



ICSEA 2011

The Sixth International Conference on Software Engineering Advances

ISBN: 978-1-61208-165-6

October 23-29, 2011

Barcelona, Spain

ICSEA 2011 Editors

Luigi Lavazza, Università dell'Insubria - Varese, Italy

Luis Fernandez-Sanz, Universidad de Alcalá, Spain

Oleksandr Panchenko, Hasso Plattner Institute for Software Systems Engineering -
Potsdam, Germany

Teemu Kanstrén, VTT Technical Research Centre of Finland - Oulu, Finland

ICSEA 2011

Forward

The Sixth International Conference on Software Engineering Advances (ICSEA 2011), held on October 23-29, 2011 in Barcelona, Spain, continued a series of events covering a broad spectrum of software-related topics.

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

The conference had the following tracks:

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

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

We take here the opportunity to warmly thank all the members of the ICSEA 2011 technical program committee as well as the numerous reviewers. The creation of such a broad and high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and efforts to contribute to the ICSEA 2011. We truly believe that thanks to all these efforts, the final conference program consists of top quality contributions.

This event could also not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the ICSEA 2011 organizing committee for their help in handling the logistics and for their work that is making this professional meeting a success.

We hope the ICSEA 2011 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in software engineering research.

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

ICSEA 2011 Chairs

Advisory Chairs

Herwig Mannaert, University of Antwerp, Belgium

Jon G. Hall, The Open University - Milton Keynes, UK

Mira Kajko-Mattsson, Stockholm University & Royal Institute of Technology, Sweden

Luigi Lavazza, Università dell'Insubria - Varese, Italy

Roy Oberhauser, Aalen University, Germany

Elena Troubitsyna, Åbo Akademi University, Finland

Luis Fernandez-Sanz, Universidad de Alcala, Spain

Research Institute Liaison Chairs

Oleksandr Panchenko, Hasso Plattner Institute for Software Systems Engineering - Potsdam, Germany

Teemu Kanstrén, VTT Technical Research Centre of Finland - Oulu, Finland

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

Georg Buchgeher, Software Competence Center Hagenberg GmbH, Austria

Simon Tsang, Telcordia - Piscataway, USA

Industry/Research Chairs

Herman Hartmann, University of Groningen, The Netherlands

Hongyu Pei Breivold, ABB Corporate Research, Sweden

Special Area Chairs

Formal Methods

Paul J. Gibson, Telecom & Management SudParis, France

Business and process techniques

Maribel Yasmina Santos, University of Minho, Portugal

Testing and Validation

Florian Barth, University of Mannheim, Germany

ICSEA 2011

Committee

ICSEA Advisory Chairs

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

ICSEA 2011 Research Institute Liaison Chairs

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

ICSEA 2011 Industry/Research Chairs

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

ICSEA 2011 Special Area Chairs

Formal Methods

Paul J. Gibson, Telecom & Management SudParis, France

Business and process techniques

Maribel Yasmina Santos, University of Minho, Portugal

Testing and Validation

Florian Barth, University of Mannheim, Germany

ICSEA 2011 Technical Program Committee

Adla Abdelkader University of Oran, Algeria
Mohammed Aboulsamh, University of Oxford, UK
Syed Nadeem Ahsan, TU-Graz, Austria
Ahmed Al-Moayed, Hochschule Furtwangen University, Germany
Zakarya Alzamil, King Saud University - Riyadh, Saudi Arabia
Vincenzo Ambriola, Università di Pisa, Italy
Francesca Arcelli, UNiversity of Milano Bicocca, Italy

Cyrille Artho, RCIS/AIST - Tokyo, Japan
Rodrigo Assad, CESAR, Brazil
Gilbert Babin, HEC Montréal, Canada
Rami Bahsoon, The University of Birmingham, UK
Muneera Bano, International Islamic University - Islamabad, Pakistan
Florian Barth, University of Mannheim, Germany
Gabriele Bavota, University of Salerno, Italy
Noureddine Bellhatir, University of Grenoble, France
Simona Bernardi, Centro Universitario de la Defensa / Academia General Militar - Zaragoza, Spain
Kenneth Boness, Reading University, UK
Crescencio Bravo Santos, University of Castilla-La Mancha - Ciudad Real, Spain
Hongyu Pei Breivold , ABB Corporate Research, Sweden
Georg Buchgeher, Software Competence Center Hagenberg GmbH, Austria
David W. Bustard, University of Ulster - Coleraine, UK
Fabio Calefato, University of Bari, Italy
José Carlos Metrôlho, Polytechnic Institute of Castelo Branco, Portugal
Bengt Carlsson, Blekinge Institute of Technology, Sweden
Alejandra Cechich, Universidad Nacional del Comahue - Neuquen, Argentina
Alexandra Suzana Cernian, University POLITEHNICA of Bucharest, Romania
Antonin Chazalet, France Telecom (NRS), France
Yoonsik Cheon, University of Texas at El Paso, USA
Morakot Choetkiertikul, Mahidol University, Thailand
Andrew Connor, Auckland University of Technology, New Zealand
Rebeca Cortázar, University of Deusto - Bilbao, Spain
Lirong Dai, Seattle University, USA
Darren Dalcher, Middlesex University - London, UK
Paulo Asterio de Castro Guerra, Tapijara Programação de Sistemas Ltda. - Lambari, Brazil
Claudio de la Riva, Universidad de Oviedo - Gijon, Spain
Steven A. Demurjian, The University of Connecticut - Storrs, USA
Giovanni Denaro, Università degli Studi di Milano - Bicocca, Italy
Antinisca Di Marco, University of L'Aquila - Coppito (AQ), Italy
Van Nuffel Dieter, University of Antwerp, Belgium
Sebastian Dochow, University of Freiburg, Germany
Lydie du Bousquet, Laboratoire d'Informatique de Grenoble, France
Lars Ebrecht, German Aerospace Centre (DLR), Germany
Juho Eskeli, VTT, Finland
Umar Farooq, SMART Technologies Inc., Canada
Fausto Fasano, University of Molise, Italy
João M. Fernandes, Universidade do Minho - Braga, Portugal
Luis Fernandez-Sanz, Universidad de Alcalá, Spain
Felipe Ferraz, C.E.S.A.R, Brazil
Stoyan Garbatov, Instituto de Engenharia de Sistemas e Computadores Investigação e Desenvolvimento - Lisboa, Portugal
José García-Fanjul, University of Oviedo, Spain
Angelo Gargantini, University of Bergamo, Italy
Christophe Gaston, CEA LIST - Gif sur Yvette, France

Michael Gebhart KIT - Karlsruhe Institute of Technology, Germany
Paul J. Gibson, Telecom & Management SudParis, France
Robert L. Glass, Griffith University - Brisbane, Australia
Vic Grout, Glyndwr University - Wrexham, UK
Sebastian Günther, Otto-von-Guericke-Universität Magdeburg, Germany
Ensar Gul, Marmara University - Istanbul, Turkey
Zhensheng Guo, Siemens, Germany
Bidyut Gupta, Southern Illinois University - Carbondale, USA
Jon G. Hall, The Open University - Milton Keynes, UK
Herman Hartmann, Synopsys - Eindhoven, The Netherlands
Zeljko Hocenski, University Josip Juraj Strossmayer of Osijek, Croatia
Noraini Ibrahim University of Technology Malaysia (UTM), Malaysia
Naveed Ikram, International Islamic University - Islamabad, Pakistan
Muhammad Ilyas, Johannes Kepler University (JKU) - Linz, Austria
Visar Januzaj, Technische Universität Darmstadt, Germany
Antonio Javier García Sánchez, Technical University of Cartagena, Spain
Dayang Norhayati Abang Jawawi, Universiti Teknologi Malaysia (UTM) - Johor, Malaysia
Marcellin Julius Nkenlifack, Univeristé de Dschang - Bandjoun, Cameroun
Nevin Vunka Jungum, University of Technology Mauritius (UTM), Mauritius
Hermann Kaindl, TU-Wien, Austria
Mira Kajko-Mattsson, Stockholm University & Royal Institute of Technology, Sweden
Ahmed Kamel, Concordia College - Moorhead, USA
Teemu Kanstrén, VTT Technical Research Centre of Finland - Oulu, Finland
Tatjana Kapus, University of Maribor, Slovenia
Iman Keivanloo, Concordia University - Montreal, Canada
Petri Kettunen, Helsinki University of Technology, Finland
Holger Kienle, Mälardalen University, Sweden
William Knottenbelt, Imperial College London, UK
Radek Kocí, Brno University of Technology, Czech Republic
Georges Edouard Kouamou, National Advanced School of Engineering - Yaoundé, Cameroon
Ondrej Krejcar, VSB - Technical University of Ostrava, Czech Republic
Natalia Kryvinska, University of Vienna, Austria
Cyril S. Ku, William Paterson University, USA
Sukhamay Kundu, Louisiana State University - Baton Rouge, USA
Eugenijus Kurilovas, Vilnius University, Lithuania
Raquel Lacuesta Gilaberte, Zaragoza University, Spain
Alla Lake, LInfo Systems, LLC - Greenbelt, USA
Luigi Lavazza, Università dell'Insubria - Varese, Italy
Cynthia Y. Lester, Tuskegee University, USA
Plinio Sá Leitão-Junior, Federal University of Goias, Brazil
Cati Lladó, Universitat de les Illes Balears, Spain
Maria Teresa Llano Rodriguez, Heriot-Watt University, UK
Sérgio F. Lopes, University of Minho, Portugal
Juan Pablo López-Grao, University of Zaragoza, Spain
Sarah Löw, University of Innsbruck, Austria
Ricardo J. Machado, University of Minho, Portugal

Oliver Maeckel, Siemens AG, Corporate Technology - Munich, Germany
Nicos Malevris, Athens University of Economics and Business, Greece
Herwig Mannaert, University of Antwerp, Belgium
Eda Marchetti, ISTI-CNR-Pisa, Italy
Leonardo Mariani, University of Milan Bicocca, Italy
Adriana Martín, UNPA & GIISCo COMAHUE, Argentina
Andrew McDonough, Atos Origin, Spain
Karl Meinke, Blekinge Institute of Technology, Sweden
Jose Merseguer, Universidad de Zaragoza, Spain
Henry Muccini, University of L'Aquila, Italy
Muhanna Muhanna, University of Nevada - Reno, USA
Natalja Nikitina, KTH (Royal Institute of Technology) - Stockholm, Sweden
Mara Nikolaidou, Harokopio University of Athens, Greece
Roy Oberhauser, Aalen University, Germany
Pablo Oliveira Antonino de Assis, Fraunhofer Institute for Experimental Software Engineering - IESE, Germany
Rocco Oliveto, University of Molise, Italy
Flavio Oquendo, European University of Brittany - UBS/VALORIA, France
Claus Pahl, Dublin City University, Ireland
Marcos Palacios, University of Oviedo, Spain
Oleksandr Panchenko, Hasso Plattner Institute for Software Systems Engineering - Potsdam, Germany
Päivi Parviainen VTT, Software Technologies Center, Finland
Aljosa Pasic, ATOS Research, Spain
Fabrizio Pastore, University of Milano - Bicocca, Italy
Asier Perillos, University of Deusto, Spain
Óscar Pereira, Instituto de Telecomunicações - University of Aveiro, Portugal
David Pheanis, Arizona State University, USA
Christian Prehofer, Ludwig-Maximilians-Universität München, Germany
Claudia Raibulet, Università degli Studi di Milano-Bicocca, Italy
Outi Räihä, Tampere University of Technology, Finland
Muthu Ramachandran, Leeds Metropolitan University, UK
Hassan Reza, University of North Dakota - School of Aerospace, USA
Samir Ribic, University of Sarajevo, Bosnia and Herzegovina
Elvinia Maria Riccobene, University of Milan - Crema, Italy
Daniel Riesco, Universidad Nacional de San Luis, Argentina
María Luisa Rodríguez Almendros, Universidad de Granada, Spain
Antoine Rollet, University of Bordeaux, France
Siegfried Rovrais, TELECOM Bretagne, France
Patrizia Scandurra, University of Bergamo - Dalmine, Italy
Giuseppe Scanniello, Università degli Studi della Basilicata - Potenza, Italy
Christelle Scharff, Pace University, USA
Rainer Schmidt, HTW-Aalen, Germany
István Siket, University of Szeged, Hungary
Bernd Steinbach, Freiberg University of Mining and Technology, Germany
Thomas Stocker, University of Freiburg, Germany
Dinesh Subhraveti, IBM Almaden Research Center - San Jose, USA
Daniel Sundmark, Mälardalen University, Sweden

Osamu Takaki, Japan Advanced Institute of Science and Technology (JAIST) - Ishikawa, Japan
Hadaytullah, Tampere University of Technology, Finland
Wasif Tanveer, University of Engineering & Technology - Lahore, Pakistan
Pierre Tiako, Langston University - Oklahoma, USA
Durga Toshniwal, Indian Institute of Technology Roorkee - Uttaranchal, India
Davide Tosi, University of Insubria - Como, Italy
Peter Trapp, Ingolstadt, Germany
Elena Troubitsyna, Åbo Akademi University, Finland
Simon Tsang, Telcordia - Piscataway, USA
Javier Tuya, Universidad de Oviedo - Gijón, Spain
Roland Ukor, FirstLinq Limited, UK
Sergiy Vilkomir, East Carolina University - Greenville, USA
Rainer Weinreich, Johannes Kepler University Linz, Austria
Martin Wojtczyk, Technische Universität München, Germany & Bayer HealthCare, USA
Maribel Yasmina Santos, University of Minho, Portugal
Michal Žemlicka, Charles University, Czech Republic
Qiang Zhu, The University of Michigan - Dearborn, USA

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission to reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

Software Product Line Agility <i>Ahmed Abouzekry and Riham Hassan</i>	1
An Agile Model-Driven Development Approach - A case study in a finance organization <i>Mina Bostrom Nakicenovic</i>	8
A Planning Poker Tool for Supporting Collaborative Estimation in Distributed Agile Development <i>Fabio Calefato and Filippo Lanubile</i>	14
Scrum Maturity Model: Validation for IT organizations' roadmap to develop software centered on the client role <i>Alexandre Yin, Soraia Figueiredo, and Miguel Mira da Silva</i>	20
Usage of Robot Framework in Automation of Functional Test Regression <i>Stanislav Stresnjak and Zeljko Hocenski</i>	30
A Test Purpose and Test Case Generation Approach for SOAP Web <i>Sebastien Salva and Issam Rabhi</i>	35
Ev-ADA: A Simulation-driven Evaluation Architecture for Advanced Driving-Assistance Systems <i>Assia Belbachir, Jean-Christophe Smal, Jean-Marc Blosseville, and Sebastien Glaser</i>	43
On the Preliminary Adaptive Random Testing of Aspect-Oriented Programs <i>Reza Meimandi Parizi and Abdul Azim Abdul Ghani</i>	49
Devising Mutant Operators for Dynamic Systems Models by Applying the HAZOP Study <i>Rodrigo Fraxino Araujo, Auri Marcelo Rizzo Vincenzi, Francois Delebecque, Jose Carlos Maldonado, and Marcio Eduardo Delamaro</i>	58
A Static Robustness Grid Using MISRA C2 Language Rules <i>Mohammad Abdallah, Malcolm Munro, and Keith Gallagher</i>	65
A Specifications-Based Mutation Engine for Testing Programs in C# <i>Andreas S. Andreou and Pantelis M. Yiasemis</i>	70
Component-based Software System Dependency Metrics based on Component Information Flow Measurements <i>Majdi Abdellatif, Abu Bakar Md Sultan, Abdul Azim Abd Ghani, and Marzanah A.Jabara</i>	76
Module Interactions for Model-Driven Engineering of Complex Behaviour of Autonomous Robots <i>Vladimir Estivill-Castro and Rene Hexel</i>	84

Case Study for a Quality-Oriented Service Design Process <i>Michael Gebhart, Suad Sejdovic, and Sebastian Abeck</i>	92
Meta-Model for Global Software Development to Support Portability and Interoperability in Global Software Development <i>Bugra Mehmet Yildiz and Bedir Tekinerdogan</i>	98
A New Approach to Software Development Process with Formal Modeling of Behavior Based on Visualization <i>Abbas Rasoolzadegan and Ahmad Abdollahzadeh Barfouroush</i>	104
Non-Functional Requirements for Business Processes in the Context of Service-Oriented Architectures <i>Oliver Charles and Bernhard Hollunder</i>	112
A framework for adapting service-oriented applications based on functional/extra-functional requirements tradeoffs <i>Raffaella Mirandola, Pasqualina Potena, Elvinia Riccobene, and Patrizia Scandurra</i>	118
PSW: A Framework-based Tool Integration Solution for Global Collaborative Software Development <i>Juho Eskeli, Jon Maurologoitia, and Carmen Polcaro</i>	124
Feature-Oriented Programming and Context-Oriented Programming: Comparing Paradigm Characteristics by Example Implementations <i>Nicolas Cardozo, Sebastian Gunther, Theo D'Hondt, and Kim Mens</i>	130
Soft Constraints in Feature Models <i>Jorge Barreiros and Ana Moreira</i>	136
Feature Modeling of Software as a Service Domain to Support Application Architecture Design <i>Karahan Ozturk and Bedir Tekinerdogan</i>	142
Adding Support for Hardware Devices to Component Models for Embedded Systems <i>Luka Lednicki, Juraj Feljan, Jan Carlson, and Mario Zagar</i>	149
A Service Component Framework for Multi-User Scenario Management in Ubiquitous Environments <i>Matthieu Faure, Luc Fabresse, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier</i>	155
A Graph-Based Requirement Traceability Maintenance Model <i>Vikas Shukla, Guillaume Auriol, and Claude Baron</i>	161
A Systematic Mapping Study on Patient Data Privacy and Security for Software System Development <i>Isma Masood and Saad Zafar</i>	166
Impact on the inclusion of security in the UPnP protocol within the Smart Home	171

Alberto Alonso Fernandez, Alejandro Alvarez Vazquez, Maria del Pilar Almudena Garcia Fuente, and Ignacio Gonzalez Alonso

OntoLog: Using Web Semantic and Ontology for Security Log Analysis 177
Clovis Nascimento, Felipe Ferraz, Rodrigo Assad, Danilo Leite, and Victor Hazin

Intrusion Detection with Symbolic Model Verifier 183
Ines Ben Tekaya, Mohamed Graiet, and Bechir Ayeb

Security Quality Assurance on Web Applications 190
Rodrigo Assad, Felipe Ferraz, Henrique Arcoverde, and Silvio Meira

On Generating Security Implementations from Models of Embedded Systems 198
Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjodin

Proposal for Ground Shipping High Volume of Data Parameter in Supersampling Unmanned Aircraft Through Radio Modem 202
Manuel Sanchez Rubio, Vicente Millet Coll, Neves Seoane Vieira, Luis De Marcos Ortega, and Jose Javier Martinez Herraiz

The Smart Persistence Layer 206
Mariusz Trzaska

UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems 213
Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjodin

A maintenance Approach of a BJI Index Configuration 221
Said Taktak and Jamel Feki

Software Cache Eviction Policy based on Stochastic Approach 227
Stoyan Garbatov and Joao Cachopo

Performance Simulation of a System's Parallelization 233
Markus Meyer, Helge Janicke, Peter Trapp, Christian Facchi, and Marcel Busch

Towards Executable Business Processes with the Problem Oriented Engineering Process Algebra 239
Dariusz W. Kaminski, Jon G. Hall, and Lucia Rapanotti

Optimal Functionality and Domain Data Clustering based on Latent Dirichlet Allocation 245
Stoyan Garbatov and Joao Cachopo

Formal Parsing Analysis of Context-Free Grammar using Left Most Derivations 251
Khalid A. Buragga and Nazir Ahmad Zafar

Functional Complexity Measurement: Proposals and Evaluations <i>Luigi Lavazza and Gabriela Robiolo</i>	257
Design Patterns for Model Transformations <i>Kevin Lano and Shekoufeh Kolahdouz-Rahimi</i>	263
Component-oriented Software Development with UML <i>Nara Sueina Teixeira and Ricardo Pereira e Silva</i>	269
Metrics in Distributed Product Development <i>Maarit Tihinen, Paivi Parviainen, Rob Kommeren, and Jim Rotherham</i>	275
Edola: A Domain Modeling and Verification Language for PLC Systems <i>Hehua Zhang, Ming Gu, and Xiaoyu Song</i>	281
A Practical Method for the Reachability Analysis of Real-Time Systems Modelled as Timed Automata <i>Abdeslam En-Nouaary and Rachida Dssouli</i>	287
Reverse Engineering of Graphical User Interfaces <i>Ines Coimbra Morgado, Ana Paiva, and Joao Pascoal Faria</i>	293
Towards Design Method Based on Formalisms of Petri Nets, DEVS, and UML <i>Radek Koci and Vladimir Janousek</i>	299
Invariant Preservation by Component Composition Using Semantical Interface Automata <i>Sebti Mouelhi, Samir Chouali, and Hassan Mountassir</i>	305
Method for CMMI-DEV Implementation in Distributed Teams <i>Tiago da Cunha Oliveira and Miguel Mira da Silva</i>	312
Advanced Object Oriented Metrics for Process Measurement <i>Shreya Gupta and Ratna Sanyal</i>	318
Quality Issues in Global Software Development <i>Sanjay Misra and Luis Fernandez-Sanz</i>	325
A Systematic Review of Self-adaptation in Service-oriented Architectures <i>Maria del Pilar Romay, Luis Fernandez-Sanz, and Daniel Rodriguez</i>	331
A Formal Specification of G-DTD: A Conceptual Model to Describe XML Documents <i>Zurinahni Zainol and Bing Wang</i>	338

Formal Specification of Software Design Metrics <i>Meryem Lamrani, Younes El Amrani, and Abdelaziz Ettouhami</i>	348
E-FOTO: Development of an Open-Source Educational Digital Photogrammetric Workstation <i>Jorge Luis N. S. Brito, Rafael A. Aguiar, Marcelo T. Silveira, Luiz Carlos T. C. Filho, Irving S. Badolato, Paulo Andre B. Pupim, Patricia F. Reolon, Joao A. Ribeiro, Jonas R. Silva, Orlando B. Filho, and Guilherme L. A. Mota</i>	356
Vitalizing Local ICT-industry by Acceleration of FLOSS-based Software Product Development: A Case Study of the ICT-industry in Okinawa <i>Jun Iio, Yasuyuki Minei, Masato Kubota, and Kazuhiro Ooki</i>	362
Empirical Case Study of Measuring Productivity of Programming Language Ruby and Ruby on Rails <i>Tetsuo Ndoa and Chi Jia</i>	367
Querying Source Code Using a Controlled Natural Language <i>Oleksandr Panchenko, Stephan Muller, Hasso Plattner, and Alexander Zeier</i>	369
Towards Complementing User Stories <i>Christian Kop</i>	374
Performance Evaluation of a Generic Deployment Infrastructure for Component- based S/W Engineering <i>Abdelkrim Benamar and Nouredine Belkhatir</i>	380
A Proof-based Approach for Verifying Composite Service Transactional Behavior <i>Lazhar Hamel, Mohamed Graiet, Mourad Kmimech, Mohamed Tahar Bhiri, and Walid Gaaloul</i>	386
Certification of MDA Tools: Vision and Application <i>Oksana Nikiforova, Natalja Pavlova, Antons Cernickins, and Tatjana Jakona</i>	393
Automatic Generation of Graphical User Interfaces From VDM++ Specifications <i>Carlos Nunes and Ana Paiva</i>	399
An Approach to Model, Configure and Apply QoS Attributes to Web Services <i>Ahmed Al-Moayed and Bernhard Hollunder</i>	405
Transformation of Composite Web Service for QoS Extension into ACMEArmani <i>Amel Mhamdi, Raoudha Maraoui, Mohamed Graiet, Mourad Kmimech, Mohamed Tahar Bhiri, and Eric Cariou</i>	411
ATL Transformation of UML 2.0 for the Generation of SCA Model <i>Soumaya Louhichi, Mohamed Graiet, Mourad Kmimech, Mohamed Tahar Bhiri, Walid Gaaloul, and Eric Cariou</i>	418
Towards the Development of Integrated Reuse Environments for UML Artifacts <i>Moataz Ahmed</i>	426

An Automated Translation of UML Class Diagrams into a Formal Specification to Detect Inconsistencies <i>Khadija El Miloudi, Younes El Amrani, and Abdelaziz Ettouhami</i>	432
UML 2.0 Profile for Structural and Behavioral Specification of SCA Architectures <i>Wided Ben Abid, Mohamed Graiet, Mourad Kmimech, Mohamed Tahar Bhiri, Walid Gaaloul, and Eric Cariou</i>	439
Process Improvement and Knowledge Sharing in Small Software Companies: A Case Study <i>Minna Kivihalme, Anne Valsta, and Raine Kauppinen</i>	447
Choosing a Business Software Systems Development and Enhancement Project Variant on the basis of Benchmarking Data – Case Study <i>Beata Czarnacka-Chrobot</i>	453
Towards Functional and Constructional Perspectives on Business Process Patterns <i>Peter De Bruyn, Dieter Van Nuffel, Philip Huysmans, and Herwig Mannaert</i>	459
Practical Experiences with Software Factory Approaches in Enterprise Software Delivery <i>Alan Brown, Ana Lopez, and Luis Reyes</i>	465
A "Future-Proof" Postgraduate Software Engineering Programme: Maintainability Issues <i>J Paul Gibson and Jean-Luc Raffy</i>	471
Using Software Engineering Principles to Develop a Web-Based Application <i>Cynthia Lester</i>	477
How to Think about Customer Value in Requirements Engineering <i>Xinwei Zhang, Guillaume Auriol, Claude Baron, and Vikas Shukla</i>	483
Migrating Functional Requirements in SSUCD Use Cases to a More Formal Representation <i>Mohamed El-Attar and James Miller</i>	487
KM-SORE: Knowledge Management for Service Oriented Requirements Engineering <i>Muneera Bano and Naveed Ikram</i>	494
Brainstorming as a Route to Improving Software Processes <i>Celestina Bianco</i>	500
Web-Based Focus Groups for Requirements Elicitation <i>Carla Farinha and Miguel Mira da Silva</i>	504
Mapping Architectural Concepts to SysML Profile for Product Line Architecture Modeling <i>Shahliza Abd Halim, Mohd Zulkifli Mohd Zaki, Noraini Ibrahim, Dayang N. A. Jawawi, and Safaai Deris</i>	510

Exploring Architecture Design Alternatives for Global Software Product Line Engineering <i>Bedir Tekinerdogan, Semih Cetin, and Ferhat Savci</i>	515
Towards CMMI-compliant MDD Software Processes <i>Alexandre Marcos Lins de Vasconcelos, Giovanni Giachetti, Beatriz Marin, and Oscar Pastor</i>	522
From Boolean Relations to Control Software <i>Federico Mari, Igor Melatti, Ivano Salvo, and Enrico Tronci</i>	528
Empirical Evidence in Software Architecture: A Systematic Literature Review Protocol <i>Nadia Qureshi, Naveed Ikram, Muneera Bano, and Muhammad Usman</i>	534
Agile Development of Interactive Software by means of User Objectives <i>Begona Losada, Maite Urretavizcaya, and Isabel Fernandez de Castro</i>	539
REFIS: A Stage-based Methodology for Eliciting Requirements <i>Felipe Ferraz, Leopoldo Ferreira, Rodrigo Assad, Renato Ferreira, and Silvio Meira</i>	546
A Metamodel for Representing Safety LifeCycle Development Process <i>Yulin Zhang, Brahim Hamid, and Damien Gouteux</i>	550
On the Extensibility of Plug-ins <i>Vanea Chiprianov, Yvon Kermarrec, and Siegfried Rouvrais</i>	557
Effective Task Allocation in Distributed Environments: A Traceability Perspective <i>Salma Imtiaz and Naveed Ikram</i>	563
An Agile Method for Model-Driven Requirements Engineering <i>Grzegorz Loniewski, Ausias Armesto, and Emilio Insfran</i>	570
Evidence in Requirements Engineering: A Systematic Literature Review Protocol <i>Talat Ambreen, Muhammad Usman, Naveed Ikram, and Muneera Bano</i>	576
Success Factors Leading to the Sustainability of Software Process Improvement Efforts <i>Natalja Nikitina and Mira Kajko-Mattsson</i>	581
Software Quality Assessment and Error/Defect Identification in the Italian Industry Preliminary Results from a State of the Practice Survey <i>Fausto Fasano, Giuseppe Scanniello, Andrea De Lucia, and Genoveffa Tortora</i>	589
Revisiting the Requirements Communication Problem from a Knowledge Management Perspective <i>Hermann Kaindl and Lukas Pilat</i>	595

Software Product Line Agility

Ahmed Abouzekry
 Computer Science Department
 Arab Academy for Science and Technology
 Cairo, Egypt
abouzekry@yahoo.com

Riham Hassan
 Computer Science Department
 Arab Academy for Science and Technology
 Cairo, Egypt
riham@cairo.aast.edu

Abstract— Software reuse constitutes a significant challenge for different development communities, while systematic reuse is a difficult target to achieve. Software Product Line (SPL) has been nominated as one of the effective approaches promoting software reuse. In this paper, we propose the Enterprise Product Line Software Process (EPLSP) that integrates practices of both the Enterprise Unified Process (EUP) and the Agile Unified Process (AUP). This integration benefits the engineering process with both reusable components architecture and fast time to market final products. EPLSP strategy focuses on the two major aspects of SPL namely the Core Assets (CA) and the Product Development (PD). CAs are those reusable artifacts and resources that form the basis for the SPL. PD involves building, acquisition, purchasing, retrofitting earlier work of software products, or any combination of these options. EPLSP promotes a clear up-front architecture in the CA while employing agility for PD. Constructing an up-front architecture for CA is effective in enhancing reusability and increasing productivity. Using agility in PD is meant to improve the time to market variable. We demonstrate the EPLSP approach with an SME case study on a Retail Management System (RMS) named FOCUS. Further, we leverage an evaluation framework to assess the effectiveness of EPLSP when applied to FOCUS. This case should define clearly the preferred areas of agility interference in the SPL, and where we need architecture to provide a sustainable production.

Keywords- Enterprise Unified Process; Agile Unified Process; Software Product line.

I. INTRODUCTION

Modules, objects, components and services are all different patterns of the reusability practice. Software Product Line (SPL) is recognized as an approach for systematic reuse [1]. SPL matches software with different industries representing it as a manufactured tangible product. Further, it is one of the most important practices in sustainable organizations for the ultimate cost and time reduction [1].

SPL as an effective reuse approach is highly recognized in software enterprises. Small and Medium Enterprises (SMEs) do not firmly apply principles, but one can still recognize a chaotic version of such principles over their determined or formal processes.

SPL consists of three main activities namely Core Asset (CA) Development, Product Development (PD) and Management. CAs represents the basic reusable components in the SPL. CAs could be a class, a blueprint, a series of programming code or even a document, while the PD provides the means of final customer usable product. SPL management activity plays critical role in coordinating, supervising, planning and other administration practices needed across the production activities.

Agile methods promote productivity and values of iterative development over heavy-weight methodologies through number of practices that enable cost effective change [2]. Agile and SPL merge of practices covers the increasing need for shorter time to market and higher product quality [7]. On the other hand, the more the SPL becomes agile, it loses some of its essential properties, as strategic, planned reuse which yields to predictable results. The SPL reuse practice requires precise support in different areas like organizational capabilities, management and technical roles, architecture optimization...etc seeking a systematic approach for reusability. Incorporating agile practices in developing SPL raises some questions like what is the extent of interfering between the agile and SPL? And could agile fit in both CAs and PD?

SPL complexity promotes the need for an up-front design and heavy architecture [8]. CA development should conform to some standards and include detailed description and using instructions even if this CA is a Commercial Off-The-Shelf (COTS) component.

In this paper, we propose the Enterprise Product Line Software Process EPLSP as a roadmap for the implementation of the SPL with integration of agile practices. EPLSP covers the essential architectural practices in CA building, to solve the asset management pitfalls, and the use of agile practices in the PD to enhance the time to market variables.

EPLSP integrates the Enterprise Unified Process (EUP) [9] with the Agile Unified Process (AUP) [10]. EUP is an extension of the IBM Rational Unified Process (RUP) [11]. AUP is a simplified version of the IBM RUP that applies agile techniques in modeling, development and management [10]. Using the EUP overcomes the problems of managing such a family of products; like change management, strategic reuse...etc. EUP enables the enterprise to apply the

governance practices and disciplines (project management, retirement management...etc.) within the process. AUP allows for exploiting the agile essence to lighten the response to market requirements needed to enhance productivity. Further, AUP enables the customization of the development process to multiple agile processes or some of their combinations like SCRUM and XP. EPLSP focuses on the extent of agility needed in the SPL practice and where agility best fits in the SPL development life cycle. Further, EPLSP depicts where SPL could most benefit from its goals in the production level.

The rest of this paper is structured as follows: Section 2 surveys the state of the art in integrating agile practices into SPL. Section 3 depicts the EPLSP process and the artifacts produced in each step. Section 4 demonstrates EPLSP on the Retail Management System (RMS) FOCUS. Finally, Section 5 concludes the paper with remarks for future work.

II. RELATED WORK

Investigating whether Agile and SPL could integrate to complement each other; there stills a debate among the research community about its extent and feasibility.

Tian and Cooper [2] argue that the combination of Agile and SPL forming the Agile Software Product Line Methodology (ASPLM) could shorten time to market maintaining the quality, in which the ASPLM leaves room for futher development work to meet customer's changing requirements, rather than pure customization of CA. They showed that CA, PD and SPL Management activities need to be investigated for possible agility.

