. A person may enter a building by crossing a door if it is unlocked. The doors being physically locked, a door unlocked for only one authorized person requiring entering the building. A green LED associated with each door is lit when the requested access is authorized, prerequisite for unlocking the door. Similarly, a red LED associated with each door is lit when the requested access is denied to the door. Each person has a magnetic card. Card readers are installed at each door to read the information on a card. Near each reader, there is a turnstile that is normally blocked; no one can cross it without the control of the system. Each turnstile is equipped with a clock, which determines in part its behavior.

A. Rewriting of requirements

Rewriting of requirements of the case study aims to highlight the properties of this application. In order to classify these requirements, we used the following labels:

- EQU-Equipment to reference the description of equipment used by the application.

- FUN-Equipment/Actor to reference an attached functionality of a device or an actor.
- MODEL-FUN-number to reference an assured by the application functionality.
- FUN-MODEL to reference the main function of the application.

In Table I, we will list the different requirements of the application of building access control. Each property is described by a relatively short text and a reference.

TABLE I. REWRITING OF REQUIREMENTS OF AN ACCESS CONTROL SYSTEM.

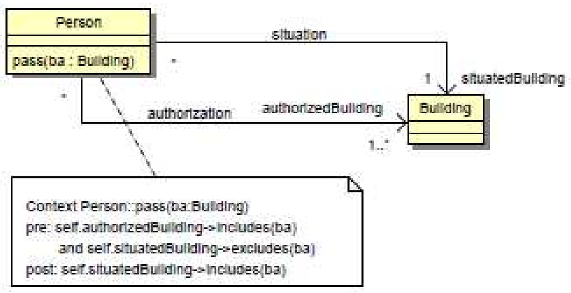
The system is responsible for controlling access of a number of people to several buildings.	FUN-MODELE
Each person is allowed to enter certain buildings (and not others). Buildings not recorded in this authorization are implicitly prohibited. This is a permanent assignment.	MODELE-FUN-1
Any person in a building is allowed to be there.	MODELE-FUN-2
The geometry of the building is used to define which buildings can communicate with each other and in what direction.	MODELE-FUN-3
A building does not communicate with itself.	MODELE-FUN-4
A person cannot move from a building where it is to a building where he wants to go if these two buildings communicate with each other.	MODELE-FUN-5
Any person authorized to be in a building should be allowed to go to another building that communicates with the first.	MODELE-FUN-6
The buildings are connected together by means of gates, which are one-way. We can therefore speak of origin and destination buildings for each door.	EQU-DOOR
A door cannot be taken if it is unlocked. A door can be unlocked for only one person at the same time. Conversely, any person involved in the unlocking of a door cannot be in one another.	FUN-DOOR-1
When a door is unlocked for a certain person, it is in the building behind the door in question. In addition, this person is allowed to go to the destination building of same door.	FUN-DOOR-2
When a door is unlocked for a certain person, it is in the building behind the door in question. In addition, this person is allowed to go to the destination building of same door.	FUN-PERSON
A green LED associated with each door.	EQU-GREENLIGHT
A green LED is lit when the requested access is allowed (pre requisite for unlocking the door).	FUN-GREENLIGHT
A red LED associated with each door	EQU-REDLIGHT
The red light of a door whose access has been denied.	FUN-REDLIGHT
The red and green lights of the same door cannot be turned on simultaneously.	FUN-LIGHT
Each person has a magnetic card that contains his permissions for different buildings.	EQU-CARD
Card readers are installed at each door to read the information on a card.	EQU-CARDREADER

B. *Refinement strategy*

Table II specifies the order of consideration of the properties and requirements of our case study: building access control. This defines our refinement strategy for incremental development of this application. The initial model is limited to the basic abstract properties of the application. Each refinement step includes a small number of properties from the abstract to the concrete. The refinement process ends when all properties from rewriting requirements were indeed taken into account. Equipment, such as door, card or LED that the application uses is introduced during the final stages of the adopted refinement process.

C. *Abstract specification*

We begin by developing a simple and very abstract class diagram that takes into account only the properties (FUN-MODELE, MODELE-FUN-2) (see Figure 9).



```
Context Person::pass(ba:Building)
pre: self.authorizedBuilding->includes(ba)
      and self.situatedBuilding->excludes(ba)
post: self.situatedBuilding->includes(ba)
```

Figure 9. Initial class diagram

TABLE II. REFINEMENT STRATEGY

Model	Equipment and Function
Initial // First	FUN-MODELE, MODELE-FUN-2 // MODELE-FUN-1
Second	MODELE-FUN-3, MODELE-FUN-4, MODELE-FUN-5, MODELE-FUN-6
Third // Fourth	EQU-DOOR, FUN-DOOR-2 // FUN-DOOR-1, FUN-PERSON

Fifth	EQU-GREENLIGHT, FUN-GREENLIGHT, EQU-REDLIGHT, FUN-REDLIGHT
Sixth // Seventh	FUN-LIGHT // EQU-CARD, EQU-CARDREADER

D. *Refinement steps*

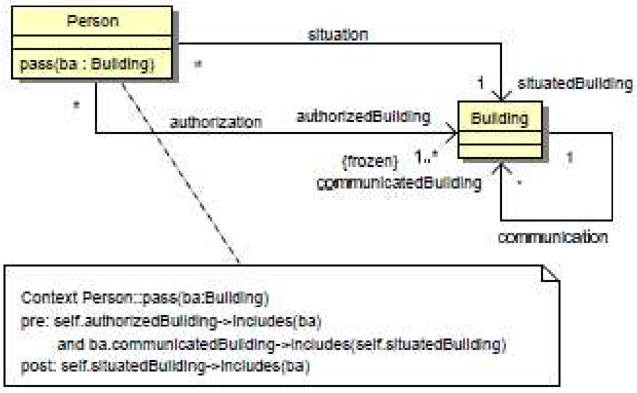
1) *First refinement*

In this step, we consider only the property (MODELE-FUN-1). This property indicates that the authorizations provided by the association "authorization" are permanent. For that, we applied data refinement based on pattern of Refinement_Operation. This refinement consists on the addition of a frozen constraint to the association "authorization" presented in Figure 9.

2) *Second refinement*

In this step, we inject into our system the properties (MODELE-FUN-3, MODELE-FUN-4, MODELE-FUN-5 and MODELE-FUN-6). These properties allow the introduction of the concept of communication between buildings. A person cannot move from one building to another only if the two buildings are interconnected.

The association “communication” is introduced into the class diagram as a recursive association on the class Building (see Figure 10). Such refinement requires a rewriting of the OCL expressions of the operation “pass”. Thus, we reused the refinement pattern Refinement_Operation.

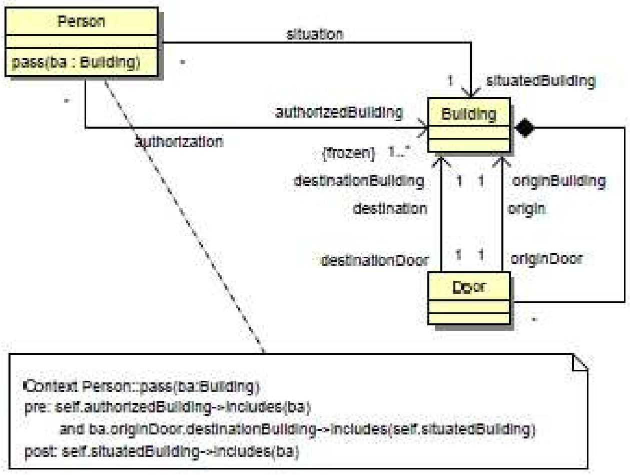


```
Context Person::pass(ba:Building)
pre: self.authorizedBuilding->includes(ba)
      and ba.communicatedBuilding->includes(self.situatedBuilding)
post: self.situatedBuilding->includes(ba)
```

Figure 10. Second refinement

5) *Third refinement*

The third refinement consists in adding new equipment (EQU-DOOR and FUN-DOOR-2). A door can make the connection between two buildings. This leads us to define the concept of door: each door has original building and a destination building. Indeed, the communication between the buildings is through a door. Thus, we must remove the association "communication", introduced in the previous refinement, and replace it with two associations between origin Building and Door and between Door and destination Building. This change can be obtained by applying the refinement pattern Class_Helper with: communication as association; origin as association1; destination as association2; Door as Helper and Building as both P1 and P2. Finally, the property (FUN-DOOR-2) is that a door is a component of a building. Thus, we introduce a composition relationship between Door and Building (see Figure 11).

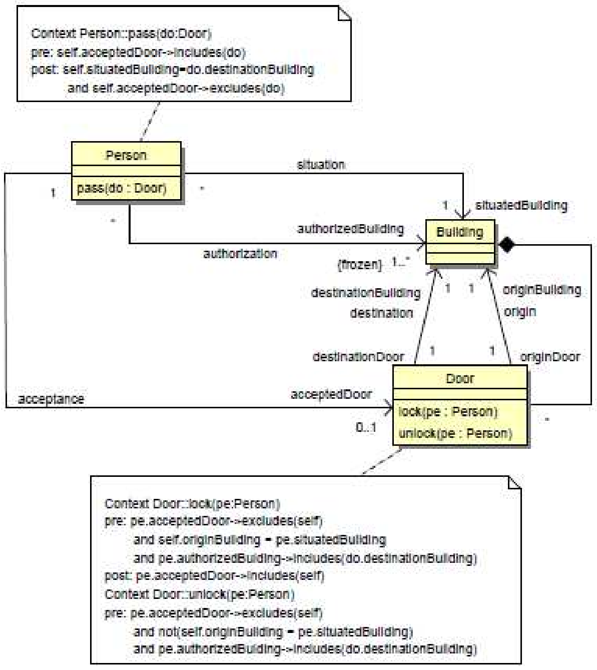


```
Context Person::pass(ba:Building)
pre: self.authorizedBuilding->includes(ba)
      and ba.originDoor.destinationBuilding->includes(self.situatedBuilding)
post: self.situatedBuilding->includes(ba)
```

Figure 11. Third refinement

4) *Fourth refinement*

The fourth step of refinement consists on the definition of the functionality of the class Door introduced in the previous step (FUN-DOOR-1).



```
Context Person::pass(do:Door)
pre: self.acceptedDoor->includes(do)
post: self.situatedBuilding=do.destinationBuilding
        and self.acceptedDoor->excludes(do)
```

```
Context Door::lock(pe:Person)
pre: pe.acceptedDoor->excludes(self)
      and self.originBuilding = pe.situatedBuilding
      and pe.authorizedBuilding->includes(do.destinationBuilding)
post: pe.acceptedDoor->includes(self)
Context Door::unlock(pe:Person)
pre: pe.acceptedDoor->excludes(self)
      and not(self.originBuilding = pe.situatedBuilding)
      and pe.authorizedBuilding->includes(do.destinationBuilding)
```

Figure 12. Fourth refinement

Such a transformation requires the revision of the semantics of the operation "pass". Indeed, property (FUN-

PERSON) is that a person must appear before a door to move from one building to another. This requires the introduction of an association "acceptance" between Person and Door. Thus, the operation "pass" should not take an instance of the class Building as formal parameter but rather an instance of the class Door. For this, we apply the refinement pattern Refinement_Operation to generate the class diagram presented in Figure 12.

5) Fifth refinement

In this step, we consider the properties (EQU-GREENLIGHT, FUN-GREENLIGHT, EQU-REDLIGHT and FUN-REDLIGHT). These properties define two new classes with their characteristics as components of the class Door. The application of refinement pattern Class_Decomposition with composition as a relationship between Door and RedLight and GreenLight as components generates the class diagram shown in Figure 13.

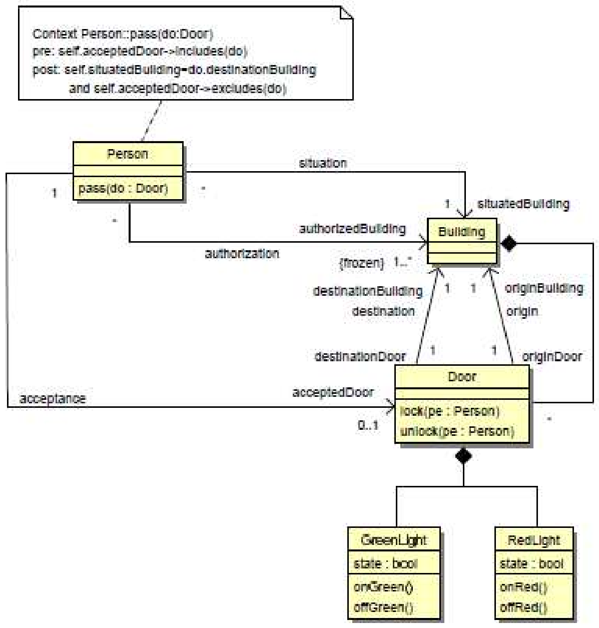


Figure 13. Fifth refinement

6) Sixth refinement

In this step, we note the similarity between the two classes RedLight and GreenLight (FUN-LIGHT). Thus, we decided to factor the common properties between these two classes. The application of refinement pattern Class_Abstraction generates the class diagram shown in Figure 14. The pattern Class_Abstraction allows introducing a new class named Light, which groups common properties between GreenLight and RedLight.

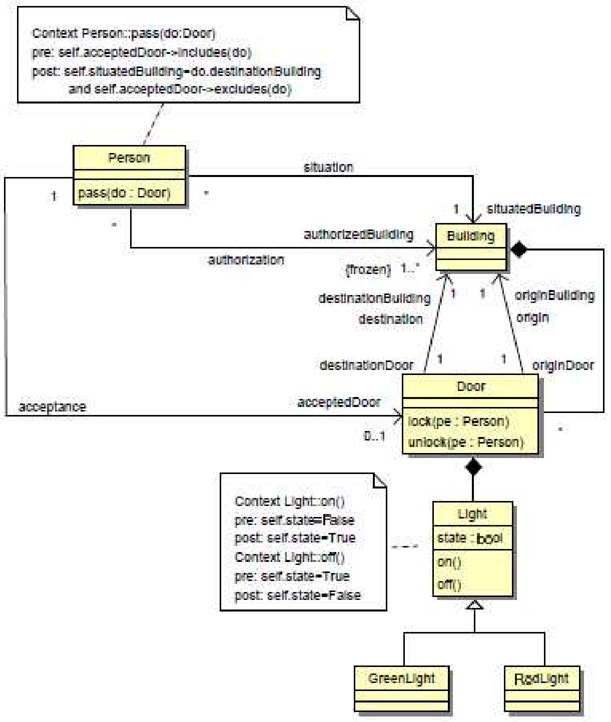


Figure 14. Sixth refinement

7) Seventh refinement

The last properties (EQU-CARD and EQU-CARDREADER) will be taken into account in this final stage of refinement.

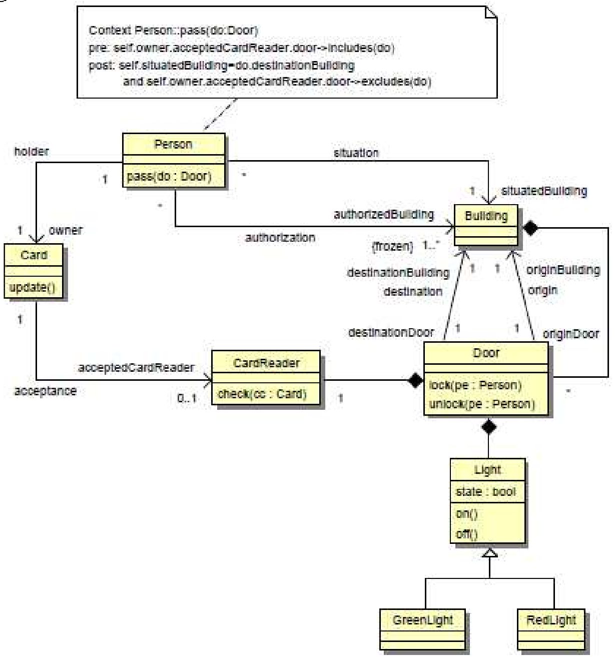


Figure 15. Seventh refinement

Two intermediate classes can be introduced:

- the class Card associated with each person,

- the class CardReader associated with each door of a building.

These classes are related as follows: Card is connected to Person, Card is connected to CardReader and CardReader is connected to Door.

The application of refinement pattern Class_Helper on the association "acceptance" generates the class diagram of Figure 15.

E. *Verification of the obtained system*

Formal verification of such refinements can be exploited if the language is equipped with formal refinement machinery, allowing the proof of the correctness of the refined specification relative to the abstract one. We proposed to use B for this purpose, using systematic derivation rules from UML into B. Such a translation of UML into B uses profitably the B formalization of the proposed refinement [6]. Indeed, the properties described in the form of B invariant --including gluing invariant-- are retrieved and instantiated when translating UML into B. As an illustration, Figure 16 and 17 shows the B formalization of pattern Class_Helper introduced in Section 4.

```
MACHINE
  B_Class_Helper_a
SETS
  OBJECTS = {p11, p12, p13,
          p21, p22, p23, h1, h2, h3}
ABSTRACT_CONSTANTS
  P1, P2
PROPERTIES
P1 ⊆ OBJECTS & P2 ⊆ OBJECTS ∧
  P1 ∩ P2 = ∅ ∧ P1 = {p11, p12, p13} ∧
  P2 = {p21, p22, p23}
VARIABLES
  p1, p2, association
INVARIANT
p1 ⊆ P1 & p2 ⊆ P2 ∧
  association ∈ p1 ↔p2
INITIALISATION
p1 := {p11,p12, p13} ||
p2 := {p21, p22, p23} ||
association := {p11↦p21, p11↦p22,
          p11↦p23, p12↦p21, p12↦p22,
          p12↦p23, p13↦p21, p13↦p22,
         p13↦p23}
END
```

Figure 16. B formalization of pattern Class_Helper

```
REFINEMENT
B_Class_Helper_r
REFINES
B_Class_Helper_a
ABSTRACT_CONSTANTS
HELPER
PROPERTIES
HELPER ⊆ OBJECTS ∧ HELPER ∩ P1 = ∅∧
HELPER ∩ P2 = ∅∧ HELPER = {h1, h2, h3}
ABSTRACT_VARIABLES
p1, p2, helper,
association1, association2
INVARIANT
helper ⊆ HELPER ∧
association1 ∈ p1↔helper ∧
association2 ∈ helper↔p2 ∧
ran(association1) = dom(association2) ∧
/*Gluing Invariant*/
dom(association) = dom(association1) ∧
ran(association) = ran(association2) ∧
ran(association1) = dom(association2) ∧
association = (association1;association2)
INITIALISATION
p1:={p11,p12,p13} k p2:={p21,p22,p23} ||
helper := {h1, h2, h3} ||
association1:={p117↦h1,p117↦h2,p117↦h3,
          p127↦h1, p127↦h2, p127↦h3,
          p137↦h1, p137↦h2, p137↦h3} ||
association2:={h17↦p21,h17↦p22,h17↦p23,
          h27↦p21, h27↦p22, h27↦p23,
          h37↦p21, h37↦p22, h37↦p23}
END
```

Figure 17. B formalization of pattern Class_Helper

The abstract machine B_Class_Helper_a formalizes the abstract level of Class_Helper pattern using the systematic translation rules of UML to B [11], while B_Class_Helper_r machine formalizes the refined level of the same pattern. The link between these two levels is described by the REFINES clause. Gluing invariant introduced in B_Class_Helper_r machine guarantees the correction of the refinement relation between the two levels of Class_Helper pattern. Formal verifications on the B models corresponding to UML/OCL class diagram are related to the coherence of the initial abstract model and the correction of each refinement step. They call the generator of proof obligations (conjectures to prove) and provers in B platform. The correction of B models, respecting requirements, is forward to the ProB tool [29], allowing animation and model checking.

TABLE III. TABLE OF THE STATE OF B SPECIFICATIONS

	nPO[1]	nPRi[2]	nPRa[3]	uUn[4]	%Pr
Initial model	9	1	8	0	100
Second refinement	4	0	4	0	100
Third refinement	7	0	7	0	100
Fourth refinement	12	3	9	0	100
Fifth refinement	12	2	10	0	100
Seventh refinement	26	0	26	0	100

[1] Number of Proof Obligations
[2] Number of Proof Obligations proved Interactively
[3] Number of Proof Obligations proved Automatically
[4] Number of Proof Obligations Unproved

Table III summarizes the proof obligations associated with our case study. The seven proposed refinement promote essentially the development of class diagrams correct by construction. However, the designer could improve the structure, without changing the semantic aspects of the class diagram obtained by refinement using wisely the refactoring technique [23][25].

VI. CONCLUSION AND FUTURE WORK

The main idea of this work is to propose intuitive refinements as patterns, providing a basis for tools supporting the refinement-driven modeling process. In this paper, we have presented our catalogue of refinement patterns. Formal verification of such refinements can be exploited if the language is equipped with formal refinement machinery, allowing the proof of the correctness of the refined specification relative to the abstract one. We proposed to use B for this purpose, using systematic derivation rules from UML into B. The proposed refinement patterns promote the identification of analysis classes that model the key concepts, resulting from requirements.

Currently, we are exploring the following two tracks: proposal of refinement patterns oriented design by retrieving and adapting ideas from GoF patterns [10]; proposal of refinement patterns oriented implementation, using the object-oriented modeling universal data structures (Eiffel) [7]. The next step of this work consists of automating detecting and application of patterns in an appropriate framework. In addition, we proposed a new approach, allowing finding a refinement strategy for the development of UML class diagrams guided by the refinement patterns. An interesting idea is to preserve the history of pattern application in development case studies in order to have a traceability of the development process, allowing to back-track on previous decisions.

REFERENCES

[1] A. Requet, “BART: A Tool for Automatic Refinement,” ABZ, London, UK, September, 2008, pp. 345-345.

[2] A. van Lamsweerde, Requirements Engineering - From System Goals to UML Models to Software Specifications, Wiley, 2009.

[3] B. Ben Ammar, M.T. Bhiri, and A. Benhamadou, “Refinement Pattern: Introduction of intermediate class Class_Helper,” Conférence en IngénierieE du Logiciel, CIEL, Rennes, France, Jun, 2012, pp. 1-6.

[4] B. Ben Ammar, M.T. Bhiri, and J. Souquières, “Event modeling for construction of class diagrams,” RSTI - ISI, vol. 13, no. 3, 2008, pp. 131–155.

[5] B. Ben Ammar, M.T. Bhiri, and J. Souquières, “Refactoring pattern of class diagrams based on the notion of delegation,” 7éme atelier sur l'Evolution, Réutilisation et Traçabilité des Systèmes d’Information, ERTSI, couplé avec le XXVI éme congrès INFORSID, Fontainebleau, France, May, 2008, pp. 1-12.

[6] B. Ben Ammar, Contribution to the Systems Engineering: Refinement and Refactoring of UML specifications, Editions universitaires europeennes, 2012.

[7] B. Meyer, Object-oriented software construction, Prentice Hall, 1997.

[8] C. A. R. Hoare, “Communicating sequential processes,” Communications of the ACM, vol. 21, no. 8, 1978, p. 666-677.

[9] C. Snook and M. Butler. 2006, “UML-B: Formal modeling and design aided by UML,” ACM Trans. Softw. Eng. Methodol. vol. 15, no. 1, January, 2006, pp. 92-122.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns, Addison-Wesley, 1995.

[11] E. Meyer and J. Souquières, “A Systematic Approach to Transform OMT Diagrams to a B Specification,” Proceedings of the World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September,1999, pp. 875-895.

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture: a system of patterns, John Wiley and Sons, 1996.

[13] G. Booch, “Object-Oriented Development,” IEEE Trans. Software Eng., vol. 12, no. 2, 1986, pp. 211–221.

[14] G. Smith, The Object-Z Specification Language. Kluwer Academic Publishers, 2000.

[15] H. Habrias and C. Stoquer, “A formal semantics for UML refining,” XII Colloque National de la Recherche en IUT, CNRIUT'06, Brest, France, Jun, 2006.

[16] H. Ledang, “Automatic Translation from UML Specifications to B,” Proceedings of the 16th IEEE international conference on Automated software engineering, San Diego, USA, November, 2004, pp. 436-440.

[17] J. R. Abrial, Modeling in Event-B - System and Software Engineering, Cambridge University Press, 2010.

[18] J. R. Abrial, The B Book - Assigning Programs to Meanings, Cambridge University Press, 1996.

[19] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual. 2 Boston, MA: Addison-Wesley, 2005.

[20] K. Beck and W. Cunningham, “A laboratory for teaching object-oriented thinking,” ACM SIGPLAN Not., vol. 24, no. 10, 1989, pp. 1–6.

[21] K. Waldèn and J. M. Nerson, Seamless object-oriented software architecture: analysis and design of reliable systems, Prentice-Hall, Inc., 1995.

[22] M. Fowler, Analysis Patterns: Reusable Object Models, Addison-Wesley Professional, 1996.

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

[24] M. Guyomard, “Specification and refinement using B: two pedagogical examples,” ZB2002 4th International B Conference, Education Session Proceedings, Grenoble, France, January, 2002.

[25] R. Straeten, V. Jonckers, and T. Mens, “A formal approach to model refactoring and model refinement,” Software and System Modeling (2), 2007, pp. 139-162.

[26] T. S. Hoang, A. Furst, and J. R. Abrial, “Event-B Patterns and Their Tool Support,” Software Engineering and Formal Methods, International Conference on, Hanoi, Vietnam, November, 2009, pp. 210-219.

[27] W. Su, J. R. Abrial, R. Huang, and H. Zhu, “From Requirements to Development: Methodology and Example,” The 13th International Conference on Formal Engineering Methods, ICFEM, Durham, United Kingdom, October, 2011, pp. 437-455.

[28] X. Castellani,“Cards stages of study of UML diagrams, Payment orders of these studies,” Technique et Science Informatiques, vol. 21, no. 8, 2002, pp. 1051–1072.

[29] The ProB Animator and Model Checker, User Manual, http://www.stups.uni-duesseldorf.de/ProB/index.php5/User_Manual, 2013.

[30] Clearsy System Engineering, Atelier B, User Manual, Version 4.0, http://www.tools.clearsy.com/resources/User_uk.pdf, 2010.

Automating Cloud Application Management Using Management Idioms

Uwe Breitenbücher, Tobias Binz, Oliver Kopp, Frank Leymann
Institute of Architecture of Application Systems
University of Stuttgart, Stuttgart, Germany
{breitenbuecher, lastname}@iaas.uni-stuttgart.de

***Abstract*—Patterns are a well-established concept to document generic solutions for recurring problems in an abstract manner. Especially in Information Technology (IT), many pattern languages exist that ease creating application architectures, designs, and management processes. Their generic nature provides a powerful means to describe knowledge in an abstract fashion that can be reused and refined for concrete use cases. However, the required manual refinement currently prevents applying the concept of patterns efficiently in the domain of Cloud Application Management as automation is one of the most important requirements in Cloud Computing. This paper presents an approach that enables automating both (i) the refinement of management patterns for individual use cases and (ii) the execution of the refined solutions: we introduce Automated Management Idioms to refine patterns automatically and extend an existing management framework to generate executable management workflows based on these refinements. We validate the presented approach by a prototypical implementation to prove its technical feasibility and evaluate its extensibility, standards compliance, and complexity.**

Keywords*—*Application Management; Automation; Patterns; Idioms; Cloud Computing

I. Introduction

Patterns are a well-established concept to document reusable solution expertise for frequently recurring problems. In many areas, they provide the basis for decision making processes, design evaluations, and architectural issues. In the domain of Cloud Computing, patterns are of vital importance to build, manage, and optimize IT on various levels: Cloud Computing Architecture and Management Patterns [1], Enterprise Integration Patterns [2], and Green IT Patterns [3] are a few examples that provide helpful guides for realizing complex Cloud applications, their management, and challenging non-functional requirements. The concept of patterns enables IT experts to document knowledge about proven solutions for problems in a certain context in an abstract, structured, and reusable fashion that supports systems architects, developers, administrators, and operators in solving concrete problems. The abstract nature of patterns enables generalizing the core of problem and solution to a level of abstraction that makes them applicable to various concrete instances of the general problem—independently from individual manifestations. Thus, many IT patterns are applicable to a big set of different settings, technologies, system architectures, and designs. Applying patterns to real problems requires, therefore, typically a *manual refinement* of the described abstract high-level solution for adapting it to the concrete use case. To guide this refinement and ease pattern application, most pattern languages document “implementations”, “known uses”, or “examples” for the described pattern solution—as already done implicitly by Alexander in his early publications on patterns [4].