Carbon et al. [3] had conducted a class-room experiment following the motivation to present preliminary results showing the successful merge between Agile and SPL. They concluded to a result that agile in SPL reduces time spent on design (Increases the speed), while SPL keeps changes to minimum (Increases quality).

On his research, Geir K. Hanssen [4] stated an answer for how to combine Agile and SPL. In a successful marriage, he stated that this combination leads to; risk reduction, organizational development, reduced maintainability, community building, openness and visibility and company culture improvement, contributing to the emergence of a software ecosystem, which refers to how organizations should exist together as an ecosystem.

One of the popular case studies conducted by the Software Engineering Institute in Carnegie Mellon University is Salion [5]. Salion is an SME with no experience in its application area. It pursued a reactive approach to its Agile SPL achieving a phenomenal reuse level of 97% with its 21 employees counting seven developers only.

Despite the success of the previous cases, they did not take in consideration the difference in nature between the CA and the PD. As any other production the sustainability

of the production depends on the systematic the whole process, which should be only achieved by architecture

III. ENTERPRISE PRODUCT LINE SOFTWARE PROCESS (EPLSP)

We propose EPLSP as a software process with the goal of effective production of SPL that better meets its market requirements. EPSLP integrates agile and SPL practices from the two extensions of IBM RUP namely EUP and AUP. EPLSP covers the Enterprise disciplines needed in the SPL to improve the change management and architectural variability in the CA phase. These parameters are improved while taking into account the increasing demand on lower time to market and quality software production through employing agile practices.

A. EUP and AUP

EUP is an information technology lifecycle that encompasses the activities of an IT department. Further, EUP adds the enterprise disciplines required to effectively manage organizations' portfolio of systems as described in Figure 1.

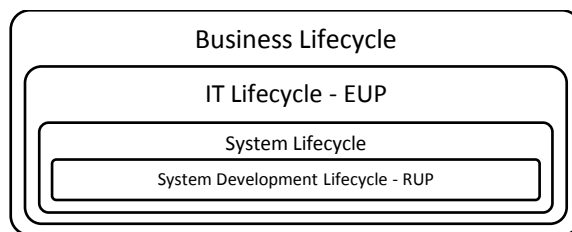


Figure 1. The Scope of different process lifecycles.

EUP extends RUP to include the operation and support of a system after being in production along with its eventual retirement, where the two new phases benefits the concept of strategic reuse promoted by the SPL. Further, EUP enhances the overall process with the separation of the disciplines into; development, support and enterprise as illustrated in Figure 2.

	Inception	Elaboration	Construction	Transition	Production	Retirement
Development Disciplines						
Business Modeling	Very High	High	Medium	Low	Very Low	N/A
Requirements	Very High	High	Medium	Low	Very Low	N/A
Analysis and Design	Very High	High	Medium	Low	Very Low	N/A
Implementation	Very High	High	Medium	Low	Very Low	N/A
Test	Very High	High	Medium	Low	Very Low	N/A
Deployment	Very High	High	Medium	Low	Very Low	N/A
Support Disciplines						
Configuration Management	Very High	High	Medium	Low	Very Low	N/A
Project Management	Very High	High	Medium	Low	Very Low	N/A
Environment	Very High	High	Medium	Low	Very Low	N/A
Operations and Support	Very High	High	Medium	Low	Very Low	N/A
Enterprise Disciplines						
Enterprise Business Modeling	Very High	High	Medium	Low	Very Low	N/A
Portfolio Management	Very High	High	Medium	Low	Very Low	N/A
Enterprise Architecture	Very High	High	Medium	Low	Very Low	N/A
Strategic Reuse	Very High	High	Medium	Low	Very Low	N/A
People Management	Very High	High	Medium	Low	Very Low	N/A
Enterprise Administration	Very High	High	Medium	Low	Very Low	N/A
Software Process Improvement	Very High	High	Medium	Low	Very Low	N/A
Levels of interference	Very High	High	Medium	Low	Very Low	N/A

Figure 2: Enterprise Unified Process [9]

AUP is an Ultra-lightweight variant of RUP, with the work disciplines and products simplified and reduced as shown in Figure 3.

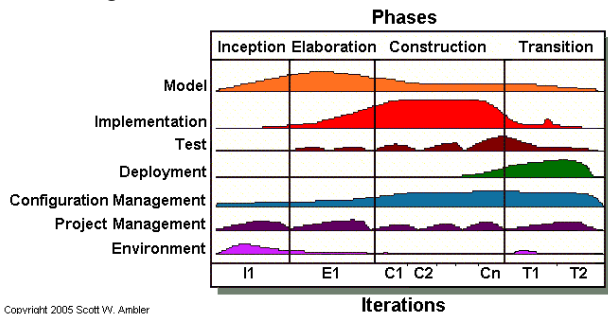


Figure 3: The Agile Unified Process.

We employ the practices of EUP and AUP that facilitate different management levels and all involved parties in the production activities to highly control tasks associated to their roles. Those practices complement the EPLSP and close the IT department circle within a tightly managed manner with the following recommendations;

- Documenting architecture using Unified Modeling Language (UML)
- Applying SCRUM as an Agile project management practice
- COTS could be used across the product line
- Configuration Management software is essential to manage releases
- Specific software to manage commonality and variability to enhance the strategic reuse option.

The different nature of the CA and the products is one of the major challenges facing the application of EUP and AUP to SPL. This marriage between EUP and AUP is intended to facilitate the application of both processes to SPL. CA needs the architecture provided by the EUP and the extension of the production and retirement phases. The need for fast response to market for the products could be achieved with agility. AUP has the same phases as EUP but simplified, so there is no need to rework the architecture of the artifacts to fit in the other SPL production activities.

B. EPLSP Process

EPLSP provides means to integrate agile practices into the SPL development life cycle. Figure 4 depicts the overall process structure in EPLSP. The initial phase on the bottom of the process consists of the domain engineering, in which it represents the knowledge needed to build the reusable artifacts like; scoping, requirement engineering, design, testing, and the realizing of the commonality and variability of the product line practice with the CA development activities. In the middle there exists the CA base which contains the reusable artifacts. The right downward arrow represents the reactive approach in which the start point is the PD.

The PD activity is split into two tasks, development task and release task for two reasons, the separation between the deployment and the production which differs in the application of disciplines, and to maintain a direct agile incremental iterative practice.

The management tent could be seen as the containing rounded box, providing SPL process with the needed management disciplines solely.

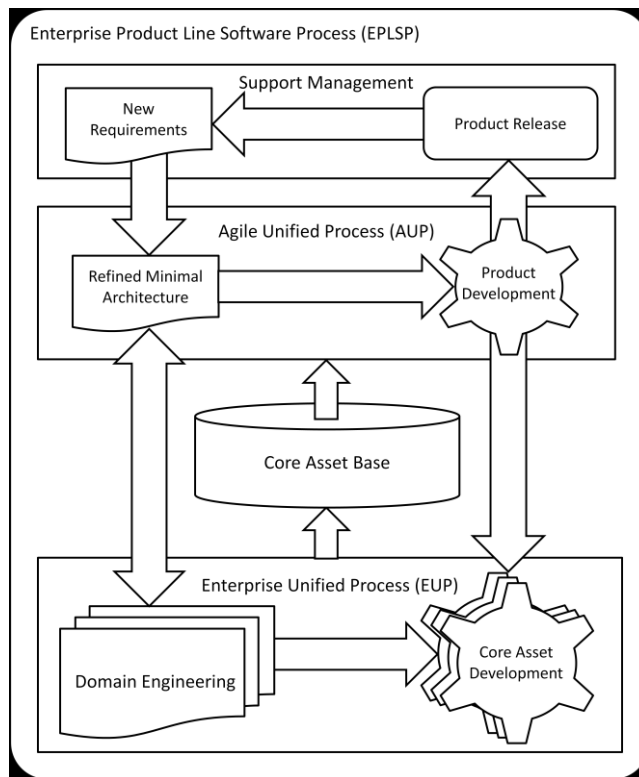


Figure 4. EPLSP Conceptual Model

CA development is the activity intended to build the reusable components of the SPL. CA development requires prior domain expertise, heavy architecture and management capabilities. This could be achieved only by a well defined engineering architectural centric process to ease the reusability of this asset. EPLSP proposes the application of the EUP as a basic process for the domain engineering and CA instantiation as shown in Figure 5.

		Project Management				Operations and Support Management	
		Configuration and Change Management					
		Inception	Elaboration	Construction	Transition	Production	Retirement
Enterprise Management	Domain Engineering	Domain Scoping and Planning	Requirement Specification	Model	Unit Testing	Quality Assurance	System Rework
		Estimates and Schedules	Architecture Definition	Building		Manage Change	
		Develop Business Cases	Build Test Cases	Develop Support Documentation	Develop User Documentation	System Support	Data Migration
	Risk and Quality Assessments	Staffing and Task allocation	Integration Testing	User Testing			
	Application Engineering	Product Development	Requirement Definition	Requirement Specification	Building		
Product Release	Training Plan			Integration Testing	User Testing		
						System Deployment	System Removal

Figure 5. EPLSP Milestones

PD activity is usually in need of the fast response to customer requirements, and early delivery of quality products. These goals could be achieved by the agile methodologies, for this reason EPLSP preferably uses AUP as a simplified version from the unified process to eliminate unneeded heavy architecture. Figure 5 determines milestones in every phase of the EPLSP.

IV. FOCUS[®] RMS

This section describes an RMS named FOCUS to demonstrate the feasibility of the EPLSP process. Further, we discuss FOCUS commonalities and the challenges we faced during and after the development process.

A. FOCUS[®] subsystems:

FOCUS[®] is a mini ERP specially developed for small and medium retail outlets. This system could work as one unit, integrated and linked over one database or every subsystem separated as a single unit as depicted in Figure 6.

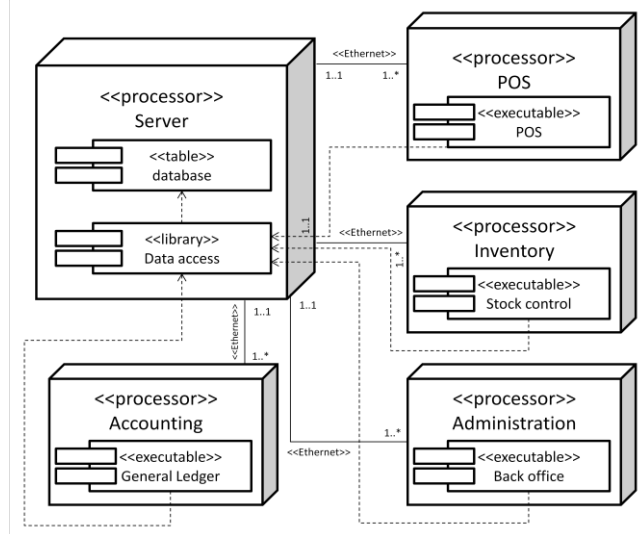


Figure 6. FOCUS[®] RMS Deployment Diagram.

FOCUS is composed of the following subsystems:

- FOCUS[®] stock control, which holds the essential stock transactions; basic entries, receiving, item cards... etc.
- FOCUS[®] Point of Sale (POS): is where daily sales transactions managed by salesperson in the checkout area of an outlet or a shop.
- FOCUS[®] General Ledger (GL): reflects automatically the daily selling, receiving and monetary transactions to journal entries and accounts, and reports financial statements.
- FOCUS[®] back office is the administrative tool, which facilitates higher management to monitor transactions, authorize permissions, link subsystems and modify system settings.

The system was primarily developed to target large sector of retail outlets with the following features; installed, not customizable, self setup with a simple instructions guide and easy to understand and apply. Since these requirements could rarely be found in SME's business software, it was planned to produce enhanced version yearly with new features; based on wide survey for user requirements.

Figure 7 depicts the system requirements and demonstrate the similarities as classes, layers and complete sub modules; like the security module, transaction file and product catalogue.

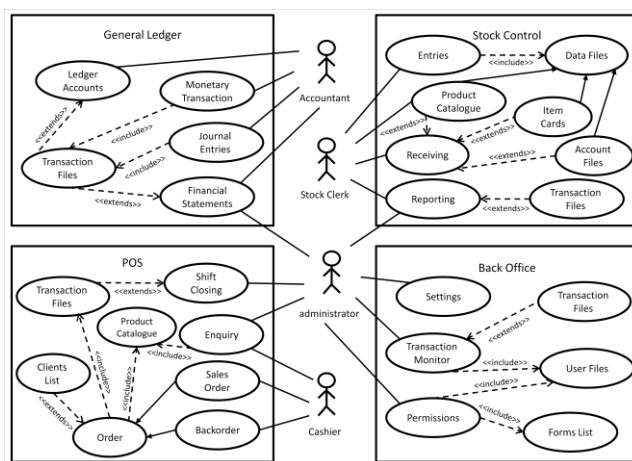


Figure 7. FOCUS[®] RMS System Requirements.

1) Company

The software was built in a small enterprise named SCOPE Communications, in which it employs 13 people; 6 only is counted as developers, and it took 18 months to release the basic version of the full system.

This basic version of the system contains 135 KLOC in total, with 160 database tables, 1100 stored procedures, 450 forms and 320 reports covering the four modules.

The core process was a simple version of the incremental, iterative process; it was described and documented using the UML. The system was built using a similar proactive approach to the SPL's, with no use of any Configuration

Management software. Test cases are prepared with two concerns; business cases depend on customer stories and technical cases over the functions, for data integrity.

2) *FOCUS Production Challenges*

From our study of the former system we have observed some challenges resulting from the application of the previous process, like;

- Recurrent costs associated with the reuse of non-architectural artifacts
- Higher risk resulted from unplanned resource allocation and estimation
- Complexity of managing the commonality and variability of artifacts
- Wasted time resulted from the duplication of code and documentation
- Corrective bug fixing rather than preventive associated with the unplanned test cases
- Customers frequent complaint from support

3) *EPLSP and FOCUS*

Applying EPLSP to FOCUS RMS will help the company well manage the SPL process, with the allocation of the architectural centric activities in the needed areas only; which is intended to well manage changes across the process, and the use of agile practices in the PD activity to improve the market response.

4) *Refactoring FOCUS®*

As a retail management system the product catalogue regarded as the main component in the solution, therefore; the selected artifact to be redesigned using the EPLSP is the product catalogue, which contains the building features of any product like name, description, type, category, price, etc.

The product catalogue is considered a sub module, and is completely used in one of the main modules, and partially used in the three other modules.

5) *Applying EPLSP to FOCUS*

The product catalogue features totally differs as the type of products or services provided by the outlet itself, however there are some common requirements in this sub module.

The architecture definition in the EPLSP elaboration phase defines a practice to manage the commonalities and variability of the product catalogue. This covers the change management problem and reduces the recurrent costs resulting from unplanned reusability.

The main goal of the EUP unique production phase is to keep systems useful and productive after deployment, in which it encompasses the operation and support of the system. Also, this phase provide some means of quality assurance by monitoring the operation of the system when working and recovering any problem. These practices help

the company manage the post deployment stage professionally, which develops customer loyalty.

We are redeveloping the product catalogue as a sub module with EPLSP maintaining the same functionality of the catalogue. We compare the development experience using EPLSP with its counterpart using the older version of the system developed with an iterative simple RUP. The metrics used for our comparison are depicted below in subsection 6.

The product catalogue itself consists of two parts. One part is recognized as a core asset, which includes the search base and the basic entry forms like category, product, limits...etc. The second part is realized as a product which includes product labeling, reports...etc.

We develop the product catalogue core asset using EUP as the part of EPLSP that incorporates a complete architecture, while developing the product part using SCRUM. In both parts we use an incremental iterative process.

In the older version of the FOCUS system, we employed a simple iterative and incremental undefined process to develop the whole SPL. The sequence of the process steps mostly relied on the task, the feature or even on the developer. The older process employed code comments and traditional UML diagrams for documentation.

Using EPLSP, we define 5 essential practices. We use a tailored version of SCRUM at the product part of the catalogue and a set of architectural templates and plans in the CA part. Further, we utilize configuration management software and a set of chosen UML diagrams for core assets and the products. We define the development incremental steps as shown in Figure 8.

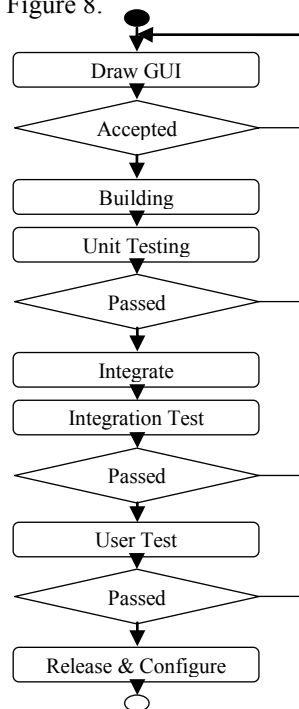


Figure 8. Development Increment.

For the product part of the catalogue, we define a set of SCRUM roles namely the project manager, the product owner and the developers. The project manager acts as the SCRUM master, while the marketing team acts as product owners. Further, we have a team of a senior developer and two junior developers.

A daily meeting is held with the team to discuss the progress and the problems. Further, a weekly meeting is held with the presence of the product owner to present the features achieved thus far. The weekly meeting aims also at collecting feedback from the product owner while developing new ideas and requirements. Finally, a monthly meeting is held to test and show the released version, which could be installed at the customer site for free. Such installation allows the support team to record comments within two or three days.

In the production phase, the product backlog is developed in cooperation between the SCRUM master and the product owners. The product backlog scenarios are prioritized while dependencies are identified. Further, the product backlog is revised and updated in every monthly meeting. The sprint backlog defines the current set of features in the construction phase, its tasks and associations to team members. These backlogs contain:

- Use cases, Class and Activity diagram.
- Test cases.
- Schedules and job orders.

Finally, the released version of the product is configured and generated with a set of user instructions.

We produce the following set of architectural documents during the development of catalogue CA part. Such documents contain the complete domain architecture that depicts the infrastructure CAs. Infrastructure CAs include the CA part of the product catalogue along with other CAs :

- Detailed business case.
- Requirements and specifications plan.
- Test plan for the 3 testing levels, unit test, integration test and user test.
- Software development plan
- Iteration plan.
- Change and configuration plan.
- Deployment and support plan.

Unlike the product development, the configured version of the core asset is augmented with the developer's manual and deployment instructions.

6) Process Validation

We utilize a number of metrics to assess the effectiveness of EPLSP and compare it to the classical iterative or incremental development process. These metrics are defined to assess the effectiveness of the merge between SPL development and agile process and it was stated and used in Salion's Agile SPL [6] as follows:

- Reusability: Salion [6] defines the reusability of its system with a percentage level that is equal to common files used in

all members of the product family divided by the total number of files generated across the product line (Reusability level% = common files/total SPL files).

- Time to market: It was proposed in the same case [6] as the manpower used per month to produce the first customer's product (# of persons-month).
- Eliminating duplicates: We measure it by the percentage of eliminated duplicates using the classic Line of Code (LOC) metrics (Eliminated Duplicates% = # of duplicated LOC/total LOC).
- Productivity: This metric is measured using popular LOC and Use Case metrics as an extension of the Function Point metrics as a complex subject concerning a relation between different resources or artifacts, the use case metrics defines an early – prior development measure of software functionality rather than the function point, which could only be used after development.(Usecase/hour, LOC-person/month...etc)
- Cost reduction: Similar to of the productivity metrics, but it is preferred to be measured by the Use Case metrics. Also either LOC or Function Point could be used, but regarding the LOC it will be subjective due to the difference in number of produced lines from one person to another within the same class. And for the function point analysis it could be determined only after the development completion; instead of early determination of cost in the case of Use Case metric.(UseCase-person/day)
- Defect Removal Efficiency (DRE): Is one of the popular quality metrics which is intended to measure the discovered errors during development in relation to the total errors and defects found.
($DRE = E / (E + D)$ in which E is the number of errors and D is the number of defects).

V. CONCLUSION

This paper proposed EPLSP to address the possible integration between SPL and agile. Applying this process to FOCUS RMS addresses most of the challenges the company faced during the production of the software using the classical process. Further, the proposed EPLSP addresses the time to market challenge, which is one of the major SPL challenges. EPLSP addresses the challenges through leveraging agility in the suitable areas of integration of the EPLSP which helps the production quality software products.

Applying EPLSP to FOCUS RMS, our potential challenges include technical and social challenges. Technical challenges include training the development staff in the EPLSP development process and reworking the design. Our social challenges confine the commitment of the upper management to change and restructuring the organization so that the new process is accommodated.

REFERENCES

- [1] Linda Northrop. 2008. Software Product Lines Essentials. *Software Engineering Institute, Carnegie Mellon University*. http://www.sei.cmu.edu/productlines/frame_report. [accessed, April 2011]
- [2] Cunningham W, Manifesto for Agile Software Development. 2001. [cited 2008-09-30]; Available from: <http://www.agilemanifesto.org>. [accessed, March 2011]
- [3] Tian, K. and K. Cooper, Agile and Software Product Line Methods: Are They So Different?, in *1st International Workshop on Agile Product Line Engineering*. 2006.
- [4] Carbon, R., et al. Integrating Product Line Engineering and Agile Methods: *Flexible Design Up-front vs. Incremental Design*. in *Workshop on Agile Product Line Engineering*. 2006.
- [5] Hanssen, G.K. and T.E. Fægri, *Process Fusion - Agile Product Line Engineering: an Industrial Case Study*. *Journal of Systems and Software*, 2007, pp. 836-849.
- [6] Clements, P. and Northrop, L., *Salion, Inc.: A Software Product Line Case Study*, Software Engineering Institute (SEI) Technical Report CMU/SEI-2002-TR-038, Carnegie Mellon University, Pittsburgh, PA, November 2002.
- [7] Snorre Gylterud, Constructing a Silver Bullet? *Combining Software Product Line Engineering and Agile Software Development, A thematic literature review*, Norwegian University of science and technology, 2008.
- [8] J. Bosch, *Design and use of software architectures: adopting and evolving a product-line approach*. Addison-Wesley, Harlow, 2000.
- [9] S. W. Ambler, J. Nalbone, M. J. Vizdos, The Enterprise Unified Process, *Extending the Rational Unified Process*, Prentice Hall, 2005.
- [10] S. W. Ambler, The Agile Unified Process (AUP), *Ambyssoft, 2005*; www.ambyssoft.com/unifiedprocess/agileUP.html. [accessed, March 2011]
- [11] Philippe Kruchten, *The Rational Unified Process: An Introduction*, 2nd ed. Addison-Wesley, 2000.
- [12] Rubin, H. A. "Macro-Estimation of Software Development Parameters: The ESTIMACS System." *Proc. SOFTFAIR: A Conference on Software Development Tools, Techniques, and Alternatives*. New York: IEEE, July 1983, pp. 109-118.

An Agile Model-Driven Development Approach

A case study in a finance organization

Mina Boström Nakićenović
 SunGard Front Arena
 Stockholm, Sweden
 email: mina.bostrom@sungard.com

Abstract—In the Sungard Front Arena, current software portfolio a business functionality called Market Server Capability (MSC) is embedded and duplicated in many components. By the application of Agile and Lean principles on model-driven development, we will get an Agile approach for constructing the architecture of a new MSC definition which will eliminate the duplication and inconsistency, while still maintaining a short implementation phase. The resulting architecture has a single modeling level, with merged PIM and PSMs. The model is designed by reverse engineering of the legacy code in a Test Driven Development fashion.

Keywords—agile; lean; MDD; TDD; finance

I. INTRODUCTION

SunGard is a large, world-wide financial services software company. The company provides software and processing solutions for financial services. It serves more than 25000 customers in more than 70 countries. SunGard Financial Systems provides mission-critical software and IT services to institutions in virtually every segment of the financial services industry. We offer solutions for banks, capital markets, corporations, trading, investment banking, etc. [1].

The Front Arena system includes functionality for order management and deal capture for instruments traded on electronic exchanges. Market access is based on a client/server architecture. The clients for market access include the Front Arena applications, while the market servers, called an Arena Market Servers (AMAS) provide services such as supplying market trading information, entering or deleting orders and reporting trades for a market.

Clients and AMAS components communicate using an internal financial message protocol for transaction handling, called Transaction Network Protocol (TNP) and built on top of TCP/IP. The TNP protocol uses its own messages, which contain TNP message records with fields [2]. Many of the TNP client components query the Market Server Capability (MSC), information about the trading functionality that one electronic exchange (market) offers. Client applications need such information in order to permit/disable the access to the different markets.

A. Problem description

When a new market (AMAS) is introduced, the information about functionality that the new market offers (which transaction i.e., TNP messages are supported) should be added to each client. MSCs describe market trading transactions (Orders, Deals, etc.), which command are supported for them (entering, modifying, etc.) and which attributes and fields could be accessed on the markets (Quantity, Broker, etc.). This information is presently hard-coded into each client application. New client application releases need to be done before the customers can start using the new AMAS. Depending on the current release plans of the client applications this can take a long time. Having to wait for the client application releases may delay the production start of the AMAS.

All components, which use the MSC functionality, must use the same MSC definition. Unfortunately the same MSCs are defined in several different files. Different components are developed in different programming languages so they do not share the same definition file. Because of historical reasons and the fact that some client components were developed within separate teams, even the components developed in the same programming language do not share the same definition file. Each client component has its own MSC definition file. There is a lot of the duplication of information in these files. Even worse they do not present exactly same data since the different clients work within different business domains, so their knowledge about the MSCs is on the different levels. Two main problems with this architecture are:

- Hard-coded MSC definition, requiring the recompilation of components when a new MSC is introduced
- Duplication of the MSC definition, introducing the risk for data inconsistency.

These problems will be resolved in the future by introducing a Dynamic Market Capabilities (DMC), a new functionality that will be used to retrieve the MSC definition dynamically, in run-time, instead of having them hard-coded. Unfortunately, it will take a long time, probably years, until the DMC solution will be completely implemented and in use (for all AMAS and all client components). Until then all components have to support the hard-coded fashion.

All new components, which will be developed during this time, have to support the hard-coded MSC way also. That is why there is a need to find an intermediate solution which will remove the duplication and which will be used under the transition phase. Since such an architecture will not be long lived company management put some time and resource constraints on the implementation. The question we address in this paper is how to create such intermediate solution, taking all conditions and constraints into account.

Introduction and problem description are presented in Section I. Section II explains, in more details, the architectures of both the present and the DMC solution as well as it introduces reasons for having an intermediate solution. In Section III, requirements and constraints are explained. The produced intermediate solution, an Agile MDD approach, is presented in Section IV. In Section V, the benefits are discussed of applying Agile and Lean principles on the MDD. Finally, Section VI presents our conclusion.

II. ARCHITECTURE OF THE MSC DEFINITION

A. The present architecture

The client components use the MSC definition from the different sources, developed in different programming languages (C++, C# and Java), where the majority of data is duplicated. The present architecture of the MSC definition is not centralized (no single definition of the model) and without control for the consistency. The lack of centralization enormously increases the risk for data inconsistency since the consistency depended on the accuracy of the developers who edits the MSC definition in a source code file. The development of the MSC definition is a continuous process, and new MSCs are defined each time when a new AMAS is developed (2-3 times per year) or when a new trading transaction is introduced (once per month). The current process flow is:

- A new AMAS is developed or a new transaction is introduced.
- A MSC is added to the MSC definition in each client component. The same information must be added to several different files.
- All client components should be recompiled in order to get the definition of the new MSC.

B. Dynamic Market Capabilities architecture

We have already done design plans for the new DMC architecture. In the DMC architecture each AMAS will be responsible to provide, to the client components, information about the MSC that the AMAS supports. The description of the MSC that the AMAS supports will be saved in one XML file. An example of an extract from a XML file, containing the MSC definition for the AMAS called OMX, is presented in the Figure 1. In this example, a MSC defines that the market OMX supports trading transaction order with the following commands:

enter, modify and delete, combined with the following fields: price and quantity.

```
<MarketCapability id="200" MarketServer="OMX">
  <Object name="Order">
    <Commands>
      <Command name="Enter"/>
      <Command name="Modify"/>
      <Command name="Delete"/>
    </Commands>
    <Fields>
      <Field name="Price"/>
      <Field name="Quantity"/>
    </Fields>
  </Object>
</MarketCapability>
```

Figure 1. Market Server Capabilities for market OMX

On the AMAS start up, AMAS reads the MSC definition from its XML file and sends them, in run time, to all client components which connect to the AMAS. In such way the client components do not have to be recompiled if something changes in the MSC definition. When a new AMAS is developed, a new XML file containing MSC definitions for the AMAS is created. On the AMAS start up, all client components connect to the AMAS and dynamically retrieve the MSC definition for that AMAS. So even in this case there will be no need for the recompilation of the client components.

C. Transition phase

The decision is that all AMAS components and all client components should be upgraded to the DMC architecture. But this transition is a complicated job. There are over 30 AMAS components and more than 5 client components that are using MSC functionality today. There is different prioritizing, from the management side, within the components' backlogs. We know, right now, that some of these components will be upgraded to the DMC in one or two years. This transition project is not marked as a critical since there is already a working architecture, although not the best one. As long as there is at least one component which has not been upgraded to the new DMC architecture, the hard-coded MSC solution must still be supported. The transition will occur gradually and the transition phase will probably take several years. Under the transition phase some new components are going to be developed; some new components are already under the development. To develop new client components according to the present architecture will introduce even more duplication. Therefore an intermediate architecture, which will eliminate the duplication, will be introduced. Such a solution should have a short implementation phase, since it must be ready before the new components are completely developed. The solution should be designed so that it eventually leads towards the new DMC architecture. It would be good if the new DMC architecture can benefit from it.

III. INTERMEDIATE SOLUTION

We work according Scrum in the company, trying to apply Lean and Agile software development

philosophy. One of the key principles of the Lean philosophy is to detect and eliminate wastes [3]. The intermediate solution should eliminate, from the present architecture, the three major points of waste.

- Duplication of the MSC information
- Amount of work done during the MSC definition updates
- Amount of time used for communication among groups, informing each other about the MSC definition changes

In order to eliminate the duplication of data we need a centralized MSC definition. In order to be able to provide support for the MSC definition in different programming languages we need to generate code in different programming languages, from the centralized MSC definition. We need a programming language independent architecture. First we considered a solution, where all client components would be refactored to reference the same central definition file, but this would require a lot of work. We did not want to refactor client's components too often, since some of them will be refactored soon regarding the DMC solution. That is why we believed that the Model-Driven Architecture (MDA) [4] approach can be the most suitable solution for the intermediate architecture. With the MDA approach we mean the general MDA concept: "A MDA defines an approach to modeling that separates the specification of system functionality from the implementation on a specific technology platform". The common denominator for all MDA approaches is that there is always a model (or models), as the central architectural input point, from which different artifacts are generated and developed. Transformations, mapping rules and code generators are called in common "MDA tools" [5].

The main idea is to have just one source, a union of all present MSC definition that is programming language independent. From such a source, which will be a central MSC definition registry, the present MSC definition source files are generated. All present MSC definition files have a similar structure. The main difference is the programming languages syntax. Because of that the code generation should not be too complicated. The way how the client components work will not be changed, the MSC definition will still be hard coded. Such a solution does not require the refactoring of the client components. But the way how the developers work will be improved. They will work just with the central MSC definition registry and add/edit the MSC definition only there. Then the MSC definition files, for each client component, will be automatically generated from the central registry. The client components will be automatically recompiled. In that way all three mentioned wastes will be eliminated.

Another key Lean principle is to focus on long-term results, which is the DMC architecture in our case. That is why we must point out that one important part of the DMC architecture is a MSC XML description file. If the MDA approach is introduced for the MSC definition, the central MSC

definition registry would be easily divided into several files (one per AMAS), later on. It is clear that the DMC architecture would benefit from having such a central MSC registry. The creation of one central MSC definition registry, with all MSC definitions for all markets, would be a good step towards the future DMC architecture introduction.

A. Limitations

Our company management is usually very careful with introducing concepts not already used in the company, since it often requires long implementation and learning time. Additionally, an investment in an intermediate solution is not always a very productive investment. On the other side, the management was aware that the intermediate architecture would increase productivity directly and make some new solutions possible right away. That is why the management listened carefully to our needs and made some general decisions. The intermediate architecture can be introduced, but the time-frame could be only several weeks. No new tools or licenses should be bought. Only tools that are already used within the company or some new, open-source tools, can be used. No investment in change management. Time for teaching/learning cannot be invested for the intermediate solution. The concepts, which our developers are already familiar with, should be used.

Considering these management decisions, we decided to explore if the organization was mature enough to introduce the MDA. Although the MDA approach has been around for a long time, for many companies it is still a new approach. A small survey which we performed showed that the MDA approach hasn't been used within the company and that a majority of the developers has never used this approach and that the UML modeling is not used in general. Also, the introduction of the full scale MDA usually implies: a long starting curve, which we cannot afford having a short time-frame and the usage of the MDA tools, which cannot be used since developers don't have enough knowledge about them and there is no possibility to invest in learning. In the following section it will be described how we managed to overcome these problems and limitations.

IV. AGILE MDD APPROACH

Our goal is to find an intermediate solution with a MDA philosophy, which satisfies the previously mentioned requirements and fulfills the constraints. In order to achieve this goal, we started from the basics of the MDA concept (models, transformations and code generators), and combined them with the following Lean and Agile principles [6]:

- "Think big, act small": Think about the DMC as a final architecture but act stepwise, introduce the intermediate solution first.
- "Refactoring": A change made to the structure of software to make it easier to understand and cheaper to modify without changing its existing behavior [7]"

- "Simplicity is essential": We have to find an applicable solution that is simple, keeping in mind that simple does not have to mean simplistic [8].