However, in some areas, these benefits face difficulties that decrease the efficiency of using patterns immensely. In the domain of *Cloud Application Management*, the immediate, fast, and correct execution of management tasks is of vital importance to achieve Cloud properties such as on-demand self-service and elasticity [5][6]. Thus, management patterns, e. g., to scale a Cloud application, cannot be applied *manually* by human operators when a problem occurs because manual executions are too slow and error prone [1][7]. Therefore, to use patterns in Cloud Application Management, their application must be automated [8]. A common way to automate the execution of management patterns is creating executable processes, e. g., workflows [9] or scripts, that implement a refined pattern solution for a certain application [1]. To achieve Cloud properties, this must be done *in advance*, i. e., before the problem occurs, for being ready to run them immediately when needed. However, these processes are tightly coupled to a single application as the refinement of a management pattern depends mainly on the technical details of the application, its structure, and the concrete desired solution [10]. For example, the pattern for scaling a Cloud application results in different processes depending on the Cloud provider hosting the application: due to the heterogeneous management APIs offered by different Cloud providers, different operations have to be executed to achieve the same conceptual effect [11]. Thus, individual pattern refinement influences the resulting management processes fundamentally. As a result, multiple individual processes have to be implemented in advance to execute one management pattern on different applications. However, if multiple patterns have to be implemented for hundreds of applications in advance, this is not *efficient* as the required effort is immense. In addition, as Cloud application structures typically evolve over time, e. g., caused by scaling, a pattern may has to be implemented multiple times for a single application. Ironically, if the implemented processes are not used during the application’s lifetime, the spent effort was laborious, costly, but completely unnecessary.

The result of the discussion above is that the gap between a pattern’s abstract solution and its refined executable implementation for a certain use case currently prevents applying the concept of patterns efficiently to the domain of Cloud Application Management—due to the mandatory requirement of automation and its difficult realization. Thus, we need a means to automate the refinement of abstract patterns to concrete executable solutions on demand. In this paper, we tackle these issues by presenting an approach that enables applying management patterns fully automatically to individual applications. We show how the required refinement of a management pattern towards a concrete use case can be automated by inserting an additional layer of *Automated Management Idioms*, which provide a fine grained refinement of a particular abstract management pattern. These Automated Management Idioms are able to create formal declarative descriptions of the management tasks to be performed automatically for individual applications that are used afterwards to generate the corresponding executable management processes using an existing management framework. This enables automating the application of abstract patterns to various concrete applications without human intervention, which increases the efficiency of using patterns in Cloud Application Management. Thereby, the presented approach (i) helps IT experts to capture their management knowledge in an executable fashion and (ii) enables automating *existing* management patterns—both independently from individual applications. To prove the relevance of our approach, we conduct a detailed case study that illustrates the high complexity of applying an abstract migration management pattern to a concrete use case. We validate the approach through a prototype that extends an existing management framework by the presented concept and the implementation of real world migration use cases to prove its technical feasibility. Furthermore, we evaluate the concept in terms of automation, technical complexity, standards compliance, separation of concerns, and extensibility.

The paper is structured as follows: in Section II, we describe background information and a case study. In Section III, we describe the management framework which is extended by our approach presented in Section IV. In Section V, we evaluate the approach and present related work in Section VI. We conclude the paper and give an outlook on future work in Section VII.

II. Background, Motivation, and Requirements

In this section, we provide background information about the domain of Cloud Application Management and analyze the requirements to apply the concept of patterns. Afterwards, we motivate the presented approach through a case study.

A. *Automating Patterns for Cloud Application Management*

In this section, we discuss why the concept of patterns and their automation are of vital importance in the domain of Cloud Application Management to optimize management and to reduce the number of errors and downtimes. Due to the steadily increasing use of IT in enterprises, accurate operation and management are of crucial importance to align business and IT. As a consequence, requirements such as high-availability, elasticity, and cheap operation of IT arise increasingly. Therefore, more and more enterprises outsource their IT to external providers to achieve these properties reliably and to automate IT management—both enabled by Cloud Computing [12]. However, the fast growing number of proprietary, mostly heterogeneous, Cloud services offered by different providers leads to a high complexity of creating composite Cloud applications and executing management tasks as the (i) offered services, (ii) non-functional capabilities, and (iii) application programming interfaces (APIs) of different Cloud providers differ significantly from each other. As a result, the actual structure of an application, the involved types of Cloud services (e. g., platform services), and the management technologies to be used mainly depend on the providers and are hardly interoperable with each other due to proprietary characteristics. This results in completely different management processes for different Cloud providers implementing the same conceptual management task. As a consequence, the *conceptual solution* implemented in such processes gets obfuscated by technical details and proprietary idiosyncrasies. It is nearly impossible to reuse and adapt the implemented knowledge *efficiently* for other applications and providers as the required effort to analyze and transfer the conceptual solution to other use cases is much too high. This results in continually reinventing the wheel for problems that were already solved multiple times—but not documented in a way that enables reusing the conceptual knowledge. A solution for these issues are management patterns, which provide a well-established means to document conceptual solutions for frequently recurring problems in a structured, reusable, and tailorable way [13]. Therefore, a lot of architectural and management knowledge for Cloud applications and their integration was captured in the past years using patterns, e. g., Enterprise Integration Patterns [2] and Cloud Computing Architecture and Management Patterns [1]. A management pattern typically documents the general characteristics of a certain (i) problem, (ii) its context, and (iii) the solution in an abstract way without getting lost in technical realization details. Thereby, they help applying proven conceptual management expertise to individual applications, which typically requires a manual refinement of the pattern's abstract solution for the concrete use case. However, this manual refinement leads to two challenges that are tackled in this paper to enable applying the concept of management patterns efficiently in the domain of Cloud Application Management: (i) technical complexity and (ii) automation of refinement and solution execution.

1) Technical Complexity of Refinement: The technical layer of IT management and operation becomes a more and more difficult challenge as each new technology and its management issues increase the degree of complexity—especially if different heterogeneous technologies are integrated in complex systems [10]. Thus, the required refinement of a pattern's abstract solution to fine grained technical management operations as well as their correct parametrization and execution are complex tasks that require technical expert management knowledge.

2) Automation of Refinement and Solution Execution: In Cloud Computing, application management must be automated as the manual execution of management tasks is too slow and error-prone since human operator errors account for the largest fraction of failures in distributed systems [7][14]. Thus, the refinement of a pattern's abstract solution to a certain use case and the execution of the refined solution must be automated.

B. *Case Study and Motivating Scenario*

In this section, we present a case study that is used as motivating scenario throughout the paper to explain the approach and to illustrate the high complexity of applying abstract management patterns to concrete use cases. Due to the important issue of vendor lock-in in Cloud Computing, we choose a migration pattern to show the difficulties of refinement and the opportunities enabled by our approach. We explain the important details of the pattern and apply it to the use case of migrating a Java-based application to the Amazon Cloud. The selected pattern is called "Stateless Component Swapping Pattern" [15] and originates from the Cloud Computing pattern language developed by Fehling et al. [1][13][15]. The question answered by this pattern is *"How can stateless application components that must not experience downtime be migrated?"*. Its intent is extracting stateless application components from one environment and deploying them into another while they are

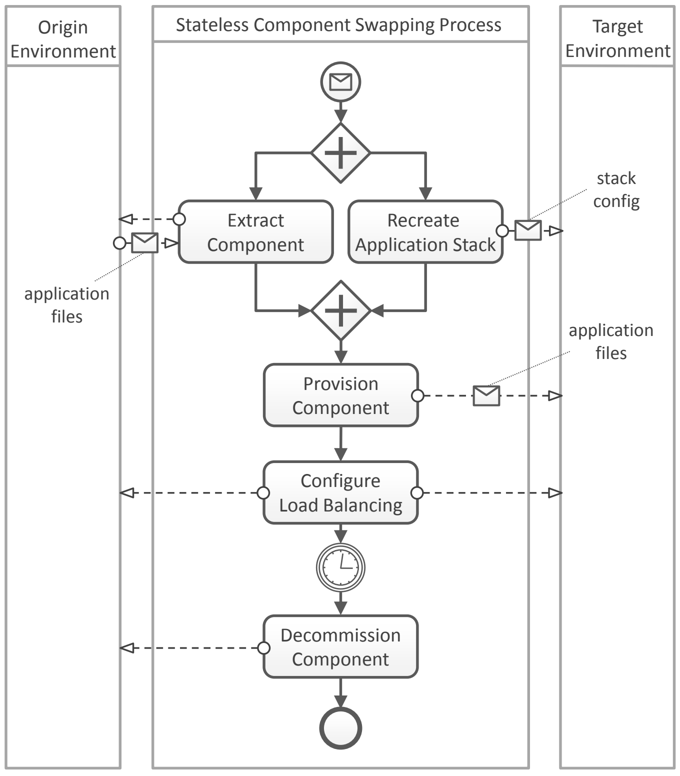


Figure 1. Abstract Stateless Component Swapping Process (adapted from [15]).

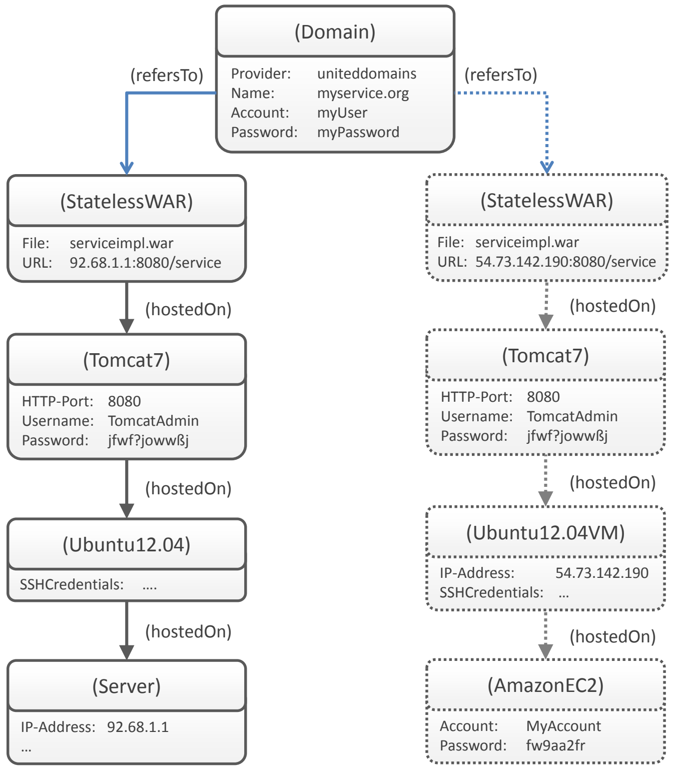


Figure 2. Use case for applying the Stateless Component Swapping Pattern.

active in both environments during the migration. Afterwards, the old components are decommissioned. The context observed by this pattern is that, in many business cases, the downtime of an application is unacceptable, e. g., for customer-facing services. A stateless application shall, therefore, be migrated transparently to the accessing human users or applications. Here, "stateless" means that the application does not handle internal session state: the state is provided with each request or kept in provider-supplied storage. Figure 1 depicts the pattern's solution as Business Process Model and Notation (BPMN) [16] diagram: the component to be migrated is first extracted from the origin environment while the required application stack is provisioned concurrently in the target environment. Then, the component is deployed on this stack in the target environment while the old component is still active. Finally, after the new component is provisioned, the reference to the migrated component is updated (load balancing) and the old component is decommissioned.

This pattern shall be applied to the following concrete use case. A stateless Webservice implemented in Java, packaged as Web Archive (WAR), is currently hosted on the local physical IT infrastructure of an enterprise. Therefore, a Tomcat 7 Servlet Container is employed that runs this WAR. The Tomcat is deployed on an Ubuntu Linux operating system that is hosted on a physical server. The Webservice is publicly reachable under a domain, which is registered at the domain provider "United Domains". This service shall be migrated to Amazon's public Cloud to relieve the enterprise's physical servers. Therefore, Amazon's public Cloud offering "Elastic Compute Cloud (EC2)" is selected as target environment. On EC2, a virtual machine with the same Ubuntu operating system and Tomcat version shall be provisioned to run the Webservice in the Cloud.

Figure 2 shows the technical details of this migration. On the left, the current Enterprise Topology Graph (ETG) [17] of the application is shown. An ETG is a formal model that captures the current state of an application in the form of a topology, which describes all components and relations including their types, configuration, and runtime information. ETGs of running applications can be discovered fully automatically using the "ETG Discovery Framework" [18]. We use the visual notation Vino4TOSCA [19] to render ETGs graphically: components are depicted as rounded rectangles containing their runtime properties below, relationships as arrows. Element types are enclosed by parentheses. On the right, the goal of the migration is shown: all components and relationships drawn with dotted lines belong to the goal state, i. e., the refined pattern solution.

To refine the pattern's abstract solution process shown in Figure 1 to this use case, the following tasks have to be performed: (i) the WAR to be migrated must be extracted from the origin environment, (ii) a virtual machine (VM) must be provisioned on EC2, (iii) a Tomcat 7 Servlet Container must be installed on this VM, (iv) the WAR must be deployed on the Tomcat, (v) the domain must be updated, and (vi) the old WAR must be decommissioned. The technical complexity of this migration is quite high as four different heterogeneous management APIs and technologies have to be combined and one workaround is required to achieve these goals: the extraction of the WAR deployed on the local Tomcat is not supported by Tomcat's management API [20]. Thus, a workaround is needed that extracts the WAR file directly from the underlying Ubuntu operating system. However, this is technically not trivial: an SSH connection to the operating system must be established, the respective directory must be found in which Tomcat stores the

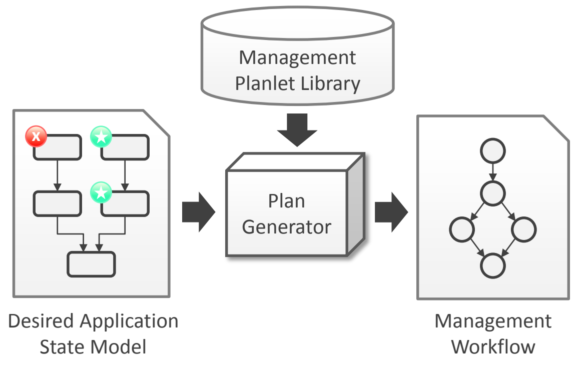


Figure 3. Management Planlet Framework Overview (adapted from [8]).

deployed WAR files, and the correct WAR must be transferred from the remote host to a local host using protocols such as Secure Copy (SCP). To provision a new virtual machine, EC2's HTTP-based management API [21] has to be invoked. However, an important security detail has to be considered here: per default, a virtual machine on EC2 is *not* accessible from the internet. Amazon employs so-called "Security Groups" to define firewall rules for accessing virtual machines. Thus, as the Webservice should be accessible from the internet, a Security Group must be defined to allow access. To install Tomcat 7, script-centric configuration management technologies such as Chef can be used. To deploy the WAR on Tomcat, Tomcat's HTTP-based management API has to be invoked. The domain can be updated using the management API of United Domains [22]. However, to avoid downtime, the old WAR must not be decommissioned until the Domain Name System (DNS) servers were updated with the new URL. Therefore, a DNS Propagation Checker must be employed. To summarize, the required technical knowledge to refine and implement the pattern's abstract solution for this concrete use case is immense. In addition, automating this process and orchestrating several management APIs that provide neither a uniform interface nor compatible data formats is complex, time-consuming, and costly as the process must be created by experts to avoid errors [10]. As a consequence, (i) handling the technical complexity and (ii) automating this process are difficult challenges.

III. Employed Management Framework

The approach we present in this paper tackles these issues by extending the "Management Planlet Framework" [8][10][23]. This framework enables describing management tasks to be performed in an *abstract* and *declarative* manner using Desired Application State Models, which can be transformed fully automatically into executable workflows by a Plan Generator through orchestrating Management Planlets. In this section, we explain the framework briefly to provide all required information to understand the presented approach. The concept of the management framework is shown in Figure 3. The *Desired Application State Model (DASM)* on the left declaratively describes management tasks that have to be performed on nodes and relations of an application. It consists of (i) the application's ETG, which describes the current structure and runtime information of the application in the form of properties, and (ii) *Management Annotations*, which are declared on nodes and relations to specify the management tasks to be executed

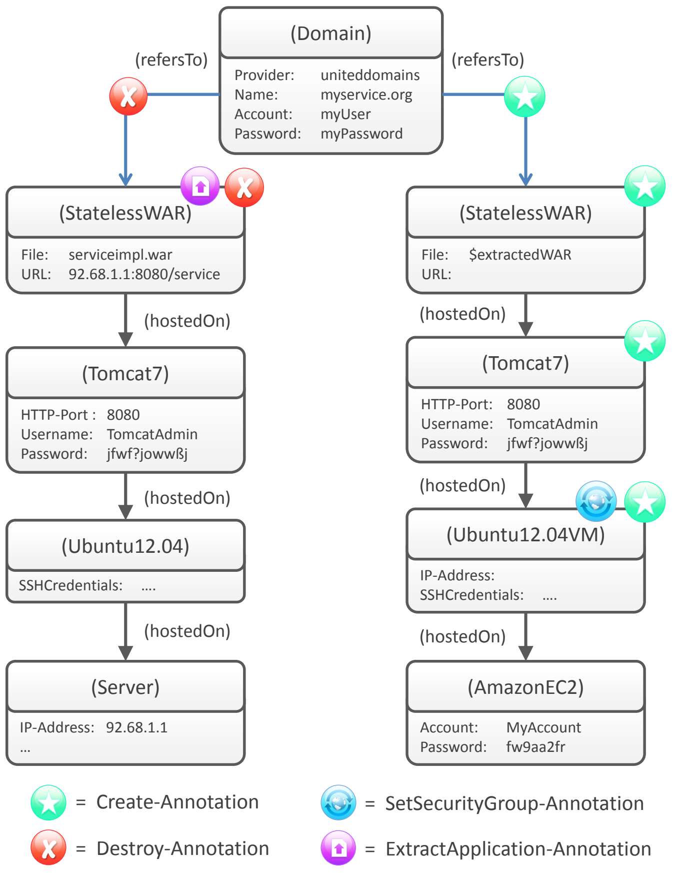


Figure 4. DASM describing the management tasks of the refined pattern.

on the associated element, e. g., to create, update, or destroy the corresponding node or relation. A Management Annotation (depicted as coloured circle) defines only the abstract semantics of the task, e. g., that a node should be created, but not its technical realization. Thus, in contrast to executable imperative management descriptions such as management workflows that define all technical details, a DASM describes the management tasks to be performed only declaratively, i. e., only the *what* is described, but not the *how*. As a consequence, DASMs are not executable and are, therefore, transformed into executable *Management Workflows* by the framework's *Plan Generator*.

Figure 4 shows a DASM that realizes the motivating scenario of Section II-B. The DASM describes the required additional nodes and relations as well as Management Annotations that declare the tasks to be performed: the green *Create-Annotations* with the star inside declare that the corresponding node or relation shall be created while the red circles with the "X" inside represent *Destroy-Annotations*, which declare that the associated element shall be destroyed. The magenta coloured *ExtractApplication-Annotation* is used to extract the application files of the WAR node, the blue coloured *SetSecurityGroup-Annotation* configures the Security Group to allow accessing the node from the internet. This DASM specifies the tasks described by the abstract solution of the Stateless Component Swapping Pattern refined to the concrete use case of migrating a Java Webservice to EC2. As Management Annotations describe tasks only declaratively, the model contains no technical details about management APIs, data formats, or the control flow.

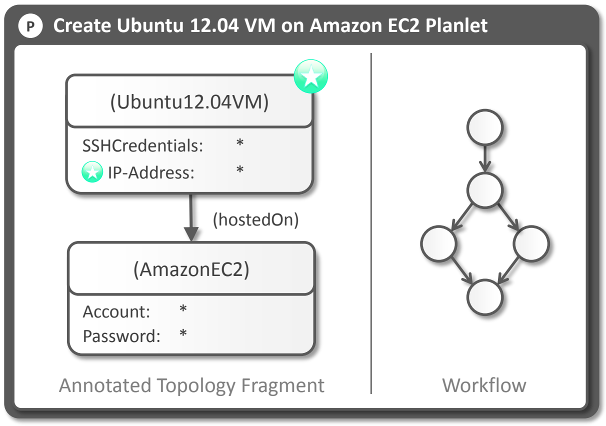


Figure 5. Management Planlet that creates an Ubuntu VM on Amazon EC2.

The framework's Plan Generator interprets DASMs and generates the corresponding executable workflows automatically. This is done by orchestrating so-called *Management Planlets*, which are workflows executing Management Annotations on nodes and relations such as deploying a WAR on Tomcat or updating a domain with a new URL. Planlets are developed by technology experts and provide the low-level imperative management logic to execute the declarative Management Annotations used in DASMs. Thus, they are reusable management building blocks implementing the *how* of the declaratively described abstract management tasks in DASMs. Management Planlets express their functionality through an *Annotated Topology Fragment*, which describes (i) the Planlet's *effects* in the form of Management Annotations it executes on elements and (ii) *preconditions* that must be fulfilled to execute the Planlet. The Plan Generator orchestrates suitable Planlets to process all Management Annotations in the DASM. The order of Management Planlets is determined based on their preconditions and effects: all preconditions of a Planlet must be fulfilled by the DASM itself or by another Planlet that is executed before.

Figure 5 shows a Planlet that creates a new Ubuntu 12.04 virtual machine on Amazon EC2. The Annotated Topology Fragment exposes the Planlet's functionality by a Create-Annotation attached to the node of type "Ubuntu12.04VM" which has a "hostedOn" relation to a node of type "AmazonEC2". The Planlet's preconditions are expressed by all properties that have no Create-Annotation attached: the desired "SSHCredentials" of the VM node as well as "Account" and "Password" of the Amazon EC2 node must exist to execute the Planlet, which takes this information to create the new VM (we omit other properties to simplify). The Planlet's effects on elements are expressed by Create-Annotations on their properties, i. e., the Create-Annotation on the "IP-Address" of the VM node means that the Planlet sets this property. The existence of this property and the "SSHCredentials" are typical preconditions of Planlets that install software on virtual machines. Thus, Planlets can be ordered based on such properties. The strength of Management Planlets is hiding the technical complexity completely [8]: the Plan Generator orchestrates Planlets based only on their Topology Fragments and the abstract Management Annotations declared in the DASM, i. e., without considering technical details implemented by the Planlet's workflow. This provides an abstraction layer to integrate management technologies.

IV. Automated Refinement of Management Patterns to Executable Management Workflows

In the previous section, we explained how DASMs can be used to describe management tasks in an abstract manner and how they are transformed automatically into executable workflows by the Management Planlet Framework. Thereby, the framework provides a powerful basis for automating patterns and handling the technical complexity of pattern refinement.

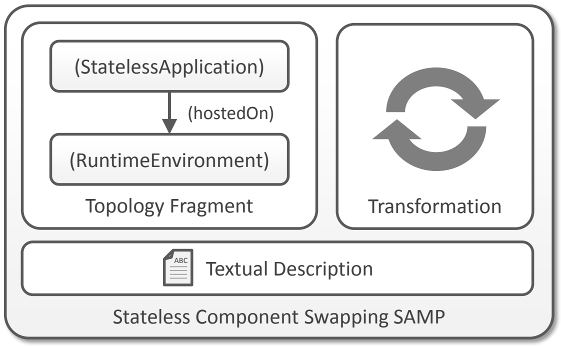


Figure 6. Stateless Component Swapping SAMP.

In a former work [8], we presented an approach to generate DASMs automatically by applying management patterns to ETGs of running applications. However, as patterns should not capture use case-specific information, the approach provides only a *semi-automated* means and requires a manual refinement of the resulting DASM. Consequently, we call these patterns *Semi-Automated Management Patterns (SAMP)*. In this paper, we extend the concept of SAMPs. Therefore, we introduce them briefly to provide required information. The input of a SAMP is the current Enterprise Topology Graph (ETG) of the application to which the pattern shall be applied, its output is a DASM that declaratively describes the pattern's solution in the form of Management Annotations to be performed. A SAMP consists of three parts, as shown in Figure 6: (i) Topology Fragment, (ii) Transformation, and (iii) textual description. The *Topology Fragment* is a small topology that defines the pattern's context, i. e., it is used to determine if a pattern is applicable to a certain ETG. Therefore, it describes the nodes and relations that must match elements in the ETG to apply the pattern to these matching elements. For example, the *Stateless Component Swapping SAMP* shown in Figure 6 is applicable to ETGs that contain a component of type "StatelessApplication" that is hosted on a component of type "RuntimeEnvironment". As the type "StatelessWAR" is a subtype of "StatelessApplication" and "Tomcat7" a subtype of "RuntimeEnvironment", the shown SAMP is applicable to the motivating scenario (cf. Figure 2). The second part is a *Transformation* that implements the pattern's solution logic, i. e., how to transform the input ETG to the output DASM that describes the tasks to be performed. However, SAMPs provide only a semi-automated means as the resulting DASM requires further manual refinement—similar to the required refinement of normal patterns for concrete use cases. Figure 7 shows the DASM resulting from applying the Stateless Component Swapping SAMP to the ETG described in the motivating scenario. As the pattern's transformation can not be aware of the desired refinement for this use case, it is only able to apply the *abstract* solution: (i) the stateless application is extracted,

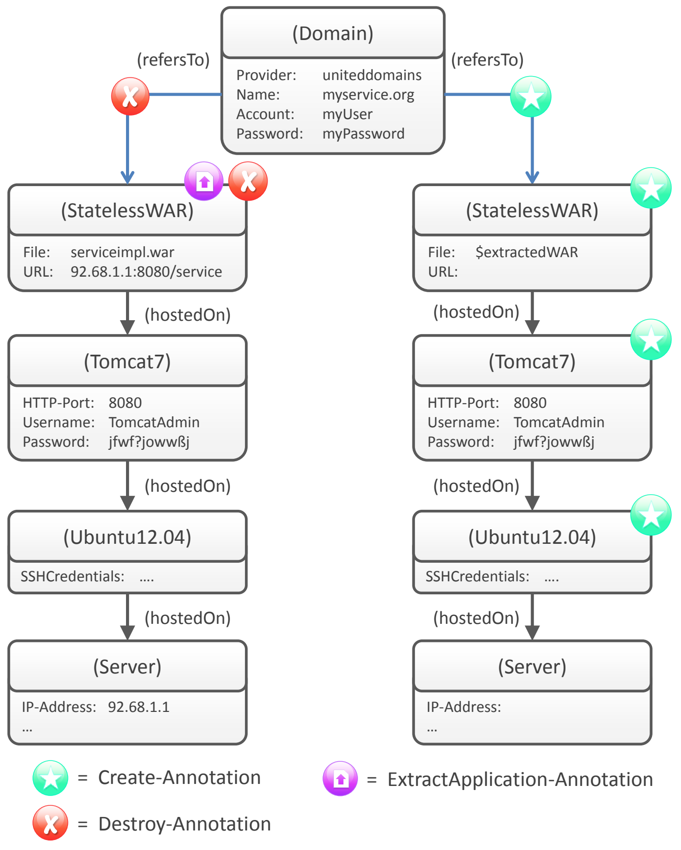


Figure 7. DASM after applying the Stateless Component Swapping SAMP.

(ii) the stack is copied, and (iii) all incoming relations of the application to be migrated are switched to the new deployment. To refine this DASM for the motivating scenario, the "Server" node type must be replaced by "AmazonEC2", the operating system must be a virtual machine of type "Ubuntu12.04VM", and a Management Annotation must be added to configure the Security Group (cf. Figure 4). However, manual refinement violates the two requirements analyzed in Section II-A as (i) the technical complexity remains (e. g., operator has to be aware of Security Groups) and (ii) human intervention is required, which breaks the required full automation. Of course, Semi-Automated Management Patterns could be created for concrete use cases as also shown in Breitenbücher et al. [8]. However, this violates the abstract nature of patterns and causes confusing dependencies between patterns and use cases, which decreases usability. In addition, with the increasing number of such use case-specific management patterns, the conceptual part of the abstract solution gets obfuscated. Therefore, we add an explicit refinement layer to separate the different layers of abstraction.

A. Overview of the Approach

In this and the next section, we present the main contribution of this paper. The approach enables applying patterns fully automatically to concrete use cases and solves the two problems analyzed in Section II-A: (i) handling the technical complexity of refinement and (ii) automating refinement and solution execution. The goal is to automate the whole process of applying a pattern selected by a human operator to a concrete use case. The reason that Semi-Automated Management Patterns can not be applied fully automatically results from the generative nature of patterns since they describe only the general core of solutions. Consequently, they must be refined manually to individual use cases. Therefore, we need a more specific means below patterns to document concrete management problems and already refined solutions. According to Buschmann et al. [24], who consider design issues of software, so-called "idioms" represent *low-level patterns* that deal with the implementation of particular design issues in a certain programming language. They address aspects of both design and implementation and provide, thus, already refined pattern solutions. For example, an idiom may describe concretely how to implement the abstract *Model-View-Controller Pattern (MVC)* in Java. Thus, this concept solves a similar kind of refinement problem in the domain of software architecture and design. Therefore, we adapt this refinement concept to the domain of Cloud Application Management.

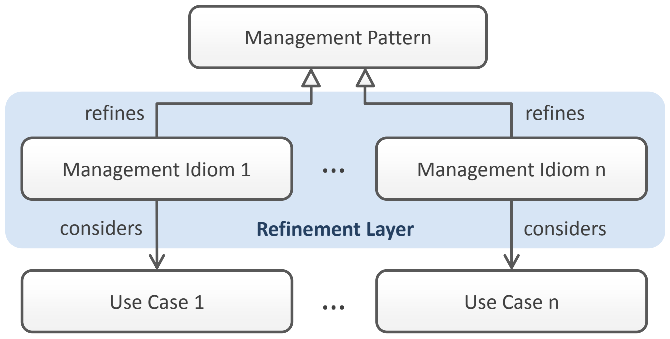


Figure 8. Additional refinement layer below management patterns.

We insert an additional refinement layer below management patterns in the form of *Application Management Idioms*, which are a tailored incarnation of a certain abstract management pattern refined for a concrete problem, context, and solution of a certain use case. Figure 8 shows the conceptual approach: a management pattern is refined by one or more Management Idioms that consider each a concrete use case of the pattern and provide an already refined solution. Thus, instead of refining an abstract management pattern manually to a certain use case, the pattern to be applied can be refined automatically by selecting the appropriate Management Idiom directly. Applying this concept to our motivating scenario, the refinement of the Stateless Component Swapping Pattern to our concrete use case of migrating a stateless Java Webservice packaged as WAR to Amazon EC2 (see Section II-B) can be captured by a "Stateless WAR from Tomcat 7 to Amazon EC2 Swapping Idiom", which describes the refined problem, context, and solution. Thus, instead of providing only the generic solution, this idiom is able to describe the tasks in detail tailored to this concrete context. Thereby, management tasks such as defining the Security Group can be described while the link to the conceptual solution captured by the actual pattern is preserved.

The additional refinement layer enables separating concerns: on the management pattern layer, the generic abstract problem, context, and solution are described to capture the *conceptual core*. On the Management Idiom layer, a *concrete refinement* is described that possibly obfuscates the conceptual solution partially to tackle concrete issues of individual use cases. As a consequence, the presented approach enables both (i) capturing generic management knowledge and (ii) providing tailored

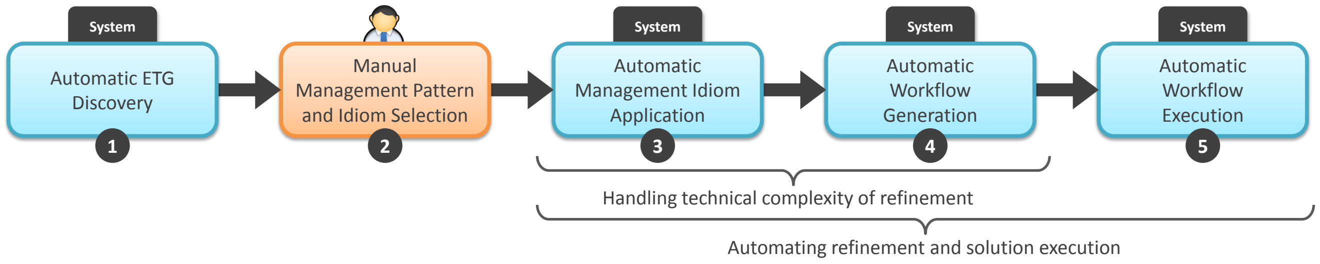

System
Automatic ETG Discovery
1
Manual Management Pattern and Idiom Selection
2
System
Automatic Management Idiom Application
3
System
Automatic Workflow Generation
4
System
Automatic Workflow Execution
5
Handling technical complexity of refinement
Automating refinement and solution execution

Figure 9. Process that describes how to apply a management pattern fully automatically to a running application using Automated Management Idioms.

refinements linking to the actual pattern. In the next section, we extend the concept of Semi-Automated Management Patterns by Automated Management Idioms following this concept.

B. Automated Management Idioms

To automate the new refinement layer, we introduce *Automated Management Idioms (AMIs)* in this section. Automated Management Idioms refine Semi-Automated Management Patterns through (i) providing a refined Topology Fragment that formalizes the particular context to which the idiom is applicable and (ii) implementing a more specific transformation tailored to the refined context for transforming the input ETG directly into an already refined DASM. Each AMI additionally defines a *Pattern Reference (PR)* linking to its original pattern.

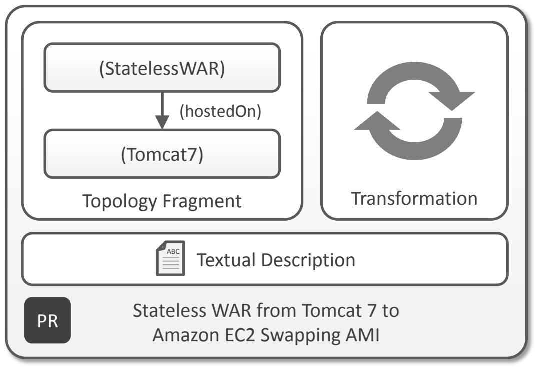


Figure 10. Stateless WAR from Tomcat 7 to Amazon EC2 Swapping AMI.

Figure 10 shows the AMI that refines the Stateless Component Swapping SAMP for our motivating scenario. The refinement of the context, to which this idiom is applicable, is expressed by the refined Topology Fragment: while the pattern is applicable to topologies that contain an abstract "StatelessApplication" that is hosted on an abstract "RuntimeEnvironment", the shown idiom refines this fragment and is only applicable to a node of type "StatelessWAR" that is hosted on a node of type "Tomcat7". The refinement of the Topology Fragment restricts the idiom's applicability and enables to implement a transformation that considers exclusively this concrete use case. Therefore, the idiom's transformation differs from the pattern's transformation by adding more details to the output DASM: it refines the "Server" node directly to "AmazonEC2" and adds the required Management Annotation that configures the Security Group accordingly in order to enable accessing the migrated WAR from the internet. The operating system is exchanged by "Ubtuntu12.04VM" while the "Tomcat7" and "StatelessWAR" nodes are simply copied from the ETG of the origin environment. Similar to the transformation of the pattern, all incoming relations of the old WAR node are redirected to the new WAR. Thus, the resulting DASM corresponds exactly to the DASM shown in Figure 4, which would be created manually by an expert to refine the pattern to this use case. As a result, applying this Automated Management Idiom to the motivating scenario's ETG results in a completely refined DASM which can be transformed directly into an executable workflow by the Management Planlet Framework. Thus, no further manual refinement is required to apply the pattern to this use case. Nevertheless, the Management Planlet Framework enables implementing this Automated Management Idiom on a high level of abstraction due to Management Annotations.

C. Fully Automated Pattern-based Management System

In this section, we explain the process of applying a management pattern in the form of an Automated Management Idiom automatically to a running application using the Management Planlet Framework. This process provides the basis for a *Fully Automated Pattern-based Management System* and consists of the five steps shown in Figure 9. First, the ETG of the application to be managed has to be discovered. This step is automated by using the ETG Discovery Framework [18], which gets an entry node of the application as input, e. g., the URL pointing to a deployed Webservice, and discovers the complete ETG of this application fully automatically including all runtime properties. In the second step, the user selects the pattern to be applied in the form of an Automated Management Idiom. This is the only manual step of the whole process. In the third step, the selected idiom is applied by the framework fully automatically to the discovered ETG. Therefore, the idiom's transformation is executed on the ETG. The output of this step is a Desired Application State Model that is based on the discovered ETG but additionally contains all management tasks to be executed in the form of Management Annotations. This DASM is used in Step 4 by the Management Planlet Framework to generate an executable management workflow, as described in Section III. In the last step, the generated workflow is executed using a workflow engine. The overall process fulfills the two requirements analyzed in Section II-A: (i) handling technical complexity and (ii) automating refinement and solution execution. The technical complexity is hidden by the framework's declarative nature: management tasks are described only as abstract Management Annotations. All implementation details are inferred automatically by Management Planlets. Thus, the user only selects the idiom to be applied, the technical realization is invisible and automated. The second requirement of automation is achieved through executable transformations of idioms and the employed Management Planlet Framework.

D. Automatic Refinement Filtering and Pattern Configuration

To apply a pattern automatically using this system, the user first triggers the automatic ETG discovery of the application to be managed, searches the Semi-Automated Management Pattern (SAMP) to be applied, and selects the desired refinement in the form of an Automated Management Idiom. Based on their Topology Fragments, non-applicable Automated Management Idioms are filtered automatically by the system. For example, if the Stateless Component Swapping SAMP is refined by two different Automated Management Idioms—one is able to migrate a PHP application hosted on an Apache HTTP Server to Amazon EC2, the other is the described WAR migration idiom shown in Figure 10—the system offers only the idioms whose Topology Fragments match the elements in the ETG. Thus, in case of our motivating scenario, only the idiom for migrating the WAR application is offered as exclusively its Topology Fragment matches the ETG. Based on this matchmaking, the system is able to offer only applicable management patterns and refinements in the form of Automated Management Idioms. Therefore, we call this concept *Automatic Refinement Filtering*.

If multiple idioms of a selected SAMP match the ETG after filtering, e. g., one idiom migrates the WAR application to Amazon EC2, another to Microsoft's public Cloud "Azure", the user is able to *configure* the refinement of the pattern by a simple manual idiom selection while its actual application and execution are automated completely. Therefore, we call this concept *Configurable Automated Pattern Refinement*. The combination of these two concepts automates the refinement in two dimensions: first, Automatic Refinement Filtering preselects applicable refinements automatically, which relieves the user from finding applicable idioms. Second, the remaining filtered idioms can be applied directly without human intervention. Thus, the only manual step in this process is selecting the pattern and the desired configuration in the form of an idiom.

V. Validation and Evaluation

In this section, we validate the presented approach by a (i) prototypical implementation and evaluate the concept in terms of (ii) standards compliance, (iii) automation, (iv) technical complexity, (v) separation of concerns, and (vi) extensibility.

A. Prototype

To validate the technical feasibility of the approach, we extended the Java prototype presented in our former work [8] by Automated Management Idioms. The system's architecture is shown in Figure 11. The Web-based user interface is implemented in HTML5 using Java Servlets. Therefore, the prototype runs on a Tomcat 7 Servlet Container. The UI calls the *Application Management API* to apply a pattern to a running application. Therefore, it discovers the application's ETG by integrating the ETG Discovery Framework [18] and employs a *Pattern and Idiom Manager* to select the pattern / idiom to be applied. The manager is connected to a local library which stores deployable SAMPs and AMIs in the form of Java-based Webservices and implements a matchmaking facility to analyze which patterns / idioms are applicable to a certain ETG. To add new patterns or idioms to the system, the library provides an interface to register additional implementations. After selection, the discovered ETG and the SAMP / AMI to

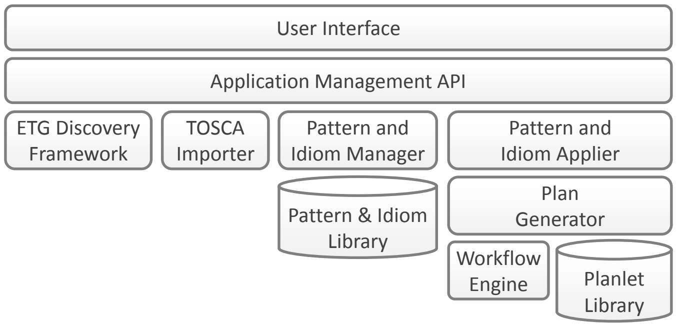


Figure 11. Fully Automated Pattern-based Management System Architecture.

be applied are passed to the *Pattern and Idiom Applier*, which executes the pattern's / idiom's transformation on the ETG and passes the resulting DASM to the *Plan Generator*. The generator is connected to a local *Management Planlet Library* that contains all available Planlets, which are implemented using the standardized Business Process Execution Language (BPEL) [25]. It generates the corresponding management workflow also in BPEL, which is deployed and executed afterwards on the employed workflow engine "WSO2 Business Process Server 2.0". We implemented several Semi-Automated Management Patterns and corresponding refinements in the form of Automated Management Idioms, e. g., for migration scenarios similar to the one used in this paper.

The Topology and Orchestration Specification for Cloud Applications (TOSCA) [26] is an OASIS standard to describe Cloud applications and their management in a portable way. Therefore, it provides a specification to model application topologies that can be linked with management workflows. As our approach is based on the same concepts, our prototype supports importing and deploying TOSCA-based applications, which can be managed afterwards using the presented approach.

B. Standards Compliance and Interoperability

Standards are a means to enable reusability, interoperability, and maintainability of software and hardware, which leads to higher productivity and helps aligning the enterprise's IT to its business. However, most available management approaches are based on non-standardized APIs or domain-specific languages (DSLs) which makes it difficult to provide and transfer the required knowledge. The presented approach tackles this issue by supporting two existing standards: (i) TOSCA is used to import standardized application descriptions that can be managed using SAMPs and AMIs while the (ii) BPEL standard is used to implement and execute the generated Management Workflows. In addition, Management Planlets and the generated workflows are implemented in BPEL, which is a standard to describe executable workflows. Thus, they are portable across standard-compliant BPEL engines. In addition, Planlets allow integrating different kinds of management technologies seamlessly [8][10]. As a result, the presented approach is agnostic to individual management technologies and Cloud providers, which supports interoperability and eases integration.

C. Automation

In Cloud Application Management, automation is of vital importance. Workflow technology enables automating the execution of management tasks and provides powerful features such as automated recoverability and compensation [9]. However, if these processes that implement a refined solution of the respective management pattern have to be created manually, it is not efficient and, in addition, error-prone. The presented approach automates the creation of workflows that implement a pattern's refined solution through the introduced Automated Management Idiom layer and the employed framework. Thus, only the choice when to trigger a management pattern is left to the human operator. This automation decreases the risk of human errors, which account for the largest fraction of failures and system downtimes in distributed systems [7][14]. In addition, the required technical knowledge to understand the different management technologies is not required at all as the whole process of determining the tasks to be performed and orchestrating the required technologies is fully automated.

D. Technical Complexity

Manually handling the technical complexity of pattern refinement for composite Cloud applications is a major issue due to heterogeneous and proprietary management technologies (cf. Section II). The presented approach tackles this issue by automating the whole refinement process from a pattern's abstract solution to the final executable workflow on two layers: (i) automating refinement by Automated Management Idioms and (ii) automated workflow generation using the Management Planlet Framework. The automated refinement removes the two manual tasks to specify concrete node and relationship types for abstract types and to add refinement-specific Management Annotations (cf. Figure 4 and Figure 7). Hence, the formerly required technical expertise on proprietary management idiosyncrasies such as Security Groups is no longer a mandatory prerequisite. Secondly, the orchestration of the different management technologies is completely handled by the Plan Generator that transforms the refined DASM fully automatically into an executable Management Workflow. Thus, this removes all manual steps and the only task that is left to the human user is choosing the AMI to be applied. This eliminates potential sources of errors on the technical layer [1].

E. Separation of Concerns

The presented approach separates concerns through splitting the process of selecting, creating, and automating a pattern, its refinement, and execution. Our approach enables IT experts to capture generic management knowledge in patterns that can be refined through Automated Management Idioms developed by specialists of certain areas. Thus, pattern creators and AMI developers are separate roles that are responsible for different kinds of knowledge: SAMP creators capture *generic* management knowledge, AMI creators refine this through implementing concrete *technology-specific* management knowledge. In addition, experts of low-level API orchestration that understand the different management technologies are able to automate their knowledge through implementing Management Planlets. Thus, even AMI creators do not have to deal with complex technology-specific details such as API invocations or parametrization: they only implement their knowledge declaratively, i. e., without defining the final API calls etc. This enables separating different responsibilities and concerns as well as a seamless integration of different kinds of knowledge.

F. Extensibility

The presented approach is extensible on multiple layers as it provides an explicit integration framework for patterns, idioms, and Planlets through using libraries. New Semi-Automated Management Patterns and Automated Management Idioms can be created based on a uniform Java interface and integrated into the system seamlessly by a simple registration at the library. All patterns and idioms in the library are considered automatically when an application shall be managed by comparing their Topology Fragments with the application's ETG. To extend the system in terms of management technologies, Planlets can be implemented for new node types, relationship types, or Management Annotations and stored in the Planlet Library. The framework's Plan Generator integrates new Management Planlets without further manual effort when processing DASMs.

VI. Related Work

Several works focus on automating application management in terms of application provisioning and deployment. Eilam et al. [27], Arnold et al. [28], and Lu et al. [29] consider pattern-based approaches to automate the deployment of applications. However, their *model-based patterns* are completely different from the abstract kind of patterns we consider in this paper. In their works, patterns are topology models which are used to associate or derive the corresponding logic required to deploy the combination of nodes and relations described by the topology, similarly to the Annotated Topology Fragments of Management Planlets. Mietzner [30] presents an approach for generating provisioning workflows by orchestrating "Component Flows", which implement a uniform interface to provision a certain component. However, these works focus only on provisioning and deployment and do not support management.

Fehling et al. [1][15] present *Cloud Application Management Patterns* that consider typical management problems, e. g., the "Stateless Component Swapping Pattern", which was automated in this paper. They propose to attach abstract processes to management patterns that describe the high-level steps of the solution. However, these abstract processes are not executable until they are refined manually for individual use cases. Their management patterns also define requirements in the form of architectural patterns that must be implemented by the application. These dependencies result in a uniform pattern catalog that interrelates abstract management patterns with architectural patterns. In addition, they propose to annotate reusable "Implementation Artifacts" to patterns that may assist applying the pattern, e. g., software artifacts or management processes. Their work is conceptually equal to our approach in terms of using processes to automate pattern application. However, most of the refinement must be done manually, which leads to the drawbacks discussed in Section I: management processes must be created in advance to be executable when needed. In addition, changing application structures caused by other pattern applications, for example, migration patterns, lead to outdated processes that possibly result in reimplementation. In Falkenthal et al. [31], we show how reusable solution implementations, e. g., Management Workflows, can be linked with patterns. However, also this approach requires at least one manual implementation of the concrete solution which is typically tightly coupled to a certain application structure.

Fehling et al. [32] also present a step-by-step pattern identification process supported by a pattern authoring toolkit. This authoring toolkit can be combined with our approach to create patterns and idioms that can be automated afterwards. Thus, the work of Fehling et al. provides the basis for creating AMIs out of patterns captured in natural text following this process.

Reiners et al. [33] present an iterative pattern evolution process for developing patterns. They aim for documenting and developing application design knowledge from the very beginning and continuously developing findings further. Non-validated ideas are documented already in an early stage and have to pass different phases until they become approved design patterns. This process can be transferred to the domain of Cloud Application Management Patterns. Our approach supports this iterative pattern evolution process as it helps to apply captured knowledge easily and quickly to new use cases. Thus, it provides a complementary framework to our approach that enables validating and capturing knowledge.

VII. Conclusion and Future Work

In this paper, we presented the concept of Automated Management Idioms that enables automating the refinement and execution of a pattern's abstract solution automatically to a certain use case by generating executable management workflows. We showed that the approach enables (i) applying the concept of patterns efficiently in the domain of Cloud Application Management through automation, (ii) abstracting the technical complexity of refinement, and (iii) reducing human intervention. The approach enables operators to apply various management patterns fully automatically to individual use cases without the need for detailed technical expertise. The prototypical validation, which extends the Management Planlet Framework, proves the concept's technical feasibility. In future work, we plan to investigate how Automated Management Idioms can be triggered automatically based on occurring events and how multiple patterns can be applied together.

Acknowledgment

This work was partially funded by the BMWi project CloudCycle (01MD11023).

References

[1] C. Fehling, F. Leymann, J. Rütschlin, and D. Schumm, "Pattern-based development and management of cloud applications." Future Internet, vol. 4, no. 1, March 2012, pp. 110–141.

[2] G. Hohpe and B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, 2003.

[3] A. Nowak et al., "Pattern-driven Green Adaptation of Process-based Applications and their Runtime Infrastructure," Computing, February 2012, pp. 463–487.

[4] C. Alexander, S. Ishikawa, and M. Silverstein, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977.

[5] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," Tech. Rep., July 2009.

[6] M. Armbrust et al., "A view of cloud computing," Communications of the ACM, vol. 53, April 2010, pp. 50–58.

[7] A. B. Brown and D. A. Patterson, "To err is human," in EASY, July 2001, p. 5.

[8] U. Breitenbücher, T. Binz, O. Kopp, and F. Leymann, "Pattern-based runtime management of composite cloud applications," in CLOSER. SciTePress, May 2013, pp. 475–482.

[9] F. Leymann and D. Roller, Production workflow: concepts and techniques. Prentice Hall PTR, 2000.

[10] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and J. Wettinger, "Integrated cloud application provisioning: Interconnecting service-centric and script-centric management technologies," in CoopIS. Springer, September 2013, pp. 130–148.

[11] S. Leonhardt, "A generic artifact-driven approach for provisioning, configuring, and managing infrastructure resources in the cloud," Diploma thesis, University of Stuttgart, Germany, November 2013.

[12] F. Leymann, "Cloud Computing: The Next Revolution in IT," in Proc. 52th Photogrammetric Week, September 2009, pp. 3–12.

[13] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer, January 2014.

[14] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in USITS. USENIX Association, June 2003, pp. 1–16.

[15] C. Fehling, F. Leymann, S. T. Ruehl, M. Rudek, and S. Verclas, "Service Migration Patterns - Decision Support and Best Practices for the Migration of Existing Service-based Applications to Cloud Environments," in SOCA. IEEE, December 2013, pp. 9–16.

[16] OMG, Business Process Model and Notation (BPMN), Version 2.0, Object Management Group Std., Rev. 2.0, January 2011.

[17] T. Binz, C. Fehling, F. Leymann, A. Nowak, and D. Schumm, "Formalizing the Cloud through Enterprise Topology Graphs," in CLOUD. IEEE, June 2012, pp. 742–749.

[18] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "Automated Discovery and Maintenance of Enterprise Topology Graphs," in SOCA. IEEE, December 2013, pp. 126–134.

[19] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and D. Schumm, "Vino4TOSCA: A visual notation for application topologies based on TOSCA," in CoopIS. Springer, September 2012, pp. 416–424.

[20] Apache Software Foundation. Apache Tomcat 7 API. [Online]. Available: http://tomcat.apache.org/tomcat-7.0-doc/

[21] Amazon Web Services. Elastic Compute Cloud API Reference. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/APIReference

[22] United Domains. Reselling API. [Online]. Available: http://www.ud-reselling.com/api

[23] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and M. Wieland, "Policy-Aware Provisioning of Cloud Applications," in SECURWARE. Xpert Publishing Services, August 2013, pp. 86–95.

[24] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. Wiley, 1996.

[25] OASIS, Web Services Business Process Execution Language (WS-BPEL) Version 2.0, OASIS, April 2007.

[26] OASIS, Topology and Orchestration Specification for Cloud Applications Version 1.0, May 2013.

[27] T. Eilam, M. Elder, A. Konstantinou, and E. Snible, "Pattern-based composite application deployment," in IM. IEEE, May 2011, pp. 217–224.

[28] W. Arnold, T. Eilam, M. Kalantar, A. V. Konstantinou, and A. A. Totok, "Pattern based soa deployment," in ICSOC. Springer, September 2007, pp. 1–12.

[29] H. Lu, M. Shtern, B. Simmons, M. Smit, and M. Litoiu, "Pattern-based deployment service for next generation clouds," in SERVICES. IEEE, June 2013, pp. 464–471.

[30] R. Mietzner, "A method and implementation to define and provision variable composite applications, and its usage in cloud computing," Dissertation, University of Stuttgart, Germany, August 2010.

[31] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, and F. Leymann, "From Pattern Languages to Solution Implementations," in PATTERNS. Xpert Publishing Services, May 2014.

[32] C. Fehling, T. Ewald, F. Leymann, M. Pauly, J. Rütschlin, and D. Schumm, "Capturing cloud computing knowledge and experience in patterns," in CLOUD. IEEE, June 2012, pp. 726–733.

[33] R. Reiners, "A pattern evolution process - from ideas to patterns." in Informatiktage. GI, September 2012, pp. 115–118.

A Method for Situational and Guided Information System Design

Dalibor Krleža
Global Business Services
IBM
Miramarska 23, Zagreb, Croatia
dalibor.krleza@hr.ibm.com

Krešimir Fertalj
Department of Applied Computing
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3, Zagreb, Croatia
kresimir.fertalj@fer.hr

***Abstract*—Model Driven Architecture is not highly used in current information system development practice. One of the reasons is that modeling languages are mostly used for documenting of the information system development and enhancement of communication within project teams. Without guidance, an information system design results in models of low quality, which cannot be used for anything more than documenting pieces of the information system. When it comes to model transformation, project team members usually reuse predefined transformations included in a modeling tool. Models of low quality that are not traceable and not complete are hard to transform. Model quality cannot be high if modeling activities are not guided and constrained. Guidance and constraints can be imposed through project activities led by a senior designer responsible for model quality. In this article, a method for situational modeling guidance is presented. This method is adaptable and situation dependent. Implemented within a modeling tool, the method should allow project team members responsible for model quality to give guidance and constraints, and to ensure model quality through the modeling tool.**

Keywords-modeling; guidance; design; pattern; transformation.

I. INTRODUCTION

The Model Driven Architecture (MDA), standardized by the Object Management Group (OMG) [1], is an information system design approach based on models and model transformations. Using MDA, an information system is designed through several models of different abstraction levels, from business oriented models to technical and platform specific models. MDA defines three different types of models having different levels: abstract and business oriented Computational Independent Model (CIM), technically oriented Platform Independent Model (PIM), and very detailed Platform Specific Model (PSM).

Model transformation is a key procedure in MDA. According to the specification [1], "model transformation is the process of converting one model to another model". Model transformation can be done manually or automatically. Manual model transformation is more common than we think. It is not unusual for a designer to start modeling from scratch by using models delivered earlier in the project. Such an approach is defined within various design and development methodologies. Chitforoush, Yazdandoost and Ramsin [2] are giving an overview of MDA specific methodologies. Most of these methodologies were developed for specific projects. Some generic design and development methodologies, such as Rational Unified Process (RUP) [3][12], also rely on model based design. However, basic purpose of methodologies is to help organize projects, giving guidance for project activities and deliverables, leaving execution to project team members.

When a modeling language is structured and formal enough, automatic transformation can be used. The Meta Object Facility (MOF), standardized by the OMG and described in [5], is a metalanguage for modeling languages that can be transformed automatically. Automatic transformation takes artifacts of a source model and converts them into artifacts of a target model by using transformation mapping. Transformation can be additionally used to establish relationships between models, or to check consistency of artifacts between a source and target model. Czarnecki and Helsen [4] elaborate a number of model transformation approaches and basic features of transformation rules. Most used are graph based transformations and transformation languages. Transformation languages can be declarative or imperative. The OMG standardized group of MOF based transformation languages named Query/View/Transformation (QVT) [7]. QVT Relational language (QVT-R) is a typical example of a declarative approach with a graphical notation. QVT Operational language (QVT-O) is an example of an imperative approach.

The focus of this article is delivery of the models that are of high quality. In order to understand what this means, we can use one of the existing quality models. One example is the quality model given by Lange and Chaudron [8]. Relying only on methodology guidance will not necessarily produce a model of high quality, because it allows designers to focus on wrong aspects and details within the model. The result can be a model of poor quality, problems with traceability and inability to transform or analyze created model. One way to solve these problems is by appointing a senior designer to the design lead role. The design lead responsibilities are to establish modeling guidance and constraints, oversee modeling work, check delivered models and to ensure model quality. According to the quality model [8], this means that all models are traceable, complete, consistent and correspondent to the information system. Establishing modeling constraints means imposing patterns that need to be used during the information system design.

In this article, a method for automated modeling guidance and imposing constraints is proposed. The proposed method utilizes existing specifications such as MDA, MOF, UML and QVT to achieve a guided process of an information system design, supported by model transformation. The proposed method will be extensible in order to naturally fit existing design and development methodologies. The purpose of the method is to ease communication between a design lead and his team members and to enable management of an information system design work through usage of a modeling tool.

In Section II, a modeling space is defined. The modeling space is a way how to combine all models of an information system together, giving them relationship and defining their purpose. In the same Section, a relationship between pattern instances and models is given. In Section III, current modeling practice in the context of methodologies is discussed, which helps understand how the pattern instances are created during the project. In Section IV, an overview of the pattern instance transformation is given. The pattern instance transformation is essential for the method proposed in this article. In Section V, the tracing and transformation language is defined. This language is used to bind pattern instances together and help to establish tracing between model artifacts. In Section VI, an overview of the method for situational and guided information system design is given.

II. Modeling Space

A modeling space can be represented as a three dimensional space containing all possible models of a designed information system. The modeling space must follow MDA philosophy, support different levels of abstraction given in MDA specification, and classification of the containing models.