In that way we got our own Agile MDD approach, an applicable intermediate solution, which will be described in detail in the following section.

A. Agile modeling and code generators

We need to model the MSC definition registry. This modeling can be done on the different modeling levels and in the different modeling languages. Considering the limitations, the UML modeling cannot be accepted as a modeling solution in our project: it is not used in general and there is no time for learning. Since the XML format is a standard format and the developers are familiar with it, we decided to use a XML description as a "natural language" for the developers. XML was good enough. We had to balance between the familiarity of the XML and abstraction benefits of UML but also a complexity of the related frameworks, keeping the project within the time-frame.

We have created two models. One is a logical model which describes the entities in the MSC definition registry. Another is the MSC definition registry by itself, expressed in a XML dialect. As a consequence of that, the logical model is expressed as a XSD schema and is used to validate the entries in the registry.

The MDA defines different model categories, like a Platform Independent Model (PIM) and a Platform Specific Model (PSM) [5]. This is an important issue if there are plenty of different platforms with specifications that differ very much. In our case the different PSMs didn't differ too much from each other and, at the same time, didn't differ too much from the PIM either. In order to keep it simple we made a pragmatic solution: to have just one model, which contains all info for all programming languages. The code generators have the responsibility for creating the right MSC information to the corresponding programming language.

We needed code generators for generating the different types of files: C++, C#, Java. We decided to use XSL transformations as the code generators. They satisfied our needs and could be widely used, since the XSL is a common standard for all developers, who program in the different programming languages. In that way a "collective code ownership" [9] is achieved for the code generators. The maintainability is also better if all developers can maintain/develop the transformations.

B. Reverse engineering of the Legacy code

We needed to do a one-time reverse engineering in order to convert a large amount of the existing MSC data, legacy code, to the new MSC XML format. We developed our own tool for this purposes since no open-source tool was completely suitable. The main question was: when to start with the reverse engineering? At the end or at the beginning of the

project? Very soon we realized that we could not design our model in detail without the data from the existing MSC definitions. We decided to adopt a Spike principle. The Spike is a full cross-section of the modeling and architecture aspects of the project for a specific scenario. The aim of the Spike approach is to develop the whole chain for only one, chosen user scenario. The first chosen scenario is a simple one, and during the incremental development process every next scenario is a more complex one [10]. We started with the round-tripping (the whole chain: model – code generation – reversing back to the model) for simple scenarios, which we expanded, in each sprint, to the more complex scenarios. In that way we could develop the reverse engineering tool, the code generators and to design the model in parallel. The results of the reverse engineering helped us with the specification of the model objects for both the logical model and for the central MCS registry. Since we could do the round-tripping very early in the project, it was a way in which we could start testing our MDD approach early, under development. Round tripping in combination with the Test Driven Development (TDD) [11] will be explained in more detail in the following section.

C. Round tripping with the TDD approach

According to the Lean principles, we wanted to specify our model just according to the existing data, without unnecessary objects or unnecessary properties, which risk never to be used. In order to be able to do that, we wanted to do the reversing first and specify the logical model and fill the data in the MSC registry upon these results. We used a TDD approach and started with writing unit tests first. For this purpose we used test framework developed and already used in the company. This framework simulates the execution of the TNP messages sent among server and client components. Because of that the test scenarios that we wrote can be reused later on, for testing AMAS components, when the DMC is introduced.

According to the TDD principles we wrote the tests first, run them on "empty" code and developed the code, until the tests passed. Since we had to test several parts of our MDD approach (the logical model, the central MSC registry, the code generators and the reverse engineering tool), we established our own TDD process for the MDD testing. The main idea was to use the same test, which reflects one Spike scenario, both to develop the reverse engineering tool and the code generators, but with the input from the different sources: the legacy code was used as input when the reversing tool was developed and the generated code files was used as input when the code generators were developed. Our TDD process is presented on the "Fig. 2". Modules presented on "Fig. 2" are parts of our MDD approach where the following abbreviations are used: RE for the reversing engineering tool, CG for the code generators, LC for the legacy code and GC for the generated code.

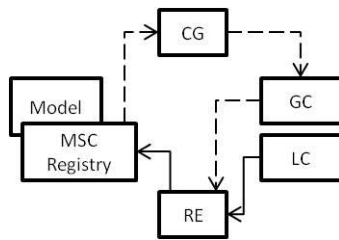


Figure 2. Our TDD process

Our TDD process will be described now through one real Spike scenario. The chosen Spike scenario is called “Get all markets” and the goal is to get all existing markets, described in the present MSC files. We started with writing a test, which consisted of sending a TNP message “TNPGETALLMARKETS”. The next step was to develop the reverse engineering tool for this scenario. The legacy code was used as input data. We developed the corresponding methods in the reversing tool, which extract markets from the existing data, producing the results in the XML format, and inserted them in our MSC registry. It was a list of all markets. Then we redesigned the model and registry entities and refactored the reversing tool according to the model changes. This process flow is presented with full arrows on the Figure 2. The TDD logic for the code generators were more complicated. What we had, so far, was the reversing tool working for the chosen scenario, and some data in the central MSC registry. We used the same test, trying to get all markets, but this time from the generated code instead (which was empty when we started), via the reversing tool (where we have some code implemented). We developed the code generators using the mentioned test. The final goal was to get the same entries in the MSC registry by the reversing of the generated code as we got by the reversing of the legacy code. After this sprint we had a list of all markets in the MSC registry, the code generators methods which generate files containing such a list and the reversing tool methods for extracting such a list from the generated files. This process is marked with dashed arrows on the Figure 2. In the following Sprints we used more advanced scenarios, such as, for example, “Get all markets where is Order supported with commands: Enter, Modify”.

At the end of each Sprint we run the whole round tripping, starting from the legacy code. In that way we could confirm that both the newly implemented code worked, as well as that the previously implemented code was not broken. As the final verification process we confirmed that all client components could be compiled without errors. We did the usual integration tests also, in order to confirm that the communication among the client components and the AMAS components has not been changed. When we completely finished with the reversing, we disabled this functionality. We needed the reversing only for extracting the existing data. It has not been

possible do the reversing nor the round tripping since the project was released.

It is important to say that we had to reverse the legacy code from the code, which was written in the different programming languages. We had to develop separate methods for the reversing from C++, Java and C#. Fortunately, the respective legacy code files had a similar structure; the syntax was the main difference. So we could develop the corresponding reversing methods based on the common objects.

The introduction of the TDD approach was important because of the following reasons:

- By developing and testing in parallel we shortened the implementation phase.
- We did not produce any wastes in the logical model (unnecessary info). We designed the model just according to the data that we got from the reverse engineering. We achieved to avoid the usual modeling mistake when a large amount of metadata is put in the model.
- We showed how the TDD can be an efficient way to work with, since this development method has not been yet widely spread within the company. When it has been introduced once, it would be easier to introduce the TDD thinking in other projects too.
- We can reuse some of these tests later on, for the DMC architecture testing.

D. Automation

We have automated some of the processes, supporting a kind of continues integration also. We reduced the amount of work and time spent for working with the MSC definition architecture. We use ClearCase (CC) as a configuration management tool and we have a build server for automatic build processes. Since all client MSC definition files were in CC, we decided to keep even the generated files in the CC repository, at least under some period. This decision was made by the management.

When the MSC definition registry file is updated and checked into CC, the following steps are executed automatically:

- The MSC definition files with hard-coded data, belonging to the client components, are checked out from CC.
- The code generators are invoked by a CC trigger script. All MSC definition files are generated.
- All generated files are checked into CC, if the generation did not fail. Otherwise the “undo checkout” operation is done.
- All client components, affected by the mentioned code generation, are recompiled. If some compilation fails, the error report is immediately sent to the component owners.

V. AGILE AND LEAN PRACTICES IN MDD

The Agile and Lean methods are light in contrast to the MDA that can become complex, because of all standards and OMG recommendations. Through the

application of the Agile and Lean principles, the MDD becomes more pragmatic and more useful. Some of the Agile and Lean principles, used in our Agile MDD approach, are explained below.

“Eliminating waste”: Eliminating the duplication of information was also according to the XP’s principle “Never duplicate your code” [9]. This principle is the heart of the MDD – to have one central input point, model (models) from which everything else is generated.

“Think big, act small”: We were thinking on the DMC as a final architecture but acted in a stepwise way, via an intermediate solution.

“Deliver as fast as possible”: The implementation phase of our Agile MDD approach was short.

“Empower the team”: Roles are turned – the managers are taught how to listen to the developers [3]. Despite the fact that managements put non-technical constraints on our project, they allowed the developers to make decisions, regarding the intermediate solution, on their own. It contributed to faster development, since the developers did not have to wait for feedback from the management, for each decision.

“Spike principle” applied on the reverse and round-trip engineering made the introduction of the TDD philosophy spontaneous and natural.

“Simplicity is essential.” We have simplified the full scale MDA. Instead of the UML modeling language we used the XML. The PIM and PSMs were merged, avoiding the maintenance of several models and transformations among them. On the other side, by merging PIM and PSMs in one model we lost a good Separation of Concerns but it was a price worth paying.

“Welcome changing requirements, even late in development.” The case-study presented an iterative development, which allowed late model changes. We worked in sprints, according to the Spike principle, which implied the frequent model changes, in each sprint.

A. Benefits of the Agile MDD approach

We got a lot of benefits by introducing the Agile MDD approach. Now we will list them:

1. Agile principles can make the starting curve for the MDD shorter. Through the application of the Agile principles the long learning curve and introduction gap of MDD methods and tools could be avoided.
2. We introduced the TDD approach, showing the effectiveness of such an approach.
3. We have prepared, in advance, for the introduction of the DMC architecture: the model specification and the reverse engineering job are already done. As well as the test cases, some of them are going to be reused.
4. The Agile MDD approach could be used instead of the full scale MDA. When all MDA recommendations could not be applied, we

adjusted them to our system and organization, with a help of Agile and Lean principles.

VI. CONCLUSION AND FUTURE WORK

The main point of this paper was to show how Lean and Agile principles helped us with producing an intermediate solution, with a short implementation phase, for the architecture of the MSC definition. In that way we coped successfully with the management constraints, achieving the implementation within the short time-frame and without investment in change management.

Our Agile MDD approach is based on the general MDA idea but is shaped then with the Lean and Agile principles. “Eliminating waste” helped us to detect main wastes. The most important was the duplication, which we eliminated by applying the MDA philosophy. “Simplicity” Agile principle reduced the MDA concept to the single modeling level, expressed in the XML dialect. By being aware of “Think big act small”, we could produce such an intermediate solution, which can be easily improved in the long-term solution. The TDD logic improved the development efficiency and decreased the total time spent on the development and testing. We got a simple and applicable solution which will easily grow to a more complex one.

“A complex system that works has usually been evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over with a simple system. [12]”

REFERENCES

- [1] SunGard, www.sungard.com. Accessed in May 2011.
- [2] TNP SDK documentation: SunGard Front Arena
- [3] Mary Poppendieck, Tom Poppendieck: Lean Software Development, An Agile toolkit. Addison Wesley, 2005.
- [4] James McGovern, Scott Ambler, Michael Stevens: A practical guide to Enterprise Architecture. Prentice Hall PTR, 2003.
- [5] MDA, www.omg.org/mda. Accessed in May 2011.
- [6] AgileManifesto, www.agilemanifesto.org. Accessed in May 2011.
- [7] Martin Fowler: Refactoring: Improving the Design of Existing Code. Addison Wesley, 1999.
- [8] James O. Coplien, Gertrud Bjornvig: Lean Architecture for Agile Software Development, Wiley 2010.
- [9] Ron Jeffries, Ann Anderson, Chet Hendrickson: ExtremeProgramming. Addison Wesley, 2001.
- [10] Ray Carroll, Claire Fahy, Elyes Lehtihet, Sven van der Meer, Nektarios Georgalas, David Cleary: Applying the P2P paradigm to management of large-scale distributed networks using Model Driven Approach, Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP Volume, Issue , 3-7 April 2006 Page(s):1 – 14.
- [11] Michael C. Feathers: Working Effectively with Legacy code. Prentice Hall PTR, 2005.
- [12] John Gall: Systemantics: How Systems Really Work and How They Fail. Quadrangle, 1975.

A Planning Poker Tool for Supporting Collaborative Estimation in Distributed Agile Development

Fabio Calefato
Dipartimento di Informatica
University of Bari
Bari, Italy
calefato@di.uniba.it

Filippo Lanubile
Dipartimento di Informatica
University of Bari
Bari, Italy
lanubile@di.uniba.it

Abstract— Estimating and planning are critical to the success of any software project, also in the case of distributed agile development. Previous research has acknowledged that conventional agile methods need to be adjusted when applied in distributed contexts. However, we argue that also new tools are needed for enabling effective distributed agile practices. Here, we present eConference3P, a tool for supporting distributed agile teams who applies the planning poker technique to perform collaborative user story estimation. The planning poker technique builds on the combination of multiple expert opinions, represented using the visual metaphor of poker cards, which results in quick but reliable estimates.

Keywords- distributed; agile; estimation; planning

I. INTRODUCTION

Software estimation and planning activities aim to create meaningful cost and schedule estimates for a project. The ability to accurately estimate the time and cost for a project is a key factor to its successful conclusion. Hence, estimating and planning are critical activities also in the case of distributed agile development. Unfortunately, agile and distributed development practices are so different that, when blended together, the key characteristics of the former exacerbate the challenges intrinsic to the latter, creating a set of brand new challenges. In fact, as any agile method, agile planning is based upon intense interactions among individuals and thus, it emphasizes the need for frequent informal interaction and communication. On the contrary, in distributed software development communication and interaction are dramatically hindered due to the absence of collocation.

Collaborative software development across distances has become commonplace for a number of years [19]. However, there are still important problems to solve that are strictly related to the effects of distance among the members of a development team [7]. It is well known that a distributed approach to software development increases difficulties related to coordination, control, and communication mechanisms, which are fundamental for any software project. Quite the opposite, agile software development methodologies are based on strong collaboration and frequent informal communication among project members [13]. Among the underlying principles that underpin agile

methodologies, personal relationships and direct communication among people are considered as the best resource in a project [4].

There is an increasing interest towards new experimental approaches that aim to combine the specific characteristics of agile methodologies with those of distributed software development [23]. Previous research has acknowledged that conventional agile methods need to be adjusted when applied in distributed contexts. However, we argue that also new tools are needed for enabling effective distributed agile practices. In particular, we argue that tools that provide better communication support are needed in order to cope effectively with the reduction of direct, synchronous interaction.

In this paper, we present eConference3P (eConference Planning Poker Plugin), a tool meant for supporting distributed agile teams who applies the planning poker technique to perform collaborative user-story estimation. The planning poker technique builds on the combination of multiple expert opinions, represented using the visual metaphor of poker cards, which results in quick but reliable estimates. Our tool has been developed as a plugin of the eConference system, a communication platform that connects to either Google Talk or Skype networks and thus, allows the organization of text- and audio-based conferences. Among the other features, eConference3P allows to visually edit user stories and import a backlog from many collaborative development environments such as Google Code, Assembla, Github, Trac, and Jira.

The remainder of this paper is structured as follows. Section 2 discusses in detail the planning poker estimation technique. Section 3 presents our agile planning prototype. Instead, related academic and industrial tools for agile estimation are illustrated in Section 4. Finally, we conclude in Section 5.

II. AGILE ESTIMATION & PLANNING POKER

Before starting a project, whatever agile methodology a team is applying, developers have to deal with iteration planning and, therefore, user story estimation. A user story is a brief description of functionality as viewed by a user or customer of a system. User stories are free-form and there is no mandatory syntax, although they are generally formulated according to the following template: "As a <role>, I want <goal/desire> so that <benefit>" [6]. In agile development,

user story estimates are not defined individually by just one developer. Instead, estimates are obtained collaboratively by (part of) the agile team, including those developers who will actually implement the user stories.

The size of user stories can be estimated in story points or ideal days. *Story points* are a relative unit of measure, used to estimate the size of user story by combining the effort, the complexity, and the risk inherent in its development. *Ideal days*, instead, are used to evaluate the size of a story in terms of the amount of time it will take to be fully developed. Both story points and ideal days values are arranged in an estimation scale. Although any sequence might work, Cohn [6] suggests using nonlinear sequences (e.g., the Fibonacci sequence 0,1,2,3,5,8,13...). Because the gaps between values become appropriately larger as the numbers increase, such sequences better reflect the greater uncertainty associated with larger estimates.

To arrive at a shared estimate, agile teams rely on three main techniques: expert opinion, analogy, and disaggregation.

In the *expert opinion*-based approach, experts assign estimates to user stories relying on their intuition. Typically, multiple expert opinions are needed because implementing a system functionality described by a user story requires a number of multidisciplinary skills that normally belong to more than one developer. The expert opinion-based approach has been found to be more effective than others [17].

In the *analogy*-based approach, estimators compare the user stories to be estimated to one or two other stories already estimated before. This approach builds on the fact that humans find easier to estimate relative size than absolute size. Thus, in the typical scenario, if an estimator believes that user story A is twice the size of story B, which was estimated at 5 story points, then A is estimated at 10 points. The comparison can be of course generalized by comparing the size of user story A to a couple of stories already estimated. Obviously, this approach suffers from a cold start problem and, therefore, works better when at least a few user stories have been already estimated.

Finally, in the *disaggregation*-based approach, before estimating the expert splits a large user story into multiple smaller ones, easier to evaluate and compare. In fact, if user

story A is much bigger than previously estimated user story B, it would be hard to say that A is fifty times as complex as B. Therefore, disaggregation works well with the analogy-based approach.

An effective way for combining the three estimation techniques is planning poker [6]. In planning poker, each estimator is given a deck of cards with a valid estimate shown on each. A feature is discussed and each estimator selects the card that represents the estimate. All cards are shown at the same time. Then, the estimates are discussed and the process repeated until agreement on the estimate is reached. Typically, a planning poker session is arranged at the beginning of a project, to estimate user stories so that the first iteration can begin. Then, further sessions may be arranged after each iteration to estimate new stories, if any.

Planning poker is an effective way to estimate user stories for at least a couple of reasons. First, it brings together a cross-functional, agile team of experts from different disciplines, whose averaged estimations tend to be more precise than individual scores [14]. Second, it fosters group discussion, as estimators need to justify their scores, which has also been found to lead to better results, especially in case of high amounts of uncertainty and missing information [18].

III. ECONFERENCE3P

eConference3P (see Figure 1) is a tool developed for supporting distributed agile teams who perform collaborative user story estimation by applying the planning poker technique. As shown in the figure, the eConference3P user interface has five main areas. The *message board* is the view that collects all the messages from the discussion that ensues upon any estimation. In particular, the message board is “threaded”, in the sense that messages get stored with respect to the user story that they are related to. We point out that our tool distinguishes the roles available in an agile team and, in particular, between project owner and developers. Relevant notes and decisions, taken through the meeting, are logged in the *decision place*, which can be only

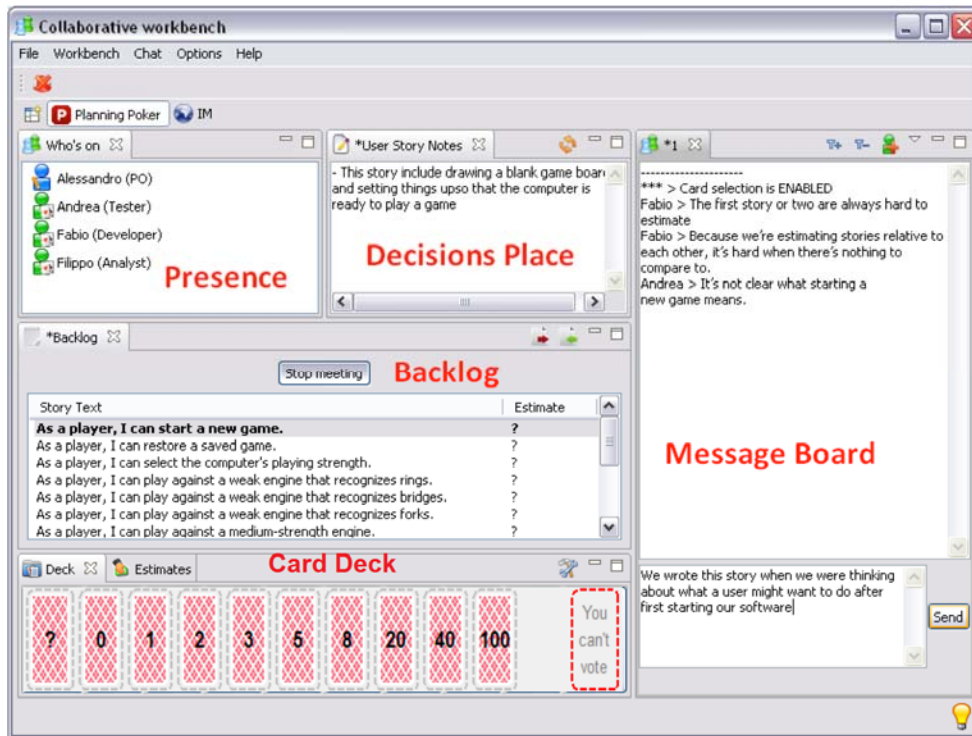


Figure 1. A screenshot of eConference3P.

edited by participants who receive from the project owner the right of acting as scribe. A project owner acts as the moderator of the planning session and, as such, the user interface of the tool enables specific actions to manage the meeting, load user stories, call for and accept the estimates, and grant/revoke rights from other participants. For the sake of space, through the rest of the paper we report the screenshots only from the perspective of the project owner. The *backlog* view allows starting and stopping the meeting, as well as importing and exporting the user stories, which are also listed together with the accepted estimates, once available. The *card deck* view shows the scale from which developers pick the score. Finally, the *presence panel* shows the team members that are participating in the planning meeting, along with their roles (e.g., project owner, developer) and their rights (i.e. to estimate, scribe, chat).

Through the rest of this section, we first describe the architecture of eConference3P, showing how its building components have been arranged together, and then, we discuss in more detail its features.

A. eConference3P Architecture

eConference3P is built around two main components, which are the results of two academic research projects named eConference and AgilePlanner. Both can be run as either standalone applications or Eclipse IDE plugins. In fact,

such components could be seamlessly and almost effortlessly integrated because both are rich client applications, developed using the Eclipse Rich Client Platform (RCP) technology, a pure-plugin development platform that is fully extensible by architectural design [8].

eConference [5] is a distributed meeting system, previously developed by our research group at the University of Bari, Italy. Its primary functionality is a closed group chat, augmented with agenda, meeting minutes editing, and typing awareness capabilities. Around this basic functionality, other features have been built to help organizers control the discussion during distribute meetings. eConference can use either XMPP, an IETF standard protocol, or Skype. In the latter case, also VoIP communication is supported.

AgilePlanner [21] is a tool for synchronous, card-based agile planning meetings, developed at the University of Calgary, Canada. AgilePlanner mimics paper index cards as it simulates a whiteboard in a meeting room and utilizes electronic index cards (see Figure 2). AgilePlanner is a client/server application with its own communication protocol. The tool is specifically intended to support distributed agile teams (i.e. work with networked clients), rather than being an offline visual editor for planning artifacts.

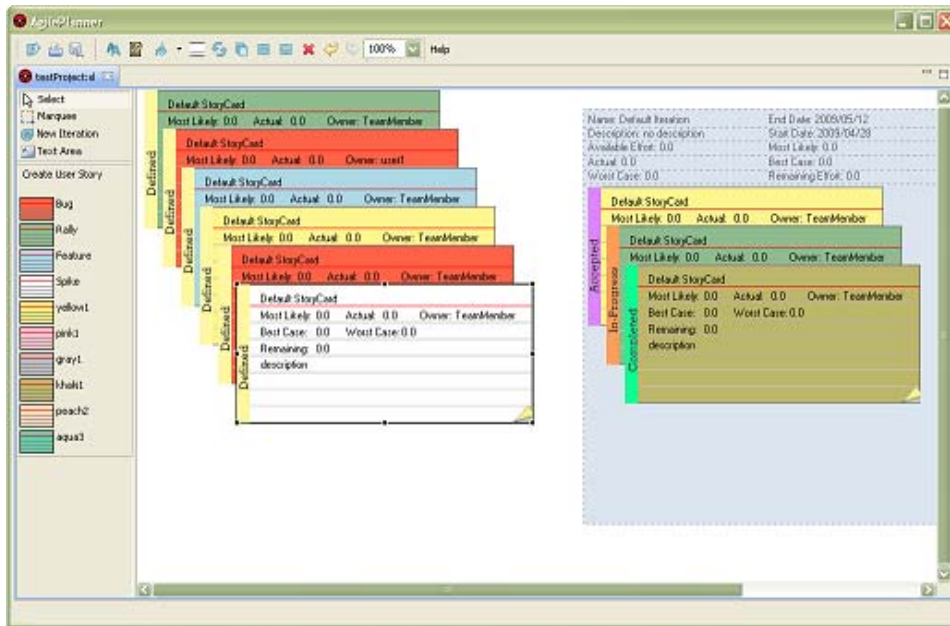


Figure 2. AgilePlanner with user story cards being assigned to an iteration.

B. eConference3P Features

In this section, we illustrate the features of our tool eConference3P against a few requirements acknowledged as critical in the field of distributed agile development [1] [20] [26].

1) Offline/online working switch.

eConference and, therefore, eConference3P too, work with XMPP-based Gmail accounts since the third release and with Skype accounts since the fourth. One of the benefits of our solution is that eConference products work without requiring any user or maintainer to install a server, thus minimizing the hassles coming from installations and configurations.

In developing our planning poker plugin for eConference, we selected AgilePlanner as the graphical editor of user stories and iterations. However, AgilePlanner required a connection to a server to work. Therefore, since we mostly needed AgilePlanner for its editing functionality, we patched it in order to support both online and offline mode [2]. In the offline mode, a user has the chance to store all the planning artifacts on a file and then load them back later. The online mode, instead, remained untouched. Because the transition from offline to online co-editing is not fluid (i.e. developers need to connect again to the Agile Planner server), at the end of the integration process, we felt that the presence of a proprietary server to install clashed with our intention of building an extensible, hassle-free planning poker tool.

2) Simultaneous interaction and manipulation of artifacts through telepointers

With respect to the supported planning activity, AgilePlanner is primarily focused on the interactive collaboration and meant for conducting real-time planning

meetings, whereas it only has limited capabilities for progress tracking during the interaction.

The user interactions of AgilePlanner include the complete manipulation of planning cards (i.e., creating, moving, and deleting cards). Different colors are used to distinguish between the cards representing bugs, spikes, features, user stories, and finally iterations and backlog, to which they are assigned. To support distributed collaboration, AgilePlanner provides telepointers, which are a groupware technology that uses a remote mouse pointer to represent mouse movement happening on other connected computers so that remote collaborators can understand other team members' mouse movements, much like they would look at others' movements in a traditional co-located meeting.

3) Real-time information sharing & estimation.

In eConference3P all the changes happening to the shared workspace are notified in real time, so that updated information is simultaneously available to each remote developer. In particular, eConference3P focuses on supporting synchronous interaction rather asynchronous interaction between distributed agile team members. When playing planning poker, near real-time interaction is fundamental to support the discussions and converge to a shared estimate, when individual scores differ.

With respect to estimation-specific features, the project owner can import, export, and edit user stories from the backlog view. In particular, as for the edit feature, selecting that menu entry will change the current perspective of the tool to that of Agile Planner, as shown in Figure 2, thus letting distant developers move user stories in or out of the backlog, as well as plan multiple interactions for long-term release planning. The project owner can also call for estimation. When an estimation procedure starts, the deck

becomes clickable, and each developer can pick the card with the desired score by dropping it in the drop zone on the right hand side of the deck view. The project owner can check who provided what estimate at any time. Instead, developers' estimates will not be visible to each other until all of them provide one. eConference3P also allows the project owner to select the estimate scale of choice before a planning meeting is started.

4) *Integration with others development environments*

Supporting integration with development environments increases the ease of access to the planning information for developers and makes it easier to track progresses. Therefore, in eConference3P we enabled the import of user stories from the most used, web-based collaborative development environments (CDEs). CDEs such as Google Code [11], Github [10], and Trac [24], offer issue-tracking features for storing items, such as bug descriptions, enhancements, and milestones. However, they are also used by developers of agile teams to store planning artifacts, such as the backlog of user stories and iterations. The import procedure locally stores the data retrieved in the same XML format supported by AgilePlanner, so that imported data can be graphically edited afterwards.

Table I shows the four CDEs supported by the import procedure. First, we notice that all the CDEs offered official APIs to programmatically query and retrieve the information from the project repository. The only exception was Trac, for which we had to develop a custom scraper that makes http requests and then parses the resulting html output to retrieve the information needed. Because this solution depends on the structure of web pages, using a scraper is considerably less stable than using an API, since even smallest changes to the graphical layout may end up breaking it. Second, we notice that Assembla [3] is the only CDE that specifically support user story entries for its repository, whereas Google Code and Trac allow customizing generic entries (called tickets) into user stories, and later retrieve them through custom search queries. Github, instead, does not offer any of the two solutions and, thus, proved to be the least effective CDE for hosting an agile project repository. Lately, we have also added support to Jira [16] and Fogbugz [9].

IV. RELATED WORK

Tools for supporting agile development have been some ten years in the making. To date, there are literally hundreds of agile project management tools, some more complete than others, which tend to focus only on a specific activity of the agile management process. These tools typically allow teams to manage agile projects following Scrum and XP agile methodologies. A list of the most used and well-known application can be found on UserStories.com [25]. Such tools, whether free or commercial, can be broadly divided into three main categories, according to their target platforms: web-based, standalone, and plugins.

Web-based is the category that accounts for the largest number of existing agile planning and management tools. This is because such applications only require a web browser to be executed on the client side. Besides, as for commercial tools, web application as are often sold in "hosted mode",

which requires no installation by customers since companies sell seats to use the service running on their own servers. Among web applications, we can first identify general purpose Wikis, used for agile project management in general, and estimation as well, by letting developers create, edit, and publish story cards and other artifacts as web pages. As such, they do not offer any specific support to agile practices and, therefore, only meet a very minimal set of requirements for agile projects management. On the contrary, there are tens of tools designed for agile project management, both commercial (e.g., Mingle, VersionOne, and Rally) and free (e.g., XPlanner, Agilo for Scrum, Agilefant, and eXPlainPMT), which offer sophisticated features to represent and manipulate project data, but none of them support the planning poker technique. The only tool that supports the homonymous agile estimation technique is PlanningPoker.com [22], which we analyze in detail in the next section.

The second category of agile project management applications is that of *standalone* tools, most of which run natively just on Windows with a very few alternatives for Linux and OS X built on Java. In this category, we identified no standalone tool supporting planning poker, other than eConference.

Finally, *Integrated Development Environments (IDE)*, such as Visual Studio and Eclipse, have also been extended through specific plugins in order to support, among the other things, agile practices and create an even more convenient development environment for closely managing and interconnecting code artifacts, such as test cases, and agile planning artifacts, such as story cards. In this category we identified WolfPoker [27], a planning poker plugin for Jazz, a commercial CDE developed by IBM that supports the customization and execution of any agile project management process of choice.

A. *Comparing Planning Poker Tools*

From the review in the previous section, we note that PlanningPoker.com (web-based) and WolfProject (Jazz plugin) are the only other existing tools that support the planning poker estimation technique as eConference3P (standalone), as shown in Table II. All the three tools support synchronous sessions (i.e., backlog editing and estimation), while PlanningPoker.com is the only one that also enables asynchronous estimation sessions.

Besides, we note that both PlanningPoker.com and WolfPoker support collocated groups of developers only in picking scores from a card deck and then visualize the estimates, while they completely lack any communication feature to support discussion. This is probably due to the fact that collocation and frequent direct communication are paramount for agile teams [4]. However, as distributed agile teams get more and more common [12], face to face communication cannot be given for granted any longer. Hence, distributed agile teams willing to adopt PlanningPoker.com or WolfPoker must also use such applications in combination with other communication tools.

TABLE I. THE CDEs CURRENTLY SUPPORTED BY THE IMPORT PROCEDURE.

CDE	API	User story	Milestone	Custom ticket type	Custom search query
Assembla	X	X	X		
Github	X				
GoogleCode	X		X	X	X
Trac			X	X	X
Jira	X				
Fogbugz	X				

eConference3P, instead, integrates text-based and audio communication to support estimate synchronous discussions with no hassles. In addition, thanks to the AgilePlanner component, eConference3P allows collaborative editing of the backlog.

Finally, eConference3P is the only tool that can import a backlog from a number of CDEs, such as Google Code, Github, and Jira, whereas WolfPoker can only read file exported from MS Project.

V. CONCLUSIONS

In this paper, we presented eConference3P, a tool for enabling effective estimation meetings for distributed agile teams. The tool was built by integrating the AgilePlanner component, to enable iteration planning through a visual editor, and the eConference meeting system, to build a better communication tool and cope with the reduction of information exchanged in distributed settings. In fact, our review of existing tools for performing planning poker agile estimation revealed a lack of support for synchronous communication. Being based on the Eclipse RCP platform, specific plugins were then added to support the planning poker estimation technique and import user stories from web-based collaborative development environments.