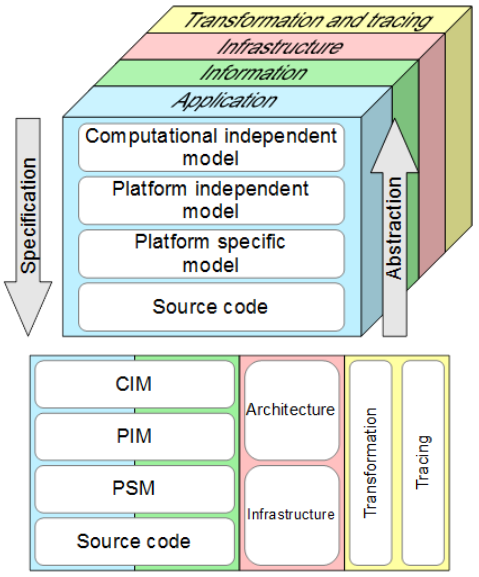


Figure 1. Structure of a modeling space

The proposed modeling space presented in Figure 1 is in three dimensions because it contains different layers representing respective aspects or viewpoints of the designed information system. The modeling space contains four layers. The application layer is comprised of models with the business logic. The information layer is comprised of information and data models. Models containing architecture details and infrastructure nodes are placed in the infrastructure layer. And finally, there needs to be a specific layer for transformation and tracing models. Of course, a number of layers and their purpose depend on a set of models representing an information system design. One model can belong to multiple layers. For example, a model containing requirements can easily be considered for application, information and infrastructure related. The proposed modeling space must also support a clear distinction between abstract and detailed models. Abstract and computing independent models are placed on top of each layer. Models with more details are closer to the bottom of the layer. Figure 1 shows the placement of different MDA model types in the proposed modeling space.

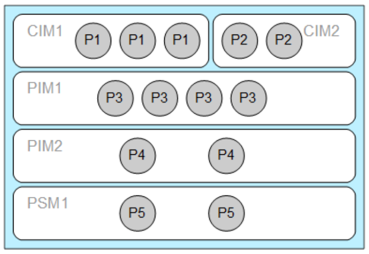


Figure 2. Models and pattern instances placed in the application layer of the modeling space

Each model is a set of artifacts. These artifacts originate from a modeling language, such as UML. A set of models together represent a design of an information system. However, there are building blocks between single artifact and a whole model that are meaningful for designers. These building blocks are patterns. Example of repeating patterns within different models is given in Figure 2. CIM1 contains repeating sets of model artifacts that can be interpreted as requirements, CIM2 contains business processes, PIM1 use cases, PIM2 components, and PSM1 implementation of components defined in PIM2.

Which patterns will appear in the modeling space depends on the design of an information system. Computational independent patterns are usually created early in the project and they depend on used architecture as well as how business analysis is performed. These high level abstract patterns have the biggest impact on the design of an information system. Platform independent patterns are derived from architecture and computational independent patterns. They represent an elaboration of computational independent patterns within an architectural context. The most detailed are platform specific patterns that represent the implementation of platform independent patterns for a specific infrastructure yielded by the previously determined architecture.

In order to establish the method proposed in this article, a library of modeling patterns and transformations must be established. Modeling patterns can be determined in several different ways. Gamma, Helm, Johnson and Vlissides [9] propose a list of basic object-oriented patterns visualized in the UML. Hohpe and Woolf [10] propose a list of enterprise integration patterns. Enterprise integration patterns are more abstract than object-oriented patterns.

Collecting modeling patterns from existing models of the already developed information system is another way. It can be done manually or automatically by detecting repetitions in existing models. Detection itself can be done by the graph matching method [11]. However, this is just a part of a collection process. Rahm et al. [15] propose graph matching method for detection of cloned fragments in graph based models. According to their definition, repetitive fragments that are similar enough can be considered for clones or patterns. A similar approach can be applied to UML models.

And finally, as already mentioned, modeling patterns can be a great way how to give a sense of direction and cooperation to a team of designers. A pattern is a class, a blueprint that binds one or more modeling artifacts together. Application of a pattern means his instantiation within at least one model in the modeling space. Applying the modeling pattern does not mean that the modeling is completed. Adding details and further elaboration of the pattern instance is needed, in order to give it enough details to fit an information system design.

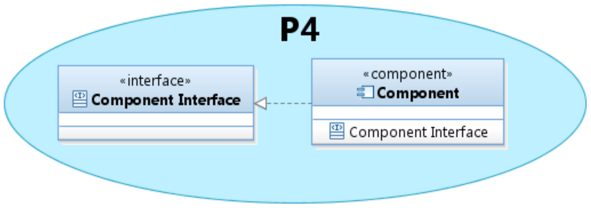


Figure 3. Example of a simple modeling pattern: component and interface

Figure 3 presents a pattern that is comprised of an empty interface and a component. After applying this pattern a pattern instance is created. Further elaboration of the pattern instance must add interface details, operations and attributes, subcomponents and additional interfaces.

III. Mapping between Methodology and Pattern Usage

CIMs are usually created very early in the project. In the RUP, business models are created in the Inception phase. It means that selecting and applying CIM related patterns and further elaboration can be done very early in the project. These patterns will be classified as functional requirements, non-functional requirements, business processes, or business use cases. Idea is to have these patterns and related transformations ready for use in the modeling library that is used for the project. Elaboration of newly created pattern instances in CIMs can be done in the Inception phase.

PIMs, part of the PSMs, architecture models and infrastructure models, are created in the Elaboration phase. In this phase, we do most of an information system design and take the most important decisions. In the Elaboration phase, patterns used in CIMs are guidance for choosing patterns that will be used next. For example, usual patterns that could be used here contain use cases, components and nodes.

The PSM is usually the last step in the design of an information system. The ultimate goal is to get the source code and deployment units. Therefore, the PSM must contain pattern instances that define a sufficient level of details for transformation into the source code, in a way that there is less work as possible for programmers. Pattern instances in the PSM are mostly implementation of pattern instances in the PIM. For example, in the Component Based Modeling (CBM), the PSM contains platform specific implementations of components defined in the PIM.

As the design of an information system advances through the project, designers can create new pattern instances or elaborate existing ones, as presented in Figure 4. A new pattern instance can be created to document business need, reflect already existing functionality that will be reused, or by transforming from already existing pattern instance in the modeling space. Transformation between pattern instances will probably be the most used option. Elaboration of the existing pattern instances is also very important. Once a new pattern instance has been created, it must be elaborated in subsequent project activities.

IV. Pattern Instance Transformation

In the MDA specification [1], various different model-to-model transformation examples can be found. Transformation can be done within the same model, between two different models, for model aggregation, or model separation. Grunske et al. [6] are presenting important notion of "horizontal" and "vertical" transformations. Horizontal transformation is done between models of the same abstraction level. Typical horizontal transformation is PIM to PIM, or PSM to PSM. Any transformation within the same model is also a horizontal transformation. Vertical transformation is done between models of different

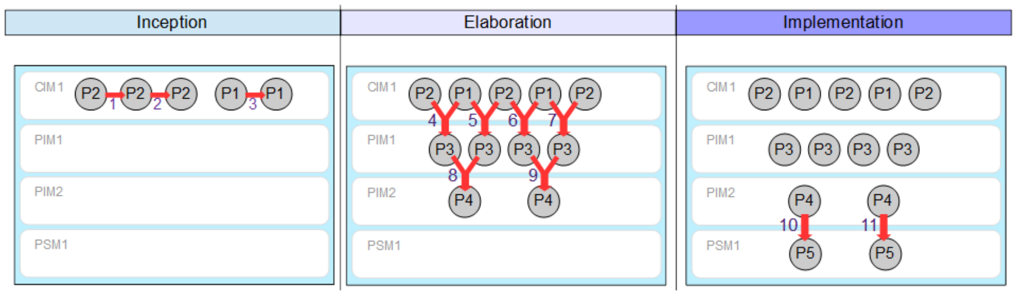


Figure 4. RUP and advancement through a design of an information system

abstraction levels, or from a model to the source code. A transformation from PIM to PSM, or from PSM to the source code is vertical transformation.

Model transformation is the procedure for translating source model into target model. A modeling space can be defined as a finite set of models $M_S = \{M_1, M_2, \dots, M_n\}$. Each model is a finite set of artifacts $M_i = \{a_1, a_2, \dots, a_m\}$. A transformation is a function $tr: M_S \rightarrow M_S$ that takes a set of artifacts ar_{So} from a set of source models $So \subseteq M_S$ such that $ar_{So} \subseteq \bigcup So$, analyses this set of artifacts and translates them into another set of artifacts ar_{Ta} in a set of target models $Ta \subseteq M_S$, such that $ar_{Ta} \subseteq \bigcup Ta$. Transformation can be done within the same model $So = Ta = M_i$, or between two disjunctive sets of models $So \neq Ta$. Since a transformation can have multiple models from source and target side, these sets do not need to be disjunctive $So \cap Ta \neq \emptyset$, meaning that the transformation can include same model M_i on source and target side, or $M_i \in So \wedge M_i \in Ta$. A transformation can use the same source and target artifacts, meaning that $ar_{So} \cap ar_{Ta} \neq \emptyset$ when $So \cap Ta \neq \emptyset$, or it can use two disjunctive sets of artifacts $ar_{So} \cap ar_{Ta} = \emptyset$.

From a pattern point of view, each pattern instance is a set of model artifacts. This definition is valid for cross model pattern instances as well. All pattern instances in the modeling space M_S form a finite set of pattern instances $M_P = \{P_i : 0 < i \leq m \wedge P_i \subseteq \bigcup M_S\}$. In this context, transformation is a function $tr: M_P \rightarrow M_p$. Such transformation takes a set of source pattern instances $p_{So} \subseteq M_p$, analyses all artifacts in these instances and translates them into artifacts that form a set of target pattern instances $p_{Ta} \subseteq M_p$. Figure 5 shows an example of transformation application to a cross model pattern instance.

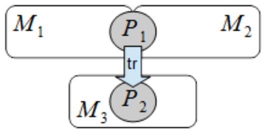


Figure 5. Cross model pattern instance and transformation

Every transformation can be encapsulated in a black box implementation. Such an approach is used in [7] along with the QVT specification. However, taking a step back and observing the QVT specification as one of the transformation approaches, every transformation can be defined as a black box having an interface that depends on the context of transformation usage.

A. Transformation rules

Czarnecki and Helsen [4] are giving important features of transformation rules. Since a transformation can be implemented in many different ways, we must observe it in more abstract and generic way. No matter if we use a declarative or imperative approach, each transformation is a set of rules that creates a relationship between a set of source artifacts and a set of target artifacts. Features and principles given in [4] can be applied to these transformation rules.

A transformation written in QVT-R [7] has two different modes: checking mode and enforcement mode. In the checking mode, transformation rules can be used to validate correctness and completeness of involved pattern instances. In the enforcement mode, transformation rules can be used for creating, updating, or deleting artifacts in target pattern instances, in order to reflect all the details found in source pattern instances.

1) Validation of pattern instances and imposing constraints

When the transformation is applied, execution of the transformation must perform several different tasks.

As the first step, transformation must validate that supplied source pattern instances are matching expected source side of the transformation. A set of mandatory transformation rules must validate source pattern instances. If all mandatory transformation rules are satisfied from the source side then the transformation can be applied to a supplied source, i.e., the transformation can be applied to the source pattern instance that contains all artifacts needed by the mandatory transformation rules.

The second step is the creation of the target pattern instances. Transformation rules must create all target pattern instances and their artifacts. Every pattern is characterized by the mandatory artifacts that define the essence of the pattern, or what makes this pattern different from other patterns. Not all artifacts created by the transformation must be considered for mandatory. Mandatory artifacts in the target pattern instances are created by the mandatory transformation rules. However, not all mandatory transformation rules must create mandatory artifacts in the target pattern instances.

The last step is to create a set of constraints that will disallow designers to change some of the artifacts in the involved pattern instances. Transformation binds involved pattern instances together by imposing constraints on their artifacts. Each pattern instance can be bound with other pattern instances through several different transformations. Constraints are imposed by the mandatory transformation rules.

Imposed constraints are used to limit designer changes in the modeling space to prevent:

1. Violating correctness and completeness of the pattern instances by changing their mandatory artifacts. Obviously, all mandatory artifacts must be constrained.
2. Breaking transformation binding by changing artifacts that are satisfying source and target side of the mandatory transformation rules. In this case, constrained artifacts do not need to be mandatory.

If we observe a target pattern instance made of l artifacts $P_i = \{a_1, a_2, \dots, a_l\}$, a subset $MP_i \subseteq P_i$ is considered for a set of mandatory artifacts of the pattern instance P_i. If we have a finite set of applied transformations $TP_i = \{tr_1, tr_2, \dots, tr_k\}$ having P_i as an involved pattern instance, we can derive a mapping function $C: TP_i \rightarrow X$, where $X \subseteq P_i$ is a set of artifacts in P_i constrained by a transformation $tr_x \in TP_i$. In the context of the previous definition about difference between mandatory and constrained artifacts, we can conclude that P_i can be in a situation where $C(tr_j) \cap C(tr_k) = \emptyset \wedge j \neq k$, and $C(tr_j) = MP_i \wedge C(tr_k) \cap MP_i = \emptyset$. Finally, $MP_i \subseteq \bigcup_{j=1}^{k} C(tr_j)$ means that not all constrained artifacts need to be mandatory, but all mandatory artifacts are constrained since we want to preserve a pattern definition.

Each pattern instance can be a result of several different pattern instances done earlier in the same project, or it can be a reason for creating several new pattern instances later in the same project. Several good examples can be found in [9]: a facade associated with a web service client can be used as a mediator between two different subsystems. In this example, the mediator is the pattern whose instance is bound by two different transformations.

2) *Pattern instance elaboration*

A transformation can be used to perform changes on involved pattern instances. This approach is used when new pattern instances are created, or existing instances are updated or deleted. Even when two pattern instances are bound with a transformation, the source pattern instance can be elaborated by adding new details and artifacts. A transformation can be made so that these newly added details automatically update target pattern instances. Artifacts that are not constrained by one of the binding transformations are handled by optional transformation rules responsible for spreading of elaboration details. Bidirectionality is a very important transformation aspect described in [7] and [13]. While transformation might constrain changes of some artifacts in target pattern instances, changes of unconstrained artifacts in pattern instances across the modeling space are encouraged. Such changes must be propagated throughout the modeling space, wherever transformation between pattern instances allows it. This propagation must be automatic and seamless.

3) *Top-level pattern instances*

Top-level pattern instances do not have predecessors. These pattern instances can be modeled manually by a designer without using any transformation, or they can be created by using a transformation. Such transformation does not need to have input source pattern instances. In order to give the transformation some instructions, input parameters can be used. Transformations that create only target pattern instances can be used both for validation and enforcement purposes. All transformation rules in this transformation are mandatory transformation rules that create an initial version of target pattern instances and impose constraints on them. However, these constraints must allow elaboration of newly created top-level pattern instances in order to allow adding needed details. Functional or non-functional requirements are typical examples of top-level patterns. An external service definition is another example of such pattern.

In the example in Figure 6, pattern instance P_1 is made of *CodebookComponent* and related interface. All mandatory artifacts are marked with red color. Artifacts added in the elaboration of P_1 are marked with brown color. Mandatory artifacts within P_1 are all we need to declare a component. Transformation tr_1 mandatory rules are responsible for translation of mandatory artifacts from P_1 to P_3. The artifact

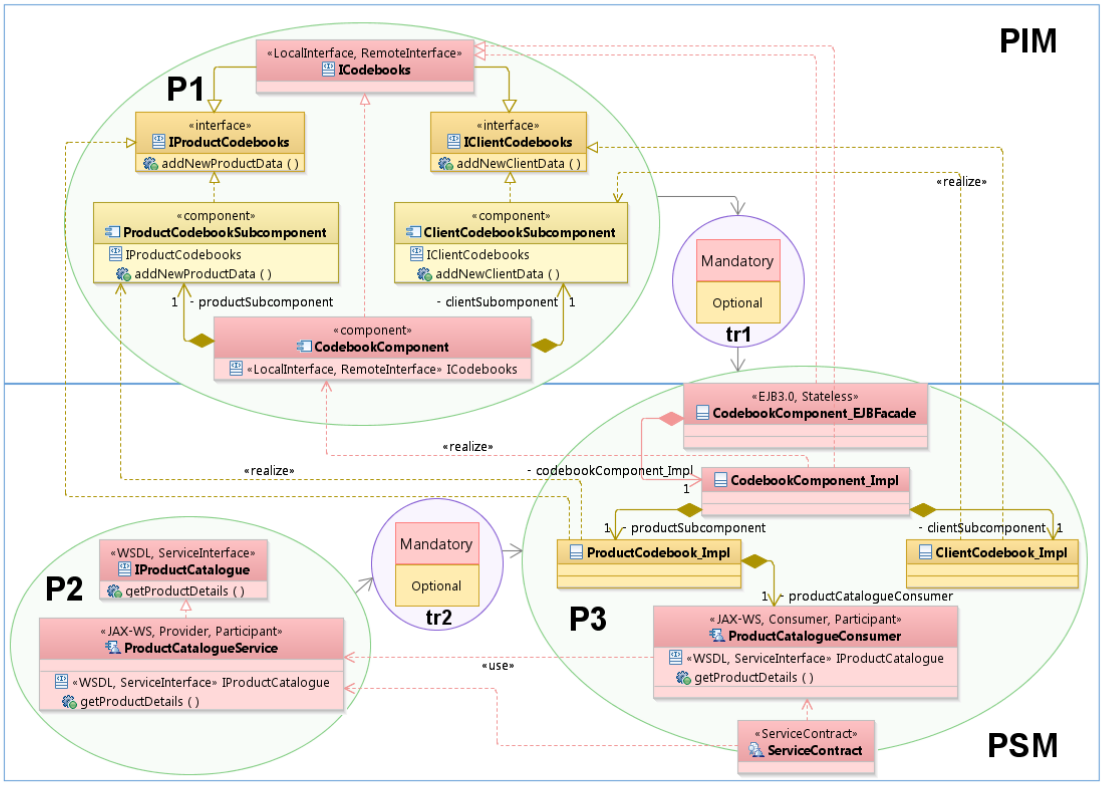

PIM
P1
«LocalInterface, RemoteInterface» ICodebooks
«interface» IProductCodebooks
addNewProductData ()
«interface» IClientCodebooks
addNewClientData ()
«realize»
«component» ProductCodebookSubcomponent
IProductCodebooks
addNewProductData ()
«component» ClientCodebookSubcomponent
IClientCodebooks
addNewClientData ()
1 - productSubcomponent
- clientSubomponent 1
«component» CodebookComponent
«LocalInterface, RemoteInterface» ICodebooks
Mandatory
Optional
tr1
«EJB3.0, Stateless» CodebookComponent_EJBFacade
«realize»
«realize»
- codebookComponent_Impl
1
CodebookComponent_Impl
1 - productSubcomponent
- clientSubcomponent 1
ProductCodebook_Impl
ClientCodebook_Impl
1 - productCatalogueConsumer
P2
«WSDL, ServiceInterface» IProductCatalogue
getProductDetails ()
«JAX-WS, Provider, Participant» ProductCatalogueService
«WSDL, ServiceInterface» IProductCatalogue
getProductDetails ()
Mandatory
Optional
tr2
P3
«JAX-WS, Consumer, Participant» ProductCatalogueConsumer
«WSDL, ServiceInterface» IProductCatalogue
getProductDetails ()
«use»
«ServiceContract» ServiceContract
PSM

Figure 6. Example of pattern instance transformation

created within P_3 is EJB3 facade as realization of *CodebookComponent*. Transformation tr_1 also created relationships between mandatory artifacts of P_3 and P_1 pattern instances. An information system designer additionally elaborated P_1 and added *ProductCodebookSubcomponent* and *ClientCodebookSubcomponent* together with related interfaces. Transformation tr_1 optional rules translated these subcomponents from P_1 into classes of P_3 and created new relationships between optional artifacts of P_3 and P_1. P_1 and P_3 are bound with transaction tr_1. It means that mandatory artifacts of P_3 are constrained by tr_1 and cannot be changed unless corresponding artifacts in P_1 are changed. Introducing relationships between P_3 and P_1, such as interface realization, simplifies the propagation of interface changes between these two pattern instances. Propagation of interface changes is a matter of transformation tr_1!

The information system designer's final decision was to reuse the existing service to read product data from a product catalog information system. P_2 is the pattern instance that represents the product catalog service provider. This pattern instance can be created by another transformation from WSDL. Transformation tr_2 is used to translate mandatory artifacts from P_2 representing the service provider, into the set of artifacts for P_3 representing the product catalog service consumer. P_3 must be further elaborated in order to connect *CodebookComponent* realization with the product catalog service consumer.

4) *Transformation applicability*

As already defined, a transformation takes a set of modeling space artifacts and translates them into another set of artifacts. Earlier definition shows that the transformation can include pattern instances as artifact containers. The size of a pattern instance can be one artifact, up to a whole model. A pattern instance can also be a set of artifacts coming from different models within the modeling space. In order to use transformation, source side of it must be satisfied. Precisely, mandatory transformation rules source side must be satisfied in order for the transformation to be able to create a set of target artifacts and impose constraints on them. If the transformation is applied to a set of pattern instances and a set of source pattern instances satisfies source side of the mandatory transformation rules, the transformation is applicable to this set of pattern instances.

Transformation and related transformation rules, especially if they are written in a declarative programming language such as QVT-R, are logic programs [14]. Transformation tr can be defined as a logic program P, comprised of mandatory and optional set of rules on source and target side. Applicability of a transformation can be derived only from source mandatory rules. If we take a finite set of the mandatory source rules $MSR = \{msr_1(X), msr_2(X), \dots, msr_n(X)\}$, where $X = p_{So}$ is a set of terms, then the applicability of the transformation can be expressed as $\mathrm{A(X)} \leftarrow msr_1(\mathrm{X}) \wedge msr_2(\mathrm{X}) \wedge \dots \wedge msr_n(\mathrm{X})$. Each mandatory source rule is comprised of atoms for checking artifacts within a source pattern instance set $msr_i(X) \leftarrow a_1(y_1, X) \wedge a_2(y_2, X) \wedge \dots a_m(y_m, X)$, where $y_i \in X$. The applicability defined this way can only determine whether a transformation can be applied to a set of source pattern instances or not. Another way is to define a measure of the applicability by expressing percentage of mandatory transformation rules that are satisfied. A finite set of satisfied rules is $MSR_S = \{r(X) \in MSR: r(X) = true\} \subseteq MSR$. The measure of the applicability can be defined as $A_m = |MSR_S|/|MSR| * 100$, or percentage of satisfied mandatory source transformation rules. This measure can help a designer to see which transformations in the modeling library are close to being applicable and what are the differences. Consulting the measure of the transformation applicability is one aspect of the design guidance.

Validation of involved pattern instances is a similar concept to the applicability of the transformation, but it must involve both source and target pattern instances.

V. TRANSFORMATION AND TRACING LANGUAGE

Relationship between model artifacts and a pattern instance is not established within the UML. Although there is the *Package* element defined within the UML, its purpose is not the same as "pattern instance" defined earlier in this article. Also, application of transformation and imposing constraints on target pattern instances must leave some trail. Creation of a Transformation and Tracing Model (TTM), automatically or manually, can help to resolve before mentioned issues. Every time a new pattern instance is created, new artifact is added into TTM representing this pattern instance. All model artifacts belonging to this pattern instance are automatically bound to it. It can be the result of the transformation, or it can be done manually meaning that a modeling tool must have capabilities for it. Also, each time when a transformation is used, this transformation is added to TTM including all relationships between pattern instances and used transformation. Each time a transformation is used, and this transformation is imposing constraints on involved pattern instances, these constraints are added to pattern instances in TTM and bound to the transformation that created them, since these constraints are the result of the transformation. In order to do this modeling, a Transformation and Tracing Language (TTL) must be defined. The UML and the TTL must be compatible, meaning that they must have a common M0 ancestor [13]. Therefore, the TTL must be a MOF metamodel. An overview of the TTL is presented in Figure 7.

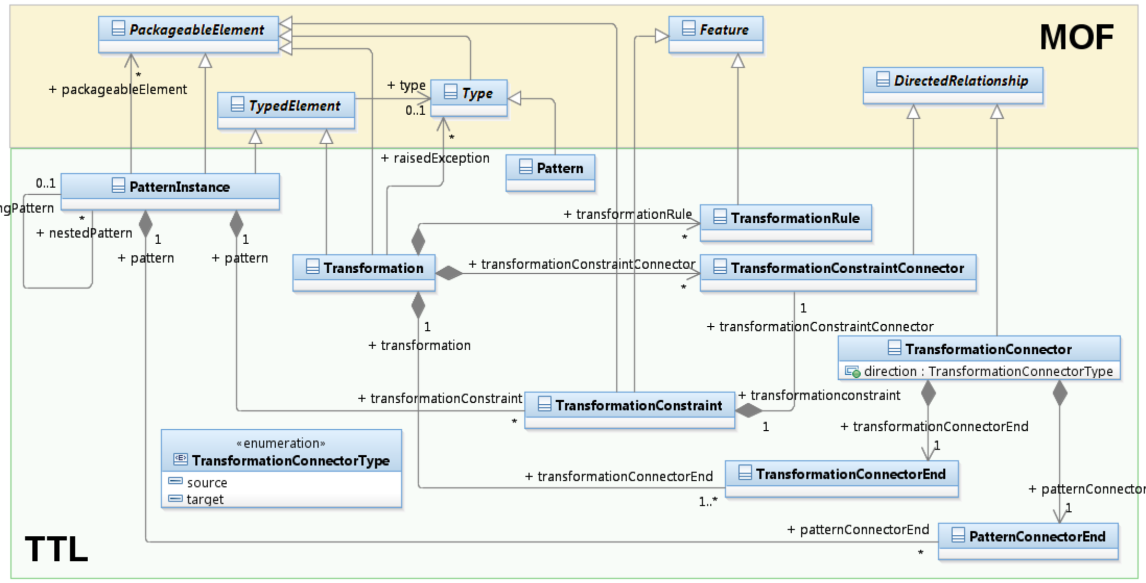


Figure 7. TTL definition

The TTL is having the following elements:

- *Pattern* - A pattern type. Allows classification of pattern instances.
- *PatternInstance* - An element similar to the UML *Package* element. Represents a container for model artifacts. This element is defined by its name and type. Pattern type (or class) can be very helpful when constructing transformation rules and it can impact the transformation applicability since transformations can be applied to the pattern instances of specific types.
- *Transformation* - An element defined by its name and type, representing applied transformation. It contains transformation rules used in the transformation. The transformation must be connected to a set of source and target pattern instances, being connected to at least one target pattern instance. Connector direction is determined by the *TransformationConnectorType* enumeration.
- *TransformationConnector*, *TransformationConnectorEnd*, *PatternConnectorEnd* - A connector is a directed relationship between a pattern instance and a transformation. Connector direction must have a visual notation. If the connector is directed from the pattern instance to the transformation, it represents the source pattern instance in the context of the transformation. If the connector is directed from the transformation to the pattern instance, it represents the target pattern instance in the context of the transformation. Connector end elements represent the point of touch between the connector and the pattern instance, or the connector and the transformation.
- *TransformationConstraint* - An element defined by its name, representing a constraint on members of a pattern instance imposed by used transformation. This element is contained by the pattern instance and connected to the transformation responsible for the creation of the constraint. This element is the result of the transformation and can be used to validate the pattern instance correctness and completeness.
- *TransformationConstraintConnector* - A relationship between resulting constraint and the transformation that created it, directed from the transformation to the constraint. Each constraint can be imposed by only one transformation, but one transformation can impose multiple constraints within multiple pattern instances.

In the TTM example in Figure 8, brown artifacts were created before $tr1$ was applied. We can say that pattern instances $p1$ and $p2$ were designed manually. Green artifacts are produced by the transformation $tr1$. Actions taken during an information system design are automatically stored in a TTM for multiple purposes: preserving correctness and completeness of the modeling space, reconstruction of activities in the design process, and analysis of the resulting design work.

VI. GUIDANCE

So far, this article gave only insights into elements needed to establish the method for situational information system design guidance. How to explain designers what is preferred designing practice and how an information system design should look alike? Many companies have well established design practices, from methodology, project activities and modeling point of view. Selection of architectures, technology and practical experience gives a

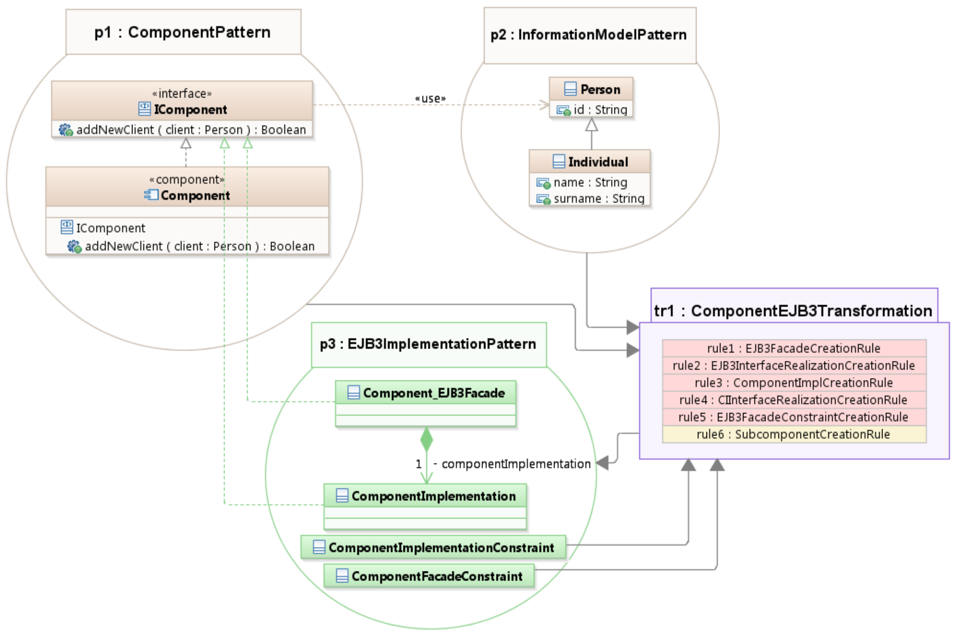

p1 : ComponentPattern
«interface»
IComponent
addNewClient (client : Person) : Boolean
«component»
Component
IComponent
addNewClient (client : Person) : Boolean
«use»
p2 : InformationModelPattern
Person
id : String
Individual
name : String
surname : String
tr1 : ComponentEJB3Transformation
rule1 : EJB3FacadeCreationRule
rule2 : EJB3InterfaceRealizationCreationRule
rule3 : ComponentImplCreationRule
rule4 : CIInterfaceRealizationCreationRule
rule5 : EJB3FacadeConstraintCreationRule
rule6 : SubcomponentCreationRule
p3 : EJB3ImplementationPattern
Component_EJB3Facade
1
- componentImplementation
ComponentImplementation
ComponentImplementationConstraint
ComponentFacadeConstraint

Figure 8. TTM example

company starting point. The method proposed in this article simply takes this experience and allows the company to document their design practices within a modeling tool.