TABLE II. A COMPARISON BETWEEN TOOLS SUPPORTING PLANNING POKER

Feature	eConference3P	PlanningPoker.com	WolfPoker
Category	Standalone	Web based	Plugin (Jazz)
Sync. sessions	Backlog editing, estimation	Backlog editing, estimation	Backlog editing, estimation
Async. sessions	Backlog editing	Backlog editing, estimation	Backlog editing
Comm. modes	Text, audio	None	None
Backlog editing	Yes (co-editing)	None	Yes
Integration w/ CDEs	Backlog import	None	Backlog import*

* only supports MS Project file format

REFERENCES

[1] Abrahamsson, R., Salo, O., Ronkainen J., and Warsta J. (2002). *Agile Software Development Methods*. VTT Publications, vol. 112.

[2] AgilePlanner for eConference, <http://code.google.com/p/agileplanner-for-econference> (last accessed: Jul. 19, 2011).

[3] Assembla, <http://www.assembla.com> (last accessed: Jul. 18, 2011).

[4] Beck, K. et al. (2001). *Manifesto for Agile Software Development*. <http://agilemanifesto.org> (last accessed: Jul. 18, 2011).

[5] Calefato, F., and Lanubile, F. (2009). *Using Frameworks to Develop a Distributed Conferencing System: An Experience Report*. *Software: Practice and Experience*, vol. 39, no. 15, pp. 1293–1311

[6] Cohn, M. (2005). *Agile Estimating and Planning*. Prentice Hall

[7] Damian, D., Sengupta, B., Lanubile, F. (2008). *Global Software Development: Where Are We Headed?* *Software Process Improvement and Practice*. vol. 13, pp. 473-475.

[8] Eclipse RCP, <http://www.eclipse.org/home/categories/rcp.php> (last accessed: Jul. 19, 2011).

[9] Fogbugz, <http://www.fogcreek.com/fogbugz> (last accessed: Jul. 18, 2011).

[10] Github, <https://github.com> (last accessed: Jul. 18, 2011).

[11] Google Code, <http://code.google.com> (last accessed: Jul. 18, 2011).

[12] Herbsleb, J.D. (2007), *Global Software Engineering: The Future of Socio-technical Coordination*. *Future of Software Engineering (FoSE'07)*, May 23-25, pp.188-198.

[13] Highsmith, J., Cockburn A. (2001) *Agile Software Development: The Business of Innovation*. *Computer*, vol. 34, no. 9, pp. 120-122.

[14] Hoest, M., and Wohlin, C. (1998). *An Experimental Study of Individual Subjective Effort Estimations and Combinations of the Estimates*. *Proc. 20th Int'l Conf on Software Engineering (ICSE'98)*, Kyoto, Japan, Apr. 19-25, pp. 332-339.

[15] Jazz, <https://jazz.net> (last accessed: Jul. 18, 2011).

[16] Jira, <http://www.atlassian.com/software/jira> (last accessed: Jul. 18, 2011).

[17] Johnson, P., Moore, C.A., Dane, J.A., and Brewer, R.S. (2000). *Empirically Guided Software Estimation*, *IEEE Software*, vol. 17, no. 6, pp. 51-56.

[18] Jorgensen, M., and Molokken, K. (2002). *Combination of Software Development Effort Prediction Intervals: Why, When and How?* *Proc.14th Int'l Conf. Sw. Eng. and Knowledge Engineering (SEKE '02)*, vol. 27, Ischia, Italy, Jul. 15-19, pp. 425-428.

[19] Lanubile F., Damian D, Oppenheimer H. (2003). *Global Software Development: Technical, Organizational, and Social Challenges*. *Software Engineering Notes*. Vol. 28.

[20] Larman, C. (2004). *Agile & Iterative Development - a Managers's Guide*, Addison-Wesley.

[21] Morgan, R., and Maurer, F. (2008). *An Observational Study Of A Distributed Card Based Planning Environment*. *Proc. 9th Int'l Conf. on Agile Processes and eXtreme Programming in Software Engineering (XP'08)*, Limerick, Ireland, Jun. 10-14, pp. 53-62..

[22] PlanningPoker.com, www.planningpoker.com (last accessed: Jul. 18, 2011).

[23] Šmite, D., Wohlin, C., Gorschek, T., and Feldt, R. (2010). *Empirical Evidence In Global Software Engineering: A Systematic Review*. *Empirical Software Engineering*, vol. 15, pp. 91–118.

[24] Trac, <http://trac.edgewall.org> (last accessed: Jul. 18, 2011).

[25] UserStories.com, <http://userstories.com/products>, (last accessed: Jul. 18, 2011).

[26] Wang, X., Maurer, F., Maurer, R., and Oliveira, J. (2010). *Tools for Supporting Distributed Agile Project Planning*, in *Agility Across Time and Space* (Smitte, Moe, Ågerfalk eds.), Springer, pp. 183-199.

[27] Wolfpoker, <http://www.researchgroup.org/wolfpoker> (last accessed: Jul. 18, 2011).

Scrum Maturity Model

Validation for IT organizations' roadmap to develop software centered on the client role

Alexandre Yin

Departamento Engenharia Informática (DEI)
Instituto Superior Técnico (IST)
Lisboa, Portugal
alexandre.yin@ist.utl.pt

Soraia Figueiredo; Miguel Mira da Silva

INESC . INOV
Instituto Superior Técnico (IST)
Lisboa, Portugal
soraia.figueiredo@inov.pt; mms@ist.utl.pt

Abstract—Within the agile development methodologies context, the topic of client relationship management is strongly focused, mainly due to the importance of collaboration between the development team and its clients. Most clients avoid or are unable to develop a close cooperation with vendor organizations, since it requires a motivation and close participation among key stakeholders in the development processes within and correct usage of the adopted software development methodology. Hence, software development projects fail and become unsuccessful because of this lack of communication. In order to increase the rate of successful projects, this paper will present the journey of the validation process for this roadmap to lead and aid software vendor organizations improve their development processes, concentrating mainly on the client's role throughout the process. This concept is called Scrum Maturity Model; therefore, our main goal is to validate this concept with organizations that use Scrum agile methodology as their main development process, which turns out to be a viable approach to reduce the rate failed development projects.

Keywords-development methodologies; agile methodologies; scrum development methodology; maturity model; action research

I. INTRODUCTION

According to a CHAOS Report [1], about 70% of IT development projects fail to deliver functional software, mostly due to a poor communication between stakeholders, who play key roles in the development process. This problem of human factors in software development collaboration is also highlighted in these three following papers [2][3][4][5].

The fact that most clients spend an extremely small amount of time and effort working closely with the software vendor organization, that develops the solution, goes against the Agile Manifesto values [6], which are the foundations for a successful agile oriented development.

The failure of Information Technology (IT) projects caused by mediocre software requirements engineering and other human/client factors is a highly researched theme among professionals and scholars. Therefore, this paper intends to provide a different insight about the current issues concerning this topic [7] [8][9][10].

The main concern that induced this research was precisely the dilemma mentioned above: lack of cooperation among stakeholders involved in an IT development project, focusing

on the type of communication between the development team and the client. This problem in communication can result from: (1) Human factors and resistance to changes; (2) Distance that separates both vendors and clients or; (3) Inexistence of a commitment that follows the definition of a contract of collaboration.

Generally, both clients and software development organization teams may fear and avoid the adoption of new methods of collaboration with a new team [10]. This harms the partnership between the two, thus resulting in inadequate requirements engineering emphasized by agile methodologies, which will, eventually, lead to an unsuccessful project.

Concerning human behavior, the distance that separates the vendor organization and the client challenges the accomplishment of a fluent and successful cooperation [11]. Apart from this exact physical distance, that hardens the communication and occasionally blocks the possibility of face-to-face meetings, a cultural distance must also be considered, since this aspect may bring a negative impact, such as cultural clashes, to the performance of the collaboration and influence the project as a whole [10].

Another cause of this problem is the inexistence of highlighted goals, such as market competition, which will motivate all stakeholders to improve their processes and maximize the outputs. According to a survey made by Gartner [13], agile methodologies could use a maturity model as a roadmap and market differential, so software development organizations might explore their processes and reach higher levels of maturity. Moreover, a paper from Software Engineering Institute (SEI) [14] reveals that Capability Maturity Model Integration (CMMI) can coexist with agile methodologies and enhance these software development organizations [15].

This paper will focus on the changes from the previous proposal [16] and recent evaluation processes of the solution for this lack of collaboration, usually, between vendor organizations and clients. Moreover, it will conceive a roadmap for improvement in order to create successful IT development projects. Since Scrum development methodology emphasizes such collaboration, the solution shall be molded as a roadmap in the form of a maturity model so as to achieve the goal of this paper.

Note that this topic of maturity models and other IT governance frameworks on agile methodologies a highly polemic among the agile community. Nevertheless, IT governance mechanisms are necessary and welcomed in organizations which are underproductive, and, thus, hold the major slice of failed projects [14].

The chosen research method was Action Research (AR) due to its success in various academic investigations in the Information Systems area and for allowing the researcher to interfere and observe introduced modifications on the studied environment. AR is comprised by a five stages cycle [17]: (1) Diagnosis – problem identification; (2) Action Planning – planning and research phase to prepare the experiment and alternative actions; (3) Action – implementation of planned actions, introduction of changes and analysis of the outputs on the environment; (4) Evaluation – it is determined if the outcomes are expected or against odds and assures that introduced actions are the only reason for the obtained success; (5) Specifying Learning – Identify general findings.

Note that AR is carried out by individuals who are interested parties in the research. This fact has led to criticisms of the validity of the research process, with accusations of inevitable researcher bias in data gathering and analysis. The justification for AR counters this criticism by suggesting that it is impossible to access practice without involving the practitioner. Practice is action informed by values and aims which are not fully accessible from the outside. The practitioner may not even be wholly aware of the meaning of his or her values until he or she tries to embody them in her action.

Nevertheless, there are some limitations with this research methodology, namely: the unfamiliarity with research methods and the representations of the process of action research may confuse, rather than enlighten.

As stated, this paper continues our previous research, hence, the first two cycles of action research were already previously applied. This paper will mainly focus on the changes to the proposal, based on past learning, and iterate more cycles of action research in order to achieve stronger validation of the proposal.

Before the presentation of the improved proposition, a brief introduction and review of the related work in this area of research shall be developed in Sections II and III. After, the changes in the proposition are detailed in Section VI; in the next section (Section V) the results of newer and various practical experimentations of the proposition will be presented. Afterwards, in Section VI, the main lessons learned shall be analyzed. Finally, Section VII will conclude with the summary of this investigation, relating all mentioned topics as a whole. In this section, some future works and approaches are given to continue the research.

II. RELATED WORK

This section intends to make a brief review of the related work in the field of agile development study.

A. Agile Methodologies

The origins of Agile methodologies are deeply connected with the concepts of iterative and incremental development. There were several ideas concerning the agile concept, hence an Agile Manifesto [6] was established.

The set of values and inherent principles listed on this Manifesto stress the importance of the clients' presence in order to obtain a better collaboration outcome, working software as the main goal and agility when facing a sudden change in requirements [18][19].

Since this approach requires a high cooperation level between the client and the development team, mainly through face-to-face meetings, it has the drawback of being partially obsolete in the current market, in which an ascending number of projects are developed at a distance [20][21][22].

B. Scrum

Scrum is an agile methodology to manage development projects through an iterative and incremental method [23][24][25]. It is divided into three main key roles: (1) Scrum Master – individual who is responsible for the Scrum process and its correct usage maximizing its benefits; also known as the facilitator of Scrum team; (2) Product Owner – individual who is accountable for the alignment of the development and business goals definition, and; (3) Team – team that is in charge of delivering the product. A team comprises 5 to 9 members with cross-functional skills, who are self-organized and self-led.

This methodology identifies four objects that are operated by the Scrum team throughout the development cycle: (1) Product Backlog – a prioritized list of everything necessary to conclude the product; (2) Sprint Backlog – a list of tasks to perform during a sprint, i.e., an up to four weeks development iteration to introduce parts of the Product Backlog into working software; (3) Release Burndown Charts – charts that show the progress of the project over time, and; (4) Sprint Burndown Charts – charts that show the progress of the sprint over time.

The interaction of the roles maneuvering these objects is set for the following meeting: (1) Release Planning Meeting – Scrum team gathers and fills in the Product Backlog; (2) Sprint Planning Meeting – development team and client closely discuss matters and define the goals for the next sprint; (3) Daily Scrum – a brief meeting for developers to identify personal issues and possible improvements in methodology usage; (4) Sprint Review – demonstration of the working software to the client and stakeholders; (5) Sprint Retrospective – team performs a self-examination regarding the last sprint in order to seek improvements on their use of Scrum Methodology and collaboration in general.

Scrum methodology is an iterative and incremental development methodology. The phase for planning and system architecture takes place in Release Planning Meeting, while the sprints are comprised by Sprint Planning Meetings, Daily Scrum, Sprint Review and Sprint Retrospective.

Although Scrum has a wide definition of concepts, that, when applied, may allow agile software development, it cannot guarantee the success of IT projects. This methodology

emphasizes close collaboration between development teams and their clients; still, most of the time this does not happen and, thus, a supplementary solution to complement this imperfection is needed.

C. Modified Agile

Modified Agile is an agile development methodology that results from the analysis of the flaws in the Agile Manifesto [6] opposing to distant outsourcing environment [11].

The main problems identified concerning this matter were the poor communication among participants of the IT projects and the exhaustive documentation needed for contract negotiation. All other values and principles mentioned in Agile Manifesto remain feasible in a distant outsourcing context.

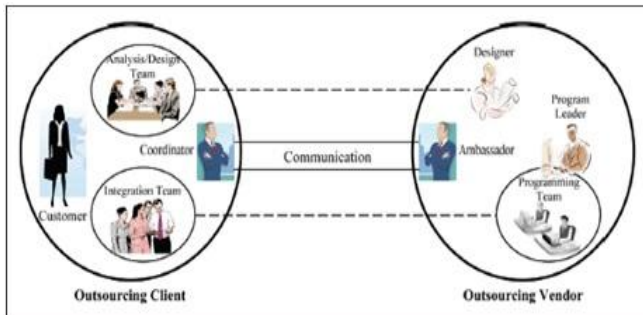


Figure 1. Modified Agile communication model proposal [11].

The solution recommended by the author of this paper is an authentic communication model and team composition structure, which will enhance the communication between clients and developers and reduce the negative effects derived from the distance factor that leads to a loss of knowledge.

In Figure 1, the introduction of two specific roles is emphasized: (1) Coordination – an individual from the client-side, who ensures the maximization of development outputs by assigning the most important business goals to be developed as a priority; (2) Ambassador – individual from the development team-side who makes sure that the product developed is aligned according to the customer’s needs and wills. These two roles must work closely as a formal communication channel, while team members from both development and the client-side might communicate among themselves through an informal channel

Although this distributed agile concept is broadly used with several case studies proving its success, there are also many failed IT projects due to human factors and inadequate collaboration between clients and vendor organizations [26][27].

III. MATURITY MODELS

The maturity models from software development processes enable the classification of the performance of the actual ones and guide organizations to encourage process improvement through a staged method, also known as maturity. These maturity models are an interesting approach to solving the problem described in Section I, since the presence of a

maturity classification can allow the comparison between competitor organizations.

A. Capability Maturity Model Integration

Capability Maturity Model Integration (CMMI) was introduced in 2002 and ever since, it has focused on process improvement approaches, which assist organizations in adopting the best type of practices from each process area and make the processes performance evolve [28][29].

In the staged representation, CMMI presents different levels that vary from one to five. One level of maturity is characterized by a set of predefined process areas, evaluated by the accomplishment of specific and generic goals applicable to the various areas. Each of these is attached to a set of practices, which reflect specific and generic goals [30]. This type of approach is highly successful worldwide amongst enterprises that wish to surpass competitors by providing improved and better products and services.

Given its broad scope coverage, CMMI does not solve the issue due to its non-focus on agile software development processes, which are the area of the current study.

B. Agile Maturity Model

Agile Maturity Model (AMM) was introduced by two researchers in an IT University in Leeds, and it was conceived in order to provide future researchers a more in-depth agile maturity model as a basis for their investigations [31].

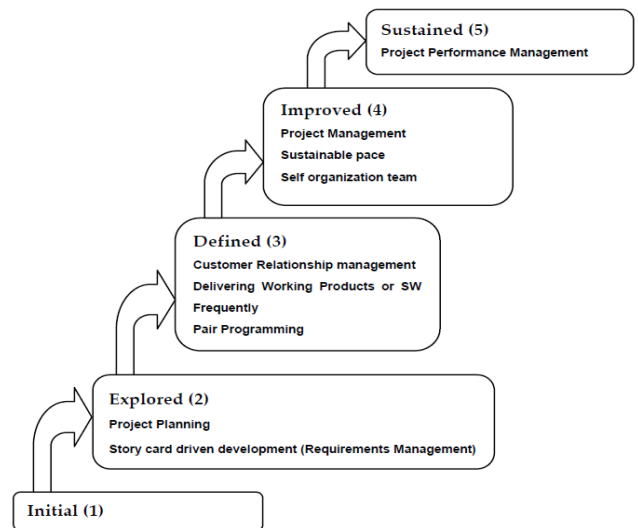


Figure 2. Agile Maturity Model staged representation [31].

This model is shown in Figure 2. and it is somehow inspired by CMMI, since it also has 5 levels, each with a set of goals for their practices: (1) Level 1: Initial – organizations belonging to this level of agile maturity do not have a clearly defined process for agile development and eminent success depends solely on the competence of individuals; (2) Level 2: Explored – it gives particular focus to project planning and requirements engineering for organizations; (3) Level 3: Defined – it stresses the importance of frequent deliveries, pair programming and customer relationship enhancement; (4)

Level 4: Improved – it focus on project management, sustainable velocity of development and self-organizing teams; (5) Level 5: Sustained – underlines the need for the management of projects’ performance, thus continuously improving processes.

The AMM provides a first approach to classifying the maturity of agile development processes, which comprises practices from various agile methodologies. Therefore, it leads us to a continuous research, since this model’s set of practices crosses too many agile methodologies that most organizations do not apply, causing increased levels of entropy.

C. Agile Maturity

Agile Maturity paper appeared as a study case from the British Telecom while developing an IT project [32]. Since it was said that big organizations had increased the barrier for a successful agile adoption, an agile maturity roadmap was presented.

The agile maturity evaluates the agile performance in seven dimensions within five levels of maturity: (1) Level 1 – represents the appearance of software engineering best-practices; (2) Level 2 – best-practices are continuous and improve within small development teams; (3) Level 3 – there is continuous integration within local component teams; (4) Level 4 – there is an incessant integration within global journey teams, i.e., distributed teams, and; (5) Level 5 – on-demand development maturity.

For each of these levels there shall be an evaluation of each the seven existing dimension: (1) Automation of regression tests; (2) Code quality metrics; (3) Automation of deployment; (4) Automation of configurations and best-practices management; (5) Interface integration tests; (6) Test driven development, and; (7) Performance scalability tests.

The combination of these five maturity levels and the seven dimensions allowed British Telecom to incrementally perform a better agile development process. However, this approach is generic and non-focused on the description of these levels and their practices, which leads to one’s need to seek another solution for the major problem stated in Section I.

IV. PROPOSAL

Following the problem focused throughout the last investigation and its various related work, the proposal of a potential solution was introduced in the previous work. Therefore, this section will present the improvements made, the results from the previous proposal through the last two cycles of action research, and propose an optimized roadmap for IT organizations, with renewed validation, so as to develop software with better quality, i.e., more focused on the client role and motivated to self-improvement and market competition.

The Scrum Maturity Model’s main purpose is to aid and guide IT software development organizations and encourage self-improvement, giving special attention to the client’s role, which is mandatory on this fast moving, global and competitive worldwide market. Furthermore, this proposal intends to help organizations that are not familiar with Scrum and wish to implement and adopt it on a staged and incremental approach.

This proposition introduces five levels for Scrum development methodology with its respective goals, objectives, specific and suggested practices. The number of levels is a standard of maturity models; thus making it easier to be measured up with other maturity models for comparison and evaluation purposes.

Next, the main improvements made from the original proposal will be presented. Note that the full details of the proposition contain the complete goals, objectives, practices and suggested metrics for each level of Scrum maturity.

A. Level 1 – Initial

This first and lowest level of maturity, which can be assigned to an organization that uses Scrum, represents the absence of goals for process improvement. The explicit definition of agile development with Scrum methodology does not exist within organizations classified as belonging to this level.

The main issues of the organizations in this level are the frequent over-time and over-budget projects, poor communication among stakeholders and unsatisfactory quality of the final product. These organizations operate on their own and unique way depending on their particular situation which makes their success highly reliant on competent and skilled individuals rather than on standardized and capable teams. In fact, organizations that do not comply with the goal defined for level 2 of Scrum maturity are downgraded to level 1 until further improvements are performed in order to achieve the next level.

B. Level 2 – Managed

In level 2, software development practices appear more structured and complete than in level 1, due to the fulfillment of the two main goals set for this level also shown in Figure 3:

Goals	Objectives	Practices	Suggested Metrics
Basic Scrum Management	Scrum roles exist	(...)	(...)
	Scrum artifacts exist	(...)	(...)
	Scrum meetings occur and are participated	(...)	(...)
	Scrum process flow is respected	(...)	(...)
Software Requirements Engineering	Clear definition of Product Owner	(...)	(...)
	Product Backlog Management	(...)	(...)
	Successful Sprint Planning Meetings	(...)	(...)

Figure 3. Goals and objectives for level 2 of Scrum Maturity.

- Basic Scrum Management – this goal dictates practices that organizations in this level must accomplish, which will ensure the minimum acceptable usage of the Scrum methodology and structure. Note that, although all Scrum roles, objects and meetings must exist in these organizations, those Scrum objects might not be correctly or effectively used, resulting on the need to have further process improvement;

- **Software Requirement Engineering** – this goal comprises a set of practices that the organizations must comply with in order to achieve satisfaction from the final product’s quality created by the vendor organization. Organizations in level 2 usually face fewer problems in the development process than the ones in level 1. However, they still have difficulty in communicating with the client-side representatives and delivering their projects as planned, concerning schedule and budget.

According to the last evaluation of the proposal, this level showed solid goals, objectives, practices and suggested metrics. For this reason, level 2 presented minor changes in the text of the practices, remaining the majority of this level intact.

C. Level 3 – Defined

Level 3 of this maturity model has its major focus on the relationship with clients and on time deliveries. Hence, this level also has two major goals, shown in Figure 4, to guide organizations and improve their processes:

Goals	Objectives	Practices	Suggested Metrics
Customer Relationship Management	Definition of "Done" exists	(...)	(...)
	Product Owner available Successful	(...)	(...)
	Sprint Review Meetings	(...)	(...)
Iteration Management	Sprint Backlog Management	(...)	(...)
	Planned Iterations	(...)	(...)
	Measured Velocity	(...)	(...)

Figure 4. Goals and objectives for level 3 of Scrum Maturity.

- **Customer Relationship Management** – this goal emphasizes the importance of the client and the efforts required to maximize the collaboration with the customer side, even considering the three main difficulties mentioned in Section I. A set of practices are defined and must be satisfied in order to solve the core problem of this investigation.
- **Iteration Management** – this goal is indirectly linked to the previous one, since both contribute to raise customer satisfaction levels. In order to achieve this goal, a set of practices must be fulfilled and implemented so that the organizations always deliver their projects and sprints on time, following their budgets.

With the implementation of level 3 of maturity, an organization can be successful on several projects. However, this success is only partial due to the lack of standardized management, which would guarantee the same quality and performance in all development processes.

Again, the previous work evaluated this level as fairly solid, and only minor changes within the description of the practices were introduced.

D. Level 4 – Quantitatively Managed

In level 4 of Scrum maturity, an organization can boost their achievements by offering standardized and regular software development process aided by the management of the process performance through measurement and analysis practices. In this level of maturity, there are two main fields:

- **Standardized Project Management** – this goal shall lead organizations to use the same development process for all projects and deliver significantly high quality and performance levels. In order to achieve this goal, an organization must complete the standardization of the performed processes;
- **Process Performance Management** – this goal demands the monitoring of all suggested practices up to level 4 of Scrum maturity. These metrics aim to provide enough feedback about actual processes and manage their performance.

Although this level seems very simple in Figure 5, it is actually extremely hard to implement the management and monitor all projects within an organization so as to fulfill all specific practices and maintain the process’ consistency. Note that suggested metrics may be used and organizations are encouraged to customize them to be more appropriate for each enterprise’s culture and best practices.

Goals	Objectives	Practices	Suggested Metrics
Standardized Project Management	Quantitative Project Management	(...)	(...)
Process Performance Management	Measurement and Analysis	(...)	(...)

Figure 5. Goals and objectives for level 4 of Scrum Maturity.

Organizations in this level adopt appealing Scrum development processes and the majority of their projects are successful. The only and last improvement left is optimization of the current processes.

With the previous evaluation process for this proposal it was possible to identify the ambiguity within level 4 for many organizations. In order to clarify it, the demand for “Standardized Projects Management” is now only applied to all agile Scrum projects within the organization, and not to all projects, since in one organization both waterfall development methodologies and agile, in different projects and clients, can coexist.

E. Level 5 – Optimizing

Organizations in level 5 of the Scrum Maturity Model are top class software developers using Scrum methodology. They focus on continuous self-improvement to excel competition and bring higher levels of satisfaction from client, development team and all stakeholders. The only goal for this level is:

- **Performance Management** – this goal allows organizations to measure and analyze their own actions and processes to self-improve.

Organizations in this level have achieved a maximum level and must not discard previous accomplishments and goals by negligence which will block continuous process improvement.

In Figure 6, the four objectives for the main goal of this Scrum maturity are illustrated, being “Causal Analysis Resolution” a newly added objective to this level of Scrum maturity.

Goals	Objectives	Practices	Suggested Metrics
Performance Management	Successful Daily Scrum	(...)	(...)
	Successful Sprint Retrospective	(...)	(...)
	Causal Analysis and Resolution	(...)	(...)
	Positive Indicators	(...)	(...)

Figure 6. Goals and objectives for level 5 of Scrum Maturity.

The main result from the previous work for the definition and first approach experimentation was that the top levels of this Scrum Maturity Model were slightly incomplete and ambiguous. Therefore, the objective “Causal Analysis and Resolution” was included to be used with Daily Scrum and Scrum Retrospective Meeting as to analyze the occurred impediments, differentiate them from incidents and problems, make causal analysis retrospective and then take corrective actions against them.

Note that the whole Scrum maturity model was constantly aligned with similar and renowned best-practices such as CMMI. This decision was based on the purpose of future comparisons with CMMI assessments versus assessments using the proposed model, in order to provided another form of the validation.

Before the results from the practical experimentation of this preposition, note that this Scrum Maturity Model is comprised by its goals, objectives, specific practices and suggested practices for each level. However, due to its size, the complete list of specific practices was not presented. Therefore, only instances from the set were given.

V. RESULTS

In order to evaluate and validate the usefulness and effectiveness of this improved proposal, a third cycle of action research was planned, which included two interviews with Scrum, agile and CMMI experts to validate the concept and details of the proposal as well as six appraisals and audits of Scrum maturity in three different enterprises so as to evaluate its usefulness, efficiency and impact made.

A. Interviews

In order to attain validation of this concept: maturity model for agile Scrum development methodology, a few experts were interviewed.

1) Expert A

Expert A, an international CMMI, Agile and Scrum expert and also partner of an Agile coaching company, granted us two interviews to present our previous proposal and discuss it regarding its viability, usefulness and value created from it.

According to Expert A, the first three levels of Scrum maturity have sufficient detail and acceptable approach. However, although level 4 and 5 have proper goals and objectives, they required some more detail, more specifically, practices to enhance the quality of Scrum Retrospective Meeting are lacking. For instance, practices such as “Question five W’s”, “Identify problems and incidents” and “Build cause-effect diagram to identify problems” would enhance the quality of the inner inspection from retrospective meeting to seek continuous improvement.

Nevertheless, in her feedback, Expert A also stated that the suggested metrics from level 4 of Scrum maturity presents an excellent feature, since not even CMMI presents suggested metrics that exists in COBIT. These suggested metrics allow the monitoring of the current state of the process and discover where to put efforts for improvement, apart from analyzing quantitative statistics from the development process.

About the concept as a whole, Expert A accepts that scattered Scrum loses integrity, however she also agrees that Scrum Maturity Model is not intended to split Scrum into five levels and areas, but rather to provide more emphasis on different areas in each level. Furthermore, it was assured that if this proposal does not become a standard worldwide, it will at least be an extraordinary tool to be used in Scrum Retrospective Meetings as self appraisal and assessment of own maturity.

2) Expert B

Expert B, also an international Agile and Scrum expert as well as a Scrum coach, works for a top five world largest IT company, and conceded us an interview to present to him the actual proposal and discuss about its viability, usefulness and created value . He was pleased with the concept which involves the evaluation of the maturity of the Scrum process, and provided precious feedback for the definition of the practices of each level and within each goal.

Most of the original proposal remains, while merely the definition of the required practices changed, remaining the goals and objectives intact.

B. Appraisals

Another way to validate this theoretical work is to apply it to organizations with strong contact with real business problems. To evaluate the proposal, the following process was adopted:

- Pre-appraisal questionnaire – First, a brief presentation of Scrum Maturity Model concept and its goals on each level will take place. Then, the organization will be asked to fill in the pre-appraisal evaluation form, which will unfold its beliefs about the level in which the organization should and will be classified;
- Appraisal – Later on, if it was never audited before, the appraisal for level 2 of Scrum maturity will begin. If they had obtained successful appraisals before, then the next level of Scrum maturity will be appraised. This process consists in auditing the organizations’ practices against the checklist of the Scrum ones which must be

accomplished in order to obtain the intended level Scrum of maturity.

- Post-appraisal questionnaire – After the appraisal, the assessed organization receives a post-appraisal questionnaire to evaluate the proposal. This phase aims to extract all feedback, both positive and negative, about the proposal and the satisfaction level with appraisal results, comparing it to the initial expectations.

Next, we will present the action taken within three IT development and consulting organizations while auditing the maturity of their development process using Scrum. A number of organizations provided more than one project in progress for the audit process, therefore, in some, more than one project manager was interviewed.

1) Organization X

Organization X, which is focused on cutting waste in software delivery through the practice of lean and agile concepts that they have been implementing for a year now, allowed an audit of their software development process to assess their maturity of Scrum usage.

They are comprised by around seven developers abroad in Ukraine, who assume the Scrum role “Team”, and three project managers in Portugal, that take on the role of “Scrum Master” involved in two or three projects at a time. This enterprise is the excellent example of distributed Scrum, which intends to manage the resources wisely without creating waste and still fulfills the needs of the client, considering the problems from cooperation and distance.

Within the pre-appraisal questionnaire, the organization predicted the possible outcome from the audit as level 2 or 3 of Scrum maturity, since they were aware of the lack of mechanisms to measure and monitor process metrics and formal processes for continuous improvement.

As the appraisal occurred, the organization was confronted with the checklist of the practices which had to be fulfilled in order to achieve the first level of Scrum maturity – level 2. According to the audit, they failed the “Basic Scrum Management” goal by missing the objective of “Scrum meetings occur and are participated”. Actually, they ignored the need of a Scrum Retrospective Meeting and neglected the importance of a formal Daily Scrum Meeting and Scrum Review Meeting.

During the post-appraisal questionnaire, the organization did not show any sight of disappointment and, instead, appeared to be very excited with the results, displaying motivation and critic analysis toward the results and opportunities for future improvements for a better development process. First, they argued that it is very difficult to communicate with clients in this fast moving generation. It was hard to convince the collaboration and their presence at the end of each sprint, which caused them to fail practices such as “Sprint Review Meeting occurs exactly once per Sprint” and “Sprint Review Meeting is attended by Stakeholders, Scrum Master, Product Owner and Team”. They also claimed against the failed “Daily Scrum occurs exactly once per workday”

practice, since the organization affirms there are casing meetings from lean development principles, for the nature of these meetings is different.

Nevertheless, at the end, the organization will rethink these failed practices, and the interviewed project manager planned to immediately launch the implementation of Scrum Retrospective Meeting, since it has great potential benefits that had not yet been considered.

When the interview ended, the interviewee gave the following feedback regarding the Scrum Maturity Model: “This proposal provides a good roadmap for IT organizations by offering goals and objectives per level to evolve and gradually improve, attacking one goal at a time.”; “For higher levels of maturity, it is required much more stability to see the improvements and, although the existence of suggested metrics is brilliant, it lacks how to implement the monitoring mechanism.”. As a final word, the Scrum Master from the organization stated: “Many organizations nowadays declare themselves as agile, but how agile they can be when there are no definitions or rules? The existence of this proposal can surely differentiate the successful agile practitioners from the others.”

2) Organization Y

Organization Y, a fast growing IT consultant enterprise focused on satisfying the market needs through agile and flexible principles, also accepted to be a part of this investigation by providing three of their four project managers to be audited with Scrum Maturity Model.

They are around forty employees, with about thirty in headquarters and ten distributed in two other branches, being one of these branches located abroad, in Vienna. Currently, they employ four project managers and the CEO arranged three meetings with three of them in order to receive some academic research feedback within his company.

a) Project Manager Y1

Project Manager Y1 has been recently promoted to perform the more technical oriented role of project manager. He has a background in the business intelligence field, and now focuses more on the leadership and management of the team of developers for consulting projects.

During the pre-appraisal questionnaire phase, while analyzing the goals required for each level, he determined levels 1 or 2 as a possible result, for he was fully aware that the organization is on the early stage of agile implementation and several goals might not be fulfilled.