A. *Guidance given through a modeling library*

We already mentioned that a modeling library is comprised of patterns and transformations. Since transformation binds two patterns instances together (as described in the section III), selection of a transformation imposes a selection of involved patterns. Similarly, selection of patterns imposes a selection of potentially applicable transformations.

Applicability and measure of applicability are important transformation features that can be used to give guidance. A designer can elaborate a model or pattern instance and occasionally check for transformations that are applicable to the model or pattern instance he is working on. If there is no transformation currently applicable, the designer can check transformations that are nearly applicable and the gap that needs to be closed in the model or pattern instance in order for this nearly applicable transformation to become applicable. Of course, many designers have enough experience to know which transformation would need to be used next even before modeling of the pattern instance is finished. If there is a problem with selected transformation, and rules in the transformation are not correct, meaning that the transformation will never become applicable, this particular transformation can be changed as part of company's design practice evolution.

As already stated before, some transformations can be used exclusively to create new top-level pattern instances. Such transformations are used to create mandatory and optional artifacts in the target pattern instances. Optional artifacts initially created in the target pattern instance can fulfill requirements for the next transformation to become applicable. Further elaboration of this pattern instance can add needed details. It means that applying a transformation can result in a chain of transformations and creation of new pattern instances if a modeling tool is permitted to execute applicable transformations automatically.

Another way is a selection of transformations from the modeling library used in the project. A design lead can manage a set of allowed transformations for his project, limiting designer's choice of applicable or nearly applicable transformations. For example, the architectural decision to use JAX-WS web services will influence the choice of transformations for the project. Similarly, the design lead can manage a set of allowed patterns implicitly by imposing a set of allowed transformations.

B. *Guidance given through a model*

More specific guidance can be given through a specific model that predetermines patterns and transformations used in an information system design process. Such model is created *a priori*, before the start of the design activities. Creation of the guidance model is an ongoing activity through the whole project. The TTL can be used for this purpose. This model must represent a selection of allowed pattern types and related transformations. Such model can be used by the designer to check guidance, or directly by a modeling tool for selection of allowed transformation list for particular pattern type. It is the same approach as in the previous Section, with additional visualization of selected design practice for the ongoing project.

VII. CONCLUSION AND FUTURE WORK

MDA is having two major practical problems: designers have too much freedom while creating information system design models and transformation scope can be very ambiguous. Usage of a pattern as the main building block for an information system design is a well known approach. In the context of this article, design of an information system is done block by block, allowing the design lead to choose blocks to be used. Such approach allows the design team to use past positive experience to select or define best patterns for the information system they are designing. Another very important element of this method is the usage of transformations to create pattern instances. Transformations must be perceived as the behavioral part of the method. Applicability and measure of the applicability are very important features of the transformation given in this article. They enable controlled application of transformations, which represents guidance for the design team.

Of course, designers are still free to model according to their preferences, as long as they are within boundaries imposed by the proposed method, which is assured by an optional part of each transformation helping team to keep artifacts of bound pattern instances synchronized. Bidirectionality feature of the transformation helps to reflect changes in both directions. Chain of pattern instances can be easily updated through transformations used to form the chain. Since a pattern instance is supposed to have significantly smaller scope than a model, keeping several pattern instances synchronized during elaboration should be much easier than with big models.

Current modeling tools are introducing a high level of automation. This automation is mostly related to elements of the modeling languages supported by a modeling tool. Changing the modeling tool behavior to follow the model in a modeling space is needed feature.

This article is giving only the main idea that can be significantly improved and extended. There are still opportunities for improvement of the mapping between proposed method and methodologies. Also, the TTL defined in this article can be extended with elements for interaction with modeling tool, model analysis capabilities and model quality assessment. Interaction between a TMM and a modeling tool can be extended with modeling events, allowing a design lead to define modeling tool actions associated with patterns and transformations. For example, a TTM can include an event handler on a pattern that can be triggered by the modeling tool when a new subcomponent is added into a pattern instance. The event handler initiates execution of a specific transformation that automatically adds interface and interface realization relationship for this newly added subcomponent.

REFERENCES

[1] OMG, MDA. "Guide, Version 1.0.1, 2003." Object Management Group.

[2] F. Chitforoush, M. Yazdandoost, and R. Ramsin, "Methodology support for the model driven architecture." Proceedings of the 14th Asia-Pacific Software Engineering Conference, IEEE, Dec. 2007, pp. 454-461, doi: 10.1109/ASPEC.2007.58.

[3] I. Jacobson, G. Booch, and J. E. Rumbaugh, "The unified software development process-the complete guide to the unified process from the original designers." Addison-Wesley, 1999.

[4] K. Czarnecki and S. Helsen, "Classification of model transformation approaches." Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, vol. 45, no. 3, Oct. 2003, pp. 1-17.

[5] OMG, "Core Specification, Version 2.4.1, 2011." Object Management Group.

[6] L. Grunske, L. Geiger, A. Zündorf, N. Van Eetvelde, P. Van Gorp, and D. Varro, "Using graph transformation for practical model-driven software engineering." Model-driven Software Development, Springer Berlin Heidelberg, 2005, pp. 91-117, doi: 10.1007/3-540-28554-7_5.

[7] OMG, "Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT), Version 1.1, 2011." Object Management Group.

[8] C. F. J. Lange and M. R. V. Chaudron, "Managing model quality in UML-based software development." 13th IEEE International Workshop on Software Technology and Engineering Practice, IEEE, Sep. 2005, pp. 7-16, doi: 10.1109/STEP.2005.16.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Elements of reusable object-oriented software." Addison-Wesley, 28th edition, 2004.

[10] G. Hohpe and B. Woolf, "Enterprise Integration Patterns." Addison Wesley, 2004.

[11] M. Gupta, R. Singh Rao, and A. Kumar Tripathi, "Design pattern detection using inexact graph matching." 2010 International Conference on Communication and Computational Intelligence, IEEE, Dec. 2010, pp. 211-217.

[12] P. Kroll and P. Kruchten, "The rational unified process made easy: a practitioner's guide to the RUP." Addison-Wesley, 2003.

[13] A. G. Kleppe, J. B. Warmer, and W. Bast, "MDA explained, the model driven architecture: Practice and promise." Addison-Wesley, 2003.

[14] A. Van Gelder, K. A. Ross, and J. S. Schlipf, "The well-founded semantics for general logic programs." Journal of the ACM (JACM), ACM, vol. 38, no. 3, Jul. 1991, pp. 619-649, doi: 10.1145/116825.116838.

[15] N. H. Pham, H. A. Nguyen, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Complete and accurate clone detection in graph-based models." Proceedings of the 31st International Conference on Software Engineering, IEEE, May 2009, pp. 276-286, doi: 10.1109/ICSE.2009.5070528.

An Analytic Evaluation of the SaCS Pattern Language – Including Explanations of Major Design Choices

André Alexandersen Hauge
Institute for energy technology, Halden, Norway
University of Oslo, Norway
andre.hauge@hrp.no

Ketil Stølen
SINTEF ICT, Oslo, Norway
University of Oslo, Norway
ketil.stolen@sintef.no

Abstract—**In this paper, we present an analytic evaluation of the Safe Control Systems (SaCS) pattern language for the development of conceptual safety designs. By a conceptual safety design we mean an early stage specification of system requirements, system design, and safety case for a safety critical system. The SaCS pattern language may express basic patterns on different aspects of relevance for conceptual safety designs. SaCS may also be used to combine basic patterns into composite patterns. A composite pattern may be instantiated into a conceptual safety design. A framework for evaluating modelling languages is used to conduct the evaluation. The quality of a language is within the framework expressed by six appropriateness factors. A set of requirements is associated with each appropriateness factor. The extent to which these requirements are fulfilled are used to judge the quality. We discuss the fulfilment of the requirements formulated for the SaCS language on the basis of the theoretical, technical, and practical considerations that were taken into account and shaped the SaCS language.**

Keywords–pattern language, analytic evaluation, design conceptualisation, safety.

I. Introduction

A pattern describes a particular recurring problem that arises in a specific context and presents a well-proven generic scheme for its solution [1]. A pattern language is a language for specifying patterns making use of patterns from a vocabulary of existing patterns and defined rules for combining these [2]. A safety critical system [3] is a system *"whose failure could result in loss of life, significant property damage, or damage to the environment"*. With a conceptual safety design we mean an early stage specification of system requirements, system design, and safety case for a safety critical system. The Safe Control Systems (SaCS) pattern language has been designed to facilitate the specification of patterns to support the development of conceptual safety designs. The intended users of SaCS are system engineers, safety engineers, hardware and software engineers.

This paper conducts an analytic evaluation of the suitability of the SaCS pattern language for its intended task. A framework for analysing languages known as the semiotic quality framework (SEQUAL) [4] is used as a basis for the evaluation. The appropriateness of a language for its intended task is in the framework characterised by six appropriateness factors [4]: domain, modeller, participant, comprehensibility, tool, and organisational. A set of requirements is presented for each appropriateness factor in order to characterise more precisely what is expected from our language in order to be appropriate. The requirements represent the criteria for judging what is appropriate of a language for conceptual safety design, independent of SaCS being appropriate or not. We motivate our choices and discuss to what extent the requirements are fulfilled.

The remainder of this article is structured as follows: Section II provides a short introduction to the SaCS pattern language. Section III discusses evaluation approaches and motivates the selection of SEQUAL. Section IV motivates the selection of requirements and conducts an evaluation of the SaCS language with respect to these requirements for each appropriateness factor. Section V presents related work on pattern-based development. Section VI draws the conclusions.

II. Background on The SaCS Pattern Language

Fig. 1 defines a composite pattern according to the syntax of SaCS [5]. The composite described in Fig. 1 is named *Safety Requirements* and consists of the basic patterns *Hazard Analysis*, *Risk Analysis*, and *Establish System Safety Requirements*. The basic patterns are specified separately in a structured manner comparable to what can be found in the literature [1][2][6][7][8][9][10][11] on patterns.

In Fig. 1, the horizontal line separates the declaration part of the composite pattern from its content. The icon placed below the identifier *Safety Requirements* signals that this is a composite pattern. Every pattern in SaCS is parametrised. An input parameter represents the information expected to be provided when applying a pattern in a context. An output parameter represents the expected outcome of applying a pattern in a given context. The inputs to *Safety Requirements* are listed inside square brackets to the left of the icon, i.e., *ToA* and *Haz*. The arrow pointing towards the brackets symbolises input. The output of the pattern is also listed inside square brackets, but on the right-hand side of the icon, i.e., *Req*. The arrow pointing away from the brackets symbolises output. An icon placed adjacent to a parameter identifier denotes its type. The parameters *ToA*, *Haz*, *HzLg*, and *Risks* in Fig. 1 are of type *documentation*, while *Req* is of type *requirement*. The inputs and outputs of a composite are always publicly accessible.

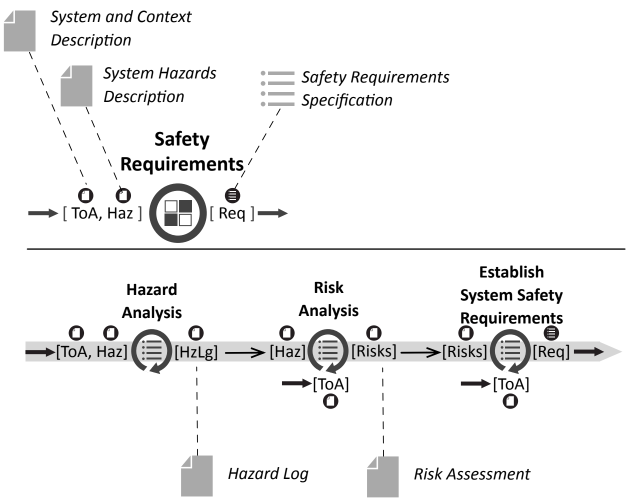


Figure 1. A composite pattern named Safety Requirements

A particular instantiation of a parameter is documented by a relation that connects a parameter with its associated development artefact. In Fig. 1, a grey icon placed adjacent to an identifier of a development artefact classifies what kind of artefact that is referenced. A dotted drawn line connecting a parameter with an artefact represents an *instantiates* relation. Instantiations of parameters expressed in Fig. 1 are:

- the document artefact *System and Context Description* instantiates *ToA*.
- the document artefact *System Hazards Description* instantiates *Haz*.
- the requirement artefact *Safety Requirements Specification* instantiates *Req*.
- the document artefact *Hazard Log* instantiates *HzLg*.
- the document artefact *Risk Assessment* instantiates *Risks*.

A one-to-many relationship exists between inputs in the declaration part of a composite and similarly named inputs with public accessibility (those pointed at by fat arrows) in the content part. The relationship is such that when *ToA* of *Safety Requirements* is instantiated (i.e., given its value by the defined relation to *System and Context Description*) then every correspondingly named input parameter contained in the composite is also similarly instantiated. A one-to-one relationship exists between an output parameter in the declaration part of a composite and a correspondingly named output parameter with public accessibility (those followed by a fat arrow) in the content part. The relationship is such that when *Req* of *Establish System Safety Requirements* is produced then *Req* of *Safety Requirements* is similarly produced.

The arrows (thin arrows) connecting basic patterns in the content part of *Safety Requirements* represent two instances of an operator known as the *assigns* relation. The *assigns* relations within *Safety Requirements* express that:

- The output *HzLg* of the pattern *Hazard Analysis* is assigned to the input *Haz* of the pattern *Risk Analysis*.
- The output *Risks* of the pattern *Risk Analysis* is assigned to the input *Risks* of the pattern *Establish System Safety Requirements*.

That the three basic patterns are process patterns follows from the icon below their respective identifiers. There are six different kinds of basic patterns in SaCS, each represented by a specific icon.

III. Evaluation Framework

Mendling et al. [12] describe two dominant approaches in the literature for evaluating the quality of modelling approaches: (1) top-down quality frameworks; (2) bottom-up metrics that relate to quality aspects. The most prominent top-down quality framework according to [12] is SEQUAL [4][13][14]. The framework is based on semiotic theory (the theory of signs) and is developed for evaluating the quality of conceptual models and languages of all kinds. Moody et al. [15] report on an empiric study involving 194 participants on the use of SEQUAL and concludes that the study provides strong support for the validity of the framework. Becker et al. [16] present a guideline-based approach as an alternative to SEQUAL. It addresses the six factors: correctness, clarity, relevance, comparability, economic efficiency, and systematic design. Mendling et al. [12] also discuss a number of bottom-up metrics approaches. Several of these contributions are theoretic without empirical validation according to the authors.

We have chosen to apply the SEQUAL framework for our evaluation as it is a general framework applicable to different kinds of languages [4] whose usefulness has been confirmed in experiments [15]. Furthermore, an analytic evaluation is preferred over a metric-based approach due to project limitations. An analytic evaluation is also a suitable complement to the experience-based evaluations of SaCS presented in [17][18].

The appropriateness of a modelling language for a specific task is in SEQUAL related to the definition of the following sets: the set of goals G for the modelling task; its domain D in the form of the set of all statements that can be stated about the situation at hand; the relevant knowledge of the modeller Km and other participants Ks involved in the modelling task; what persons involved interpret the models to say I; the language L in the form of the set of all statements that can be expressed in the language; relevant tool interpretation T of the models; and what is expressed in the models M.

Fig. 2 is adopted from [13] and illustrates the relationships between the different sets in SEQUAL. The quality of a language L is expressed by six appropriateness factors. The quality of a model M is expressed by nine quality aspects.

In the following, we will not address the different quality aspects of a model M but rather address the quality of the SaCS pattern language.

The appropriateness factors indicated in Fig. 2 are related to different properties of the language under evaluation. The appropriateness factors are [4]:

- *Domain appropriateness*: the language should be able to represent all concepts in the domain.

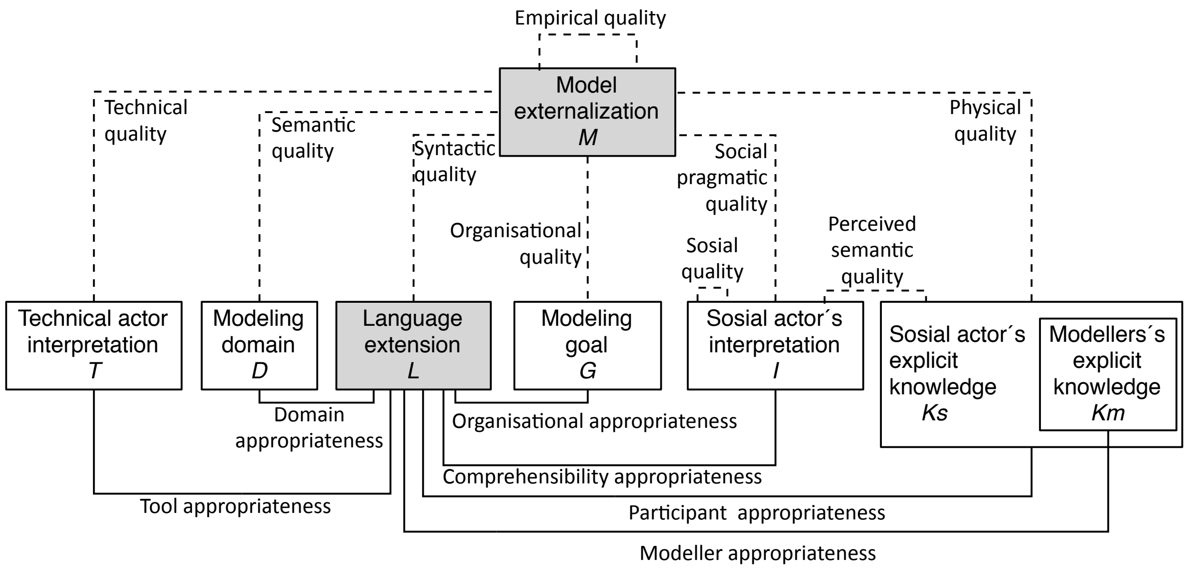


Figure 2. The quality framework (adopted from [19])

- *Modeller appropriateness*: there should be no statements in the explicit knowledge of the modeller that cannot be expressed in the language.
- *Participant appropriateness*: the conceptual basis should correspond as much as possible to the way individuals who partake in modelling perceive reality.
- *Comprehensibility appropriateness*: participants in the modelling should be able to understand all the possible statements of the language.
- *Tool appropriateness*: the language should have a syntax and semantics that a computerised tool can understand.
- *Organisational appropriateness*: the language should be usable within the organisation it targets such that it fits with the work processes and the modelling required to be performed.

A set of requirements is associated with each appropriateness factor. The extent to which the requirements are fulfilled are used to judge the quality of the SaCS pattern language for its intended task. The requirements are defined on the basis of requirements found in the literature on SEQUAL [4].

TABLE I. OVERVIEW OF EVALUATION CRITERIA

ID	Requirement
DA.1	The language must include the concepts representing best practices within conceptual safety design.
DA.2	The language must support the application of best practices within conceptual safety design.
MA.1	The language must facilitate tacit knowledge externalisation within conceptual safety design.
MA.2	The language must support the modelling needs within conceptual safety design.
PA.1	The terms used for concepts in the language must be the same terms used within safety engineering.
PA.2	The symbols used to illustrate the meaning of concepts in the language must reflect these meanings.
PA.3	The language must be understandable for people familiar with safety engineering without specific training.
CA.1	The concepts and symbols of the language should differ to the extent they are different.
CA.2	It must be possible to group related statements in the language in a natural manner.
CA.3	It must be possible to reduce model complexity with the language.
CA.4	The symbols of the language should be as simple as possible with appropriate use of colour and emphasis.
TA.1	The language must have a precise syntax.
TA.2	The language must have a precise semantics.
OA.1	The language must be able to express the desired conceptual safety design when applied in a safety context.
OA.2	The language must ease the comprehensibility of best practices within conceptual safety design for relevant target groups like system engineers, safety engineers, hardware and software engineers.
OA.3	The language must be usable without the need of costly tools.

IV. The Evaluation

A necessary step in the application of SEQUAL [4][13] is to adapt the evaluation to account for the modelling needs. This amounts to expressing what the different appropriateness factors of the framework represent in the particular context of the evaluation in question. In particular, the modelling needs are detailed by the definition of a set of criteria for each of the appropriateness factors.

Table I introduces the criteria for evaluating the suitability of the SaCS pattern language for its intended task. In the first column of Table I, the two letters of each requirement identifier identify the appropriateness factor addressed by the requirement, e.g., DA for Domain Appropriateness.

The different appropriateness factors are addressed successively in Section IV-A to Section IV-F according to the order in Table I. Each requirement from Table I is discussed. A requirement identifier is presented in a **bold** font when first introduced in the text followed by the associated requirement and an evaluation of the extent to which the requirement is fulfilled by SaCS.

A. Domain appropriateness

DA.1 The language must include the concepts representing best practices within conceptual safety design.

In the SaCS language, there are currently 26 basic patterns [17][18] on different concepts within conceptual safety design.

Each pattern may be referenced by its unique name. Three of the currently available basic patterns are referenced in Fig. 1 and are named *Hazard Analysis*, *Risk Analysis* and *Establish System Safety Requirements*.

Fig. 3 presents the icons used for basic SaCS patterns and indicates a categorisation. The three icons to the left are used for categorising patterns providing development guidance with a strong processual focus. The three icons to the right are used for categorising patterns providing development guidance with a strong product focus. Different kinds of patterns express different concepts and best practices within development of safety critical systems. The combined use of patterns from different categories facilitates development of conceptual safety designs.

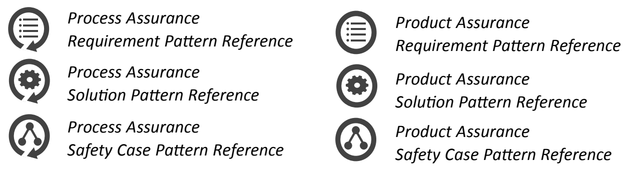

Process Assurance Requirement Pattern Reference
Process Assurance Solution Pattern Reference
Process Assurance Safety Case Pattern Reference
Product Assurance Requirement Pattern Reference
Product Assurance Solution Pattern Reference
Product Assurance Safety Case Pattern Reference

Figure 3. Icons for the different kinds of basic pattern references

Habli and Kelly [20] describe the two dominant approaches in safety standards for providing assurance of safety objectives being met. These are: (1) the process-based approach; (2) the product-based approach. Within the process-based approach, safety assurance is achieved on the basis of evidence from the application of recommended or mandatory development practices in the development life cycle. Within the product-based approach, safety assurance is achieved on the basis of product specific evidences that meet safety requirements derived from hazard analysis. The practice within safety standards as described above motivate our categorisation into the process assurance and the product assurance pattern groups.

The safety property of a system is addressed on the basis of a demonstration of the fulfilment of safety objectives. Seven nuclear regulators [21] define a safety demonstration as *"a set of arguments and evidence elements that support a selected set of dependability claims - in particular the safety - of the operation of a system important to safety used in a given plant environment"*. Although it is the end system that is put into operation, evidences supporting safety claims are produced throughout the system life cycle and need to be systematically gathered from the very beginning of a development project [21]. The safety case approach represents a means for explicitly presenting the structure of claims, arguments, and evidences in a manner that facilitates evaluation of the rationale and basis for claiming that safety objectives are met. The safety case approach is supported by several authors [10][20][21][22]. What is described above motivates the need for patterns supporting safety case specification in addition to patterns on requirements elicitation and system design specification.

As indicated above, in the design of the SaCS pattern language we have as much as possible selected keywords and icons in the spirit of leading literature within the area. This indicates that we at least are able to represent a significant part of the concepts of relevance for conceptual safety design.

DA.2 The language must support the application of best practices within conceptual safety design.

Safety standards [23] may demand a number of activities to be performed in which certain activities must be applied in a specific sequence. Safety standards [23] may also describe the expected inputs and outputs of different activities and in this sense state what is the expected content of deliverables that allows a transition from one activity to the next. According to Krogstie [4], the main phenomena in languages that accommodate a behavioural modelling perspective are states and transitions between states. In this sense, the language should support the modelling of the application of best practices according to a behavioural modelling perspective.

Fig. 4 presents the icons for the different kinds of parameters and artefact references in SaCS. The *documentation parameter* and the *documentation artefact reference* types (represented visually by the icons presented in Fig. 4) are defined in order to allow a generic classification of parameters and artefacts that may not be classified as requirement, design, or safety case. An example may be the result of risk analysis that is an intermediate result in conceptual safety design and an input to an activity on the specification of safety requirements [23][24]. The process of deriving safety requirements on the basis of an assessment of hazards is expressed by a chain of patterns as presented in Fig. 1. The outcome of applying the last pattern in the chain is a requirements specification. The last pattern cannot be applied before the required inputs are produced.

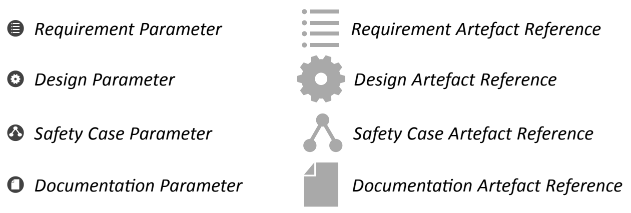

Requirement Parameter
Requirement Artefact Reference
Design Parameter
Design Artefact Reference
Safety Case Parameter
Safety Case Artefact Reference
Documentation Parameter
Documentation Artefact Reference

Figure 4. Icons for the different kinds of parameters and artefact references

Fig. 5 presents the symbolic representation of the different relations in SaCS. Relations define transitions between patterns or dependencies between elements within a composite pattern definition. The reports [17][18] define the concepts behind the different relations and exemplify the practical use of all the concepts in different scenarios. Fig. 1 is explained in Section II and exemplify a composite pattern containing five instances of the *instantiates* relation and two instances of the *assigns* relation.

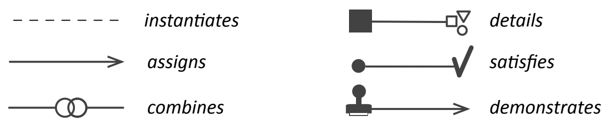


Figure 5. Symbols for the different kinds of relations

The need for the different relations presented in Fig. 5 is motivated by the practices described in different standards and guidelines, e.g., IEC 61508 [23], where activities like hazard identification and hazard analysis are required to be performed sequentially and where the output of one activity is assigned as input to another activity. Thus, we need a concept of assignment. In SaCS, this is defined by an *assigns* relation between patterns. When performing an activity like hazard analysis, the results from the application of a number of methods may be combined and used as input. Two widely known methods captured in two different basic SaCS patterns are Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA). A concept for combining results is needed in order to model that the results from applying several patterns as FMEA and FTA are combined into a union consisting of every individual result. In SaCS, this is defined by a *combines* relation between patterns. A *details* relation is used to express that the result of applying one pattern is further detailed by the application of a second pattern. Functional safety is an important concept in IEC 61508 [23]. Functional safety is a part of the overall safety that depends on a system or equipment operating correctly in response to its inputs. Furthermore, functional safety is achieved when every specified safety function is carried out and the level of performance required of each safety function is met. A *satisfies* relation between a pattern for requirements elicitation and a pattern for system design expresses that the derived system satisfies the derived requirements. Safety case patterns supports documenting the safety argument. A *demonstrates* relation between a safety case pattern and a design pattern expresses that the derived safety argument represents a safety demonstration for the derived system.

Fig. 6 and Fig. 7 illustrate how the intended instantiation order of patterns may be visualised. The direction of the arrow indicates the pattern instantiation order; patterns (or more precisely the patterns referred to graphically) placed closer to the starting point of the arrow are instantiated prior to patterns placed close to the tip of the arrow. Patterns may be instantiated in parallel and thus have no specific order; this is visualised by placing pattern references on separate arrows.

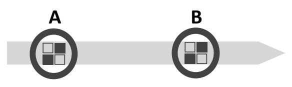


Figure 6. Serial instantiation

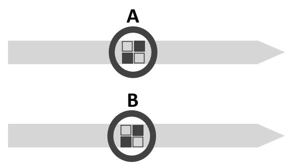


Figure 7. Parallel instantiation

As argued above, the SaCS language facilitates the application of best practices within safety design and mirrors leading international standards within the area; in particular IEC 61508. We therefore think it is fair to say that the language to a large extent fulfils DA.2.

B. Modeller appropriateness

MA.1 The language must facilitate tacit knowledge externalisation within conceptual safety design.

As already mentioned, the current version of the language contains 26 basic patterns. The basic patterns are documented in [17] and [18]. The patterns are defined on the basis of safety engineering best practices as defined in international standards and guidelines [21][23][24][25][26][27] and other sources on safety engineering. The limited number of basic patterns currently available delimit what can be modelled in a composite pattern. Defining more basic patterns will provide a better coverage of the tacit knowledge that can be externalised. A user may easily extend the language. A basic pattern, e.g., the pattern *Hazard Analysis* [17] referenced in Fig. 1, is defined in a simple structure of named sections containing text and illustrations according to a common format. The format is thoroughly detailed in [5].

Table II compares the overall format of basic SaCS patterns to pattern formats in the literature. We have chosen a format that resembles that of Alexander et al. [2] with the addition of the sections "Pattern signature", "Intent", "Applicability", and "Instantiation rule". The signature, intent, and applicability sections of basic patterns are documented in such a manner that the context section provided in [2] is not needed. The format in [2] is a suitable basis as it is simple, well-known, and generally applicable for specifying patterns of different kinds. The format provided by Gamma et al. [8] is also simple and well-known, but tailored specifically for capturing patterns for software design.

All in all, we admit that there may be relevant tacit knowledge that is not easily externalised as the SaCS language is today. However, the opportunity of increasing the number of basic patterns makes it possible to at least reduce the gap.

MA.2 The language must support the modelling needs within conceptual safety design.

IEC 61508 [23] is defined to be applicable across all industrial domains developing safety-related systems. As already mentioned, a key concept within IEC 61508 is functional safety. Functional safety is achieved according to [23] by adopting a broad range of principles, techniques and measures.

A key concept within SaCS is that principles, techniques, methods, activities, and technical solutions of different kinds are defined within the format of basic patterns. A limited number of concerns are addressed by each basic pattern. A

TABLE II. PATTERN FORMATS IN THE LITERATURE COMPARED TO BASIC SACS PATTERNS [5]

	[6]	[2]	[8]	[9]	[10]	[28]	[29]	[30]	[5]
Name	✓	✓	✓	✓	✓	✓	✓	✓	✓
Also known as			✓		✓				
Pattern signature									✓
Intent			✓		✓				✓
Motivation			✓		✓				
Applicability			✓		✓				✓
Purpose							✓		
Context	✓	✓				✓	✓		
Problem	✓	✓		✓		✓		✓	✓
Forces	✓					✓			
Solution	✓	✓		✓		✓	✓	✓	✓
Structure			✓		✓				
Participants			✓		✓				
Collaborations			✓		✓				
Consequences			✓		✓				
Implementation			✓		✓				
Sample code			✓						
Example					✓				
Compare							✓		
Instantiation rule									✓
Related patterns		✓	✓	✓	✓	✓			✓
Known uses	✓	✓		✓		✓			✓

specific combination of patterns is defined within a composite pattern. A composite pattern is intended to address the overall challenges that appear in a given development context. Individual patterns within a composite only address a subset of the challenges that need to be solved in the context. A composite may be defined prior to work initiation in order to define a plan for the application of patterns. Another use may be to refine a composite throughout the work process. This is exemplified in [17] and [18]. A composite may also be defined once patterns have been applied in order to document the work process. A composite representing a plan may be easily reused for documentation purposes by adding information on the instantiation of parameters.

C. Participants appropriateness

PA.1 The terms used for concepts in the language must be the same terms used within safety engineering.

Activities such as *hazard identification* and *hazard analysis* [26], methods such as *fault tree analysis* [31] and *failure mode effects analysis* [32], system design solutions including *redundant modules* and voting mechanisms [33], and practices like arguing safety on the basis of arguing that safety requirements are satisfied [21], are all well known safety engineering practices that may be found in different standards and guidelines [23][24][27]. The different concepts mentioned above are all reflected in basic SaCS patterns. Moreover, as already pointed out, keywords such as process assurance, product assurance, requirement, solution, safety case, etc. have all been selected based on leading terminology within safety engineering.

PA.2 The symbols used to illustrate the meaning of concepts in the language must reflect these meanings.

One commonly cited and influential article within psychology is that of Miller [34], on the limit of human capacity to process information. The limit, according to Miller, is seven plus or minus two elements. When the number of elements increases past seven, the mind may be confused in correctly interpreting the information. Thus, the number of symbols should be kept low in order to facilitate effective human information processing.

Lidwell et al. [35] describe iconic representation as *"the use of pictorial images to make actions, objects, and concepts in a display easier to find, recognize, learn, and remember"*. The authors describe four forms for representation of information with icons: similar, example, symbolic, and arbitrary. We have primarily applied the symbolic form to identify a concept at a higher level of abstraction than what may be achieved with the similar and example forms. We have also tried to avoid the arbitrary form where there is little or no relationship between a concept and its associated icon. Fig. 3, Fig. 4, Fig. 5, and Fig. 6 present the main icons in SaCS. In order to allow a flexible use of icons and keep the number of icons low, we have chosen to not define a dedicated icon for each concept but rather define icons that categorises several related concepts. A relatively small number of icons was designed in a uniform manner in order to capture intuitive representations of related concepts. As an example, the referenced basic patterns in Fig. 1 have the same icons linking them by category, but unique identifiers separating them by name.

PA.3 The language must be understandable for people familiar with safety engineering without specific training.

The SaCS language is simple in the sense that a small set of icons and symbols are used for modelling the application of patterns, basically: pattern references as in Fig. 3, parameters and artefact references as in Fig. 4, relations as in Fig. 5, and instantiation order as in Fig. 6. Guidance to the understanding of the language is provided in [5], where the syntax and the semantics of SaCS patterns are described in detail. The SaCS language comes with a structured semantics [5] that offers a schematic mapping from syntactical elements into text in English. Guidance to the application of SaCS is provided by the examples detailed in [17][18]. Although we have not tested SaCS on people unfamiliar with the language, we expect that users familiar with safety engineering may comprehend the concepts and the modelling on the basis of [5][17][18] within 2-3 working days.

D. Comprehensibility appropriateness

CA.1 The concepts and symbols of the language should differ to the extent they are different.

The purpose of the graphical notation is to represent a structure of patterns in a manner that is intuitive, comprehensible, and that allows efficient visual perception. The key activities performed by a reader in order to draw conclusion from a diagram are according to Larkin and Simon [36]: searching and recognising relevant information.

Lidwell et al. [35] present 125 patterns of good design based on theory and empirical research on visualisation. The patterns describe principles of designing visual information for effective human perception. The patterns are defined on the basis of extensive research on human cognitive processes. Some of the patterns are commonly known as Gestalt principles of perception. Ellis [37] provides an extensive overview of the Gestalt principles of perception building upon classic work from Wertheimer [38] and others. Gestalt principles capture the tendency of the human mind to naturally perceive whole objects on the basis of object groups and parts.

One of several Gestalt principles applied in the SaCS language is the principle of similarity. According to Lidwell et al. [35], the principle of similarity is such that similar elements are perceived to be more related than elements that are dissimilar.

The use of the similarity principle is illustrated by the composite pattern in Fig. 1. Although each referenced pattern has a unique name, their identical icons indicate relatedness. Different kinds of patterns are symbolised by the icons in Fig. 3. The icons are of the same size with some aspects of similarity and some aspects of dissimilarity such that a degree of relatedness may be perceived. An icon for pattern reference is different in shape and shading compared to an icon used for artefact reference (see Fig. 3 and Fig. 4). Thus, an artefact and a pattern should be perceived as representing quite different concepts.

CA.2 It must be possible to group related statements in the language in a natural manner.

There are five ways to organise information according to Lidwell et al. [35]: category, time, location, alphabet, and continuum. The *category* refers to the organisation of elements by similarity and relatedness. An example of the application of the principle of categorisation [35] in SaCS is seen in the possibility to reduce the number of relations drawn between patterns when these are similar. Patterns in SaCS may have multiple inputs and multiple outputs as indicated in Fig. 1. Relations between patterns operate on the parameters. The brackets [ ] placed adjacent to a pattern reference denotes an ordered list of parameters. In order to avoid drawing multiple relations between two patterns, relations operate on the ordered parameter lists of the patterns by list-matching of parameters.

Fig. 8 exemplifies two different ways for expressing visually the same relationships between the composite patterns named *A* and *B*. The list-matching mechanism is used to reduce the number of relation symbols drawn between patterns to one, even though the phenomena modelled represents multiple similar relations. This reduces the visual complexity and preserves the semantics of the relationships modelled.

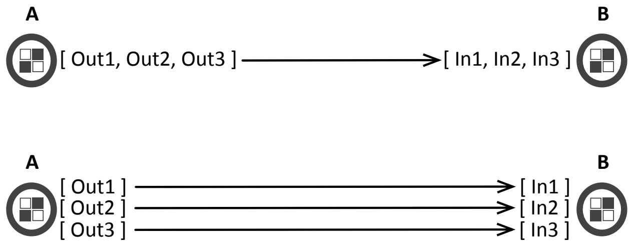


Figure 8. Alternative ways for visualising multiple similar relations

CA.3 It must be possible to reduce model complexity with the language.

Hierarchical organisation is the simplest structure for visualising and understanding complexity according to Lidwell et al. [35]. The SaCS language allows concepts to be organised hierarchically by specifying that one pattern is detailed by another or by defining composite patterns that reference other composite patterns in the content part.

Fig. 9 presents a composite pattern named *Requirements* that reference other composites as part of its definition. The contained pattern *Safety Requirements* is defined in Fig. 1. The contained pattern *Functional Requirements* is not defined and is referenced within Fig. 9 for illustration purposes. *Requirements* may be easily extended by defining composites supporting the elicitation of, e.g., performance requirements and security requirements, and later model the use of such patterns in Fig. 9. In Fig. 9, the output of applying the *Requirements* pattern is represented by the parameter *ReqSpec*. The *ReqSpec* parameter represents the result of applying the *combines* relation on the output *Req* of the composite *Safety Requirements* and the output *Req* of the composite *Functional Requirements*.

CA.4 The symbols of the language should be as simple as possible with appropriate use of colour and emphasis.

A general principle within visualisation according to Lidwell et al. [35] is to use colour with care as it may lead to misconceptions if used inappropriately. The authors points out that there is no universal symbolism for different colours.

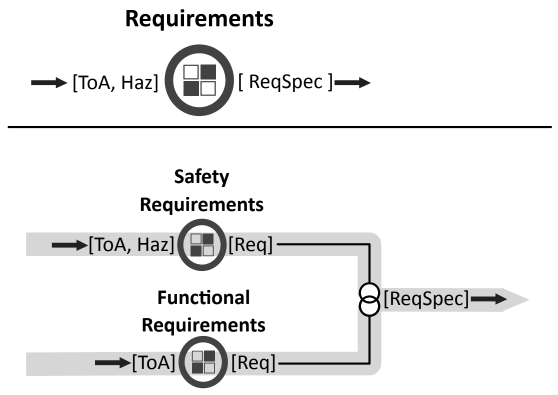


Figure 9. Composition of composites

As colour blindness is common the SaCS language applies different shades of grey in visualisations.

Fig. 10 illustrates how the SaCS language makes use of the three Gestalt principles of perception [35][39][38] known as: Figure-Ground; Proximity; and Uniform Connectedness. The Gestalt principles express mechanisms for efficient human perception from groups of visual objects.

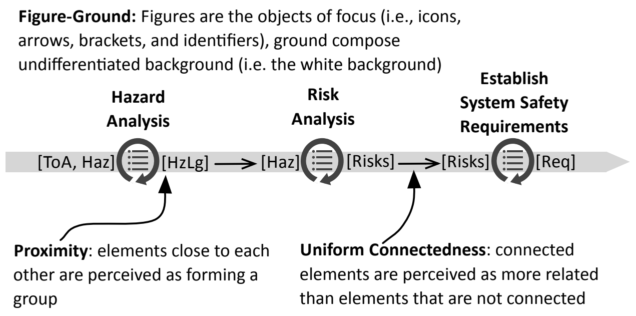


Figure-Ground: Figures are the objects of focus (i.e., icons, arrows, brackets, and identifiers), ground compose undifferentiated background (i.e. the white background)
Hazard Analysis
Risk Analysis
Establish System Safety Requirements
[ToA, Haz]
[HzLg]
[Haz]
[Risks]
[Risks]
[Req]
Proximity: elements close to each other are perceived as forming a group
Uniform Connectedness: connected elements are perceived as more related than elements that are not connected

Figure 10. A fragment of Fig. 1 illustrating the use of Gestalt principles

E. Tool appropriateness

TA.1 The language must have a precise syntax.

The syntax of the SaCS language (see [5]) is defined in the EBNF [40] notation. EBNF is a meta-syntax that is widely used for describing context-free grammars.

TA.2 The language must have a precise semantics.

A structured semantics for SaCS patterns is defined in [5] in the form of a schematic mapping from pattern definitions, via its textual syntax in EBNF [40], to English. The non-formal representation of the semantics supports human interpretation rather than tools, although the translation procedure as described in [5] may be automated. The presentation of the semantics of patterns as a text in English was chosen in order to aid communication between users, possibly with different technical background, on how to interpret patterns.

F. Organisational appropriateness

OA.1 The language must be able to express the desired conceptual safety design when applied in a safety context.

The application of the SaCS pattern language produces composite patterns that are instantiated into conceptual safety designs. A composite pattern expresses a combination of basic patterns. The basic patterns express safety engineering best practices and concepts inspired by international safety standards and guidelines, e.g., [23][24][27]. International safety standards and guidelines describe concepts and practices for development of safety critical systems that may be perceived as commonly accepted. The SaCS pattern language is tested out in two cases. The first concerned the conceptualisation of a nuclear power plant control system, while the second addressed the conceptualisation of a railway interlocking system, fully detailed in [17] and [18], respectively. In both cases it was possible to derive a conceptual safety design using the SaCS language as support as well as model how patterns were applied as support.

OA.2 The language must ease the comprehensibility of best practices within conceptual safety design for relevant target groups like system engineers, safety engineers, hardware and software engineers.

We have already explained how basic patterns represent concepts and best practices inspired by safety standards and guidelines. Each basic pattern addresses a limited number of phenomena. Basic patterns are combined into a composite pattern where the composite addresses all relevant challenges that occur in a specific context. A composite pattern as the one presented in Fig. 1 ease the explanation of how several concepts within conceptual safety design are combined and applied.

Wong et al. [41] reviewed several large development projects and software safety standards from different domains with respect to cost effectiveness and concludes that although standards provide useful and effective guidance, safety and cost effectiveness objectives are successfully met by effective planning and by applying safety engineering best practices evidenced in company best practices throughout the development life cycle. Compared to a standard or a guideline, a composite pattern in the SaCS language may be used to capture such a company specific best practice. In order to accommodate different situations, different compositions of patterns may be defined.

OA.3 The language must be usable without the need of costly tools.

Every pattern used in the cases described in [17][18] was interpreted and applied in its context by a single researcher with background from safety engineering. A conceptual safety design was produced for each case. Every illustration in [5][17][18] and in this paper is created with a standard drawing tool.

V. Related Work

In the literature, pattern approaches supporting development of safety critical systems are poorly represented. In the following we shortly discuss some different pattern approaches and their relevancy to the development of conceptual safety designs.

Jackson [42] presents the problem frames approach for requirements analysis and elicitation. Although the problem frames approach is useful for detailing and analysing a problem and thereby detailing requirements, the problem classes presented in [42] are defined on a very high level of abstraction.

The use of boilerplates [43][44] for requirement specification is a form of requirement templates but nonetheless touches upon the concept of patterns. The boilerplate approach helps the user phrase requirements in a uniform manner and to detail these sufficiently. Although boilerplates may be useful for requirement specification, the focus in SaCS is more towards supporting requirement elicitation and the understanding of the challenges that appear in a specific context.

Withall [45] describes 37 requirements patterns for assisting the specification of different types of requirements. The patterns are defined at a low level; the level of a single requirement. The patterns of Withall may be useful, but as with the boilerplates approach, the patterns support more the specification of requirements rather than requirements elicitation.

Patterns on design and architecture of software-based systems are presented in several pattern collections. One of the well-known pattern collections is the one of Gamma et al. [8] on recurring patterns in design of software based systems. Without doubt, the different pattern collections and languages on system design and architecture represent deep insight into effective solutions. However, design choices should be founded on requirements, and otherwise follow well established principles of good design. The choice of applying one design pattern over another should be based on a systematic process of establishing the need in order to avoid design choices being left unmotivated.

The motivations for a specific design choice are founded on the knowledge gained during the development activities applied prior to system design. Gnatz et al. [46] outline the concept of process patterns as a means to address the recurring problems and known solutions to challenges arising during the development process. The patterns of Gnatz et al. are not tailored for development of safety critical systems and thus do not necessarily reflect relevant safety practices. Fowler presents [7] a catalogue of 63 analysis patterns. The patterns do not follow a strict format but represent a body of knowledge on analysis described textually and by supplementary sketches.

While process patterns and analysis patterns may be relevant for assuring that the development process applied is suitable and leads to well informed design choices, Kelly [10] defines patterns supporting safety demonstration in the form of reusable safety case patterns. The patterns expressed are representative for how we want to address the safety demonstration concern.

A challenge is to effectively combine and apply the knowledge on diverse topics captured in different pattern collections and languages. Henninger and Corrêa [47] survey different software pattern practices and states *"software patterns and collections tend to be written to solve specific problems with little to no regard about how the pattern could or should be used with other patterns"*.

Zimmer [48] identifies the need to define relationships between system design patterns in order to efficiently combine them. Noble [49] builds upon the ideas of Zimmer and defines a number of relationships such as *uses, refines, used by, combine*, and *sequence of* as a means to define relationships between system design patterns. A challenge with the relations defined by Noble is that they only specify relations on a very high level. The relations do not have the expressiveness for detailing what part of a pattern is *used, refined*, or *combined*. Thus, the approach does not facilitate a precise modelling of relationships.

Bayley and Zhu [50] define a formal language for pattern composition. The authors argue that design patterns are almost always to be found composed with each other and that the correct applications of patterns thus relies on precise definition of the compositions. A set of six operators is defined for the purpose of defining pattern compositions. The language is exemplified on the formalisation of the relationships expressed between software design patterns described by Gamma et al. [8]. As we want the patterns expressed in the SaCS language to be understandable to a large community of potential users, we find this approach a bit too rigid.

Smith [51] presents a catalogue of elementary software design patterns in the tradition of Gamma et al. [8] and proposes the Pattern Instance Notation (PIN) for expressing compositions of patterns graphically. The notation uses simple rounded rectangles for abstractly representing a pattern and its associated roles. Connectors define the relationships between patterns. The connectors operate on the defined roles of patterns. The notation is comparable to the UML collaboration notation [52].

UML collaborations [52] are not directly instantiable. Instances of the roles defined in a collaboration that cooperates as defined creates the collaboration. The main purpose is to express how a system of communicating entities collectively accomplishes a task. The notation is particularly suitable for expressing system design patterns.

Several notations [53][54][55] for expressing patterns graphically use UML [52] as its basis. The notations are simple, but target the specification of software.

VI. Conclusion

We have presented an analytical evaluation of the SaCS pattern language with respect to six different appropriateness factors. We arrived at the following conclusions:

- *Domain*: In the design of the SaCS language we have as much as possible selected keywords and icons in the spirit of leading literature within the area. This indicates that we at least are able to represent a significant part of the concepts of relevance for conceptual safety design.
- *Modeller*: There may be relevant tacit knowledge that is not easily externalised as the SaCS language is today. However, the opportunity of increasing the number of basic patterns makes it possible to at least reduce the gap.
- *Participants*: The terms used for concepts have been carefully selected based on leading terminology within safety engineering. The SaCS language facilitates representing the application of best practices within safety design and mirror leading international standards; in particular IEC 61508.
- *Comprehensibility*: The comprehension of individual patterns and pattern compositions is supported by the use of terms commonly applied within the relevant industrial domains as well as by the application of principles of good design in visualisations, such as the Gestalt principles of perception [35][38].
- *Tool*: Tool support may be provided on the basis of the syntax and semantics of the SaCS language [5].
- *Organisational*: Organisations developing safety critical systems are assumed to follow a development process in accordance to what is required by standards. Wong et al. [41] reviewed several large development projects and software safety standards from different domains with respect to cost effectiveness and concludes that although standards provide useful and effective guidance, safety and cost effectiveness objectives are successfully met by effective planning and by applying safety engineering best practices evidenced in company best practices throughout the development life cycle. SaCS patterns may be defined, applied, and combined in a flexible manner to support company best practices and domain specific best practices.

Acknowledgment

This work has been conducted and funded within the OECD Halden Reactor Project, Institute for energy technology (IFE), Halden, Norway. The second author has partly been funded by the ARTEMIS project CONCERTO.

References

[1] F. Buschmann, K. Henney, and D. C. Schmidt, Pattern-Oriented Software Architecture: On Patterns and Pattern Languages. Wiley, 2007, vol. 5.

[2] C. Alexander, S. Ishikawa, and M. Silverstein, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977, vol. 2.

[3] J. C. Knight, "Safety Critical Systems: Challenges and Directions," in Proceedings of the 24th International Conference on Software Engineering (ICSE'02). ACM, 2002, pp. 547–550.

[4] J. Krogstie, Model-based Development and Evolution of Information Systems: A Quality Approach. Springer, 2012.

[5] A. A. Hauge and K. Stølen, "Syntax & Semantics of the SaCS Pattern Language," Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, Tech. Rep. HWR-1052, 2013.

[6] A. Aguiar and G. David, "Patterns for Effectively Documenting Frameworks," in Transactions on Pattern Languages of Programming II, ser. LNCS, J. Noble, R. Johnson, P. Avgeriou, N. Harrison, and U. Zdun, Eds. Springer, 2011, vol. 6510, pp. 79–124.

[7] M. Fowler, Analysis Patterns: Reusable Object Models. Addison-Wesley, 1996.

[8] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[9] R. S. Hanmer, Patterns for Fault Tolerant Software. Wiley, 2007.

[10] T. P. Kelly, "Arguing Safety – A Systematic Approach to Managing Safety Cases," Ph.D. dissertation, University of York, United Kingdom, 1998.

[11] B. Rubel, "Patterns for Generating a Layered Architecture," in Pattern Languages of Program Design, J. Coplien and D. Schmidt, Eds. Addison-Wesley, 1995, pp. 119–128.

[12] J. Mendling, G. Neumann, and W. van der Aalst, "On the Correlation between Process Model Metrics and Errors," in Proceedings of 26th International Conference on Conceptual Modeling, vol. 83, 2007, pp. 173–178.

[13] A. G. Nysetvold and J. Krogstie, "Assessing Business Process Modeling Languages Using a Generic Quality Framework," in Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05) Workshops. Idea Group, 2005, pp. 545–556.

[14] J. Krogstie and S. D. F. Arnesen, "Assessing Enterprise Modeling Languages Using a Generic Quality Framework," in Information Modeling Methods and Methodologies. Idea Group, 2005, pp. 63–79.

[15] D. L. Moody, G. Sindre, T. Brasethvik, and A. Sølvberg, "Evaluating the Quality of Process Models: Empirical Testing of a Quality Framework," in Proceedings of the 21st International Conference on Conceptual Modeling, ser. LNCS. Springer, 2013, vol. 2503, pp. 380–396.

[16] J. Becker, M. Rosemann, and C. von Uthmann, "Guidelines of Business Process Modeling," in Business Process Management, ser. LNCS, vol. 1806. Springer, 2000, pp. 30–49.

[17] A. A. Hauge and K. Stølen, "A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Nuclear Power Production," Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, Tech. Rep. HWR-1029, 2013.

[18] ——, "A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Railway Signalling," Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, Tech. Rep. HWR-1037, 2013.

[19] I. Hogganvik, "A Graphical Approach to Security Risk Analysis," Ph.D. dissertation, Faculty of Mathematics and Natural Sciences, University of Oslo, 2007.

[20] I. Habli and T. Kelly, "Process and Product Certification Arguments – Getting the Balance Right," SIGBED Review, vol. 3, no. 4, 2006, pp. 1–8.

[21] The members of the Task Force on Safety Critical Software, "Licensing of safety critical software for nuclear reactors: Common position of seven european nuclear regulators and authorised technical support organisations," http://www.belv.be/, 2013, [accessed: 2014-04-10].

[22] C. Haddon-Cave, "The Nimrod Review: An independent review into the broader issues surrounding the loss of the RAF Nimrod MR2 aircraft XV230 in Afghanistan in 2006," The Stationery Office (TSO), Tech. Rep. 1025 2008-09, 2009.

[23] IEC, "IEC 61508 Functional safety of electrical/electronic/programmble electronic safety-related systems, 2nd edition," International Electrotechnical Commission, 2010.

[24] CENELEC, "EN 50129 Railway applications – Communications, signalling and processing systems – Safety related electronic systems for signalling," European Committee for Electrotechnical Standardization, 2003.

[25] European Commission, "Commission Regulation (EC) No 352/2009 on the Adoption of Common Safety Method on Risk Evaluation and Assessment," Official Journal of the European Union, 2009.

[26] ERA, "Guide for the Application of the Commission Regulation on the Adoption of a Common Safety Method on Risk Evaluation and Assessment as Referred to in Article 6(3)(a) of the Railway Safety Directive," European Railway Agency, 2009.

[27] IEC, "IEC 61513 Nuclear power plants – Instrumentation and control systems important to safety – General requirements for systems, 2nd edition," International Electrotechnical Commission, 2001.

[28] A. Ratzka, "User Interface Patterns for Multimodal Interaction," in Transactions on Pattern Languages of Programming III, ser. LNCS, J. Noble, R. Johnson, U. Zdun, and E. Wallingford, Eds. Springer, 2013, vol. 7840, pp. 111–167.

[29] D. Riehle and H. Züllinghoven, "A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor," in Pattern Languages of Program Design, J. Coplien and D. Schmidt, Eds. Addison-Wesley, 1995, pp. 9–42.

[30] K. Wolf and C. Liu, "New Client with Old Servers: A Pattern Language for Client/Server Frameworks," in Pattern Languages of Program Design, J. Coplien and D. Schmidt, Eds. Addison-Wesley, 1995, pp. 51–64.

[31] IEC, "IEC 61025 Fault Tree Analysis (FTA), 2nd edition," International Electrotechnical Commission, 2006.

[32] ——, "IEC 60812 Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA), 2nd edition," International Electrotechnical Commission, 2006.

[33] N. Storey, Safety-critical Computer Systems. Prentice Hall, 1996.

[34] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," Psychological Review, vol. 63, no. 2, 1956, pp. 81–97.

[35] W. Lidwell, K. Holden, and J. Butler, Universal Principles of Design, 2nd ed. Rockport Publishers, 2010.

[36] J. H. Larkin and H. A. Simon, "Why a Diagram is (Sometimes) Worth Ten Thousand Words," Cognitive Science, vol. 11, no. 1, 1987, pp. 65–100.

[37] W. D. Ellis, A Source Book of Gestalt Psychology. The Gestalt Journal Press, 1997.

[38] M. Wertheimer, "Laws of Organization in Perceptual Forms," in A sourcebook of Gestalt Psychology, W. D. Ellis, Ed. Routledge and Kegan Paul, 1938, pp. 71–88.

[39] J. Wagemans, J. H. Elder, M. Kubovy, S. E. Palmer, M. A. Peterson, M. Singh, and R. von der Heydt, "A Century of Gestalt Psychology in Visual Perception: I. Perceptual Grouping and Figure-Ground Organization," Psychological Bulletin, vol. 138, no. 6, 2012, pp. 1172–1217.

[40] ISO/IEC, "14977:1996(E) Information technology - Syntactic metalanguage - Extended BNF," International Organization for Standardization / International Electrotechnical Commission, 1996.

[41] W. E. Wong, A. Demel, V. Debroy, and M. F. Siok, "Safe Software: Does It Cost More to Develop?" in Fifth International Conference on Secure Software Integration and Reliability Improvement (SSIRI'11), 2011, pp. 198–207.

[42] M. Jackson, Problem Frames: Analysing and Structuring Software Development Problems. Addison-Wesley, 2001.

[43] E. Hull, K. Jackson, and J. Dick, Requirements Engineering, 3rd ed. Springer, 2010.

[44] G. Sindre, "Boilerplates for application interoperability requirements," in Proceedings of 19th Norsk konferanse for organisasjoners bruk av IT (NOKOBIT'12). Tapir, 2012.