As the appraisal for level 2 of Scrum maturity occurred, soon the missing practices was identified. They missed the “Sprint Retrospective Meeting occurs exactly once per Sprint” practice. Unfortunately, this missing feature made this organization fail level 2, although many other practices were accomplished.

Then, within the post-appraisal questionnaire, the project manager agreed with the results, although slightly disappointed with the obtained level. The grounds for this result, he said, was that many unimplemented practices were not given the importance they should have and, although it is possibly very

rewarding, they wanted to focus more on the current client needs without having to worry about overworking their employees. Another explanation is that, given the dimension of his team, so much formality in the development process was not really necessary, as long as the results show up and the clients are satisfied.

For evaluation purposes, it was allowed for the project manager to inspect the next level, which turned out to be another failed appraisal, but this time for level 3, the organization failed the “Sprint Backlog Items are split into tasks” practice when all other practices were accomplished. With this result, the project manager was relieved as he believed that they could achieve up to level 3 of Scrum maturity with a relative small amount of effort, even though it required immense work in employees’ culture to implement them.

To conclude, he agreed that the concept itself has potential to grown into a certification, which will provide more market differentiation. Another interesting point is that it might not be very expensive to concentrate efforts and obtain an acceptable level 3 of Scrum maturity.

b) *Project Manager Y2*

Project Manager Y2 is in charge of four development projects, each of them with only one or two developers located in Vienna focusing on the improvement of applications for smart phones. The main challenges for him are how to coordinate and perform the role of middle man between the client’s needs and developers’ performance with Scrum methodology, since he has less than a year experience with this development methodology.

Within the pre-appraisal questionnaire, given Manager Y2 relative inexperience, the project manager did not have high expectations and pointed out level 2 as a possible outcome.

During the appraisal, they failed many practices such as: “Release Burndown Chart exists”, “Sprint Burndown Chart exists” and “Sprint Retrospective Meeting occurs exactly once per Sprint”.

In the post-appraisal questionnaire phase, the project manager explained that due to the unawareness of the technical capabilities from the project management tools, it was not possible to maintain updated and correct burndown charts. Concerning the missing retrospective meeting, he stated that it is very difficult to have a formal meetings with the distributed team located in Vienna, seriously affecting the performance of this communication.

Again, for evaluation purposes, it was allowed for the interviewee to inspect the fully detailed Scrum Maturity Model, and advanced to the next level’s audit. They did not accomplish practices like: “Definition of ‘Done’ is achieved in each iteration” and “During Sprint Review Meeting Product Owner and other stakeholders provide feedback”.

In the end, the project manager was satisfied to learn more about agile Scrum methodologies, and where he should improve in further projects. He stated that this maturity model might be an important tool to measure their current performance and guide them to continuous improvement.

c) *Project Manager Y3*

Project Manager Y3, a very experienced and enthusiastic Scrum and agile practitioner, is leading the company to implement the backbone for Scrum adoption. It has been almost a year since they started trying to reach this objective, and, at the moment, they are in the final stage. For him, continuous improvement is the core strategy to achieve a competitive advantage. In order to achieve this goal, he leads the implementation and integration of several support systems to aid the development process, since he believes that no agile is solid enough without the required backbone tools. Now, he is in charge of a development project with three developers and a three month length deadline.

The pre-appraisal questionnaire phase revealed that he had high expectations and confidence in their maturity, choosing the level 4 or 5 as the expected result from the appraisal.

When the appraisal began, they succeed to fulfill level 2 practices, and then level 3. No problems were encountered so far. Surprisingly, level 4 was also achieved, because all his previous projects were managed with a standard method and he had a data mining module that defined, monitored and measure their development process and metrics. At the last appraisal for level 5, unfortunately, they failed the practices: “Successful Retrospective Meetings result in concrete improvement proposals” and “Successful Retrospective Meetings’ lessons learned are recorded to a knowledge base”.

Within the post-appraisal questionnaire, the project manager was satisfied with the results, seeing his efforts recognized by external parties and not totally disappointed with the obtained level 4 of Scrum maturity, since they were working on the quality of retrospective meetings now.

His final feedback for this proposal is the following: “This proposal is an excellent tool for deeper insight, to rethink their agile path. Moreover, this preposition motivates the adoption of Scrum by separating several objectives via levels. Agile is easy to learn, however very hard to master. Thus, it is very important for prepositions like these to exist in order to aid organizations to correctly adopt Scrum.”

3) *Organization Z*

Organization Z is a worldwide renowned company that provides technology solutions and services around the world. In their office located in Portugal, they employ around four hundred professionals, delivering both consulting service and software solutions. Their development projects are normally very big involving more than forty people and a twelve-month period per project.

a) *Project Manager Z1*

Project Manager Z1 is the senior software architect and performs team coaching regularly. He worked for a leading and pioneer company using agile methodologies, where he learned a lot about agile best practices from the elite from that generation. Currently, the project he is working with involves forty people, three scrum teams and a year of schedule, and it applies Scrum methodology with this particular client for the first time. They are on the production and deployment phase.

In the pre-appraisal questionnaire assessment, Manager Z1 suggested level 3 as the possible result, since he was aware that the company missed the goals “Measurement and Analysis Management” and “Performance Management”.

During the appraisal for level 2 of Scrum maturity, Manager Z1’s project succeeded to accomplish all practices for level 2, except “Sprint Burndown Chart exists” practice.

In the post-appraisal questionnaire phase, Manager Z1 intensely argued about the need of a sprint burndown chart, which is only used to manage small two weeks sprints and creates waste by joining efforts to manually build such a chart. Note that the organization uses manual means to follow Scrum methodology.

By analyzing the next levels, Manager Z1 felt frustrated again, because he would fail level 3 due to the inexistence of the sprint burndown chart stressed in the goal “Iteration Management”. However, to achieve levels 4 and 5, he agreed that more efforts were needed and that they intend to move further in their question of continuous improvement as a competitive advantage.

As final words, he said: “What I see here is a very interesting approach in agile methodologies study. The roadmap is very good for new enterprises to adopt Scrum and a nice differentiation model for companies in the development industry.”

b) Project Manager Z2

Project Manager Z2 is also a well experienced Scrum practitioner within the organization, and is currently managing a project with four years already, which involves three Scrum teams. This project’s particularity is that the client does not collaborate as closely as the company would wish, so Scrum was only applied as internal communication and work methodology.

In the pre-appraisal questionnaire, after the overview of the maturity mode, he selected level 2 as most likely result of the appraisal.

As the appraisal started, “Sprint Burndown Chart exists” practice was found to be missing just like in the last project manager. Moreover, they did not have “Sprint Review Meeting occurs exactly once per Sprint” practice formally implemented, only some demonstrations once or twice a year. Yet another missing practice was “Sprint Retrospective Meeting occurs exactly once per Sprint”, as according to company’s culture, it only happens right after the Scrum Review Meeting.

During the post-appraisal questionnaire, he commented as the following: “Agile methodologies stress communication a lot. Its qualities are not shown in tiny projects, but in large scale projects in which real problems occur. In these big projects, flexible and constant communication is needed to maximize and optimize the work performed. This proposal presents a staged maturity model to guide Scrum implementation and Scrum performance and usage to differentiate enterprises, which is a magnificent idea.”

VI. EVALUATION

Given the results previously presented, in this section, a critic study for the Scrum Maturity Model will be analyzed and presented.

Regarding the interviews, it was possible for us to realize that the first three levels were well structured, while top levels needed some rework, which is already done. Moreover, it was stated by professionals that the proposition is a very good approach for Scrum adoption, self-inspection and continuous improvement.

This study considered the six performed appraisals in three sample organizations from Portugal, represented by a small, a medium and a large-sized company. Although the average level of maturity is not very high, many of the audited organizations were able to easily reach level 3 by focusing efforts to implement the missing goals, objectives and practices.

The most common missing practices for the first level of Scrum maturity, level 2, were “Sprint Review Meeting occurs exactly once per Sprint” and “Sprint Retrospective Meeting occurs exactly once per Sprint”. In level 3, “Definition of ‘Done’ is achieved in each iteration” is the most commonly failed practice. Top levels were scarcely achievable due to their requirements for mechanisms and concepts for measurement; analysis of process metrics; causal analysis; resolution of problems; and, impediments identified, which were not popular among IT development organizations.

Although many organizations define themselves as agile and Scrum followers, another interesting finding is that many of the basics were not taken into account, and only main and popular values and principles were retained, resulting in these low levels of Scrum maturity.

Through this assessment, it was possible to conclude that the proposal provides a good roadmap for organizations that want to implement Scrum methodology from scratch, align their position for benchmarking purposes or for organizations that want to self-improve.

All feedback collected from both interviews with experts and professionals in the development industry gave us a great deal of confidence and insight to continue our research, refine it and possibly scale its usage and define it as a standard.

VII. CONCLUSION AND FUTURE WORK

In Section I, followed by some discussion and analysis, the main problem was a visible lack of collaboration, in most cases, between vendor organizations and clients as they tried to achieve the development of a successful IT project. This problem is a widely researched topic amongst IT experts, due to its vital importance on the success of software development projects [1][2][3][4][5][6][7][8].

Inspired by the related work and maturity models, the improved proposition, from previous research, with five levels of Scrum maturity presents a roadmap for organizations to implement Scrum methodology and compare the performance of software development process amongst competitors.

The main focus of this paper was the validation phase of the current proposal within cycles of AR, which are comprised

by two interviews with two agile and CMMI experts and six appraisals and post-appraisal assessment. The proposal was evaluated and validated by them, and it is our intention to share our findings with the scientific community. Since this proposition is continuously evolving, the current research shall be repeated until the community agrees on a final iteration and accept it as standard.

We are aware that the evaluation process has limitations, but despite credibility issues regarding this process, the experienced validation phase is worthy to be share with the scientific community, given the interest of the process and its results.

Along with the analysis of the motivation for this research, it was pointed out that further investigation on human factors and on the change of management areas might benefit and enhance the performance of this maturity model. Another interesting research topic would be the classification of the partnership and client maturity, since, as referred to in Section I, clients are usually the major impediment for successful IT projects.

In the end, and we once more stress, the proposition of maturity model is highly polemic within agile community. Nevertheless, the concept Scrum Maturity Model has proved successful as the roadmap for organizations that seek self-improvement and guidance.

ACKNOWLEDGMENT

We would like acknowledge the participation of all experts, organizations and their projects manager for their validation, feedback and experience sharing. A special thanks will go to Microsoft and Tiago Andrade e Silva for their support and guidance throughout this validation process.

REFERENCES

- [1] Group, S.: The Chaos Report 2009 (2007) Retrieved from http://www1.standishgroup.com/newsroom/chaos_2009.php last accessed 25 August 2011
- [2] Kraut, R. E. and Streeter, L. A.: Coordination in software development: *Communications of the ACM*, 38(3), 69-81. ACM (1995)
- [3] Herbsleb, J. D. and Moitra, D.: Global software development: *IEEE Software*, 18(2), 16-20. IEEE (2001)
- [4] Highsmith, J. and Cockburn, A.: Agile software development: the business of innovation: *Computer*, 34(9), 120-127. IEEE (2001)
- [5] Cockburn, A. and Highsmith, J.: Agile software development, the people factor: *Computer*, 34(11), 131-133. IEEE (2001)
- [6] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., et al.: Agile Manifesto (2001) Retrieved from <http://agilemanifesto.org/principles.html> last accessed 25 August 2011
- [7] Leffingwell, D. and Widrig, D.: Managing Software Requirements: A Unified Approach: AddisonWesley Longman Publishing Co Inc Boston MA USA, 491. Addison Wesley (2000)
- [8] Charette, R. N.: Why Software Fails: *Ieee Spectrum*, 42(9), 42-49. IEEE INSTITUTE OF ELECTRICAL AND ELECTRONICS (2005)
- [9] Reel, J. S.: Critical success factors in software projects: *IEEE Software* 16(3), 18-23. IEEE (1999)
- [10] Wilson S.: Failed IT Projects (The Human Factor) (1998) Retrieved from <http://ac-support.europe.umuc.edu/~meinkej/inss690/wilson.htm> last accessed 25 August 2011
- [11] Batra, D.: Modified agile practices for outsourced software projects.: *Communications of the ACM* 52(9), 143. AMCIS (2009)
- [12] Holmström, H., Fitzgerald, B., Ågerfalk, P. J., and Conchúir, E. Ó.: Agile Practices Reduce Distance in Global Software Development: *Information Systems Management*, 23(3), 7-18. Taylor & Francis (2006)
- [13] Norton, D.: The Current State of Agile Method Adoption. *Analysis* (2008) Retrieved from <http://my.gartner.com/portal/server.pt?open=512&objID=260&mode=2&PageID=3460702&resId=837321&ref=QuickSearch&sthkw=agile+methods> last accessed 25 August 2011
- [14] Glazer, H., Dalton, J., Anderson, D., Konrad, M., and Shrum, S.: CMMI® or Agile : Why Not Embrace Both!: Carnegie Mellon University, Software Engineering Institute (2008)
- [15] Laudon, K. C., Laudon, J. P.: *Management Information Systems*: Pearson (2009)
- [16] Yin, A., Figueiredo, S., and Mira da Silva, M.: Scrum Maturity Model: Roadmap for IT organizations to develop software centering on the client role: submitted to 23th International Software & Systems Engineering and their Applications (2011)
- [17] Baskerville, R. L.: Investigating information systems with action research: *October*, 2(October), 1-32. Association for information Systems (1999)
- [18] Fowler, M., and Highsmith, J.: The Agile Manifesto: Software Development, 9(August), 28-35. San Francisco, CA: Miller Freeman, Inc (2001)
- [19] Larman, C. and Basili, V. R.: Iterative and Incremental Development: A Brief History: *Computer*, 36(6), 47-56. IEEE (2003)
- [20] Layman, L., Williams, L., and Cunningham, L.: Motivations and Measurements in an Agile Case Study: *Journal of Systems Architecture* 52(11) 654-667 Elsevier North-Holland, Inc. (2006)
- [21] Chow, T. and Cao, D.: A Survey Study of Critical Success Factors in Agile Software Projects: *Journal of Systems and Software*, 81(16), 961-971. Elsevier Science Inc. (2008)
- [22] Cockburn, A.: *Crystal Clear: A Human-Powered Methodology for Small Teams*. (Series in Agile Software Development). Addison-Wesley Professional (2004)
- [23] Schwaber and K., Beedle, M.: *Agile Software Development with Scrum* : (Series in Agile Software Development). Prentice Hall (2002)
- [24] Beedle, M., Devos, M., Sharon, Y., Schwaber, K., and Sutherland, J.: SCRUM: An Extension Pattern Language for Hyperproductive Software Development: *Pattern Languages of Program Design*, 4, 637-651 (1999)
- [25] Kircher, M., Jain, P., Corsaro, A., and Levine, D.: Distributed eXtreme Programming: *Challenges*, 20-23. XP01 (2001)
- [26] Sutherland, J., Viktorov, A., Blount, J., and Puntikov, N.: Distributed Scrum: Agile Project Management with Outsourced Development Teams. 40th Annual Hawaii International Conference on System Sciences HICSS07 0, 274a-274a. Ieee (2007)
- [27] Braithwaite, K. and Joyce, T.: XP Expanded: Distributed Extreme Programming: *Communication*, 180-188. Springer Berlin / Heidelberg (2005)
- [28] Chrissis, M. B., Konrad, M., and Shrum, S.: *CMMI Guidelines for Process Integration and Product Improvement*: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (2003)
- [29] Menezes, W.: To CMMI or Not to CMMI: Issues to Think About: *CrossTalk The Journal of Defense Software Engineering*, (February 200), 9-11. (2002)
- [30] Development, C.: CMMI® for Development, Version 1.3 CMMI-DEV, V1.3.: *Engineering*. Carnegie Mellon University (2010) Retrieved from <http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm> last accessed 25 August 2011
- [31] Patel, C. and Ramachandran, M.: Agile Maturity Model (AMM): A Software Process Improvement framework for Agile Software Development Practices: *International Journal of Software Engineering*, 2(1), 3-28. Software Engineering Competence Center (2009)
- [32] Benefield, R.: Seven Dimensions of Agile Maturity in the Global Enterprise: A Case Study: *System Sciences HICSS 2010 43rd Hawaii International Conference*, 1-7. IEEE (2010)

Usage of Robot Framework in Automation of Functional Test Regression

Stanislav Stresnjak
Siemens CMT
Osijek, Croatia
e-mail: stanislav.stresnjak@siemens.com

Zeljko Hocenski
Computer and Software Engineering Department
University Josip Juraj Strossmayer in Osijek
Osijek, Croatia
e-mail: zeljko.hocenski@etfos.hr

Abstract — Manual testing is a time consuming process. In addition, regression testing, because of its repetitive nature, is error-prone, so automation is highly desirable. Robot Framework is simple, yet powerful and easily extensible tool which utilizes the keyword driven testing approach. Easy to use tabular syntax enables creating test cases in a uniform way. Ability to create reusable high-level keywords from existing keyword ensures easy extensibility and reusability. Simple library API, for creating customized test libraries in Python or Java, is available, while command line interface and XML based output files ease integration into existing build infrastructure, for example continuous integration systems. All these features ensure that Robot Framework can be quickly used to automate test cases. This paper describes how it is used for automation of existing functional regression test cases within short time and with great success and thus saving costs and enhancing the quality of the software project.

Keywords-software testing; integration testing; regression testing; test automation; robot framework

I. INTRODUCTION

In order to integrate a component within a larger system, three major properties, the fitness, the correctness, and the robustness, have to be tested [1]. The fitness of a component for an application is in general treated as the compatibility of the provided interface of the component and the specification of the required interface of the application. The correctness of a component is its ability to return the correct output when provided with the correct input, while the robustness concerns the absence of a behavior possibly jeopardizing the rest of the system, especially under wrong input. When lot of components is present, integration testing became quite complex and one of the software development improvement steps pertains to testing process improvements which can hardly be done without test automation.

There are various tools for test automation available – commercial and open source, but few are suitable for black box testing (for a black-box testing, see [2]). Many of available tools are most suitable for the unit tests performed by the developers. When it comes to the integration testing or functional verification – not so many tools are available.

Many of the testing tools provided by vendors are very sophisticated and use existing or proprietary coding languages. Effort to automate existing manual tests is similar

to a programmer, using a coding language, writing program in order to automate any other manual process [3].

This paper is organized as follows. Section 2 explains how the tool choosing is done. Section 3 describes why specific tool was chosen. Section 4 describes the implementation of the tool. Section 5 is about benefits of the automation. Section 6 draws conclusions.

II. CHOOSING THE TOOL

What was needed was a tool simple enough to make fast automation and in the same time powerful so these tests can be extended and produce less error prone. The tool should be platform independent. Client tests were run on Linux and Windows and server tests were run on Linux and Solaris. The tool obtained complete platform independence. And the main focus was on regression testing of the integration functional tests. This includes various protocols testing using proprietary protocol simulator as main tool that triggers application logic under test. Although most of the tests were already executed at least once, it became difficult to run regressions, as with end milestone approaching number of test cases began to grow (speaking about few hundreds of the test cases dealing with various scenarios and protocols – CAP [4], TCP [5], SIP [6], LDAP [7], Diameter [8], SOAP [9], SMPP [10], SMTP [11], POP3 [12]) and more important it was rather problematic to check all the logs for errors. When various servers, against which tests were run, were introduced, situation got even more complicated because of their different configuration they had. Not to mention error-prone process because of large number of small actions that should be repeated.

Basic procedure was the same for all test cases – create configuration, start tracing on the platform, run test script, stop tracing on the platform, check script traces, and check platform traces. It was important not to omit generation of report at the end with statistics which could take great amount of time and effort because it is needed to update test cases list, mark those which have failed, make some notes why they failed and for few hundred of test cases – it can take a while.

First idea was to write just a simple shell script that would execute all the tests and analyze the results from log files – but after a while (when it is realized that tests will be required to run with different configurations against different

servers) it is realized that could be benefited from real test framework.

Keyword-driven testing, which enables executing of the test scripts at a higher level of abstraction, was considered to be used as a framework. The idea of keyword driven testing is similar to that of a service or subroutine in programming where the same code may be executed with different values [13], what would make it a perfect choice for the required automation.

III. WHY ROBOT FRAMEWORK

After careful analysis Robot Framework [14] was found to satisfy all needed requirements. It is created in Python which can be implemented on all major platforms. Therefore, multiplatform requirement was completely fulfilled. Among other open source tools, Robot Framework seems to be one of the very few tools, which supports multi platform environment and it is maintained regularly, as it is listed on [15]. The tool is sponsored by Nokia Siemens Networks and released under Apache 2.0 license, meaning it is allowed to be used for free (quite important topic, not only these days).

Robot Framework is a generic, application and technology independent framework. It has a highly modular architecture illustrated in the Figure 1.

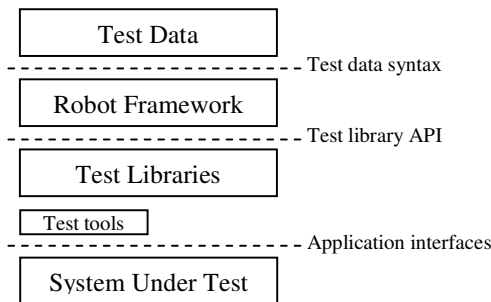


Figure 1. High level architecture [14]

The test data is in simple, easy-to-edit tabular format. When Robot Framework is started, it processes the test data, executes test cases and generates logs and reports. The core framework does not know anything about the target under test, and the interaction with it is handled by test libraries. Libraries can either use application interfaces directly or use lower level test tools as drivers [14].

What was missing was the GUI - for easy test case adding and editing. After considering options, it was decided to use RIDE, which stands for Robot Framework Integrated Development Environment [16]. Its purpose is to be an easy-to-use editor for creating and maintaining test data for Robot Framework. It is still in alpha state, but surprisingly stable for 0.3 version.

Robot Framework is a keyword-driven test automation framework [17]. Test cases are stored in HTML files (in a form of an ordinary HTML tables, as shown in TABLE I.) and make use of keywords implemented in test libraries to drive the software under test, while test suites are created

from files and directories so it's convenient to store into any version of control system.

TABLE I. USING HTML FORMAT

Setting	Value	Value	Value
Library	OperatingSystem		
Library	lib/MyLibrary.py		

Variable	Value	Value	Value
\${MESSAGE}	Hello, World!		

Test case	Action	Argument	Argument
My Test	[Documentation]	Example test	
	[Setup]	Some Setup	
	[Timeout]	5 minutes	
	Log	\${MESSAG E}	
	Check If Directory Exist	/tmp	
	[Teardown]	Some Finish	
Another Test	Should Be Equal	\${MESSAG E}	Hello, World!

Keyword	Action	Argument	Argument
Check If Directory Exist	[Arguments]	\${path}	
	Directory Should Exist	\${path}	

It is possible to create new higher-level keywords by combining and grouping existing keywords together. These keywords are called user keywords to differentiate them from lowest level library keywords that are implemented in test libraries. The syntax for creating user keywords is very close to the syntax for creating test cases, which makes it easy to learn - TABLE I. Rules that should be followed is that keyword names should be descriptive, clean and they should explain what the keyword does, not how it does it.

IV. REAL LIFE EXAMPLE

A. Test suite creation

One way to mitigate mistakes, which arise when new tool usage is started, is to create scripts that will provide immediate pay back [1]. That is, create scripts that won't take too much time to create yet will obviously save manual testing effort and, more important, by creating the scripts you will learn more about the tool's functionality and learn to design even better scripts. Not much is lost if these scripts are thrown away since some value has already been gained from them. Since Robot Framework is based on keywords, and combination of keyword can form a new user keyword - it can be seen as a script.

Robot Framework has some libraries already defined (for example, OperatingSystem, Telnet, String, Collection, etc.), but since it is Python based tool, it is easy to extend it with

libraries written in Python or Java. What is needed is just to write your own function and return some value (if needed).

```
def FTP_Delete(self, host, user, pwd,
file_remote):
    ftp = ftplib.FTP()
    ftp.connect(host, 21)
    try:
        try:
            ftp.login(user, pwd)
            ftp.delete(file_remote);
            return True
        finally:
            ftp.quit()
    except:
        traceback.print_exc()
        return False
```

Figure 2. New library keyword (FTP Delete) definition in Python

Writing and including own library with newly defined keywords it is easy – example for deleting file on FTP server is shown in Figure 2. When using newly defined keywords in the Robot Framework it is only necessary to replace “_” with spaces and new keyword is ready for usage.

RIDE has keyword completion feature that shows the keywords that are found either from the test suite, resource being edited, from its imported resource files or libraries. Also arguments are validated automatically for all known keywords and validation is shown on the grid editor and visualized as different cell backgrounds (everything ok – white background, too many or too few arguments - red background, optional argument - light gray, and if no arguments are allowed then cell background is dark gray). This feature works for built-in and user defined keywords.

Descriptive keywords are one of the Robot Framework features, and with RIDE possibility to create keywords, it is possible to describe test case first and then to actually create keywords and fill them with actions.

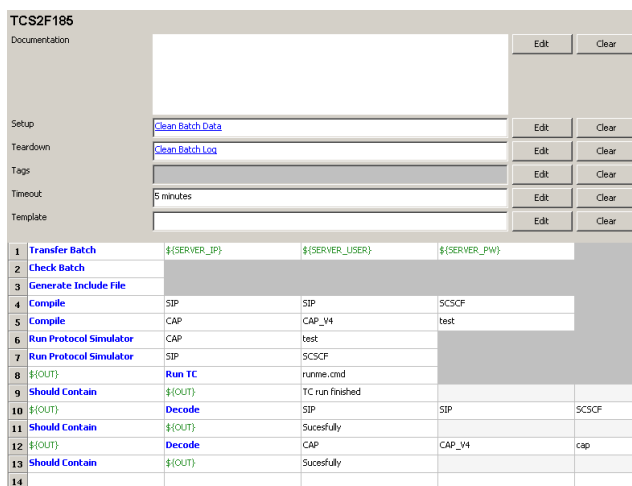


Figure 3. Test case definition in RIDE

Other thing that can happen is to find out that some sequence is needed to be used repeatedly. In that case it is

possible to group that sequence, and define it as new keyword. It is easy task in RIDE - it is just needed to mark the sequence and RIDE will extract those lines and create the new keyword with auto recognition if parameters are needed. After new keyword creation RIDE will replace the sequence and change the test case accordingly.

Keywords and variable definition can be saved into resource file, so it can be used in various suites. It is a good idea if the keyword could be useful also to other tests to move it to shared resource. This way, those keywords can be used later by other tests and duplicate work is avoided.

Usually, there is a need for some setup and cleaning actions – this is also supported and, not only on the test case level, but setup and teardown actions can also be defined on the suite level.

TABLE II. TEST CASE DEFINITION IN HTML FORMAT

Test case	Action	Argument	Argument
TCS2F185	[Setup]	Clean Batch Data	
	[Timeout]	5 minutes	
	Transfer Batch	\$(SERVER_IP }	\$(SERVER_USER }
	Check Batch		
	Generate Include File		
	Compile	CAP	test
	Compile	SIP	SCSF
	Run Protocol Simulator	CAP	Test
	Run Protocol Simulator	SIP	SCSF
	\$(OUT)	Run TC	runme.cmd
	Should Contain	\$(OUT)	TC run finished
	\$(OUT)	Decode	SIP
	...	CAP	
	Should Contain	\$(OUT)	Call finished sucesfully
	[Teardown]	Clean Batch Log	

All this helps to read test cases, even for non technical persons, since we used live language grammar and our test case have execution defined as “Transfer Batch”, “Check Batch”, “Generate Include File”, “Compile”, “Run Protocol Simulator”, “Decode Output”, “Should Contain something” as shown in Figure 3. and in native HTML format in TABLE II.

B. Test case execution

It is possible to execute suite or just some test cases directly from the RIDE GUI, however there is a need to run test cases from the command line so its execution could be easily automated – for example from some continuous integration server. Since Robot Framework is command line tool this is usually done this way. That way various switches

can be used. All possible switches are shown and explained with running tool with “—help” switch. One of many things that can be specified (via test case name pattern matching) is the critical test cases definition. In order to complete the test suite successfully, all critical test cases have to pass.

After executing our test suite HTML report is generated, as shown on Figure 4. and the background color undoubtedly tells whether the whole test suite finished correctly. Critical test cases must be specified with a caution. If critical test cases pass successfully, regardless of other test cases results, the report will be marked as OK. However, statistics will show the number of test cases failed and specify these cases, if any.

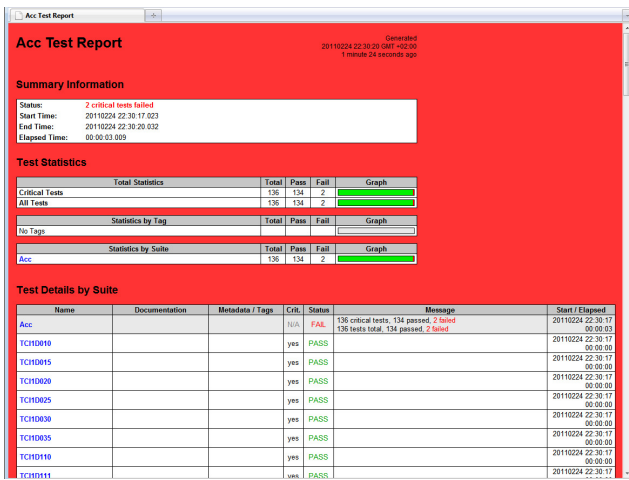


Figure 4. Test case report file

For further manual analysis, there is also detailed log file generated, as shown on I (also configurable with command line switch) with all actions, detailed description of the input and output parameters and keyword output with marked actions that went wrong. There is a keyword “Log” defined, so it is also possible to write additionally whatever need to the log file.

Since all output, as input also, is in the HTML format and already nicely formatted – it is very convenient to use it for reporting.

Robot Framework also generates XML output file which can be used for further analysis. In the source distribution there are interesting tools, for example “risto.py”, used for generating graphs about historical statistics of test executions and “robotdiff.py” tool for generating diff reports from multiple Robot Framework output files.

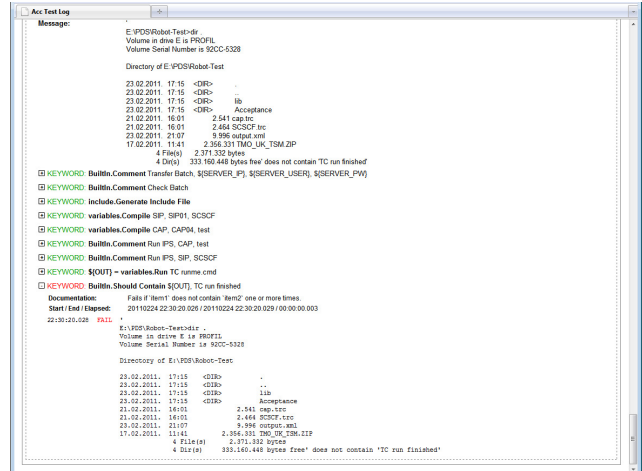


Figure 5. Test case log file

V. BENEFITS OF THE AUTOMATION

An automated test suite can explore the whole product every day. A manual testing effort will take longer to revisit everything. So, the bugs automation does find will tend to be found sooner after the incorrect change was made. Debugging is much faster, which is also meaning – cheaper, when there’s only been a day’s worth of changes. This raises the value of automation.

Automated tests, if written well, can be run in sequence, and the ordering can vary from day to day. This can be an inexpensive way to create something like task-driven tests from a set of feature tests.

Before Robot Framework execution of the test suite took about two days with one person executing test cases sequentially and looking for traces and, most important, being busy all that time. With Robot Framework whole process take only few hours, but only one batch command is needed to run, so person is not busy during test suite execution and can work on other topics, as shown in Table III.

TABLE III. USED TIME COMPARISON

	Time used (in hours)	
	Manual	Automated
Preparation of one test case	8:00	8:00
Execution of one test case	0:02	0:02
Check of one test case	0:05	0:01
Automation of one test case	-	2:00
Report for one test case	0:03	0:00
Total time used for one test case	8:10	10:03
One test run cycle	0:10	0:03

	Time used (in hours)	
	Manual	Automated
For 100 test cases - one suite run	16:40 tester involved	5:00 machine time
20 suite runs	333:20 tester involved	100:00 machine time
20 suite runs with automation time included (suites run time + automation time for all test cases)	333:20	300:00

VI. CONCLUSION

Benefit of working with Robot Framework is that writing test cases follows natural work flow with test case preconditions, action, verification and finally cleanup. Real language is used for keyword description, so it's easy to follow test case – even for non technical person, which, together with its simple usage and easy library extension, make it great tool for test case automation.

Everything is checked automatically and all reports are automatically generated and published on the web pages. This also saved lot of time when decision to introduce continuous integration was made.

The cost of automating a test is best measured by the number of manual tests prevented from running and the bugs it will therefore caused to miss [21], and this is probably the biggest strength of the Robot Framework.