[45] S. Withall, Software Requirement Patterns (Best Practices), 1st ed. Microsoft Press, 2007.

[46] M. Gnatz, F. Marschall, G. Popp, A. Rausch, and W. Schwerin, "Towards a Living Software Development Process based on Process Patterns," in Proceedings of the 8th European Workshop on Software Process Technology (EWSPT'01), ser. LNCS, vol. 2077. Springer, 2001, pp. 182–202.

[47] S. Henninger and V. Corrêa, "Software Pattern Communities: Current Practices and Challenges," in Proceedings of the 14th Conference on Pattern Languages of Programs (PLOP'07). ACM, 2007, pp. 14:1–14:19, article No. 14.

[48] W. Zimmer, "Relationships between Design Patterns," in Pattern Languages of Program Design. Addison-Wesley, 1994, pp. 345–364.

[49] J. Noble, "Classifying Relationships between Object-Oriented Design Patterns," in Proceedings of Australian Software Engineering Conference (ASWEC'98), 1998, pp. 98–107.

[50] I. Bayley and H. Zhu, "A Formal Language for the Expression of Pattern Compositions," International Journal on Advances in Software, vol. 4, no. 3, 2012, pp. 354–366.

[51] J. M. Smith, Elemental Design Patterns. Addison-Wesley, 2012.

[52] OMG, "Unified Modeling Language Specification, Version 2.4.1," Object Management Group, 2012.

[53] H. Byelas and A. Telea, "Visualization of Areas of Interest in Software Architecture Diagrams," in Proceedings of the 2006 ACM Symposium on Software Visualization (SoftVis'06), 2006, pp. 105–114.

[54] J. Dong, S. Yang, and K. Zhang, "Visualizing Design Patterns in Their Applications and Compositions," IEEE Transactions on Software Engineering, vol. 33, no. 7, 2007, pp. 433–453.

[55] J. M. Vlissides, "Notation, Notation, Notation," C++ Report, 1998, pp. 48–51.

Privacy by Design Permission System for Mobile Applications

Karina Sokolova*[†], Marc Lemercier*

*University of Technology of Troyes
Troyes, France
{karina.sokolova, marc.lemercier}@utt.fr

Jean-Baptiste Boisseau[†]

[†]EUTECH SSII
La Chapelle Sain Luc, France
{k.sokolova, jb.boisseau}@eutech-ssii.com

***Abstract*—The Privacy by Design concept proposes to integrate the respect of user privacy into systems managing user data from the design stage. This concept has increased in popularity and the European Union (EU) is enforcing it with a Data Protection Directive. Mobile applications have emerged onto the market and the current law and future directive is applicable to all mobile applications designed for EU users. By now it has been shown that mobile applications do not suit the Privacy by Design concept and lack for transparency, consent and security. The actual permission systems is judged as unclear for users. In this paper, we introduce a novel permission model suitable for mobile application that respects Privacy by Design. We show that such adapted permission system can improve the transparency and consent but also the security of mobile applications. Finally, we propose an example of the use of our system on mobile application.**

Keywords–permission, permission system, mobile, privacy by design, privacy, transparency, control, Android, iOS, application, development, software design, pattern, mobility, design, modelling, trust

I. Introduction

Mobile devices gain in popularity. Thousands of services and applications are proposed on mobile markets and downloaded every day. Smart devices have a high data flow processing and storing large amounts of data including private and sensitive data. Most applications propose personalized services but simultaneously collect user data even without the user's awareness or consent. More and more users feel concerned about their privacy and care about services they use. The TRUSTe survey conducted in February 2011 shows that smartphone users are concerned about privacy even more than about the security (the second in the survey results) [1].

Nowadays, people realize the lack of privacy especially while using new technological devices where information is massively collected, used and stored (Big Data notion). The privacy regulation aiming to control personal data use is set up in many countries. European Union privacy regulation includes the European Data Protection Directive (Directive 95/46/EC) and the ePrivacy Directive. United States regulation includes Children's Online Privacy Protection Act (COPPA) and The California Online Privacy Protection Act of 2003 (OPPA). Canada is under the Personal Information Protection and Electronic Documents Act (PIPEDA) concerning privacy.

The Privacy by Design (PbD) notion proposes to integrate privacy from the system design stage [2] to build privacy-respecting systems. PbD proves systems can embed privacy without sacrificing either security or functionality. Some PbD concepts are already included in European data legislation; the notion is considered to be enforced in European Data Protection Regulation, therefore systems made with PbD are compliant with the law. An application made with PbD notion is not only a benefit for users, along with the opportunity to provide a truly personalized service, but also a legal obligation for developers.

The PbD concept was firstly presented by Dr. Ann Cavoukian. She proposes seven key principles of PbD enabling the development of privacy-respective systems. The system should be proactive, not reactive, embed privacy feature from the design, integrate Privacy by Default, respect user privacy, data usage should be transparent to the end user and the user should have access to the mechanism of control of his data. Full functionality and the end-to-end security should be preserved without any sacrifice. [2]

Mobile privacy was discussed in 'Opinion 02/2013 on apps on smart devices' by the Article 29 Data Protection Working Party [3], where the opinion on mobile privacy and some general recommendations were given. The article states that both the Data Protection Directive and the ePrivacy Directive are applicable to mobile systems and to all applications made for EU users. Data Protection Regulation is also applicable to mobile systems. The article defines four main problems of mobile privacy: lack of transparency, lack of consent, poor security and disregard for purpose limitation.

Many reports propose recommendations about mobile privacy improvement repeating basic privacy notions (e.g., data minimization, clear notices) but the exact patterns or a technical solutions are missing [4].

The permission system is embedded in the mobile systems and is a crucial part of mobile security and privacy. Nowadays, permission systems do not follow Privacy by Design notions. Many works are concentrated on analysing and modelling the actual permission systems [5][6][7], on improving actual permission systems to give more control to the user [8][9][10] or to add additional transfer permissions [11], on visual representation of permissions [12], on user perceptions of current permission systems [13][14][15], on the data flow analyses (possible data leakage detection) and the actual permissions enforcement and verifications [16][17][18][19][20][21][22][23]. To our knowledge no work has been conducted on redefining the permission system to fit the Privacy by Design notion.

The remainder of the paper is organized as follows: Section 2 describes current permission systems of iOS and Android and points out problems regarding Privacy by Design. Section 3 introduces our proposal: the pattern of the privacy-respecting permission system. We show that it can cope with the transparency, consent and purpose disregard problems and also improve the security. Section 4 shows the application of our novel permission system to the real mobile application. The paper ends with a conclusion and future works.

II. Existing Mobile Permission Systems

In this section, we present current iOS and Android permission systems and evaluate those systems regarding the PbD notion. We take into account the full functionality allowed by the permission system, privacy by default, transparency and the control notions.

- Full functionality: possibility to use all functionalities available on the platform.
- Privacy by Default: the default configurations of the system are privacy protective.
- Transparency: user should clearly understand what data is used, how and for what purpose.
- Control: user should have a full control over his personal data usage.

We consider the privacy policy to be very important for the proactive and transparent system therefore we present the state of application privacy policy in both systems.

A. Permissions

iOS and Android have different strategies concerning the access to the device data. The iOS platform gives to non-native applications access only to the functionalities listed in privacy settings: location services, contacts, calendar, reminder, photos, microphone and Bluetooth (sensitive data, such as SMS and e-mails are not shared at all). Recently, the connection to Facebook, Twitter, Flickr and Vimeo was added to the platform (iOS7). Full functionality is given up for privacy reasons as applications cannot use the full power of the platform but only a limited number of functionalities.

An iOS application should have permission to access information listed above. By default, the installed application has no permission granted. The application displays a pop-up explaining what sensitive data it needs before to access it. The user can accept or decline permission. If permission is declined, the corresponding action is not executed. If the permission is accepted, the application obtains access to the corresponding data. The user is asked to grant permission only once, but he/she can activate or deactivate such permission for each application in privacy settings integrated by default into the iOS. Thereby iOS maintains transparency, control and privacy by default.

The Android system remains on the sharing principle. Full functionality is preserved: applications have access to all native applications' data and can expose the data themselves. Applications need permission to access the data, but differently from iOS, users should accept the full list of permissions before installing an application. While all permissions are granted, an application has full access to the related data. Some Android permissions tagged as 'dangerous' can be prompted to the user every time the data is going to be accessed, but it is rarely the case. Users see the list of dangerous permissions on the screen before installing the application.

TABLE I
Android and iOS Permission Systems Comparison

	Full Functionality	Default Settings	Transparency	Control
Android	+	-	-	-
iOS	-	+	-/+	+

Android proposes more than 100 default permissions and developers can add supplementary permissions. Multiple works show that users do not understand many of default permissions and fail to judge the application privacy and security correctly using the full permission list [13][15]. Permissions do not clearly show what data is used for and how. Moreover, some other studies show the abusive usage of android permissions by developers [24].

Some users do not check the Android permission list because they need a service and they know that all permissions should be accepted to obtain it. Android permission list looks like a license agreement on a desktop application that everybody accepts but very few actually read [25]. An Android user does not have any control over permissions once the application is installed: permissions cannot be revoked. Android does not include an iOS-like system permission manager (privacy settings) by default, therefore the user has to activate or deactivate the entire functionality to disable the access to related data (e.g., Wi-Fi or 3G for Internet connection; GPS for geolocation) or to use additional privacy enhancing applications.

Both iOS and Android default permission systems mostly inform about data access, but not about any other action that can be completed with the data. For example, no permission is needed to transfer the data. Android and iOS include permissions for functionalities that can be related to personal data transfer, such as Bluetooth and Internet. Permissions can be harmless to users, but there is no indication of whether personal data is involved in a transaction. This decreases the transparency of both platforms.

Android and iOS permissions do not include purpose explanation. An iOS application helps to understand the purpose by asking permission while in use, but if an application has a granted permission once for one functionality it could use it again for a different purpose without informing the user. Android users can only guess what permission is used for and whether the use is legitimate.

Table I shows the system differences regarding four main privacy notions: full functionality, transparency, control and privacy by default. One can see that the current Android permission system is missing in transparency, control and default privacy; iOS sacrifices the functionality and also lack of transparency. Permissions are often functionality-related and users fail to understand and to judge them. Personal data usage is unclear and the purpose is missing.

B. Privacy Policy

Users should choose applications they can trust. Apple ensures that applications available on the market are potentially harmless, although Android users should judge the application for themselves with the help of information available on the market. The AppStore and Google Play provide similar information: name, description, screenshots, rating and user reviews.

The transparency and the proactivity of the system can be improved by including the privacy policy in the store. A user can be informed about the information collected and stored before he downloads the application. Without any privacy policy, the user can hardly evaluate the security and privacy of the application, only the functionality and stability of the system. In their feedback, users often evaluate the functionalities and user interfaces and report bugs, but they rarely indicate privacy and security problems.

iOS does not require developers to include the privacy policy in the application but only in applications directed at children younger than 13 years old. Apple encourages the use of privacy policy in the App Store Review Guidelines and iOS Developer Program License Agreement. Apple specify that developers should insure the application is compliant with all laws of the country the application is distributed in. On viewing the App Store Review Guidelines one can see that all Privacy by Design fundamental principles and data violation possibilities are covered by Apple verification. However, the exact evaluation process used by Apple remains secret and some privacy-intrusive applications may appear in the store. Until recently, Apple authorized the use of device identification. This identification number was not considered private. Many applications used this number to uniquely identify their users therefore many applications were considered privacy intrusive [26].

Google Play Terms of Service do not require any privacy policy to be added to the Android applications. Google provides an option to include the privacy policy but does not verify or enforce it. Google Developers Documentation provides recommendations and warns that the developer has a responsibility to ensure the application is compliant with the laws of the countries in which the application is distributed.

Some developers include license agreement and privacy policy. According to [27] only 48% of the top 25 Android paid applications, 76% of the top 25 Android free applications, 64% of iOS paid applications and 84% of iOS free applications have included the privacy policy. Android includes the permission list in the store and this can be considered a privacy policy, but, as previously discussed, the list is unclear to the final user.

III. Privacy respecting permission system

Mobile phones have significant data flow: information can be received, stored, accessed and sent by the application. Data can be entered by the user, retrieved from the system sensors or applications, come from another mobile application, arrive from the server or from other devices. Data can be shared on the phone with another application, with the server or another device.

We propose to focus permissions on data and the action that can be carried out on this data, rather than on the technology used. The definition of purpose of the data usage is included in our permission system.

Privacy Policy should be short and clear. Users should have a global vision of the data usage and functionalities before they install an application. Users rarely read long involved policies, especially when they want a service and feel they have no choice but to accept all permissions. Our permission system enables a simple policy to be generated with a list of permissions.

A. Privacy by Design Permission System

The permission system is integrated into the mobile operating system; well designed, it makes a proactive privacy-respecting tool embedded into the system.

We model our permission system with an access control model. We choose discretionary access control where only data owner can grant access. The user should be able to control the data, therefore we consider the user is a unique owner of all information related to him.

R_{app} is a set of $rules$ assigned to the application. We define a $rule$ as an assignment of the $\mathcal{R}ight$ over an $\mathcal{O}bject$ to the $\mathcal{S}ubject$.

$$\forall rule \in \mathcal{R}_{app},\, rule = (s,\, r,\, o) \tag{1}$$

where $s \in \mathcal{S}ubject,\, r \in \mathcal{R}ight,\, o \in \mathcal{O}bject$

We define the mobile application as a $\mathcal{S}ubject$. $\mathcal{O}bjects$ are the user-related data, such as e-mail, contact list, name and surname, phone number, address, social networks friend list, etc.

$$\mathcal{S}ubject = Mobile\, Application \tag{2}$$

$$\mathcal{O}bject = \{Phone\#,\, Name,\, Contacts,\, \cdots\} \tag{3}$$

To define Right we have to introduce $\mathcal{A}cation$ and $\mathcal{P}purpose$.

We define a set of actions denoted $\mathcal{A}ction$ as all actions can be carried out on user private data by the application: load, access, process, store and transfer of the data. We define the $\mathcal{A}ction$ as follows:

$$\mathcal{A}ction = \{\mathcal{L}oad, \mathcal{A}ccess, \mathcal{P}rocess, \mathcal{S}tore, \mathcal{T}ransfer\} \tag{4}$$

$\mathcal{P}urpose$ is assigned by the application and depends on the service. For example, purpose could be 'retrieve forgotten password', 'display on the screen', 'calculate the trust score', 'send news', 'automatically retrieve nearest restaurant' and 'automatically attach location to the message'.

$$\mathcal{P}urpose = \{\mathcal{R}etrieve\, forgotten\, password,\, \cdots\} \tag{5}$$

We define the set of rights denoted $\mathcal{R}ight$ for all actions except the $\mathcal{S}tore$ action as follows.

$$\forall r \in \mathcal{R}ight,\, \forall a \in \mathcal{A}ction - \{\mathcal{S}tore\}\,, r = (a,\, p) \tag{6}$$

where $p \in \mathcal{P}urpose$.

We define the set of rights denoted $\mathcal{R}ight$ with the action equal to $\mathcal{S}tore$ having an additional parameter $\mathcal{T}ime$ informing about the time storage. We define the period $[0, T]$ as an application lifetime.

$$\forall r \in \mathcal{R}ight,\, if\, a = \{\mathcal{S}tore\}\,, r = (a,\, p,\, t) \tag{7}$$

where $p \in \mathcal{P}urpose,\, a \in \mathcal{A}ction,\, t \in [0, T]$

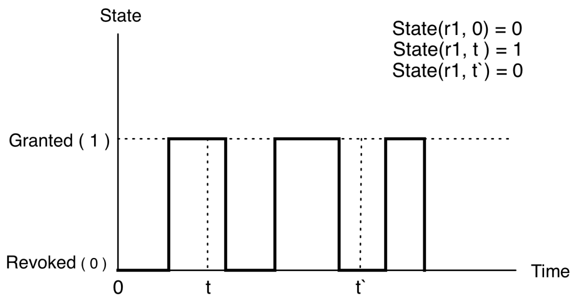


Figure 1. Example of state modification diagram for a given permission

The time storage can indicate the number of days, hours or months data is stored or the time regarding the application lifecycle: until the application is closed, until the application is stopped, until the application is uninstalled. All personal data available during the deinstallation of the application is deleted regardless of defined period, as it cannot exceed the application lifetime.

Each $rule$ should be explicitly asked of the user to be assigned. Thus each $rule$ has a $\mathcal{S}tate$: granted or revoked. To respect the Privacy by Default notion the default $\mathcal{S}tate$ of the permission in installed applications is $Revoked$. We propose to define the $\mathcal{S}tate$ as follows:

$$\forall rule \in \mathcal{R}_{app}, \forall t \in]0, T]$$
$$\mathcal{S}tate(rule, t) = \begin{cases} Granted, & user\,accepts\,the\,rule \\ Revoked, & user\,declines\,the\,rule \end{cases} \quad (8)$$

$$\forall rule \in \mathcal{R}_{app}, \mathcal{S}tate(rule, 0) = Revoked \quad (9)$$

The State of a rule $r1 \in \mathcal{R}_{app}$ changes over the application lifetime. The diagram in Figure 1 shows an example of the state modification.

A given user should be informed about the use of the permission: the $rule$ should be $defined$ and $displayed$ for each $o \in \mathcal{O}bject$. The Figure 2 shows the recapitulative schema of the rule definition.

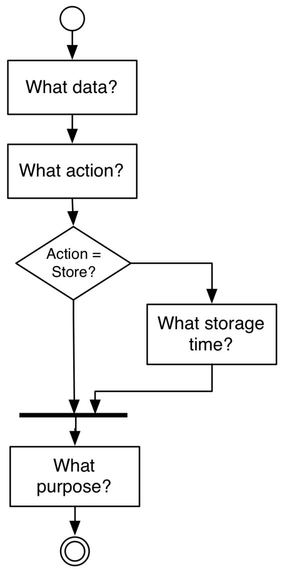


Figure 2. Activity diagram for the rule definition

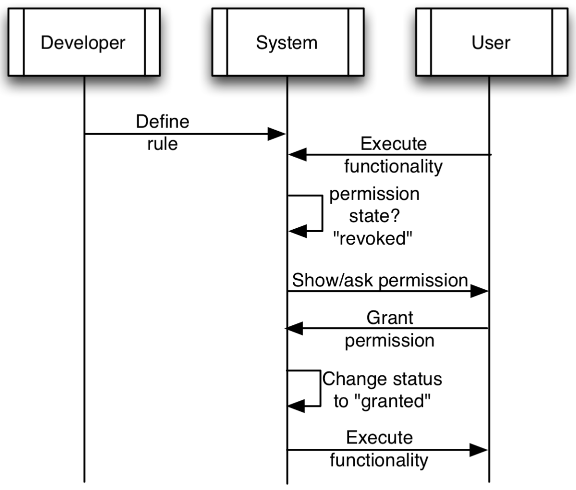


Figure 3. Sequence diagram: first use of one permission

Users should be able to grant or revoke the displayed permission. Finally, user should dispose the settings with all $rule \in \mathcal{R}_{app}$ to be able to $Grant$ or to $Revoke$ individual permissions in later use.

The sequence diagram in Figure 3 shows the pattern in action when the permission is used for the first time.

Thus, we obtain the pattern the developer can follow to design the permission system. The developer should $define$ the permission for all personal data ($\mathcal{O}bject$) used in the application ($\mathcal{S}ubject$).

The permission ($rule$) is stored inside the application with its current $State$. The default $State$ is $Revoked$. Developers should verify that the permission is $displayed$ and $requested$ at least once and that it is $available$ in settings for modification. The simple privacy policy can be generated from the list of defined rules and added to the store.

IV. Application

In this section, we propose an example of permission system made for the application of trust evaluation of friends on social networks named Socializer 1.0 [28]. We choose this application because its service is based on private information and cannot be anonymous, the PbD notion should be integrated into this application. This application needs user friend lists of different social networks (Facebook, Twitter, LinkedIn) and the contact list to view friends and common friends to calculate the overlap of friends in different social networks and contact list and to evaluate the trust of Facebook friends.

Contact list is found on the smartphone, therefore the application needs an $Access$ right.

$$r_1 = (s, (Access, \mathcal{P}_{r1}), ContactList) \quad (10)$$

where s is a $Subject$ defined as the application Socializer 1.0; $\mathcal{P}_{r1}$ is a purpose defined as a set of p_1, p_2 and p_3: $\mathcal{P}_{r1} = \{p_1, p_2, p_3\}$; p_1= view the list of contacts on the screen; p_2= calculate the overlap; p_3= calculate the trust.

Social networking friends lists should usually be retrieved from the server of a given social network thereby the load and store actions should be defined. The Facebook friends list with the contact list is essential to assure the overlap and trust functionality.

$$r_2 = (s,\ (Load, \mathcal{P}_{r2}),\ FacebookFriendList) \tag{11}$$

$$r_3 = (s,\ (Store, \mathcal{P}_{r2}, t_1),\ FacebookFriendList) \tag{12}$$

where t_1 is a storage time defined as: while the application is installed; $\mathcal{P}_{r2}$ is a purpose defined as a set of p_2, p_3 and p_4: $\mathcal{P}_{r1} = \{p_2, p_3, p_4\}$; p_4= view the Facebook friend list on the screen.

For each Facebook friend, the list of common friends with the user is necessary for trust calculation.

$$r_4 = (s,\ (Load, p_3),\ FacebookCommonFriendLists) \tag{13}$$

A list of friends from other social networks improves the scores of overlap and trust.

$$r_5 = (s,\ (Load, \mathcal{P}_{r5}),\ TwitterFriendList) \tag{14}$$

$$r_6 = (s,\ (Store, \mathcal{P}_{r5}, t_1),\ TwitterFriendList) \tag{15}$$

where $\mathcal{P}_{r5}$ is a purpose defined as a set of p_5 and p_6: $\mathcal{P}_{r5} = \{p_5, p_6\}$; p_5= view the Twitter friend list on the screen; p_6= improve the overlap and trust score with Twitter friends;

$$r_7 = (s,\ (Load, \mathcal{P}r_7),\ LinkedInFriendList) \tag{16}$$

$$r_8 = (s,\ (Store, \mathcal{P}r_7, t2),\ LinkedInFriendList) \tag{17}$$

where $\mathcal{P}_{r7}$ is a purpose defined as a set of p_6 and p_7: $\mathcal{P}_{r7} = \{p_{6'}, p_7\}$; $p_{6'}$= improve the overlap and trust score with LinkedIn friends; p_7= view the LinkedIn friend list on the screen.

The second functionality of the application is to evaluate the behaviour of Facebook and Twitter friends to indicate potentially dangerous contacts. The behaviour evaluation is calculated by analysing the messages published by the given friend over time. The application needs a permission to load messages.

$$r_9 = (s,\ (Load, p_8),\ TwitterFriendMessages) \tag{18}$$

where p_8 is a purpose defined as 'calculate the Twitter friends behavior'.

$$r_{10} = (s,\ (Load, p_9),\ FacebookFriendMessages) \tag{19}$$

where p_9 is a purpose defined as 'calculate the Facebook friends behaviour'.

The third functionality proposes to view today Facebook and Twitter messages on the screen for the user.

$$r_{11} = (s,\ (Store, p_{10}, t_2),\ TodayTwitterFriendMessages) \tag{20}$$

where p_{10} is a purpose defined as 'view today Twitter messages'; t_2 is a storage time defined as: one day.

$$r_{12} = (s,\ (Store, p_{11}, t_2),\ TodayFacebookFriendMessages) \tag{21}$$

where p_{11} is a purpose defined as 'view today Facebook messages'.

The user has the option to share the scores by posting new messages on Facebook and Twitter. The user can also contribute to the research by sending the trust and behaviour anonymized statistics to the developer. Those actions should be taken with the user's express consent.

$$r_{13} = (s,\ (Transfer, p_{12}),\ FacebookFriendTrustScore) \tag{22}$$

where p_{12} is a purpose defined as 'share results on Facebook'.

$$r_{14} = (s,\ (Transfer, p_{13}),\ FacebookFriendTrustScore) \tag{23}$$

where p_{13} is a purpose defined as 'share results on Twitter'.

$$r_{15} = (s,\ (Transfer, p_{14}),\ TrustAndBehavior) \tag{24}$$

where p_{14} is a purpose defined as 'contribute to the improvement of the methodology'.

The final application has 15 rules that should be accepted by the user.

$$\mathcal{R}_{app} = \{r_1, r_2, r_3, \cdots, r_{14}, r_{15}\} \tag{25}$$

The rules r_1, r_2, r_3 and r_4 have a common purpose, all rules should be accepted to achieve the functionality mentioned in the purpose: 'calculate the trust'. Similarly, r_5 should be grouped with r_6 and r_7 with r_8. The rules from r_9 to r_{15} should be accepted one by one to achieve the aforementioned purpose (to get the functionality). Finally, we obtain 10 permissions to be added to the application to propose control to the user. The Table II recapitulates permissions.

To compare with actual permission systems, (a) iOS requires contact list, Facebook and Twitter access permissions. (b) Android requires 'internet', 'read_contacts' and 'get_accounts' access permissions. Facebook and Twitter connections are managed with APIs that requires permissions to be declared on the platform but the permission management will not be available for users in the mobile application by default. iOS permissions give certain transparency to the user but Android permissions are vague.

We obtained more fine-grained control of the application and the data including permissions to all necessary personal data, actions carried out on this data and corresponding purposes. The recapitulation table (Table II) clearly shows what data are used for what purpose. This kind of table can be added to the privacy policy to improve transparency.

V. Conclusion and Future Work

We modelled a permission system for mobile application regarding Privacy by Design. This permission system is data-oriented, thus the final user can easily understand what personal data is involved. We include the action that is missing from current iOS and Android permission systems, such as load and transfer, that improves transparency of the application.

The novelty is to include the purpose of the data usage into the permission system. The clear purpose will help users to understand better why the data is used and to judge whether this permission is needed. Purpose in permission also forces developers to apply the minimization principle: a developer cannot use the data if he cannot define the clear purpose of usage. The compulsory purpose definition should help guard against the abusive permission declaration 'in case'. Finally, purpose gives the user more fine-grained control, as the same

TABLE II
TABLE RECAPITULATING PERMISSIONS NEEDED FOR THE APPLICATION (LAST COLUMN IS A PERMISSION NUMBER)

Object	Action	Purpose	#
Contacts list	Load	View; Calculate Overlap and Trust	1
Facebook friends list	Load; Store		
Facebook common friends	Load		
Twitter friends list	Load; Store	View; Improve Overlap and Trust	2
LinkedIn friends list		View; Improve Overlap and Trust	3
Twitter messages	Load	Tw. friends behaviour	4
Facebook messages		Fb. friends behaviour	5
Today Tw. messages	Store	View Tw. messages	6
Today Fb. messages		View Fb. messages	7
Trust score	Transfer	Publish to Twitter	8
		Publish to Facebook	9
Trust and behaviour	Transfer	Contribute to research	10

data can be allowed to be used for one functionality but not for another. It is important for our system to integrate clear purpose and not a vague explanation (e.g., 'measure the frequency of application utilization' instead of 'improve user experience').

PbD states that the user should have a control over his data and be Privacy by Default, therefore permissions used in the application are revoked by default. Users should be clearly informed and asked to grant permission. Moreover, users should keep control of permissions during all the application use-time, therefore the permission setting must be available.

Our permission system helps developers to be compliant with the law; it defines what permissions the developer should add to the application, but in the current state it cannot ensure that all necessary permissions are really added. Our pattern indicates to the developer what should be added to the application to be more transparent, but if he decides to transfer data without asking permission, the pattern allows this (even if it goes against European law). The generated privacy policy can give the first indication permitting evaluation if the data usage is reasonable and the purpose is clear. Manual verification of an application can show the anomaly in permission system usage.

We aim to build a framework for the automatic management of a new permission system to simplify the developers work. We target the Android system first as it is more crucial due to the more open communication and data sharing and the vagueness of the current permission system.

The impact of new privacy-respective permission systems on users and developers could be measured by conducting the real-life experience. We aim to measure the impact of integration of the new permission system on design and development time, as well as particular situations and difficulties in applying the pattern. We also aim to evaluate user perceptions. We have an additional hypothesis that the explicative application with high transparency improves the user experience and leads to more positive perception of the same application, therefore the use of our permission system gives benefits to the application owner.

This work can be continued by developing an enforcement system automatically verifying whether all necessary permissions are properly defined. Many works propose systems monitoring mobile data flow, therefore the permission verification system can be based on one of the already proposed systems. Another important aspect for the developer is to be able to prove the application is compliant with the law. The system generating on-demand reports on the data, including the private data usage, can be developed.

REFERENCES

[1] TRUSTe. Consumer Mobile Privacy Insights Report. [retrieved: Apr., 2011]

[2] A. Cavoukian, “Privacy by design: The 7 foundational principles,” 2009.

[3] E. data protection regulators, “Opinion 02/2013 on apps on smart devices,” EU, Tech. Rep., Feb. 2013.

[4] K. D. Harris, “Privacy on the go,” California Department of Justice, Jan. 2013, pp. 1–27.