REFERENCES

[1] B. Lei, X. Li, Z. Liu, C. Morisset, and V. Stolz, Robustness Testing for Software Components, Science of Computer Programming, Volume 75 Issue 10, 2010, pp. 879-897

[2] R. Patton, Software Testing, Sams Publishing, 2005

[3] K. Zallar, Practical Experience in Automated Testing, METHODS & TOOLS, Global knowledge source for software development professionals, Volume 8, Spring 2000, pp. 5-9

[4] 3GPP, Customised Applications for Mobile network Enhanced Logic (CAMEL) Phase 4; CAMEL Application Part (CAP) specification (Release 6), TS 29.078 6.3.0, September 2004

[5] RFC: 793, TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION, Information Sciences Institute University of Southern California, September 1981

[6] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, Network Working Group, Request for Comments: 3261, June 2002

[7] M. Wahl, T. Howes, and S. Kille, Network Working Group Request for Comments: 2251, Lightweight Directory Access Protocol (v3), December 1997

[8] H. Hakala, L. Mattila, J-P. Koskinen, M. Stura, and J. Loughney, Network Working Group Request for Comments: 4006, August 2005

[9] E. O'Tuathail and M. Rose, Network Working Group Request for Comments: 3288, Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP), June 2002

[10] SMPP Developers Forum, Short Message Peer to Peer Protocol Specification v3.4 Issue 1.2, October 1999

[11] J.B. Postel, RFC 821 - SIMPLE MAIL TRANSFER PROTOCOL, Information Sciences Institute University of Southern California, August 1982

[12] M. Rose, Network Working Group Request for Comments: 1460, Post Office Protocol - Version 3, June 1993

[13] A.M. Jonassen Hass, Guide to Advanced Software Testing, ARTECH HOUSE INC, 2008

[14] <http://code.google.com/p/robotframework/>, May 2011

[15] [http:// www.opensourcetesting.org/](http://www.opensourcetesting.org/), May 2011

[16] <http://code.google.com/p/robotframework-ride/>, May 2011

[17] P. Laukkanen, Data-Driven and Keyword-Driven Test Automation Frameworks, Master Thesis, HELSINKI UNIVERSITY OF TECHNOLOGY, February 2006

[18] R.W.Rice, Surviving the top ten challenges of software test automation, In Proceedings of the Software Testing, Analysis & Review Conference (STAR) East 2003. Software Quality Engineering, 2003.

[19] W.E.Lewis, Software Testing and Continuous Quality Improvement, AUERBACH PUBLICATIONS, 2005

[20] J.Bach, Test Automation Snake Oil, Windows Technical Journal, pp. 40-44, October 1996.

[21] B.Marick, When Should a Test Be Automated? Proc. 11th Int'l Software/Internet Quality Week, May 1998.

A Test Purpose and Test Case Generation Approach for SOAP Web Services

Sébastien Salva, Issam Rabhi

LIMOS CNRS UMR 6158

PRES Clermont University, Campus des Cézeaux
Aubière, FRANCE

sebastien.salva@u-clermont1.fr, rissam@isima.fr

Abstract—SOA Web services are now supported by most of major software development companies and industry. To be reliable, these ones require to be developed with respect to a complete software development life cycle and, in particular, they need to be tested. Test purpose-based methods are black box testing techniques which take advantage of reducing the time required for test derivation in comparison with exhaustive methods. Nevertheless, test purposes must be constructed manually. This paper proposes some test purpose generation methods for SOAP Web services, modelled by Symbolic Transition Systems (STS). Prior to generate test purposes, we augment the specification with the SOAP environment, to benefit from the messages generated by SOAP processors which give new information about the operations and the faults received. Then, we describe the test case generation from test purposes by synchronizing them with the specification. Test cases are finally translated into XML to be used later by the Soapui tool.

Keywords-Stateful Web services; STS; SOAP; test purpose generation.

I. INTRODUCTION

Web services represent a remarkable branch in the evolution of software development since they offer substantial advantages such as the externalization of Business or social applications available on the Internet, or the reuse of software accompanied by cost reduction. During the recent years, industry has embraced Web services as well-accepted channel for conducting E-Businesses on the Web.

Nevertheless, to ensure that Web services hold their promises, it is crucial that testing activities play an important role in the development process. Indeed, to achieve trustworthy Web services in an environment like the Internet, these ones must also be tested to check various aspects such as robustness, security and conformance which is the topic of this paper. Several testing methods concerning Web services testing have been proposed recently [1], [2], [3]. Some of them, dealing with conformance testing, are said exhaustive i.e., the test case selection is performed to ensure that a faulty implementation is detected by a least one test case. This exhaustiveness often implies to construct test cases for covering all the actions of a specification at least one time. So, the test case generation is often costly and eventually may lead to a state space explosion.

Test purpose-based methods represent an interesting alternative which solve the previous issues and which can be used to test various properties (not the whole system). The test selection is then guided and thereby reduced since test purposes target the test of some implementation parts only. But, although using this approach greatly reduces test costs, the main encountered issue is that test purposes are formulated manually. This task is particularly difficult when the system is large, has real-time constraints or is distributed. Few works propose to solve this issue. For instance, Henniger et al. propose to generate automatically test purposes for distributed systems [4] by considering the specific properties of these latter. None method has been proposed for service-oriented applications though. This is why we present, in this paper, some test purpose generation methods for stateful Web services to test the following specific properties: the operation existence, the critical states and the exception handling. To test them, we augment the specification, modelled with the STS formalism (Symbolic Transition System [5]), with the SOAP environment. Indeed, this one gives more information about the operations and the faults produced by Web services under test. Then, we describe the test case generation. We define a synchronous product which combines the specification and test purposes to produce test cases which can be executed on the implementation and which contain the test purpose properties. Finally, test cases are translated into XML to be executed with the Soapui tool [6], which is dedicated to the functional testing of Web services.

This paper is structured as follows: In Section II, we define both Web service and test purpose modelling. Section II-A provides an overview on some related works about Web service testing. We describe the advantages granted by SOAP for testing in Section III and define the specification completion. Test purpose generation methods are given in Section IV. Section V describes the testing method: we detail the test case generation and the testing framework. Finally, Section VI describes some experimentation results and Section VII concludes with some perspectives.

II. WEB SERVICE AND TEST PURPOSE MODELLING

We formalize, in this paper, Web services with Symbolic Transition Systems (STS [5]). This extended automaton

model associates a behaviour with a specification composed of transitions labelled by actions and of internal and external variables sets, which may be used to send or receive concrete values and to set guards which must be satisfied to fire transitions. Below, we only summarize the STS suspension definition where quiescence (the lack of observation) is taken into account with the δ symbol. The complete definition can be found in [5].

Definition 1 A (suspension) Symbolic Transition System STS is a tuple $\langle L, l_0, V, V_0, I, \Lambda, \rightarrow \rangle$, where:

- L is the finite set of locations, with l_0 the initial one,
- V is the finite set of internal variables, I is the finite set of external or interaction ones. We denote D_v the domain in which a variable v takes values. The internal variables are initialized with the assignment V_0 , which is assumed to take an unique value in D_V ,
- Λ is the finite set of actions, partitioned by $\Lambda = \Lambda^I \cup \Lambda^O$: inputs, beginning with ?, are provided to the system, while outputs (beginning with !) are observed from it. $a(p) \in \Lambda$ is an action where $p = (p_1, \dots, p_k)$ is a finite set of external variables. We denote $type(p) = (t_1, \dots, t_k)$ the type of the variable set p . δ denotes the quiescence i.e., the lack of observation from a location,
- \rightarrow is the finite transition set. A transition $(l_i, l_j, a(p), \varphi, \varrho)$, from the location $l_i \in L$ to $l_j \in L$, also denoted $l_i \xrightarrow{a(p), \varphi, \varrho} l_j$ is labelled by $a(p) \in \Lambda \times I$, $\varphi \subseteq D_V \times D_p$ is a guard which restricts the firing of the transition. Internal variables are updated with the assignment $\varrho : D_V \times D_p \rightarrow D_V$ once the transition is fired.

The STS model is not specifically dedicated to Web services. These latter may be invoked with methods called operations. This is why, for modelling, we assume that an action a in Λ represents either the invocation of an operation op which is denoted $opReq$ or the return of an operation op with $opResp$. For an STS S , we denote $\mathcal{OP}(S)$ the operation set found in Λ . We also assume that service handlers, which may be used to modify SOAP messages, are actions of the specification. An example is illustrated in Figures 1(a) and 2 (solid transitions only). This one describes a part of the Amazon Web Service devoted for e-commerce (AWSEC-commerceService). For sake of simplicity, we consider only two operations, "CartCreate" which creates a virtual cart composed of items, and "CartAdd", which fills the existing cart with new items. Initially, a customer has to create a cart by giving a correct identifier (AWSAccessKeyID), an item identifier and a quantity. If the cart is instantiated (the *CartCreate* operation response is composed of the variable *Isvalid* equal to "true"), this one can be upgraded with the *CartAdd* operation. Note that we do not include all the parameters for readability reasons.

An STS is also associated to an LTS (Labelled Transition System) to define its semantics. Intuitively, the LTS semantics corresponds to a valued automaton without symbolic variables: the states are labelled by internal variable values while transitions are labelled with actions and parameter values. The semantics of an STS $S = \langle L, l_0, V, V_0, I, \Lambda, \rightarrow \rangle$ is expressed by an LTS $\|S\| = \langle Q, q_0, \Sigma, \rightarrow \rangle$.

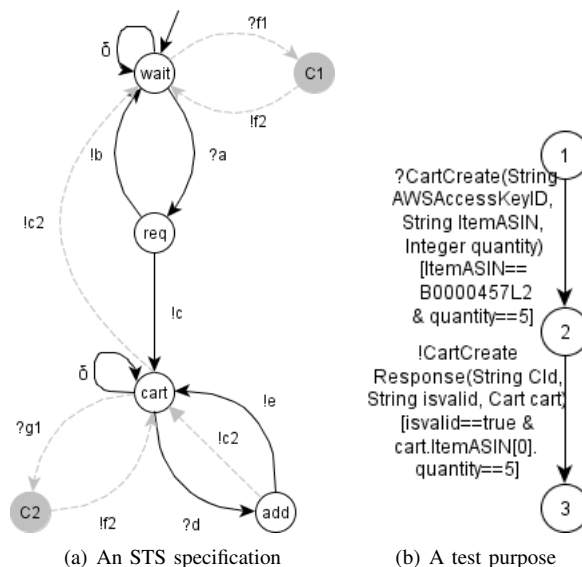


Figure 1.

?a	?CartCreate(String AWSAccessKeyID, String ItemASIN, Integer quantity) id:=AWSAccessKeyID q:=quantity
!b	!CartCreateResponse(String[] errors, String isvalid) [isvalid==false & id<<"ID"]
!c	!CartCreateResponse(String CID, String isvalid, Cart cart) [isvalid==true & id=="ID" & q>0] CartId:=CID
!c2	!(soapfault,cause) [q<=0]
?d	?CartAdd(String AWSAccessKeyID, String ItemASIN, Integer quantity, String CID) [CID==Cartid] id:=AWSAccessKeyID q:=quantity
!e	!CartAddResponse(String isvalid) [isvalid==true & id=="ID" & q>0]
?f1	?CartAdd(String AWSAccessKeyID, String ItemASIN, Integer quantity, String CID)
!f2	!a(p) [a(p)≠(soapfault,"Client") & a(p)≠(soapfault,"the endpoint...")]
?g1	?CartCreate(String AWSAccessKeyID, String ItemASIN, Integer quantity)
?sc1	?CartCreate(String AWSAccessKeyID, String ItemASIN, Integer quantity) [ItemASIN==B0000457L2 & quantity==5] id:=AWSAccessKeyID q:=quantity
!sc2	!CartCreateResponse(String CID, String isvalid, Cart cart) g1=[isvalid==true & id=="ID" & q>0 & cart.ItemASIN[0].quantity==5] CartId:=CID
!sc3	!CartCreateResponse(String CID, String isvalid) g2=[isvalid==true & id=="ID" & q>0 & cart.ItemASIN[0].q ≠ 5] CartId:=CID
!sc4	!CartCreateResponse(String CID, String isvalid) [¬g1 & ¬g2]

Figure 2. Specification symbols

Test purposes describe the test intention. We assume that they are composed of specification properties which

should be met in the implementation under test. Usually, test purposes do not represent complete specification paths. Therefore, they are often synchronized with the specification to generate executable test cases. Consequently, for a specification $S = \langle L, l_0, V, V_0, I, \Lambda, \rightarrow \rangle$, we also formalize a test purpose with a deterministic and acyclic STS $TP = \langle L_{TP}, l_{0TP}, V_{TP}, V_{0TP}, I, \Lambda, \rightarrow_{TP} \rangle$ where internal variables of the test purpose and the specification are exclusive ($V \cap V_{TP} = \emptyset$), \rightarrow_{TP} is composed of transitions modelling specification properties. So, for any transition $l_j \xrightarrow{a(p), \varphi_j, \theta_j} l'_j \in \rightarrow_{TP}$, it exists a transition $l_i \xrightarrow{a(p), \varphi_i, \theta_i} l'_i \in \rightarrow$ and a value set $(x_1, \dots, x_n) \in D_I$ such that $\varphi_j \wedge \varphi_i(x_1, \dots, x_n) \models \text{true}$. A test purpose example for the AWSECommerceService is illustrated in Figure 1(b). This one aims to create a cart composed of five items whose identification number (ASIN) is *B0000457L2*.

A. Related work on Web service testing

Some test purpose-based methods dealing with Web services can be found in literature. These ones propose to transform and adapt an initial specification to be used with existing test purpose-based techniques [2], [3].

These test purpose-based method assume having an existing test purpose set constructed manually. Few works have also proposed automatic test purpose generation techniques: for instance, in [4], some generation techniques are proposed for distributed systems by identifying significant action sequences of the distributed components. From each sequence, a test purpose is generated. To our knowledge, none method has been given for service-oriented applications. However, these ones are composed of specific properties, e.g., SOAP faults, operations with data or exceptions. Therefore, these applications require new test purpose generation techniques which take into consideration these properties. We introduce, in this paper, some of these techniques for stateful Web services, which aim at testing the operation existence, the critical states and the exception handling. Then, we define a synchronous product to achieve the test case selection.

III. THE ADVANTAGES OFFERED BY THE SOAP ENVIRONMENT FOR TESTING

Web services are deployed in specific environments, e.g., SOAP for SOAP Web services, to structure messages in an interoperable manner and to manage operation invocations. In particular, the SOAP environment consists in a SOAP layer which serializes messages with XML and of SOAP receivers (SOAP processor + Web services) [7] which is software, in Web servers, that consumes messages. The SOAP processor is a Web service framework part which represents an intermediary between client applications and Web services and which serializes/deserializes data and calls the corresponding operations.

SOAP processors complete Web service behaviours by adding new messages, called SOAP faults, which give details

about the faults raised in the server side. They return SOAP faults composed of the causes "Client" or "the endpoint reference not found" if services or operations do not exist. SOAP processors also generate SOAP faults when a service instance triggers exceptions. In this case, the fault cause is equal to the exception name. However, exceptions correctly managed in the specification and in the service code (with try...catch blocks) are distinguished from the unhandled ones since a correct exception handling produces SOAP faults composed of the cause *SOAPFaultException* only. So, SOAP faults can also be used to test whether the exception handling is correct by identifying the received causes. Consequently, taking into account these messages while generating test purposes sounds very interesting to check the satisfaction of some service properties. So, we propose to augment the specification with SOAP faults. We denote $(soapfault, cause)$ a SOAP fault where the variable *cause* is the reason of the SOAP fault receipt.

Let $S = \langle L, l_0, V, V_0, I, \Lambda, \rightarrow \rangle$ be a Web service specification. S is completed by means of the algebraic operation *addsoap* in S which completes the specification with SOAP faults as stated previously. The result is a new STS S^\dagger . The operation *addsoap* is defined as follow: *addsoap* in $S =_{def} S^\dagger = \langle L_{S^\dagger}, l_0, V, V_0, I, \Lambda_{S^\dagger}, \rightarrow_{S^\dagger} \rangle$ where L_{S^\dagger} , Λ_{S^\dagger} and \rightarrow_{S^\dagger} are defined by the following rules:

$$\begin{array}{l}
 \bullet \frac{l_1 \xrightarrow{?opReq(p), \varphi, \theta} s l_2, l_1 \xrightarrow{?op'Req(p), \varphi', \theta'} l \notin \rightarrow_S}{l_1 \xrightarrow{?op'Req(p), \emptyset, \emptyset} s^\dagger l', l' \xrightarrow{!a(p), \varphi, \emptyset} s^\dagger l, \varphi = [a(p) \neq l' \notin L_S]} \\
 \frac{}{(soapfault, "CLIENT") \wedge a(p) \neq (soapfault, "the endpoint...")} \\
 \\
 l \xrightarrow{?opReq(p), \varphi, \theta} s l', \varphi' = \bigwedge \neg \varphi_i \\
 \bullet \frac{l' \xrightarrow{!opResp(r_i), \varphi_i, \theta_i} s l'_i}{l' \xrightarrow{!(soapfault, cause), \varphi', \emptyset} s^\dagger l}
 \end{array}$$

The first rule completes the initial specification on the input set by assuming that each unspecified operation request returns a SOAP fault message. The second rule completes the output set by adding, after each transition modelling an operation request, a transition labelled by a SOAP fault. Its guard corresponds to the negation of the guards of transitions modelling responses. A completed specification is illustrated in Figure 1(a) (with solid and dashed transitions). Note that unhandled exceptions are caught by specific exceptions called *unhandledException* in Java or C# and translated later by SOAP faults composed of the *unhandledException* cause. So, unhandled exceptions are supported by our work since we differentiate them from the SOAPFault exceptions thanks to the received SOAP faults.

IV. AUTOMATIC TEST PURPOSE GENERATION METHODS

Although test purposes sound interesting to reduce test costs, these ones also raise an important drawback since

they are usually formulated manually. So, we contribute to solve this issue by introducing some automatic generation techniques for Web services. We propose three test purpose generation approaches which aim at testing the operation existence, the critical locations, and the exception handling.

A. Operation existence testing

This approach generates test purposes for testing whether operations in $\mathcal{OP}(S^\dagger)$, with S^\dagger an STS specification, are implemented and can be invoked. With the completion of the specification, detailed in the previous section, it becomes possible to test the existence of any operation, even those which do not return any response, i.e., any observable reaction. Indeed, if an operation is not implemented as it is described in the specification, the SOAP processor will return a SOAP fault composed either of the cause "Client" or of the cause "the end point reference not found". So, for a specification $S^\dagger = \langle L_{S^\dagger}, l_{0_{S^\dagger}}, V_{S^\dagger}, V_{0_{S^\dagger}}, I_{S^\dagger}, \Lambda_{S^\dagger}, \rightarrow_{S^\dagger} \rangle$, the test purpose set is given by:

$$TP = \bigwedge_{op \in \mathcal{OP}(S^\dagger)} \{tp = \langle L, l_0, V, V_0, I_{S^\dagger}, \Lambda_{S^\dagger}, \rightarrow \rangle \text{ where} \\ \rightarrow = \{l_0 \xrightarrow{?opReq(p), \emptyset, \emptyset} l_1, l_1 \xrightarrow{!a(p), \varphi, \emptyset} l_2, \text{ with } \varphi = \\ [a(p) \neq (soapfault, "Client") \wedge a(p) \neq (soapfault, \\ \text{"the end point reference not found"})]\}$$

The specification of Figure 1(a) is composed of two operations, so we obtain two test purposes. These ones will be synchronized later with the specification to test any operation invocation.

B. Critical location testing

The second technique aims at testing the specification critical locations. This method is especially suitable when the specification locations have a precise meaning. It is not obvious to set which location is critical since no general and formal definition is given in literature. So, in this paper, we suggest that the critical locations are those the most potentially encountered in the acyclic specification paths. Algorithm 1, derived from the DFS (Depth First path Search) one, returns the critical location set CS , from a specification. Then, for each critical location $l \in CS$, we construct test purposes to test all the outgoing transitions of l .

The test purpose set, expressed below, is composed of specification paths finished by output actions to observe the implementation reactions while testing. For a specification $S^\dagger = \langle L_{S^\dagger}, l_{0_{S^\dagger}}, V_{S^\dagger}, V_{0_{S^\dagger}}, I_{S^\dagger}, \Lambda_{S^\dagger}, \rightarrow_{S^\dagger} \rangle$, the test purpose set is given by:

$$TP = \bigwedge_{l \in CS} \{tp = \langle L, l_0, V, V_0, I_{S^\dagger}, \Lambda_{S^\dagger}, \rightarrow \rangle \text{ where } \rightarrow \text{ is} \\ \text{constructed with the following rules:}$$

$$\begin{aligned} & \bullet \frac{l \xrightarrow{!a(p), \varphi, \emptyset}_{S^\dagger} l', a(p) \neq \delta}{l_0 \xrightarrow{!a(p), \varphi, \phi(\emptyset)} l'} \\ & \bullet \frac{l \xrightarrow{?a(p), \varphi, \emptyset}_{S^\dagger} l', p=l' \xrightarrow{a_1(p), \varphi_1, \varphi_1} l'_1 \dots}{l_0 \xrightarrow{?a(p), \varphi, \phi(\emptyset)} l', l' \xrightarrow{a_1(p), \varphi_1, \phi(\emptyset_1)} l'_1 \dots} \\ & \frac{l'_{n-1} \xrightarrow{a_n(p), \varphi_n, \emptyset_n} l'_n, a_n(p) \in \Lambda_{S^\dagger}^O / \{\delta\}}{l'_{n-1} \xrightarrow{a_n(p), \varphi_n, \phi(\emptyset_n)} l'_n} \end{aligned}$$

In both rules, we use a renaming function $\phi : V \rightarrow V'$, $\phi(v) \rightarrow v'$ to obtain exclusive test purpose internal variables. The first rule is used when an outgoing transition, from a critical location, is labelled by an output. In this case, this transition is added to the test purpose. The second rule is used when a transition is labelled by an input. The test purpose is completed with this transition followed by a specification path finished by an output.

Algorithm 2, given below, constructs a test purpose set from one critical location l . For each outgoing transition t of l , if t is labelled by an output action then t is a test purpose (rule R_1). Otherwise, we extract a path p with the *Cover* procedure such that the test purpose $t.p$ is finished by a transition labelled by an output action (rule R_2).

In the specification of Figure 1(a), we have two critical locations *req* and *cart*. For each, one test purpose is constructed, with the previous rules, whose purpose is to test all the outgoing transitions with paths finished by an output action. For instance, for the *req* location, we obtain a straightforward test purpose composed of one transition labelled by !b and another one labelled by !c.

Algorithm 1: Critical location search

```

1 Critical(STS, location);
   input : An STS  $S = \langle L, l_0, V, V_0, I, \Lambda, \rightarrow \rangle$ , the
           initial location  $l_0$ 
   output: A location set  $CL$ 
2  $\forall t \in \rightarrow, label(t) := "UNEXPLORED"$ ;
3 foreach  $t = (l, l_i, a_i, \varphi_i) \in OutgoingTransition(l)$ 
   do
4   if  $Label(t) == "UNEXPLORED"$  then
5      $Label(t) := "VISITED"$ ;
6      $Count(l) := Count(l) + 1$ ;
7      $Critical(S, l_i)$ ;
8   else
9      $Count(l_i) := Count(l_i) + 1$ ;
10  $CL := \{location\ l \mid Count(l) \leq \frac{\sum_{l_i \in L} Count(l_i)}{card(L)}\}$ ;

```

C. Exception handling testing

As described in Section III, SOAP processors return SOAP faults when exceptions are triggered in a Web ser-

Algorithm 2: Test purpose generation dedicated to critical locations

```

1 TPgen(STS,location);
   input : An STS
            $S^\dagger = \langle L_{S^\dagger}, l0_{S^\dagger}, V_{S^\dagger}, V0_{S^\dagger}, I_{S^\dagger}, \Lambda_{S^\dagger}, \rightarrow_{S^\dagger} \rangle$ ,
           a critical location  $l \in L_{S^\dagger}$ 
   output: A test purpose
            $tp = \langle L, l0, V, V0, I_{S^\dagger}, \Lambda_{S^\dagger}, \rightarrow \rangle$ 

2  $\forall t \in \rightarrow_{S^\dagger}, label(t) := \text{"UNEXPLORED"};$ 
3 foreach  $t_i = l \xrightarrow{a(p), \varphi, \varrho} l_i \in \rightarrow_{S^\dagger}$  do
4   if  $a(p) \in \Lambda_{S^\dagger}^O$  then
5      $\rightarrow := \rightarrow \cup \{l0 \xrightarrow{a(p), \varphi, \phi(\varrho)} l_i\};$ 
6   else
7      $p' := \emptyset; Cover(l_i, p');$ 
8      $\rightarrow := \rightarrow \cup \{l0 \xrightarrow{a(p), \varphi, \phi(\varrho)} l_i.p'\};$ 
9 Cover(location  $l$ , path  $p$ );
10 if  $\exists l' \xrightarrow{a(p), \varphi, \varrho} l' \in \rightarrow_{S^\dagger}$  with  $a(p) \in \Lambda_{S^\dagger}^O$  then
11    $p := p.l \xrightarrow{a, \varphi, \varrho} l'; l := null;$ 
12 else
13   foreach  $t_i = l \xrightarrow{a(p), \varphi, \varrho} l_i \in \rightarrow_{S^\dagger}$  labelled by
14     "UNEXPLORED" do
15        $label(t_i) := \text{"VISITED"}; Cover(l_i, p.t_i);$ 
16       if  $l == null$  then
17         break;
18        $label(t_i) := \text{"UNEXPLORED"};$ 
    
```

vice operation at runtime. SOAP processors also enable to differentiate the exceptions resulting of unexpected Web service crashes from those which are thrown in Web service operations (with try...catch blocks for instance). In the last case only, we obtain SOAP faults composed of the "Soap-FaultException" cause.

With the specification completion described in Section III, we can construct test purposes to test whether the exception handling is correctly implemented and not managed by SOAP processors. However, to trigger exceptions, test purposes must be formulated over predefined value sets, that we denote $U(t)$. These ones are composed of unusual values well known for relieving bugs, for any simple or complex type t . For instance, $U(string)$ is composed of the values "&", "\$", null or "_", which usually trigger exceptions. For a specification $S^\dagger = \langle L_{S^\dagger}, l0_{S^\dagger}, V_{S^\dagger}, V0_{S^\dagger}, I_{S^\dagger}, \Lambda_{S^\dagger}, \rightarrow_{S^\dagger} \rangle$, the test purpose set is given by:

$$TP = \bigwedge_{l \xrightarrow{?opReq(p), \varphi, \varrho} l_i \in \rightarrow_{S^\dagger}} \{tp = \langle L, l0, V, V0, I_{S^\dagger}, \Lambda_{S^\dagger}, \rightarrow \rangle\}$$

$\rightarrow \rightarrow$ where $\rightarrow = \{l0 \xrightarrow{?opReq(p), \varphi', \phi(\varrho)} l_1, l_1 \xrightarrow{(!soapfault, "SOAPFaultException"), \emptyset, \emptyset} l_2 \text{ with } \varphi' = \varphi \wedge p = (p_1, \dots, p_n) \in U(type(p_1)) \times \dots \times U(type(p_n))\}$

The specification of Figure 1(a) contains two operation requests. If we suppose that $card(U(type(p_1)) \times \dots \times U(type(p_n))) = n$, we obtain at most $2n$ test purposes. It is observed that the larger the unusual values sets, the larger the test purpose set will be. To limit it, instead of using a cartesian product, we have chosen to use pairwise testing [8] which helps to reduce the coverage of the variable domain by constructing discrete combinations for pair of parameters only.

V. TESTING METHODOLOGY

Each test purpose is synchronized with the specification to produce products, formalized with STSs, which combine the specification behaviour with the test purpose properties. We extract from these synchronous products complete paths (from their initial location until a final one) with an algorithm which performs a reachability analysis to check whether the guards of each path can be satisfied to guarantee its execution. These paths are also completed to express the incorrect (unspecified) behaviour. We obtain test cases ended by locations labelled by *pass*, *fail*, *inconclusive* which represent the test case local verdict. These ones are finally translated into an XML format to be used with the Soapui tool. Each of these steps are described in detail below.

A. Synchronous product

A test purpose represents a test requirement which should be met in the implementation. To test this statement, both the specification and the test purpose are synchronized to produce paths which model test purpose runs with respect to the specification.

Similarly to the specification $S^\dagger = \langle L_{S^\dagger}, l0_{S^\dagger}, V_{S^\dagger}, V0_{S^\dagger}, I_{S^\dagger}, \Lambda_{S^\dagger}, \rightarrow_{S^\dagger} \rangle$, a synchronous product $SP = S^\dagger \times TP$, with $TP = \langle L_{TP}, l0_{TP}, V_{TP}, V0_{TP}, I_{S^\dagger}, \Lambda_{S^\dagger}, \rightarrow_{TP} \rangle$ a test purpose, is defined as an STS $SP = \langle L_{S^\dagger} \times L_{TP}, l0_{S^\dagger} \times l0_{TP}, V_{S^\dagger} \cup V_{TP}, V0_{S^\dagger} \cup V0_{TP}, \Lambda_{S^\dagger}, \rightarrow_{SP} \rangle$, where the transition relation \rightarrow_{SP} is defined with the following rules. The R_2 and R_3 rules perform the product of one specification transition with one test purpose one by synchronizing actions, variables updates and guards. We have written the specific rule R_3 for output actions to add locations labelled by *inconclusive*. This rule yields two transitions: the first one is composed of a guard satisfying both the specification and the test purpose ones ($\varphi_i \wedge \varphi'_i$). The second transition, ended by an *inconclusive* location, is composed of the guard $\varphi_i \wedge \neg \varphi'_i$ which satisfies the specification transition but not the test purpose one. So, reaching such an *inconclusive* location during the tests means that the test purpose transition is not satisfied although the specification is not faulty.

$$\begin{array}{l}
 R_1 : \frac{(l_i, l_{i'}) \in L_{SP}, l_i \xrightarrow{a(p), \varphi_i, e_i} S \uparrow l_{i'} \xrightarrow{b(p), \varphi_{i'}, e_{i'}} T P l_{j'}}{(l_i l_{i'}) \xrightarrow{a(p), \varphi_i, e_i} S P (l_j l_{j'})} \\
 R_2 : \frac{(l_i, l_{i'}) \in L_{SP}, l_i \xrightarrow{?a(p), \varphi_i, e_i} S \uparrow l_{j, l_{i'}} \xrightarrow{?a(p), \varphi_{i'}, e_{i'}} T P l_{j'}}{(\exists x \in D(I_{S \uparrow} \cup V_{S \uparrow} \cup V_{T P}) \varphi_i \wedge \varphi_{i'}(x) = \text{true})} \\
 (l_i l_{i'}) \xrightarrow{?a(p), \varphi_i \wedge \varphi_{i'}, e_i \wedge e_{i'}} S P (l_j l_{j'}) \\
 R_3 : \frac{(l_i, l_{i'}) \in L_{SP}, l_i \xrightarrow{!a(p), \varphi_i, e_i} S \uparrow l_{j, l_{i'}} \xrightarrow{!a(p), \varphi_{i'}, e_{i'}} T P l_{j'}}{(l_i l_{i'}) \xrightarrow{!a(p), \varphi_i \wedge \varphi_{i'}, e_i \wedge e_{i'}} S P (l_j l_{j'})} \\
 \frac{(\exists x \in D(I_{S \uparrow} \cup V_{S \uparrow} \cup V_{T P}) \varphi_i \wedge \varphi_{i'}(x) = \text{true})}{(l_i l_{i'}) \xrightarrow{!a(p), \varphi_i \wedge \neg \varphi_{i'}, e_i \wedge e_{i'}} S P (l_j \text{inconclusive})}
 \end{array}$$

The synchronous product of the test purpose given in Figure 1(b) and the completed specification is depicted in Figure 3. The yellow transition, which reaches an inconclusive location, models a response which does not contradict the specification but does not satisfy the test purpose. Transitions labelled by ?f1 !f2 still belong to the product. They represent a *CartAdd* operation request which may be called before the *CartCreate* operation given in the test purpose.

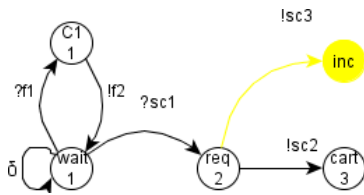


Figure 3. A synchronous product

B. Test case extraction

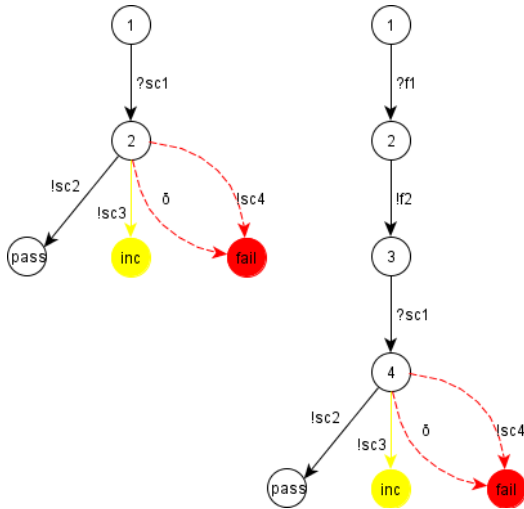


Figure 4. The final test cases

Final test cases are constructed with the following steps from the previous synchronous product *SP*.

- **Synchronous product path extraction with reachability analysis:** acyclic paths are extracted from *SP* with Algorithm 3. This one computes a set *P* of *SP* paths *p*. With the *Cover* subroutine (lines 4-14), it explores *SP* with backtracking and solves the constraints of *p* with the *solving* subroutine to ensure that *p* may be completely executed. *solving* takes a path *p* and returns a variable assignment ϱ_0 which satisfies the complete execution of *p*. If the constraint solvers [9], [10] cannot compute a value set allowing to execute *p*, then *solving* subroutine returns an empty set (lines 19-20). We use the solvers in [9] and [10] which work as external servers that can be called by the test case generation algorithm. The solver [10] manages "String" types, and the solver [9] manages most of the other simple types. In practice, to reduce the time required for solving guards and to prevent from an explosion of values possibilities, we assume that the variable domains are limited by using value sets extracted from database for instance.
- **"pass" verdict addition:** for each path $p \in P$, the final locations not already labelled by "inconclusive" are labelled by "pass" which means that some behaviours modelled by test purposes has been reached,
- **Incorrect behaviour completion:** each path $p \in P$ is completed on the incorrect response set: $\forall l \in L$ such that l has the outgoing transitions $l \xrightarrow{!opResp(r_1), \varphi_1, e_1} l_1, \dots, l \xrightarrow{!opResp(r_k), \varphi_k, e_k} l_k$, we add: (1) $l \xrightarrow{\delta, \emptyset, \emptyset} fail$, (2) $l \xrightarrow{!opResp(r), \varphi, \emptyset} fail$, $\varphi = [\neg(\varphi_1 \vee \dots \vee \varphi_k)]$. (1) δ models the location quiescence i.e., the lack of observation. We suppose that if no response is observed after a defined timeout *Tmax*, then the Web service under test is faulty. (2) If the called operation does not return an expected response, then the implementation does not satisfy both the test purpose and the specification. Thus, a fail verdict is reached too. Note that when an operation is called, we cannot observe the response provided by another operation. So, this case is not considered in this completion.

Final test cases are given in Figure 4. We obtain two acyclic paths from the previous synchronous product enhanced with the possible verdicts.

C. Test execution and verdict

The implementation under test *I* is assumed behaving like an LTS semantics, composed of valued transitions (Section II). We assume that there is no security constraint or firewall between the tester and the implementation, which modifies the SOAP messages and thus the implementation behaviour. To produce a verdict on the test purpose satisfaction, the tester executes each test case by traversing the test case tree: it successively calls an operation with parameters and waits for a response from *I* while following the corresponding

Algorithm 3: Testcase(STS): P;

input : An STS $SP = \langle L_{SP}, l_{0SP}, V_{SP}, V_{0SP}, I_{SP}, S_{SP}, \rightarrow_{SP} \rangle$

output: A set P of STS paths

- 1 $\forall t \in \rightarrow_{SP}, label(t) := "UNEXPLORED";$
- 2 $p := \emptyset;$
- 3 $Cover(l_{0SP}, p, 0);$
- 4 $Cover(\text{location } l, \text{ path } p, \text{ int } n);$
- 5 **if** $\exists (l, l', a, \varrho, \varphi) \in \rightarrow_{SP}$ labelled by "UNEXPLORED"
then
- 6 **foreach** $t_i = (l, l_i, a_i, \varrho_i, \varphi_i) \in \rightarrow_{SP}$ labelled by
 "UNEXPLORED" **do**
- 7 **if** $Solving(p.t_i) \neq \emptyset$ **then**
- 8 $label(t_i) := "VISITED";$
- 9 $Cover(l_i, p.(l(n), l_i(n+1), a_i, \varrho_i, \varphi_i), n +$
10 $1);$
- 10 $label(t_i) := "UNEXPLORED";$
- 11 **else**
- 12 $\varrho := Solving(p);$
- 13 $V_{0p} = \varrho // V_{0p}$ is the variable initialization of $p;$
- 14 $P := P \cup p;$
- 15 $Solving(\text{path } p) : \varrho;$
- 16 $p = (l_0, l_1, a_0, \varphi_0, \varrho_0) \dots (l_k, l_{k+1}, a_k, \varphi_k, \varrho_k);$
- 17 $c = \varphi_0 \wedge \varphi_1(\varrho_0) \wedge \dots \wedge \varphi_k(\varrho_{k-1});$
- 18 $(x_1, \dots, x_n) = solver(c) // solving of the guard c
 composed of the variables (X_1, \dots, X_n) such that
 $c(x_1, \dots, x_n)$ true;$
- 19 **if** $(x_1, \dots, x_n) == \emptyset$ **then**
- 20 $\varrho := \emptyset$
- 21 **else**
- 22 $\varrho := \{X_1 := x_1, \dots, X_n := x_n\}$

branch. When a branch is completely executed, a local verdict is reached. For a test case t , we denote the local verdict $v(t) \in \{\text{pass, inconclusive, fail}\}$. The final verdict is given by:

Definition 2 Let I be a Web service under test, P be a test purpose set and TC be a generated test case set. The verdict of the test over P , denoted $Verdict(I/P)$ is

- *pass*, if for all $t \in TC, v(t) = \text{pass}$. The pass verdict means that test purposes in P are satisfied,
- *inconclusive*, if it exists $t \in TC, v(t) = \text{inconclusive}$ and for all $t \in TC, v(t) \neq \text{fail}$. This verdict means that some test purposes in P are not satisfied although the implementation does not sound faulty,
- *fail*, otherwise. At least on test purpose in P is not satisfied and the implementation is faulty.

VI. EXPERIMENTATION

	Existence	Critical locations	Exception handling
test purposes	22	2	22
test cases	44	22	210
fail verdicts	0	0	39

Figure 5. Test results on the Amazon AWSECommerceService Service

At the moment, we have implemented an incomplete tool which performs the test purpose generation from a completed STS and the synchronous products between the specification and test purposes only. So, test cases are not generated from synchronous products but are extracted manually. To experiment them on real Web services, test cases are extracted and written into the Soapui format. So, these ones can be executed with the Soapui tool [6] which aims to experiment Web services with unit test cases.

We applied this preliminary tool on several Web services to experiment the test purpose generation. Figure 5 describes the results obtained for the Amazon AWSECommerceService (09/10 version) which is a representative sample because it is composed of a large operation set (22 operations) and of many data structures. We limited the test purpose number to 10 per operation for the exception handling method. We obtained fail verdicts only for the exception handling tests. Indeed, we obtained some SOAP faults composed of the cause *Client*, meaning that the requests are incoherent although the test cases satisfy the specification. We also received unspecified messages corresponding to errors composed of a wrong cause. For instance, instead of receiving SOAP faults, we obtained the response "Your request should have at least 1 of the following parameters: *AWSAccessKeyId, SubscriptionId*" when we called the operation *CartAdd* with a quantity equal to "-1", or when we searched for a "Book" type instead of the "book" one, whereas the two parameters *AWSAccessKeyId, SubscriptionId* were right.

In comparison with the other test purpose-based methods for service-oriented applications [2], [3] or tools, our approach takes into account the SOAP environment for testing. This one generates messages which help to conclude if operations exist as it is stated in the specification and which help to identify the exceptions resulting of unexpected Web service crashes from those which are thrown in Web service operations. These features helped to detect the incorrect SOAP faults, composed of the cause *Client* in the previous experimentation. These errors cannot be detected by the previous methods. But the major benefit of this approach concerns the automatic generation of test purposes. Most of the test purpose-based method assume having an existing test purpose set, constructed manually. As stated earlier, this manual construction requires time and is difficult when the system is large.

VII. CONCLUSION

We have presented some automatic test purpose generation methods dedicated to Web services, which aim to test the operation existence, the critical locations, and the exception handling. Then, we have defined a synchronous product of the test purpose with the specification to construct test cases, which are finally translated into XML and then executed by means of SOAPUI.

An immediate line of future work is to take into consideration test purposes describing incorrect behaviours. Such test purposes may be composed of properties which do not belong to the specification. These ones can be used for testing behaviours which should be met in the implementation but also behaviours which should not. With such test purposes, we could propose new generation methods for robustness or security testing.

REFERENCES

- [1] J. García-Fanjul, J. Tuya, and C. de la Riva, "Generating test cases specifications for compositions of web services," in *in Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006)*, A. Bertolino and A. Polini, Eds., Palermo, Sicily, ITALY, June 9th 2006, pp. 83–94.
- [2] M. Lallali, F. Zaidi, A. Cavalli, and I. Hwang, "Automatic timed test case generation for web services composition," in *The 6th IEEE European Conference on Web Services (ECOWS'08)*, I. C. S. Press, Ed., Dublin, November 2008, 53–63.
- [3] T.-D. Cao, P. Felix, and R. Castanet, "Wsof: An automatic testing tool for web services composition," in *Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 7–12. [Online]. Available: <http://dx.doi.org/10.1109/ICIW.2010.9>
- [4] O. Henniger, M. Lu, and H. Ural, "Automatic generation of test purposes for testing distributed systems," in *FATES*, ser. Lecture Notes in Computer Science, A. Petrenko and A. Ulrich, Eds., vol. 2931. Springer, 2003, pp. 178–191.
- [5] L. Frantzen, J. Tretmans, and T. Willemse, "Test Generation Based on Symbolic Specifications," in *Formal Approaches to Software Testing – FATES 2004*, ser. Lecture Notes in Computer Science, J. Grabowski and B. Nielsen, Eds., no. 3395. Springer, 2005, pp. 1–15. [Online]. Available: <http://www.cs.ru.nl/~lf/publications/FTW05.pdf>
- [6] Eviware, "Soapui," 2011, <http://www.soapui.org/>.
- [7] W.-I. organization, "Web services basic profile," 2006, http://www.wsi.org/docs/charters/WSBasic_Profile_Charter2-1.pdf.
- [8] M. B. Cohen, P. B. Gibbons, and W. B. Mugridge, "Constructing test suites for interaction testing," in *Proc. Intl. Conf. on Software Engineering (ICSE)*, 2003, pp. 38–48.
- [9] N. Een and N. Sörensson, "Minisat," 2003, <http://minisat.se>.
- [10] A. Kiezun, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst, "Hampi: a solver for string constraints," in *ISSTA '09: Proceedings of the eighteenth international symposium on Software testing and analysis*. New York, NY, USA: ACM, 2009, pp. 105–116.

Ev-ADA: A Simulation-driven Evaluation Architecture for Advanced Driving-Assistance Systems

Assia Belbachir, Jean-Christophe Smal and Jean-Marc Blossevillle

Laboratoire de Mesure de la Mobilité Coopérative (LEMCO)
 IFSTTAR
 25 allée des marronniers
 78000 Versailles, France
 Email: *firstname.name@ifsttar.fr*

Sébastien Glaser

Laboratoire sur les Interactions Véhicules Infrastructure
 Conducteurs (LIVIC)
 IFSTTAR
 14, route de la Minière
 78000 Versailles, France
 Email: *firstname.name@ifsttar.fr*

Abstract—This paper reports the architecture of a simulator which is able to evaluate sensors, path planners and controllers of the advanced driving-assistance systems (ADAS). The outstanding feature of this simulator is that it is able to evaluate algorithms by giving scores. The implementation of the algorithms requires several tools such as Pro-SiVIC™. To have a good evaluation of the developed algorithms, we give a list in this paper of the requirements for an ADAS simulator. The simulator architecture and the developed algorithms are tested in several ADAS scenarios. Using Pro-SiVIC™ as a simulator, we are now able to evaluate different algorithms for ADAS.

Keywords-Simulation architecture; Pro-SiVIC™; Evaluation; ADAS.

I. INTRODUCTION

Advanced driving-assistance systems (ADAS) received an increasing attention from the car industry recently. To attract industrial attention, pieces of hardware and softwares are developed. However, the software developments cannot work from the first time and can make costly damage. This is why, there is a strong need to ease the development and the validation process of different parts of hardware and software components. In this sense, using computational simulation techniques can be a candidate solution to this problem since it is cheaper in terms of time, money and human resource needed. By generating different types of vehicles, a simulator should be able to evaluate the vehicle’s behavior. Up until now, several simulators are developed. They can be logically divided into two main groups. The first group focuses on simulating only one specific behavior, such as the camera perception or the path planning [1]. The second group simulates all the system’s components behaviour at the same time especially for ADAS [2], [3]. The proposed simulator in this paper belongs to the second group, since the overall aim is to simulate and evaluate different sensors, path planning and control algorithms for ADAS. We use Pro-SiVIC™ [4] and RTMaps [5]. The former is able to generate the design (hardware) of different vehicles (e.g., the wheel’s dimension, the environment, etc.) and the latter is used to implement different perception, path planning and control algorithms. In our case, we want to

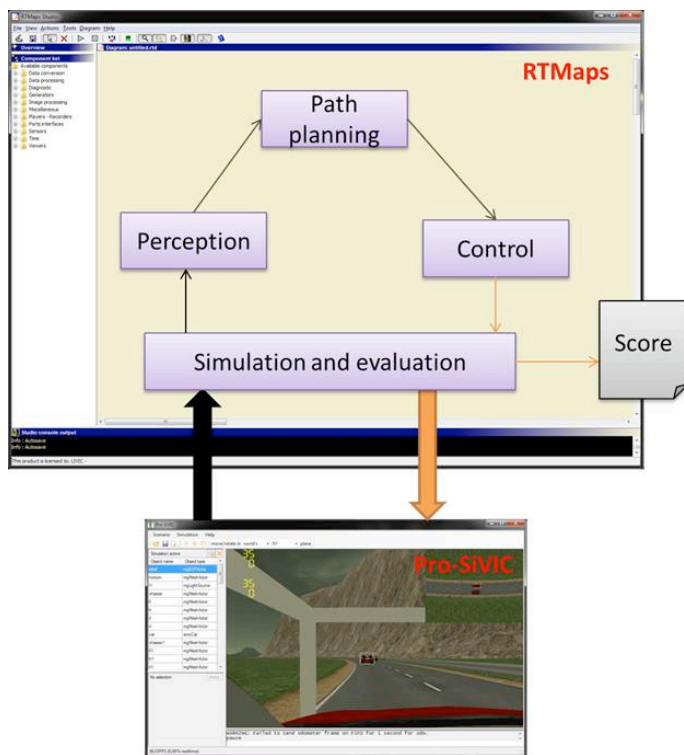


Figure 1. The general architecture to simulate the perception, path planning and control algorithms.

simulate and evaluate algorithms for ADAS. However, there are two questions rising: “What do we need to simulate?” and “How can we evaluate all the behaviors?” As a brief answer to the first question is that we are trying to simulate the perception, the path planning and the control part (see Figure 1). For the second question, the evaluation tests need to be realistic. We define realistic tests such as:

- Definition of different scenarios: the defined scenarios should simulate different road traffic cases, several kinds of road (e.g., motor-way), etc.
- Evaluation requirements: we need to define some criteria to evaluate algorithms. For that we define eight simulator

requirements for the ADAS (see Figure 2) and discuss them in Section III.

The originality of this paper is that, while simulating, we are able to *evaluate* algorithms. We applied our work to a project called: ABV (Automatisation de la conduite à Basse Vitesse sur des itinéraires sécurisés : low speed automation, on a safe trip). This project aims to automatize the driving in low speed (less than 50km/h). The system should be able to advice or take decisions for the safety of the driver and pedestrians/cars on the road. As safety is a main matter, we must evaluate each function of the system, choosing the best option available to realize them.

Our paper is organized as follow: first of all we explain the background and discuss briefly the used tools mainly Pro-SiVIC™. Section III describes our simulator requirements to evaluate different algorithms. Section IV shows the deployed architecture Ev-ADA and some experiments to show the system versatility. We conclude the paper by a discussion and future work.

II. BACKGROUND

Several simulators have already been developed [6] such as MORSE [7], Player/Stage [8] and Gazebo [9]. In general, these simulators are used to imitate the behavior of a robotic system. However, these simulators do not evaluate algorithms by giving scores. Our aim is first of all to simulate the system and secondly to assess how the system is running. We defined several criterion to assess the ADAS system running. We used the simulator Pro-SiVIC™, that is a platform for prototyping sensors. We used ^{RT}Maps to be able to implement the loop of perception–path planning–control by using different algorithms. The coupling between Pro-SiVIC™ and ^{RT}Maps brings to ^{RT}Maps the ability to observe simulated data from Pro-SiVIC™. As follow we explain how can Pro-SiVIC™ and ^{RT}Maps works together.

A. Simulation using Pro-SiVIC™

Pro-SiVIC™ is developed in order to be independent of applications type. To be realistic, Pro-SiVIC™ integrates all functionalities allowing the most realistic possible graphical in the environment. *mg Engine* is the graphical 3D engine used. To reduce the computing board process, *mg Engine* uses a tree of binary positioning (BSP) (for more details see [10]). To ensure its portability under numerous operating systems, this application is developed in C++ under LGPL with OpenGL and SDL libraries. In general several functionalities can be developed such as:

1. Simulated sensors: Several sensors can be simulated such as camera, inertial platform, odometer, telemeter, etc.

*Camera (module *sivicCamera*):* It simulates different sets of camera configured by using the Pro-SiVIC™ parameters or by using the parameters related to OpenGL.

*Inertial Navigation System (module *sivicInertial*):* this module simulates the inertial sensor.

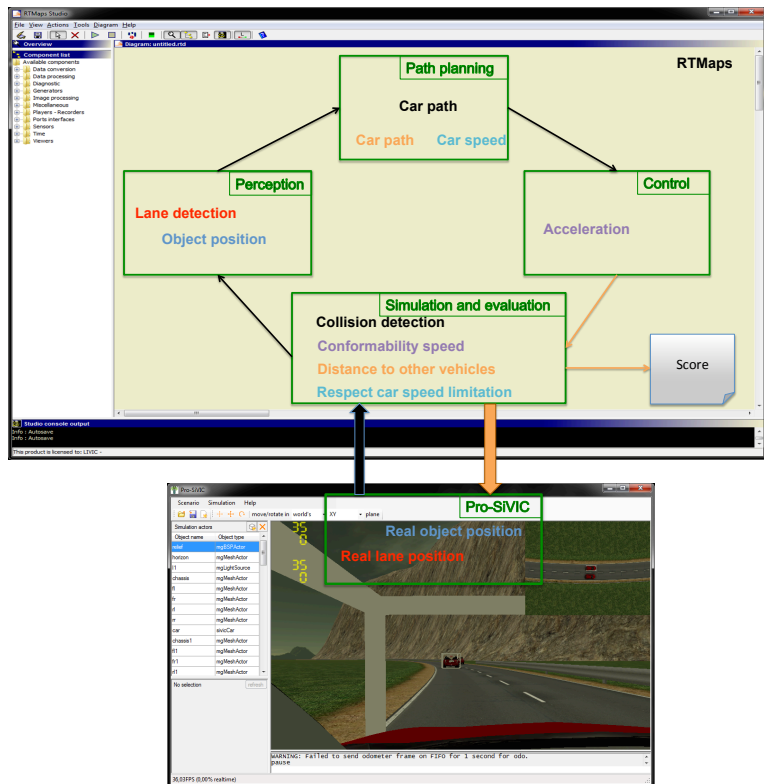


Figure 2. The general architecture to simulate the perception, path planning and control algorithms with different simulator requirements.

Odometer (module *sivicOdometer*): It provides the distance covered by a vehicle.

Telemetric scanner (module *sivicTelemeter*): This module simulates a laser scanner. Depending on the type of the telemeter, several methods can be implemented such as ray tracing or others.

2. Vehicle model: Three axes are defined : Roll, pitch and head. A generic model is able to reproduce the movement of the vehicle taking into account shock absorbers, viscosity and tie adherence [10]. In Pro-SiVIC™ other car models can be implemented and used from external libraries.

3. Mode changes: Several control modes are possible. The vehicle can be internally controlled by Pro-SiVIC™ features or externally controlled as in our case using ^{RT}Maps.

B. Simulation under ^{RT}Maps

We implemented sensors, path planner and lateral, longitudinal controllers under ^{RT}Maps. [11]. The path planner redefines a path when the vehicle trajectory should be changed for example in case of an obstacle in front of the vehicle.

III. SIMULATOR REQUIREMENTS FOR ADAS

ADAS are systems that assist the driver in his driving process. The main objective of these systems is to increase car safety and road safety. Such ADAS systems are adaptive

cruise control (ACC), lane departure avoidance, lane keeping, emergency braking, etc. Every system is specialized in one topic (path, control, etc.). our work has been to define and to group all the required criterion to assess the ADAS simulated components. Eight requirements, explained below, have been selected :

- 1) *Lane detection error*: During these last years, a lot of algorithms were developed for road lane detection. Different types of sensor are used, such as LIDAR, RADAR, Camera [12], etc. Our simulator should be able to compare the real position of the lane with the perceived position lane. In the next subsection we detail how the error is computed ($\delta Lane$).
- 2) *Pedestrian detection error*: A lot of algorithms have been designed to detect pedestrians on the road. The objective of this detection process is to avoid collisions with pedestrian. It is hard to sense, process data and avoid the pedestrian when the car is at high speed. Intensive work has been done on this topic [13], however to ensure the correct pedestrian detection implies that the vehicle speed is limited. For assessing this process part, the error between the simulated pedestrian position and the detected pedestrian position has been computed (δPos_i , where i represent the pedestrian object).
- 3) *Car position detection error*: ADAS perception systems should detect other vehicles or objects in order to avoid collisions. Our assesment process computes the error of the simulated position of the car and the relative estimated position with other vehicles or objects(δPos_i , where i represent the car object).
- 4) *Car localization error*: Some dedicated process (odometer, GPS like etc.) is used to localize the vehicle on the road. A localization error should be computed to evaluate the localization correctness.
- 5) *Path planning error*: A huge number of algorithms have been developed for path planning, originally for robotics applications. These algorithms have as main criterion to avoid collision with existent objects and to reduce the computation time. Our objective is to evaluate the capability of the algorithms to avoid collisions with other objects, at any time ($\delta Collision$).
- 6) *Control/command error*: Algorithms of control allow to control the path execution. In general the speed and the direction of the vehicle are controlled (e.g., Longitudinal and lateral ref. to e -value project). ($\delta Speed$)

- 7) *Driver safety estimation*: It is the main requirement that should be taken into account in all ADAS systems. ADAS should warn the driver in case of high risk or should take the control to prevent accidents. For example, while driving too close to the preceding car, a sound signal can be used to prevent the driver or braking can be triggered($\delta Dist_secur$).
- 8) *Driver comfort estimation*: Even if the comfort cannot be fully evaluated, some criterion should be respected, related to speed changes for example. Next section explains how each requirement is computed and let the simulator evaluates the perception–path planning–control/command loop regarding this comfort criteria ($Accel_{comfort}$).

All these requirements are used to evaluate any ADAS system. In the next section, we explain how we can merge these requirements to define a final score.

IV. A SIMULATION DRIVEN EVALUATION ARCHITECTURE FOR ADAS

To satisfy the aforementioned requirements, developers need a tool that support sensor, path planner and control command specification and development. For this purpose, we used Pro-SiVIC™ with ^{RT}Maps that are fully able to support the algorithm specification and development tasks.

Our simulator is composed by Pro-SiVIC™ and ^{RT}Maps, where we have added a component that assess algorithms by giving scores. The detailed computed scoring process is explained in the following subsections.

A. Pro-SiVIC™ components and the link with ^{RT}Maps

Modeling cars, under Pro-SiVIC™ needs several components. Car description uses observers and sensors. Each car is also described with parameters, the wheel's dimension, the weight, etc. that are estimated from real car measurements. This ability to implement different vehicle shapes make our simulator versatile. The road shape is generated by using PathEdit. The latter is used to generate the vehicle path in a specific road. According to the road description, PathEdit generates a trajectory as a set of position coordinates and speed set points on the road.

The implementation of the environment in Pro-SiVIC™ is easy, the vehicle path being as well easily loaded. The dynamic model of a car is taken into account and can be modified under Pro-SiVIC™. Observers have been implemented to allow ^{RT}Maps to take the vehicle position in the simulated time.

B. Scores

We developed a component called *ABVsim* under ^{RT}Maps from observations (e.g., CarObserver). This component pro-

Structure: Ego structure.

```

1: struct Ego {
    int id; index_lane; number_ego; gear;
    Vehicle_Type type;
    double position_x; position_y; heading_xy;
    position_x_standard_deviation;
    position_y_standard_deviation;
    heading_xy_standard_deviation;
    yaw_rate; speed_x; speed_y;
    acceleration_x; acceleration_y;
    yaw_rate_standard_deviation;
    speed_x_standard_deviation;
    speed_y_standard_deviation;
    acceleration_x_standard_deviation;
    acceleration_y_standard_deviation;
    steering_angle; timestamp;
    Indicators: indicators;
    float revolutions_wheel_rear_right; revolutions_wheel_rear_left;
    revolutions_wheel_front_right; revolutions_wheel_front_left;
    revolutions_motor; weight_empty;
    position_x_rear; position_x_front;
    position_y_rear; position_y_left;
    radius_wheel_rear; radius_wheel_front;
    speed_x_minimum; speed_x_Max;
    acceleration_x_minimum; acceleration_x_Max;
    curvature_Max; ratio_steering_wheel_on_front_wheel;
    Clutch clutch;
    Charge charge;
    Status status;
};
    
```

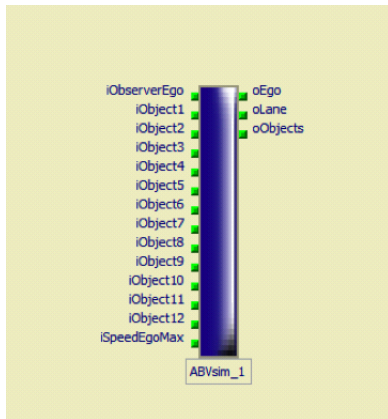


Figure 3. Illustration of different input and output in the ABVsim component.

vides a data structure such as required in the **ABV** project (see Figure 3).

Fifteenth entries are developed described as follow:

- *iObserverEgo* is an entry that observes from Pro-SiVIC™ the position of the car.
- *iObject1, ..., iObject12* are entries to observe other object's position such as pedestrian or cars.
- *iSpeedEgoMax* is the maximal allowed speed of the vehicle.

Three outputs are developed such as:

- *oEgo* is the output of the vehicle position, speed, etc.
- *oLane* is the output of the lane detection using the

appropriate sensor. This structure contains also the error of the lane detection.

- *oObject* is related to the existing objects in the environment such as pedestrian or cars.

Different scores are computed for each set of sensors, path planning, control and safety/comfort algorithms as follow:

$$Score_{sensor} = \delta Lane + (\delta \sum_{i=1}^{12} Pos_i) \quad (1)$$

$$Score_{planning} = \delta Collision \quad (2)$$

$$Score_{control} = \delta Speed + \delta Direction \quad (3)$$

$$Score_{comfort/security} = Accel_{comfort} + \delta Dist_{secur} \quad (4)$$

The sensing score ($Score_{sensor}$) is associated to the lane error detection ($\delta Lane$) and the detection error of other object positions ($\delta \sum_{i=1}^{12} Pos_i$). $\delta Lane$ is a normalized distance between the real lane position and the estimated lane position. The normalized value is between 0 and 1.

The path planning score is related to the collision criterion. If, while running path planning algorithms, the car collides with another object, the $\delta Collision$ is equal to zero.

The controller score $Score_{control}$ represents the compliance with the maximal speed and the direction to be followed. When the vehicle exceed the maximal speed the value $\delta Speed$ is equal to zero. When the car does not follow the road, the $\delta Direction$ value is equal to zero.

The $Score_{comfort/security}$ is the main objective of ADAS systems. $Accel_{comfort}$ represents a score between the maximal acceleration allowed for a vehicle and the actual vehicle acceleration. $\delta Dist_{secur}$ is related to the distance between the vehicle and other vehicles. In general, this distance should corresponds to a car interval of 2 seconds.

All these scores are normalized between 0 and 1. The higher the normalized score value, the better the score is. The normalization procedure for each value is as follows. $\delta Lane$ is normalized by dividing the result by the traffic lane width. Pos_i is normalized by the car/pedestrian dimension. $\delta Speed$ is normalized by the maximal allowed speed. $Accel_{comfort}$ is normalized by the maximal acceleration that the vehicle can drive. $\delta dist_{secur}$ is normalized by the whole driven distance.

V. SIMULATION RESULTS

In our simulation, we are using Windows 7 Professional under Intel(R) Core(TM) i7 CPU 950 @ 3.07GHz, 64 bits.

In this scenario, we run two vehicles. One vehicle is a Mini Cooper and the second vehicle is a Megan Renault. All the algorithms are implemented in the Mini Cooper car

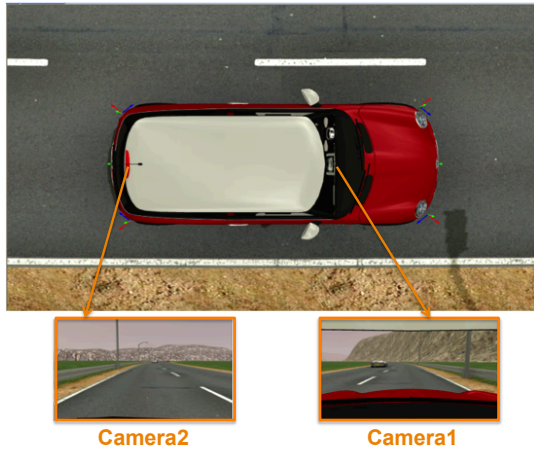


Figure 4. Illustration of different sensors in Ego car.

called Ego. This one follows the second car (Megan Renault) called Car1.

Our path planning algorithm allows the vehicle to follow another vehicle keeping a minimal safety distance with the preceding vehicle (1) and respecting a maximal speed (2).

Ego structure is represented in Figure 4, where two cameras are implemented in the upper front of Ego. A path-planning algorithm is implemented based on Camera1. This camera detects the road surface marking. Ego should follow Car1 at any time and at the same time do not exceed the maximal predefined speed. Several tests are implemented, where we vary the maximal speed and safety distance between Ego and Car1.

All the tests are evaluated in a horse-ring circuit represented in Figure 5.

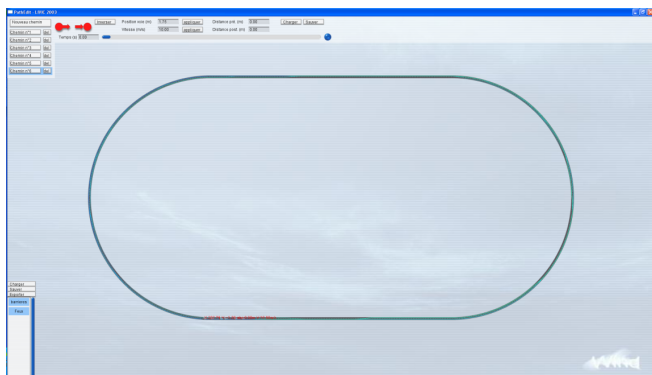


Figure 5. Illustration of a horse-ring circuit for both Ego and Car1.

Different strategies can be used to compute a score. An example of total score utilization is shown as follow:

$$\begin{aligned}
 Score &= \frac{1}{\gamma + \alpha + \beta + \Gamma} \\
 &* [\gamma Score_{sensor} \\
 &+ \alpha Score_{planning} \\
 &+ \beta Score_{control} \\
 &+ \Gamma Score_{comfort/security}] \quad (5)
 \end{aligned}$$

γ, α, β and Γ are coefficients. Depending on the coefficient values, some related parameters can be more important than others.

In our case, the control and security/comfortability are the main part that the system should respect, this is why :

$$\beta + \Gamma > \alpha + \gamma \quad (6)$$

We use a weight of $\beta=1, \Gamma= 2$ and a weight of $\alpha=\gamma=1$.

Case studies of our score		
Speed Ego < Speed Car1	0s	$(0.89 + 0.0 + 0.7 + 0.5) / 4 = 0.52$
	1s	$(0.89 + 1.0 + 0.7 + 0.5) / 4 = 0.77$
	2s	$(0.89 + 1.0 + 0.7 + 0.5) / 4 = 0.77$
	3s	$(0.89 + 1.0 + 0.7 + 0.5) / 4 = 0.77$
	4s	$(0.89 + 1.0 + 0.7 + 0.5) / 4 = 0.77$
Speed Ego > Speed Car1	0s	$(0.89 + 0.0 + 0.7 + 0.5) / 4 = 0.52$
	1s	$(0.89 + 0.0 + 0.7 + 0.5) / 4 = 0.52$
	2s	$(0.89 + 0.0 + 0.7 + 0.6) / 4 = 0.54$
	3s	$(0.89 + 1.0 + 0.7 + 0.7) / 4 = 0.82$
	4s	$(0.89 + 1.0 + 0.7 + 0.7) / 4 = 0.82$

Figure 6. Illustration of the obtained score using the equation 5

Figure 6 represents different case studies of our score. When the maximal Ego speed is less than Car1 ones, the higher score is 0.77. When the maximal Ego speed is greater than Car1 ones, the higher score is 0.82. Due to the low Ego speed, this one can not follow Car1. This difference of score is only related to the speed divergence. This scenario shows that our platform is able to evaluate different implemented algorithms on a simulation mode.

VI. CONCLUSION AND FUTURE WORKS

Our contribution aims at defining an architecture and a framework to evaluate various types of advanced driving-assistance systems (ADAS). In our experiments, an ego car is used to follow another car on a horse-ring road. Each algorithm part (perception, path planning, task control) is evaluated using different types of scores. To extend the ProSiVIC architecture, an evaluator based on proposed criteria has been implemented. These criteria are: (1) Lane detection error, (2) Pedestrian detection error, (3) Car position detection error, (4) Car localization error, (5) Path planning error, (6) Control/command error, (7) Driver safety estimation (8) Driver comfort estimation.

The evaluation of the tested algorithms related to lane detection (based on camera), path planning, control command and comfort/safety of the driver, gives a satisfied score. Our Ev-ADA simulator is now able to evaluate different types of algorithms working on different types of scenarios.