[5] W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka, “Towards Formal Analysis of the Permission-Based Security Model for Android,” in Wireless and Mobile Communications, 2009. ICWMC '09. Fifth International Conference on. IEEE Computer Society, 2009, pp. 87–92.

[6] K. W. Y. Au, Y. F. Zhou, Z. Huang, P. Gill, and D. Lie, “Short paper: a look at smartphone permission models,” in SPSM '11 Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM Request Permissions, Oct. 2011, pp. 63–67.

[7] R. Stevens, J. Ganz, V. Filkov, P. T. Devanbu, and H. Chen, “Asking for (and about) permissions used by Android apps.” MSR, 2013, pp. 31–40.

[8] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, “MockDroid: trading privacy for application functionality on smartphones,” in HotMobile '11: Proceedings of the 12th Workshop on Mobile Computing Systems and Applications, ser. HotMobile '11. ACM Request Permissions, Mar. 2011, pp. 49–54.

[9] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, “These aren't the droids you're looking for: retrofitting android to protect data from imperious applications,” in Proceedings of the 18th ACM conference on Computer and communications security, ser. CCS '11. ACM Request Permissions, Oct. 2011, pp. 639–652.

[10] M. Nauman, S. Khan, and X. Zhang, “Apex: extending Android permission model and enforcement with user-defined runtime constraints,” in ASIACCS '10: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security. ACM Request Permissions, Apr. 2010, pp. 328–332.

[11] S. Holavanalli et al., “Flow Permissions for Android,” in Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, 2013, pp. 652–657.

[12] J. Tam, R. W. Reeder, and S. Schechter, “Disclosing the authority applications demand of users as a condition of installation,” Microsoft Research, 2010.

[13] S. Egelman, A. P. Felt, and D. Wagner, “Choice Architecture and Smartphone Privacy: There's a Price for That,” in Proceedings of the 11th Annual Workshop on the Economics of Information Security (WEIS), 2012.

[14] M. Lane, “Does the android permission system provide adequate information privacy protection for end-users of mobile apps? ,” in 10th Australian Information Security Management Conference, Dec. 2012, pp. 65–73.

[15] P. G. Kelley et al., “A conundrum of permissions: Installing applications on an android smartphone,” in Proceedings of the 16th International Conference on Financial Cryptography and Data Security, ser. FC’12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 68–79.

[16] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, “Android permissions demystified,” in CCS ’11: Proceedings of the 18th ACM conference on Computer and communications security. ACM Request Permissions, Oct. 2011, pp. 627–638.

[17] P. Berthomé and J.-F. Lalande, “Comment ajouter de la privacy after design pour les applications Android? (How to add privacy after design to Android applications?),” Jun. 2012.

[18] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, “PiOS: Detecting Privacy Leaks in iOS Applications.” 2011.

[19] C. Gibler, J. Crussell, J. Erickson, and H. Chen, “AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale,” in Trust and Trustworthy Computing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 291–307.

[20] T. Vidas, N. Christin, and L. Cranor, “Curbing android permission creep,” in In W2SP, 2011.

[21] Y. Zhang et al., “Vetting undesirable behaviors in android apps with permission use analysis,” in CCS ’13: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. ACM, Nov. 2013, pp. 611–622.

[22] W. Xu, F. Zhang, and S. Zhu, “Permlyzer: Analyzing permission usage in Android applications,” Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on, 2013, pp. 400–410.

[23] W. Luo, S. Xu, and X. Jiang, “Real-time detection and prevention of android SMS permission abuses,” in SESP ’13: Proceedings of the first international workshop on Security in embedded systems and smartphones. ACM Request Permissions, May 2013, pp. 11–18.

[24] A. P. Felt, K. Greenwood, and D. Wagner, “The effectiveness of application permissions,” in WebApps’11: Proceedings of the 2nd USENIX conference on Web application development. USENIX Association, Jun. 2011, pp. 7–7.

[25] R. Böhme and S. Köpsell, “Trained to accept?: a field experiment on consent dialogs,” in CHI ’10: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM Request Permissions, Apr. 2010, pp. 2403–2406.

[26] E. Smith, “iPhone applications and privacy issues: An analysis of application transmission of iPhone unique device identifiers UDIDs,” October 2010.

[27] “Mobile Apps Study,” Future of Privacy Forum (FPF), pp. 1–16, Jun. 2012.

[28] C. Perez, B. Birregah, and M. Lemercier, “A smartphone-based online social network trust evaluation system,” Social Network Analysis and Mining, vol. 3, no. 4, 2013, pp. 1293–1310.

Effectively Updating Co-location Patterns in Evolving Spatial Databases

Jin Soung Yoo and Hima Vasudevan

Computer Science Department
Indiana University-Purdue University Fort Wayne
Indiana, USA 46805
Email: yooj,vasuh02@ipfw.edu

***Abstract*—Spatial co-location mining has been used for discovering spatial event sets which show frequent association relationships based on the spatial neighborhood. This paper presents a problem of finding co-location patterns on evolving spatial databases which are constantly updated with fresh data. Maintaining discovered spatial patterns is a complicated process when a large spatial database is changed because new data points make spatial relationships with existing data points on the continuous space as well as among themselves. The change of neighbor relations can affect co-location mining results with invalidating existing patterns and introducing new patterns. This paper presents an algorithm for effectively updating co-location analysis results and its experimental evaluation.**

***Keywords*–Spatial association mining; Co-location pattern; Incremental update**

I. INTRODUCTION

As one of the spatial data mining tasks, spatial association mining is often used for discovering spatial dependencies among objects [1]–[4]. A spatial co-location represents a set of spatial features which are frequently observed together in a nearby area [3]. Examples of frequently co-located features/events include symbiotic species such as West Nile incidents and stagnant water sources in epidemiology, and interdependent events such as a car accident, traffic jam, policemen and ambulances in transportation. In business, co-location patterns can be used for finding relationships among services requested by mobile users in geographic proximity.

Most of the spatial association mining works [3]–[10] assume that all data is available at the start of data analysis. However, many application domains including location-based services, public safety, transportation and environmental monitoring collect their data periodically or continuously. For example, a police department accumulates, on average, 10,000 crime incidents per month [11]. For Earth observation, daily climate measurement values are collected at every 0.5 degree grid of the globe [12]. For keeping the analysis result coherent with respect to the most recent database status, discovered patterns should be updated.

The problem of updating spatial co-location patterns presents more challenges than updating frequent itemsets in a traditional transaction database. In the classical association analysis, the database update means the simple addition of new transaction records, or the deletion of existing records. Newly added transaction records are separately handled from existing records because the database is a collection of disjoint transaction records. In contrast, when a spatial database is updated, a new data point can make neighbor relationships with existing data points as well as other new data points on the continuous space. Thus, all neighbor relationships in the updated database should be examined for the maintenance of co-location patterns. The spatial pattern mining process is a computational and data intensive task, therefore simply re-executing a state-of-the-art co-location mining algorithm, whenever the database is updated, can result in an explosion of required computational and I/O resources. This paper proposes an algorithm for effectively updating discovered co-location patterns with the addition of spatial data points.

The remainder of this paper is organized as follows. Section II presents the basic concept of co-location pattern mining and the related work. Section III describes our algorithmic design concept for incremental co-location mining and the proposed algorithm. Its experimental evaluation is presented in Section IV. This paper will conclude in Section V.

II. BASIC CONCEPT AND RELATED WORK

The preliminary knowledge of spatial co-location pattern mining and the related work are presented in this section.

A. Basic concept of spatial co-location mining

Let $E = \{e_1, \dots, e_m\}$ be a set of event types, and $S = \{o_1, \dots, o_n\}$ be a set of their objects with geographic location. When the Euclidean metric is used for the neighbor relationship R, two objects o_i and o_j are neighbors of each other if the ordinary distance between them is not greater than a neighbor distance threshold d. A **co-location** X is a set of event types, $\{e_1, \dots, e_k\} \subseteq E$, whose objects are frequently neighbors to each other on space. The **co-location instance** I of X is defined as a set of event objects, $I \subseteq S$, which includes all types in X and makes a clique under R.

The prevalence strength of a co-location is often measured by participation index value [3]. The **participation index** $PI(X)$ of $X = \{e_1, \dots, e_k\}$ is defined as

$$PI(X) = \min_{e_i \in X}\{PR(X, e_i)\}, \quad (1)$$

where $1 \leq i \leq k$, and $PR(X, e_i)$ is the **participation ratio** of event type e_i in X, which is the fraction of objects of event e_i in the neighborhood of instances of $X - \{e_i\}$, i.e. $PR(X, e_i) = \frac{|distinct\,objects\,of\,e_i\,in\,instances\,of\,X|}{|objects\,of\,e_i|}$. If $PI(X)$ is greater than a given minimum prevalence threshold, we say X is a *prevalent co-located event set* or a *co-location*.

B. Related work

The problem of mining association rules based on spatial relationships (e.g., proximity and adjacency) was first discussed by Koperski et al. [1]. Shekhar, et al. [3] defines the co-location pattern and proposes a join-based co-location mining algorithm. Morimoto [2] studies the same problem to discover frequent neighboring service class sets. A space partitioning and non-overlap grouping scheme is used for finding neighboring objects. Yoo et al. [4], [10] propose join less algorithms to reduce the number of expensive spatial join operations in finding co-location instances. Celik et al. [8] extends the notion of co-location to a local zone-scale pattern. Eick et al. [13] proposes a framework for mining regional co-location patterns and Mohan et al. [14] presents a graph based approach for regional co-location discovery. Recognizing the dynamic nature of database, much effort has been devoted to the problem of incrementally mining frequent itemsets in classical association rule mining literature [15]–[19]. However, to find the problem to update co-location patterns in spatial data mining literature is rare. The most similar work with ours is He et al. [20] which is compared in our experimental evaluation.

III. INCREMENTAL CO-LOCATION MINING

Let $S_{old} = \{o_1, \ldots, o_n\}$ be a set of old data points in a spatial database and $S_{in} = \{o_{n+1}, \ldots, o_{n+h}\}$ be a set of new data points added in the database. Let S be all data points in the updated database, i.e., S=$S_{old} \cup S_{in}$. There are two types of co-location in the update. The *retained co-location* is an event set prevalent in both S_{old} and S. The *emerged co-location* is an event set not prevalent in S_{old} but prevalent in S. We propose an algorithm of Effective Update of COLOCation patterns (EUCOLOC). The proposed algorithm has two update stages. The first update stage examines only neighbor relationships of new data points, and finds all retained co-locations and some emerged co-locations. If an emerged set is found from the first update, the second update stage is triggered for finding other emerged co-location patterns in the updated database. Figure 3 shows the pseudo code of EUCOLOC algorithm.

A. Neighborhood Process

Directly finding all co-location instances forming clique neighbor relationships from spatial data is computationally expensive. Instead, we process the neighbor relationships related to the new data points S_{in}.

Definition 1: The neighborhood of a new object $o \in S_{in}$, **new neighborhood** $n_{new}(o)$, is defined to $\{o, o_2, \ldots, o_p | o_i \in S \wedge R(o, o_i)$=true $\wedge$ o's event type $<$ o_i's event type$\}$, where $2 \leq i \leq p$.

We assume there is a total ordering among the event types (i.e., a lexicographic order $\preceq_e$). R is a neighbor relationship function. Next, if an existing data point has a neighbor relationship with at least one new data point, its neighborhood is changed.

Definition 2: The **changed neighborhood** of an old object $o \in S_{old}$, $n_{chg}(o)$, is defined to $\{o, o_2, \ldots, o_p | o_i \in S \wedge \exists o_i \in S_{in} \wedge R(o, o_j)$=true $\wedge$ o's event type $<$ o_i's event type $\}$, where $2 \leq i \leq p$.

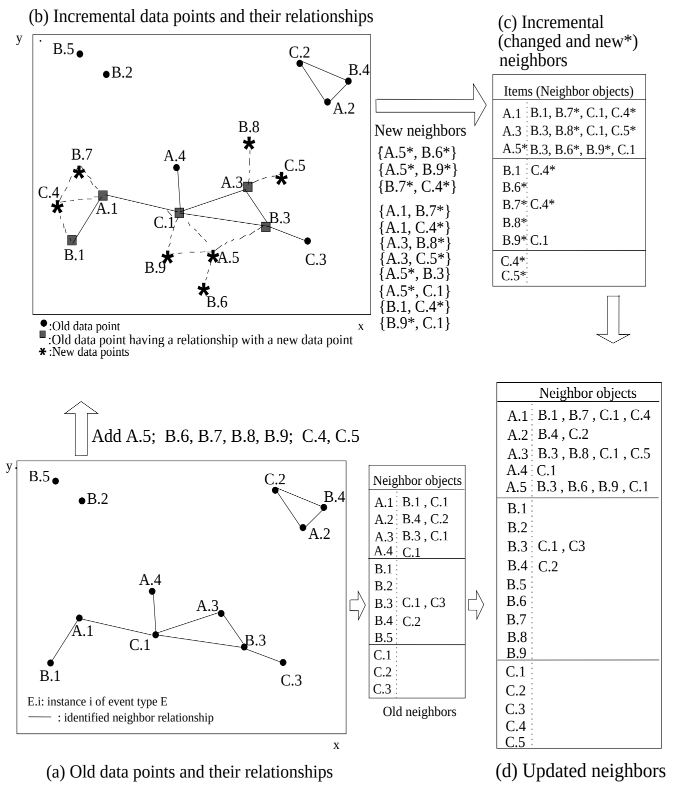


Figure 1. New, Changed and Updated Neighborhoods

Let $N_{new} = \{n_{new}(o_1), \ldots, n_{new}(o_h)\}$ be a set of all new neighborhoods for S_n and $N_{chg} = \{n_{chg}(o_1), \ldots, n_{chg}(o_q)\}$ be a set of all changed neighborhoods where $\{o_1, \ldots, o_q\} \subseteq S_{old}$. We call the union of N_{new} and N_{chg} to **incremental neighborhood set** (N_{inc}).

When an increment S_{in} is added as shown in Figure 1 (b), the EUCOLOC algorithm first finds all neighbors (NP) of new data points in S using a geometric method or a spatial query method (*Algorithm Line 2*). The incremental neighborhood set (N_{inc}) is prepared by finding new neighborhoods from NP and detecting changed neighborhoods from the old neighborhoods N_{old} (*Line 3* & Figure 1 (c)). Figure 1 (d) shows the entire neighborhood information (N) of the updated database (*Line 4*).

B. First update and detection

Let an event set be a *border event set* if the event set's all proper subsets are prevalent, but not prevalent itself. The border sets are used for detecting an emerging co-location without the generation and testing of many unnecessary candidates. The candidate event sets for the first update are previous co-located event sets (P_{old}) and border event sets (B_{old}) (*Line 7*). The incremental co-location instances of the candidate event sets are searched from the incremental neighborhoods (N_{inc}) without examining the entire neighbor relationships. (*Line 8* & Figure 1 (c)). A filter-and-refine search strategy is used for finding co-location instances. Let $SI = \{o_1, o_2, \ldots, o_k\}$ be a set of objects of a candidate set $c = \{e_1, e_2, \ldots, e_k\}$ where $e_1 < e_2 \ldots < e_k$. If the first object o_1 has neighbor relationships with all other objects in the set, SI is called a *star instance* of c. The start instances of $\{e_1, e_2, \ldots, e_k\}$ are

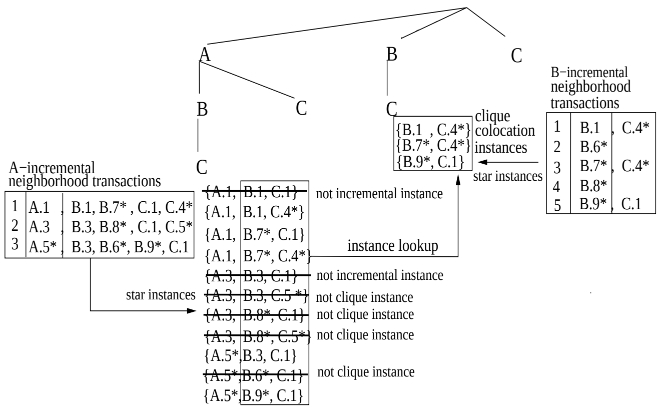

A
B
C
B
C
C
C
A−incremental neighborhood transactions
1 A.1, B.1, B.7*, C.1, C.4*
2 A.3, B.3, B.8*, C.1, C.5*
3 A.5*, B.3, B.6*, B.9*, C.1
star instances
~~{A.1, B.1, C.1}~~ not incremental instance
{A.1, B.1, C.4*}
{A.1, B.7*, C.1}
{A.1, B.7*, C.4*}
instance lookup
~~{A.3, B.3, C.1}~~ not incremental instance
~~{A.3, B.3, C.5*}~~ not clique instance
~~{A.3, B.8*, C.1}~~ not clique instance
~~{A.3, B.8*, C.5*}~~ not clique instance
{A.5*, B.3, C.1}
~~{A.5*, B.6*, C.1}~~ not clique instance
{A.5*, B.9*, C.1}
{B.1, C.4*}
{B.7*, C.4*}
{B.9*, C.1}
clique colocation instances
star instances
B−incremental neighborhood transactions
1 B.1, C.4*
2 B.6*
3 B.7*, C.4*
4 B.8*
5 B.9*, C.1

Figure 2. Event subsets and their instance search space

collected from the neighborhoods of e_1 according to Definition 1 and 2. The candidate instance $SI = \{o_1, o_2, \ldots, o_k\}$ of $c = \{e_1, \ldots, e_k\}$ becomes a true co-location instance of c if its subinstance $\{o_2, \ldots, o_k\}$ forms a clique. The cliqueness of the subinstance can be checked by simply querying the co-location instances of c's sub event set $\{e_2, \ldots, e_k\}$ if the subinstance has at least one new point, as shown in Figure 2.

The participation index of a candidate is computed with its incremental co-location instances (CI_c) and previous *instance metadata* (old_PB_info) which has the object information of its old co-location instances (*Line 13*). The prevalence of a candidate $c = \{e_1, \ldots, e_k\}$ is updated with

$$incPI(c) = \min_{e_i \in c}\{incPR(c, e_i)\}, \qquad (2)$$

where $1 \le i \le k$, and $incPR(c, e_i)$ is the updated participation ratio of event type e_i with the incremental co-location instances of c, $incPR(c, e_i) = \frac{|O_i \bigcup I_i|}{|S_{old_i}| + |S_{in_i}|}$, where $|S_{old_i}|$ is the total number of old objects of e_i, $|S_{in_i}|$ is the total number of new objects of e_i, O_i is a set of distinct objects of e_i in the old co-location instances of c, and I_i is a set of distinct objects of e_i in the incremental co-location instances of c. If the participation index is greater than min_prev, the event set is a co-location ($\in P$) (*Line 14-15*). If this co-location is from the border set B_{old}, it also becomes an emerged co-location ($\in ES$) (*Line 16-17*).

C. Second update stage

If any emerged set is found from the first update stage, there is a possibility of finding other emerged event sets according to the following lemma.

Lemma 1: Let X be a co-located event set that is prevalent in the updated set $S = S_{old} \cup S_{in}$ but not prevalent in the old set S_{old}. Then there exists a subset $Y \subseteq X$ such that Y is an emerged event set.

Proof: Let Y be a minimal cardinality subset of X that is prevalent in S, not in S_{old}. Since Y is a prevalent event set in S, so are all of is proper subsets. However, by the minimality of Y, none of these subsets are new prevalent sets in S. Thus, Y is a border set in S_{old}, and $Y \subseteq X$ as claimed. ■

In the second update, a candidate is an event set which has at least one emerged event set as its subset (*Line 29*).

```
1:  procedure PREPROCESS
2:      $NP \leftarrow$ search_neigh_pairs($S_{in}$, $S_{old}$, $R$)
3:      $N_{inc} \leftarrow$ gen_incr_neigh_trans($NP$, $N_{old}$)
4:      $N \leftarrow$ gen_upd_neigh_trans($N_{old}$, $N_{inc}$)
5:  end procedure

6:  procedure FIRSTUPDATEDETECTION
7:      $C \leftarrow P_{old} \cup B_{old}$
8:      $SI \leftarrow$ scan_incr_star_inst($C$, $N_{inc}$)
9:      $k \leftarrow 2$
10:     while $C_k \neq \emptyset$ do
11:         for all $c \in C_k$ do
12:             $CI_c \leftarrow$ find_incr_clique_inst($SI_c$, $NP$)
13:             PI $\leftarrow$ compute_incPI($old\_PB\_info$, $CI_c$)
14:             if PI $\geq min\_prev$ then
15:                 $P \leftarrow P \cup c$
16:                 if $c \in B_{old}$ then
17:                     $ES \leftarrow ES \cup c$
18:                 end if
19:             else
20:                 $B \leftarrow B \cup c$
21:             end if
22:         end for
23:         $k \leftarrow k$+1
24:     end while
25: end procedure

26: procedure SECONDUPDATE
27:     if $ES \neq \emptyset$ then
28:         $k \leftarrow 3$
29:         $C_k \leftarrow$ gen_sizeK_candidates($P_{k-1}$, $ES_{k-1}$)
30:         while $C_k \neq \emptyset$ do
31:             $SI \leftarrow$ scan_star_instances($C_k$, $N$)
32:             for all $c \in C_k$ do
33:                 $CI_c \leftarrow$ find_clique_instances($SI_c$)
34:                 if compute_PI($CI_c$) $\geq min\_prev$ then
35:                     $P \leftarrow P \cup c$; $ES = ES \cup c$
36:                 else $B \leftarrow B \cup c$
37:                 end if
38:             end for
39:             $k \leftarrow k$+1
40:             $C_k \leftarrow$ gen_sizeK_candidates($P_{k-1}$, $ES_{k-1}$)
41:         end while
42:     end if
43:     $P_{old} \leftarrow P$; $B_{old} \leftarrow B$; $N_{old} \leftarrow N$; $S_{old} \leftarrow S_{old} \cup S_{in}$
44:     return $P$;
45: end procedure
```

Figure 3. EUCOLOC algorithm

The star instances of candidates are collected from the entirely updated neighborhood transactions (N) (*Line 31*). The true co-location instances are filtered from the candidate instances. The prevalence value of a candidate is calculated using the original participation index (Equation (1)) because this set is a new candidate with no previous instance metadata. If the candidate is prevalent, it becomes an emerged co-location. Otherwise, the set is included in the border set for future update. The second update is repeated with the increase of the pattern size until no more candidate (*Line 30-41*).

IV. EXPERIMENTAL EVALUATION

We compared the performance of EUCOLOC with two other co-location mining algorithms. One (denoted as IMCP in this paper) has an update function [20]. The implementation of this algorithm is based on our understanding of the work. The other (denoted as GeneralColoc) does not have an update function [4]. All the experiments were performed on a Linux system with 8.0 GB memory, and 2.67 GHz CPU.

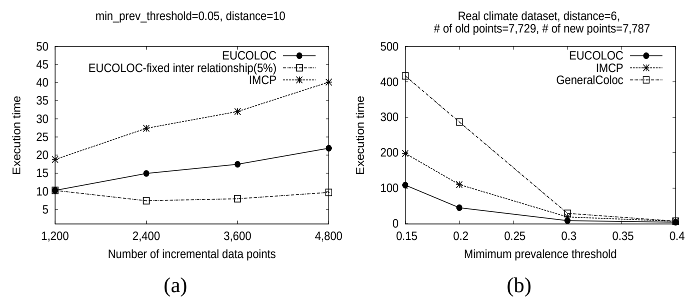


Figure 4. Experiment Result (a) By incremental data size (b) By prevalence threshold

In the first experiment, we compared the performance of EUCOLOC and IMCP by varying the incremental size of synthetic data. The number of distinct event types was 50. The number of old data points was 10,020. The first incremental set has 1,200 data points. The second incremental set is two times bigger than the first set. The third one was three times bigger than the first set, and so on. The ratio of old data points which have relationships with new points was increased with the increase of new data (i.e., 5%, 10%, 15% and 20%). As shown in Figure 4 (a), the execution times of both EUCOLOC and IMCP increased with the incremental data size. The EUCOLOC showed better performance than IMCP. When the ratio of relationships with old data points was fixed to 5%, the execution times of EUCOLOC were stable, or very slowly increased. The performance of EUCOLOC depends on the inter-neighbor relationship ratio.

We also conducted the evaluation of EUCOLOC with real climate measurement data [12]. The total number of processed event types was 18. 7,728 event records were used for the old data. 7,787 new event records were added for the incremental data. We used 6 as a neighborhood distance, which means 6 cells on latitude-longitude spherical grids, where each grid cell is 1 degree $\times$ 1 degree. About half of the old event objects had neighbor relationships with the new ones. Figure 4 (b) shows the result. EUCOLOC showed slowly increasing execution time than other algorithms when the prevalence threshold was decreased.

V. CONCLUSION

In this paper, we presented an algorithm for efficiently mining co-location patterns in evolving spatial databases. The proposed algorithm has two update stages. The first update stage is 1) to avoid the generation and testing of many unnecessary candidates using the border concept, 2) to search only incremental neighborhoods for the update, and 3) to update the prevalence value of current co-locations with their incremental instances and minimal previous co-located object information. The second update stage is used for only finding new co-located event sets (emerged ones). The initial experimental evaluation shows our algorithmic design decision is effective in updating discovered co-location patterns. The proposed algorithm can be easily extended to handle the case of deleted data points. Our approach can be adopted for the cases of change of important parameters, such as neighbor distance and prevalence threshold. In the future, we plan to explore these problems.

REFERENCES

[1] K. Koperski and J. Han, “Discovery of Spatial Association Rules in Geographic Information Databases,” in Proceedings of the International Symposium on Large Spatial Data bases, 1995, pp. 47–66.

[2] Y. Morimoto, “Mining Frequent Neighboring Class Sets in Spatial Databases,” in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2001, pp. 353–358.

[3] Y. Huang, S. Shekhar, and H. Xiong, “Discovering Co-location Patterns from Spatial Datasets: A General Approach,” IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 12, 2004, pp. 1472–1485.

[4] J. S. Yoo and S. Shekhar., “A Join-less Approach for Mining Spatial Co-location Patterns,” IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 10, 2006, pp. 1323–1337.

[5] J. S. Yoo and S. Shekhar, “A Join-less Approach for Spatial Co-location Mining: A Summary of Results,” in Proceedings of the IEEE International Conference on Data Mining, 2005, pp. 813–816.

[6] H. Xiong, S. Shekhar, Y. Huang, V. Kumar, X. Ma, and J. S. Yoo, “A Framework for Discovering Co-location Patterns in Data Sets with Extended Spatial Objects,” in Proceedings of the SIAM International Conference on Data Mining, 2004, pp. 78–89.

[7] J. Yoo and M. Bow, “Finding N-Most Prevalent Colocated Event Sets,” in Proceedings of the International Conference on Data Warehousing and Knowledge Discovery, 2009, pp. 415–427.

[8] M. Celik, J. M. Kang, and S. Shekhar, “Zonal Co-location Pattern Discovery with Dynamic Parameters,” in Proceedings of the IEEE International Conference on Data Mining, 2007, pp. 433 – 438.

[9] J. S. Yoo and M. Bow, “Mining Spatial Colocation Patterns: A Different Framework,” Data Mining and Knowledge Discovery, vol. 24, no. 1, 2012, pp. 159–194.

[10] J. S. Yoo and S. Shekhar, “A Partial Join Approach for Mining Co-location Patterns,” in Proceedings of the ACM International Symposium on Advances in Geographic Information Systems, 2004, pp. 241–249.

[11] “San Francisco Crime Incidents,” https://data.sfgov.org/.

[12] “Earth Observation Data,” http://data.giss.nasa.gov/.

[13] C. F. Eick, R. Parmar, W. Ding, T. F. Stepinski, and J. Nicot, “Finding Regional Co-location Patterns for Sets of Continuous Variables in Spatial Datasets,” in Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2008, pp. 1–10.

[14] P. M. et al., “A Neighborhood Graph based Approach to Regional Co-location Pattern Discovery: A Summary of Results ,” in Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2011, pp. 122–132.

[15] N. Ayan, A. Tansel, and E. Arkyn, “An Efficient Algorithm to Update Large Itemsets with Early Pruning,” in Proceedings of the International Conference on Knowledge Discovery and Data Mining, 1999, pp. 287–291.

[16] D. Cheung, J. Han, V. Ng, and C. Y. Wong, “Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique,” in Proceedings of the IEEE International Conference on Data Engineering, 1996, pp. 106 – 114.

[17] S. Thomas and S. Chakravarthy, “Incremental Mining of Constrained Associations,” High Performance Computing (HiPC), vol. 1970, 2000, pp. 547–558.

[18] D. Cheung, S. D. Lee, and D. Kao, “A General Incremental Technique for Maintaining Discovered Association Rules ,” in Proceedings of the International Conference on Databases Systems for Advanced Applications, 1997, pp. 185 – 194.

[19] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka, “An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases,” in Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining, 1997, pp. 263–266.

[20] J. He, Q. He, F. Qian, and Q. Chen, “Incremental Maintenance of Discovered Spatial Colocation Patterns,” in Proceedings of Data Mining Workshop, 2008, pp. 399 – 407.