This work opens perspectives. As future works, we plan to evaluate other algorithms in other case studies, varying not just the speed, but also external parameters such as the weather, the traffic, etc. As a matter of fact, the versatility of Pro-SiVIC allows us to evaluate algorithms in various conditions including raining, cloudy, dark weather associated with different car traffic situations.

We will be also able to compare different algorithms between them in the same reproduced conditions. This work will contribute to obtain the best ADAS systems suitable for drivers, safety criteria included. Nevertheless, even if, working with simulation tools reduces works, time and resources, we should recognize that real experimentations will be necessary to take account driver perceptions.

ACKNOWLEDGMENT

The authors would like to thank all the LEMCO team and Dominic Gruyer for allowing us the use of the simulator. We would like to thank also CIVITEC for their daily help.

REFERENCES

- [1] T. Siméon, J. p. Laumond, and F. Lamiroux, "Move3d: a generic platform for path planning," in *4th Int. Symp. on Assembly and Task Planning*, 2001, pp. 25–30.
- [2] M. Parent, "Advanced urban transport: Automation is on the way," *IEEE Intelligent Systems*, vol. 22, pp. 9–11, 2007.
- [3] "Haveit eur. project [online]. available: <http://www.haveit-eu.org/>," Last access date 10/2011.
- [4] N. Hiblot, D. Gruyer, J.-S. Barreiro, and B. Monnier, "Pro-sivic and roads, a software suite for sensors simulation and virtual prototyping of adas," *DSC2010 Driving Simulation Conference*, 2010.
- [5] F. Nashashibi, B. Steux, P. Coulombeau, and C. Lurgeau, "'rtmaps a framework for prototyping automotive multi-sensor applications," *In Proc. of the IEEE Intelligent Vehicles Symposium*, 2000.
- [6] S. Petters, D. Thomas, M. Friedmann, and O. Stryk, "Multilevel testing of control software for teams of autonomous mobile robots," in *Proceedings of the 1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, ser. SIMPAR '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 183–194.
- [7] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular Open Robots Simulation Engine: MORSE," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, 2011.
- [8] B. Gerkey, R. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *11th International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal, 2003. [Online]. Available: citeseer.ist.psu.edu/gerkey03playerstage.html
- [9] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004, pp. 2149–2154.
- [10] M. S. Glaser, "Modélisation et contrôle d'un véhicule en trajectoire limitée : application au développement d'un système d'aide à la conduite," Ph.D. dissertation, Ecole Doctorale Sitevry (Université Evry-Val-D'Esonne), 12 mars 2004.
- [11] S. Glaser, V. Benoit, S. Mammari, D. Gruyer, and L. Nouvelière, "Maneuver-based trajectory planning for highly autonomous vehicles on real road with traffic and driver interaction," *Trans. Intell. Transport. Sys.*, vol. 11, pp. 589–606, September 2010.
- [12] J. McCall and M. Trivedi, "Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 7, no. 1, pp. 20–37, 2006.
- [13] L. Oliveira and U. Nunes, "Context-aware pedestrian detection using lidar," in *In: IEEE Intelligent Vehicles Symposium*, 2010, pp. 773–778.

On the Preliminary Adaptive Random Testing of Aspect-Oriented Programs

Reza Meimandi Parizi, Abdul Azim Abdul Ghani
 Department of Information Systems, University Putra Malaysia,
 43400 Serdang, KL, Malaysia
 {parizi, azim}@fsktm.upm.edu.my

Abstract— Adaptive random testing (ART) is a new family of random-based test data generation and selection strategies that enhances the effectiveness of tests over the classical random testing (RT). ART has been widely investigated and studied in numerous research papers over the recent years. These studies have included proposing various techniques for implementing and improving the intuition behind ART (evenly spread of test cases over the input domain, measured by some distance measures) generally for procedural programs with numerical input domain and most recently object-oriented programs. However, there is currently no work available in the literature that discusses the applicability of ART to aspect-oriented programming (AOP), as it is gaining popularity in software development. Inspired by this, this paper aims to investigate the possible ways that ART can be applied to AOP. This investigation focuses on a multi-perspective analysis of the current ART-based techniques. In this respect, we identified three related perspectives based on the current state of art in the area of ART. Each perspective was analyzed in terms of its applicability and possibility for aspect-oriented programs, particularly its constituent distance measure. As a result, our study gives rise to some interesting points and outlines a number of potential research directions in applying ART to AOP. This can pave the way for efficient development on applying of ART to AOP and finally AOP success.

Keywords—software testing; random testing; adaptive random testing; aspect-oriented programming; aspect testing.

I. INTRODUCTION

Aspect-oriented programming [1],[2],[3] is one of the prominent modularization techniques emerged to cope with the complexity of software development process. To realize the benefits of aspect-oriented programming, the programs developed by this programming paradigm should be effectively tested. The reason is that the aspect-related defects [4],[5], stemmed from the unique characteristics of AOP, can affect the quality of these programs and consequently their general benefits, i.e., enhanced modularity and maintainability.

Software testing as the most widely used practice of ensuring the program's correctness, is useful to help finding these defects (i.e., their presence) and thus to provide a higher level of software quality. However, it has to be said that there is comparatively little work on testing of AOP in the literature and very little on automated testing of AOP such as [6],[7],[8]. This obviously indicates an insufficiency of testing approaches for the aspect-oriented programs at the current time and provides a primary motivation for leveraging the current testing

techniques and/or developing new techniques for these programs.

Adaptive random testing proposed by Chen *et al.* [9] (as a recent derivative of random testing [10]) is an active and interesting research topic, which has shown [11],[12],[13],[14],[15] to have higher fault detection effectiveness compared to classical random testing, with facility of test automation. This is why Jaygarl *et al.* [16] has noted that ART is one of the most effective technique in automated test generation. The essential idea of ART techniques is that the evenly spread random test cases over the whole input domain allows finding faults through fewer test cases than with classical random testing. ART has shown to reduce the number of tests required to reveal the first fault by as much as 50% over classical random testing [17]. Adaptive random testing has seen remarkable progress during the recent past years in order to address the notion of evenly spread of test cases. It seems reasonable to conjecture that ART would continue to be active and become popular among the other random-based testing strategies.

In line with importance of AOP testing and on the other hands its current insufficiency, we believe the idea behind adaptive random testing can be worthwhile and attractive for automated testing of aspect-oriented programs since current research on testing of AOP, especially automated has not been adequately performed and is still in stage of infancy. In order to investigate the applicability of ART to AOP, we identified three perspectives/directions based on scouring the current ART-based techniques in the literature. Corresponding to each perspective and its underlying technique (i.e., distance measure), we analyzed and discussed the feasibility of the given technique to AOP.

As far as we are aware, this is the first attempt made in the literature to discuss the applicability of ART for aspect-oriented programs. In other words, this paper takes some initial steps towards addressing the ART concept for automated test data generation and selection of the aspect-oriented programs. The specific contributions made by the paper are:

- It makes the current vague realization of ART to AOP more understandable by providing thought-provoking perspectives on this matter. Specifically, it gives a theoretical analysis and comparison of three known ART criteria adopted (presented under three identified perspectives) to calculate the distance among different test cases for aspect-oriented programs.
- It analyzes and potentially guides the application of ART in AOP and discusses the potential of using current ART techniques and their results to foster the development of

new testing techniques in area of aspect-oriented software development (AOSD).

The remainder of this paper is organized as follows. Section II provides the background on ART and overviews the current state of the art in this field of research; Section III presents and analyzes the perspectives on adaptive random testing of AOP; Section IV summarizes the results of the analyses; and Section V reports the conclusion and future work.

II. ADAPTIVE RANDOM TESTING (ART)

A. Overview and Classification

Random testing [18],[10],[19] as one of the eldest techniques that include automated test input generation and selection has been studied and applied in different programming paradigms and application domains for decades. The first emergence of the random testing was meant for programs with numerical input domain, however with passage of time and emerging different paradigms the interest in random testing has been substantially increased due to the merits it offers. This matter is evident by various studies in the literature that have extended/applied the RT to the area of their interest.

Random testing is normally referred as the opposite of systematic testing such as functional or structural testing. The techniques in this family, i.e., random-based, can be generally classified into *classical/pure random testing* (the word classical and pure are interchangeability used in this paper) and *enriched random testing* due to the strategies they use for test input generation and selection, see Figure 1.

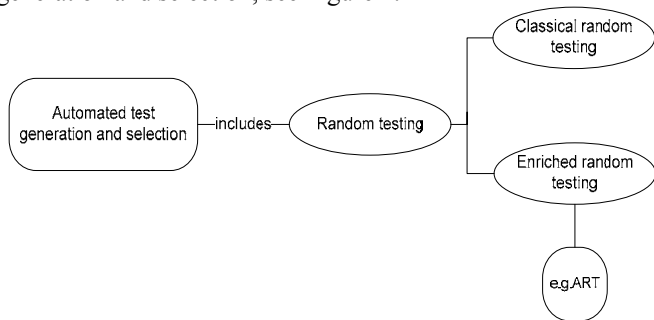


Figure 1. General classification of random testing techniques

By enriched, we mean those strategies that have been equipped with some guidance to their normal random generation process to pick up test inputs that give higher effectiveness in results, in contrast to the classical random testing in which test inputs are only picked at just random. In other words, both classical RT and enriched RT randomly generate test inputs from the input domain, but enriched RT uses additional guidance/criteria to help systematically test case selection rather than randomly selection. Note, in classical random testing test cases are generated by selecting random values of the input variables, which means the generation and selection are not two separated process but rather both imply each other and carried out randomly, see Fig. 2. (Note, in the classical RT, the test generation and test selection processes are the same but in the figure they have been separated for only the purpose of contrasting).

ART [9],[20] is the most dominant family of the enriched RT that suggests a selection criterion of “enforcing the test cases to be evenly spread over the entire input domain”. Spreading evenly the test cases over the input domain is not only the basic idea underlying the ART but also Quasi-Random Testing (QRT) [21] and somewhat the Diversity-Oriented Test Data Generation (DOTG) [22]. These techniques emphasize on the idea of existence a correlation between the fault detection effectiveness and the evenness of the test case distribution in which the more even distribution of the test cases over the input domain the more fault detection capability with fewer test cases is gained.

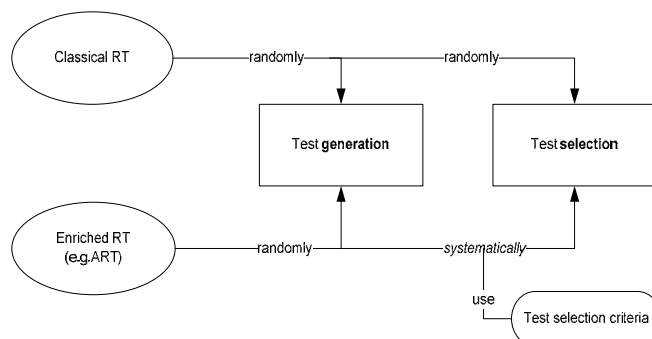


Figure 2. The contrasts between the classical and enriched random testing

In ART has been tried to enhance the fault detection effectiveness of classical RT by imposing some additional criteria on the test inputs selection process. As we mentioned before, the basic intuition of ART technique is that the evenly spread random test cases over the whole input domain allows finding faults through fewer test cases than with purely random testing. In literature several algorithms and variations of the techniques have been proposed to address the “even spread” intuition. The different ART algorithms give different test case selection criteria to ensure an even spread of the test cases. These algorithms attempt to maintain the benefits of random testing while increasing its effectiveness. For instance, one of the test case selection criterions used in one typical ART algorithm called the Fixed Size Candidate Set ART (FSCS-ART) [9] is as follows, which ensures the evenly spread of the test cases by means of a distance measure. The technique defines two test sets: the Executed Set, containing the test data that have been executed, and the Candidate Set, containing a set of randomly selected test data. The Executed Set is initially empty and the first test datum is randomly chosen. The Executed Set is then incrementally updated with the elements selected from the Candidate Set until a fault is revealed. The choice of the test datum from the Candidate Set requires the measurement of the distances of each candidate to all test data in the Executed Set. The chosen candidate is the datum that has the maximal value for the minimal distance among the distances to each test data in the Executed Set (furthest away from the already used inputs).

B. State of the Art in ART

Based on the idea of ART great deals of related algorithms, i.e., various implementation of the idea, have been proposed (distance-based ART, DART [23] was the first ART

algorithm). The different algorithms give different test case selection criteria towards achieving this idea. Some of these algorithms are closely related to the ART, however, with slight changes. Example of these include the Restricted Random Testing (RRT) [24] or Ordinary Random Testing [25], while a plenty of them, as explained below, emphasize on the improvement to ART itself since its emergence [9].

Although ART has shown to be able to improve the fault detection effectiveness of RT, it requires additional computation overhead (considered as main problem associated with ART) to evenly spread test cases [26]. On this regard, a great deal of research has been proposed to minimize the boundary effect [27] and the overhead of primary ART algorithm. Mirror ART (called MART) [28], Fuzzy ART [13], ART by restriction [29], ART by localization [30], ART through dynamic partitioning [31], ART with CG constraints [32] are examples of these improvements which alleviate the pitfalls of the original ART algorithm, especially its overhead.

Further advancement to ART has also been provided by lattice-based ART. Lattice-based ART (L-ART) is a distinctive ART method that generates test cases by systematically placing and then randomly shifting lattice nodes in the input domain. The first introduction of L-ART [33] showed that L-ART is capable of yielding a better fault detection capability than RT, at the same generation cost. However, the test cases of L-ART may be highly concentrated on certain parts of the input domain and cause a *skewed distribution* of test cases. This skewed distribution of test cases can cause a tight coupling between the fault detection capability and the failure region location in the input domain. This means, when failure regions coincidentally reside in the area where L-ART selects a high density of test cases, L-ART may show a better fault detection capability than when failure regions are in the low density area. In reality, however, failure regions can be in any part of the input domain, therefore this dependency of fault detection capability on the failure region location is undesirable.

The issue of skewed test case distributions was addressed in an enhanced version of L-ART presented by Chen *et al.* [34]. The new L-ART not only had a less-skewed test case distribution, but also demonstrated better and more consistent fault detection capability compared to the original L-ART. This superiority of the fault detection capability of the new L-ART has been shown to be better than the results by Restricted ART by random partitioning [35], ART by bisection with restriction [36] and localization [37], ART through iterative partitioning revisited [38] and not revisited [39], ART with enlarged and high dimensional input domains [40], ART with randomly translated failure region [41], ART using Voronoi diagram [42], ART by balancing [43].

Distribution Metric Driven ART [44] has been conducted to measure how evenly an ART algorithm can distribute its test cases according to some distribution metrics such as discrepancy and dispersion, which reflect different aspects of the test case distribution. Discrepancy and dispersion are two commonly used metrics for measuring the equidistribution of sample points. Intuitively, low discrepancy and low dispersion, not in isolation, indicate that sample points are reasonably equidistributed [45] and finally implies an even spread of test cases. These distribution metrics have not only been used to measure and compare the equidistribution of various ART

algorithms but also they have recently been adopted as criteria for the test case selection process aiming at improving the evenness of test case distribution and the fault detection capability of ART [45], [46].

More recently, a new family of ART [47] algorithms, namely adaptive random testing with dynamic non-uniform candidate distribution (ART-DNC) has been proposed. ART-DNC uses a new test profile called *failure driven* instead of uniform distribution or operational profiles used in the original ART algorithm to maximize the effectiveness of fault detection. These new algorithms showed better fault detection capabilities in contrast with the original ART and RT. Moreover, a new ART approach [48] based on the application of an evolutionary search algorithm, called Evolutionary Adaptive Random Testing (EART), was proposed lately.

As could be seen from above, there are so many different growing approaches that address the concept of ART and its further improvements. This matter may raise the question how the results of this work can be related to each other to come up with a completed and optimally effective ART approach. Recently, the work in [49] has taken into account this issue. This work presented a classification, amalgamation of the influential research work related to ART by highlighting the connections, and dependency relationships among the current work in this area.

The review of the current state of the art, as given in this section, shows that none of the presented work has discussed the applicability of adaptive random testing to AOP yet. This has primarily provided the motivation for the research in this paper to address this gap.

III. PERSPECTIVES ON ADAPTIVE RANDOM TESTING OF AOP

In this section, we present and discuss three perspectives on adaptive random testing of AOP. For each of the perspective, the discussion is based on the following:

- Its underlying technique and difference measure it encompasses
- Analysis (i.e., theoretical) of its applicability/feasibility to AOP

A. Overview

It has been generally believed that how evenly an ART technique spreads test cases has an impact on how effectively it detects software failures, and an even distribution of test cases brings a good fault detection capability [11],[12],[13],[14],[15],[50]. However, this matter has only been proven for the numerical and recently objects input types, where there is no evidence on the other complex contexts such as aspect-oriented yet.

In order to be able to apply a typical ART technique (such as FSCS-ART) to a given program the following two issues should be generally figured out [49]:

(1) A strategy to help random sampling from the input domain of the program under test. In other words, this strategy is used to generate random test inputs/data.

(2) A mechanism to compare any two members of the input domain and determine the distance between them to select those test inputs that ensures the evenly spread of the test cases over the input domain. The distance measure should be able to represent the probability of common failure behavior

between two inputs. In other words, the distance measure can be viewed as a difference measure that tries to maximize the diversity of the inputs in which the smaller the distance, the more likely the test cases will show a similar failure behavior. Up to the present time, ART and its all variations in the literature are limited to programs with numeric inputs. On this regards, these studies have calculated the distance between two test cases, i.e., values from input domain, using the Euclidean measure.

Nevertheless, the first issue is common between any pure random testing and adaptive random testing techniques in which a given strategy needs to provide random generation of the test inputs (i.e., random testing). The second issue is meant to be only for ART techniques, i.e., solely unique to the adaptive random testing. It is worth mentioning that the first issue, which is RT, for different programming paradigms/languages and many application domains has been popularly resolved for decades, e.g., [51],[52],[53], [54]. In particular, there have been some recent attempts [55],[56] towards application of random testing to aspect-oriented programs, however the second issue has received lesser attention as the major challenge towards applying the concept of ART to AOP. Therefore, we place emphasis on discussing the second issue as the target objective in this paper.

The main question that we seek to provide insight into it is *how the concept of distance measure can be lifted or applied to aspect-oriented programs*. The answer to this question can consequently help developing adaptive random testing techniques towards automated testing of aspect-oriented programs.

According to the current evidence from literature, there are three perspectives in which this question can provoke discussion in the applying the notion of distance measure (second issue) or more generally ART to AOP. These perspectives are presented and discussed in the following subsections. Furthermore, in our discussion AspectJ [57],[58] is adopted as the target language. The reason is that the AspectJ is the most commonly used aspect-oriented programming language that warrants special attention.

B. Category and Choice-based Perspective

1) *Underlying technique*: This perspective is based on the concepts of categories and choices [59] to which the failure behavior of test cases (i.e., their ability to trigger faults) can be predicated according to the similarity of computation in the executions of them [49]. With regard to this idea, a difference measure (hereafter category and choice distance, CCD) for the category-partition method was first proposed by Kuo [60], who claimed that this measure can be used to help applying ART to a broad range of software input types.

The category-partition method is a specification-based testing approach. In this approach, the parameters and environment conditions that define the behavior of the program under test are first identified, which called as *categories*. Then, for each category, a set of mutual values that possibly triggers similar computation forms the *choices*. The more categories in which two inputs have various choices, the more diversifiable computation they trigger. Therefore, the number of categories containing differing choices is used as predictor of this difference measure, i.e., CCD.

In order to illustrate this difference measure, a simple object recognition system that is capable of distinguishing shapes, sizes and colors is presented as follows (taken from [49]). Suppose that the color of objects can only be light-red, red, deep-red, light-blue, blue, deep-blue, light-green, green and deep-green, and objects are spheres, cubes or pyramids in shape. The size is in the range (0,10] in m^3 . The system behavior depends only on the object shape, the base color (i.e., red, blue or green), and whether the object is larger than $1 m^3$. In this case, three categories can be defined: Color, Shape and Size; three choices for the Color category: red, blue and green; three choices for the Shape category: sphere, cube and pyramid; and two choices for the Size category: large and small. Some choices contain more than one possible value. For example, the red choice has light-red, red and deep-red as its possible values and large has any size more than $1 m^3$. Consider two program inputs (i.e., test cases) T_1 and T_2 , where T_1 is a light-red sphere of size $3.2 m^3$, and T_2 is a deep-blue sphere of size $2.7 m^3$. T_1 has the choices (red), (sphere) and (large) while T_2 has the choices (blue), (sphere) and (large). Therefore, there is only one category, color, in which T_1 and T_2 differ, thus the difference between the two inputs is 1 according to the given distance measure. This is to say that, these two tests are computationally similar as there is not much differences and thus might possibly have a similar failure behavior.

2) *Analysis*: The primary intension of Kuo [60] was to suggest the CCD difference measure as a generic metric for developing ART algorithms of non-numeric input types, but his primary work has not provided any practical example or case study to discuss this matter for modern programs such as object-oriented (OO) or aspect-oriented (AO). Thus, one might think of how this measure could be possibly generalized to these programs with non-numeric input types.

Following the same source of motivation that the CCD difference measure can be possibly applied to a broad range of program input types (as claimed by Kuo [60]), we have here analyzed its feasibility of the application to object- and aspect-oriented programs. To this end, we need to define what would be the categories and choices with respect to these programs and how truly they can represent the essential idea of ART.

In adoption of this measure to the object-oriented programs (as complementary to AOP), categories can be viewed as classes and their associated choices can be considered as instances of those classes, say objects. Therefore, the number of classes containing differing object's values would be a refined definition of the CCD measure for OO programs. Given this, recall the previous example (i.e., recognition system) and test inputs T_1 and T_2 , we now assume this system is an object-oriented application containing three classes: Color, Shape and Size that does the same functionality but implemented in different programming paradigm, e.g., Java. In this case, we define three classes to represent the three categories, Color, Shape and Size respectively. Accordingly, three objects are instantiated to be as choices of the Color category that is red, blue and green. Likewise, three objects for the Shape category: sphere, cube and pyramid; and two objects for the Size category: large and small. According to the definition, there is only one class, color, in which T_1 and T_2

has different object' values, thus the difference between the two inputs is 1.

It can be said that the adaptive random testing of OO programs with respect to this category and choice-based measure (i.e., CCD) is possible to be performed. However, effectiveness of this measure would be another research effort that is worth further investigating.

Concerning the aspect-oriented programs, we now further assume that the recognition system example is an aspect-oriented application written in AspectJ that include the same classes as well as one more feature implemented in one aspect to keep track of the object's movement. The aspect is used to monitor the movement of the recognized objects to refresh the object's display whenever they actually move. Note, tracking movement of object is a crosscutting concern for the system, where it has been implemented as an aspect straightforwardly. If the aforementioned distance measure is chosen to be used for addressing the notion of evenly spread of test cases on this system, the only way to perform the adaptive random testing is to apply the given measure on the base code of the aspect-oriented program (by employing the aforementioned CCD for OO programs). The reason is that the aspects in most of AO languages (including AspectJ) do not have independent identity or existence in the system and cannot be instantiated. This articulates an aspect-related property known as *obliviousness* [61] in which objects, generally base code, are not aware of the aspects in the system. Consequently, such unique properties and characteristics related to AOP perhaps avoid adopting the categories and choices concepts to aspects, generally aspect code. (Typically, a given AO program such as AspectJ is comprised of two parts known as *base code* and *aspect code*. The base code contains all the classes and objects and provides the context execution (join points information) for the aspects. The aspect code contains all the existing aspects in the program and run based upon reaching certain join points in the base code. For more information on this please refer to [58]).

To sum up, we can state that the CCD measure is possible to be applied to adaptive random testing of AOP, however, it will not consider the direct testing of aspect code, specifically the aspect's constructs such as pointcuts and advice (as the focus is more on relationships between the affected/advised classes and aspects, i.e., base code). In this case, the tests mostly stress the integration between aspects and affected classes.

C. Object-based Perspective

1) *Underlying technique*: This perspective was inspired by two recent work on adaptive random testing of object-oriented programs. Since OO programs are considered as complementary parts to AO programs, thus the discussion regarding the prior application of ART to OO would be clearly helpful and connected to the objective of the paper, i.e., investigating the applicability of ART to AOP. Nevertheless, this work has been proposed for object-oriented programs written in Eiffel and Java languages, as briefly presented in the following.

a) *ART for Eiffel*: Ciupa *et al.* [17] propose adaptive random testing for object-oriented programs written in Eiffel, called ARTOO. Their approach initially share the idea of the DART approach [23] to select input objects (considered as test

data/cases) from a testing pool. Since DART for object-oriented programs needs to calculate the distance between two arbitrary objects, accordingly they developed a new distance measure, *object distance* [62],[63] to be applied in adaptive random testing of OO programs. The proposed object distance was made up of the summation of three measure components namely elementary distance (i.e., the distance between the direct values of data types associated with objects), type distance (i.e., the distance between types of objects irrespective of object values), and field distance (i.e., the distance between matching fields of the objects). In addition to these three components, some weights and normalization were incorporated to the calculation process.

ARTOO is capable to automatically specify how to calculate the difference measure, however exponential calculation time, i.e., time complexity, imposed by increasing the dimension of the input domain is a major issue associated with object distance. For instance, checking the distance of integer type values are easier and quicker; however, calculating an object distance takes considerable much longer time (ARTOO takes 160% longer time compared to normal random testing [17]). Recently, in response to this issue, ARTOO has been further enhanced by Jaygarl *et al.* [16] for the purpose of more efficient testing of object-oriented programs. In this work, they suggested a simplified object distance that calculates object distance with lesser time complexity. They divided input data types into three categories— primitive types (including boxed types and a string type), array types, and object types. This separation was able to reduce unnecessary calculation of the ARTOO's object distance.

b) *ART for Java*: Lin *et al.* [64] propose a divergence-oriented technique to adaptive random testing of Java programs. The primary idea of this approach is to provide the program under test with a pool of test data each of which has considerable difference from the others (i.e., high divergence), and then to use the ART technique to select test data from the pool for the program under test. Unlike ARTOO that came up with a well-defined distance measure, this work employed only an intuitive divergence measure that was simply measured as distances of the objects in the pool, without providing any details about what this measure is and how it was calculated. This obviously makes the analysis of this measure's applicability to AOP difficult and therefore, it shall be excluded from the discussion in the analysis section in the following. Nevertheless, from an abstract point of view, since AspectJ is an AO extension of Java, the approach proposed by this work is likely to be applied to AOP, i.e., AspectJ programs. However, prior to that, a clear definition of the used distance measure along with further configurations to consider crosscutting constructs, e.g., advice and pointcuts, into the test generation process would be required.

2) *Analysis*: In the first place, one might think that the unique characteristics of AOP (including obliviousness property) can completely bar the notion of object distance (calculating the distance between two arbitrary objects) from applying to AOP and to some extent makes no sense of it, i.e., constructing difference measure between two arbitrary aspects is not feasible. The reason is that, contrary to the objects in

object-oriented programs, in most of AOP languages such as AspectJ a given aspect does not have independent identity or existence in the system (i.e., the base code has no references to the given aspects) and cannot be instantiated. Note, in some special cases, it is possible to create several instances of a given aspect in AspectJ but by default, a unique instance of an aspect is only created and shared by all the objects when the application is launched. The aspect is then said to be a *singleton* [65].

However, it is important to note that it is just an instinctive misunderstanding. Because, in object-oriented programs (where the object distance was proposed for), the test data/cases to the programs are regarded as objects. Thus, in line with the idea of ART, measuring the distance between two objects would represent the difference between two test cases. Whereas, in the context of aspect-oriented programs it makes no sense to similarly measure the difference between two arbitrary aspects, while it should be between the tests for the aspects not aspects themselves.

Therefore, similar to the first perspective or specifically the category and choice-based measure (i.e., CCD), the object distance measure can only be used in the context of base code of the AO programs towards their adaptive random testing (i.e., the tests that stress the integration between aspects and affected classes). Because, the objects will form the base part of AO programs, i.e., base code.

It is also worth mentioning that, the object distance has an added advantage of requiring less effort compared to the first measure. This is why the object distance was originally developed and well-defined for OO programs, thus unlike the first measure no further effort would be required to leverage the underlying technique to OO programs, prior its application to AOP.

Finally, the explanations on the analysis of the object distance lead us to conjecture that the idea of the ART, using this measure, cannot be currently applied to aspect code of AOP (only base code). Hence, future research might include in-depth investigation of ART notion's applicability to AOP inspired by this measure, of course with a focus on adaptive random testing of aspects, i.e., aspect code. If one can figure out the feasibility or applicability of this matter then a metric model on top of object distance, as next step, will be required. This model should be designed in a way to capture an appropriate distance between arbitrary test cases (not aspects) for a given aspect under test to ensure the evenly spread of test cases (maybe "aspect distance" similar to its corresponding in object-oriented programs, object distance).

D. Coverage-based Perspective

1) *Underlying technique*: This perspective was motivated by some work related to coverage-based test case selection and prioritization [66],[67] in the context of regression testing. This work proposed methods to measure the distance between test cases based on coverage information such as statement and branch coverage, as presented below.

Zhou [66] proposes a metric, called the *Coverage Manhattan Distance (CMD)* as in (1), to measure the difference between any two arbitrary test cases, applicable to adaptive random testing. This measure uses the branch coverage information associated with the test cases. The formal definition of this measure is as follows. Given x as one

test case, and E_x as a vector that records the branch coverage information related to x . The vector is defined to be $E_x = (x_1, x_2, \dots, x_n)$, where $x_i \in \{0, 1\}$ for $1 \leq i \leq n$, and n is the total number of branches in a given program. The value of x_i is set to 1 if and only if the i th branch of the program has been exercised by execution of x ; otherwise x_i is set to 0. Similarly, let y be another test case, and $E_y = (y_1, y_2, \dots, y_n)$ records the branch coverage information of y . The Coverage Manhattan Distance (CMD) between x and y is captured by:

$$CMD(x, y) = \sum_{i=1}^n |x_i - y_i| \quad (1)$$

Similar to the work by Zhou, Jiang *et al.* [67] suggested a distance measure based on the *Jaccard distance* of the two sets to be used as measured distance between two test cases. The Jaccard distance between two test cases x and y is defined as: $D(x, y) = 1 - |A \cap B| / |A \cup B|$, where A and B are the sets of the coverage of elements such as statements or branches exercised by x and y , respectively.

Empty-intersection set is a problem associated with Jaccard measure. That is, whenever the intersection between set A and B is empty the Jaccard measure just returns the maximum value of 1. This problem can result in capturing the distance between the test cases in a wrong way and consequently misguide the ART algorithm in picking the test case candidates (see [66] for example on this problem). However, this is not the case with CMD measure, whereas it is capable of yielding result that is more effective. This superiority led us to put emphasis on the CMD measure in the analysis of its capability to AOP, in the next sub-section.

2) *Analysis*: The two preceding measures, i.e., category and choice-based and object distance, focus on the input values (according to the program's input domain/space) as their sources of measurements. This dependency on input values makes these measures to be only applicable to certain types of programs (or at least more suited to some). On the contrary, CMD measure relies on a totally different source, which is independent of the input values. In our view, this measure is promising as it has the advantage (i.e., by using coverage information) that enables ART to be applied to a border range of programs with lesser limitations. In addition, the coverage fulfillment has been the most analyzed and required test criterion through the testing studies, which CMD has also taken into account.

In adoption of this measure to AOP, towards the ART, there can be two interesting ways of further exploration:

First, we suggest including the *aspectual branch coverage* [8] instead of the traditional branch coverage in the original CMD measure to record the required coverage information. Aspectual branch coverage is a coverage metric that captures the aspectual behavior, specifically the branch coverage within the aspect code (i.e., including branches from predicates in advice and methods in aspects). This metric has been previously used to guide the test generation in area of AOP testing [8],[6]. As a result, the selection of the test cases according to this adopted CMD measure (one may call it *Aspectual Coverage Manhattan Distance, ACMD*) would be based on test cases that are able to cover new aspectual

branches that have not been covered by the previous executed test cases.

In order to make the point clear, a simple example showing the applicability of the coverage Manhattan distance to an aspect code is presented below. Given the aspect `ODRuleAspect` shown in Figure 2 (adapted from AspectJ examples by Laddad [58]):

```
public aspect ODRuleAspect
    pointcut debitExecution(Account account, float
        withdrawalAmount) : execution(void
        Account.debit*(float) && this(account) &&
        args(withdrawalAmount);
    before(Account account, float withdrawalAmount)
    : debitExecution(account, withdrawalAmount) {
        Customer customer = account.getCustomer();
        if (customer == null) return;
        if (account.getAvailableBalance() >
            withdrawalAmount) {
            float deductedAmount =
                account.getAvailableBalance() -
                withdrawalAmount;
            ...
        } else System.out.println("not enough
            money!");
        }
        ...
    }

public class Account {
    private float balance;
    private int accountNumber;
    private Customer customer;
    public Account(int accountNumber, Customer
        customer) { ... }
    public void debit(float amount) { ... }
    ...
}
```

Figure 2. An AspectJ example

In this case, there are two predicates (surrounded by a red box in Figure 2) which result in four aspectual branches in the given aspect, that is $n=4$. Suppose x and y are two test cases, where each of which contains a different instance of `Account` class, say $Ac1$ and $Ac2$ respectively. In addition, two calls to `debit` method (plus two parameter values for method's calls) on these instances are required to trigger the execution of the advice. Thus, for instance $Ac1.debit(95.60)$ and $Ac2.debit(64.35)$ would form the test cases x and y respectively. Assume, $Ac1.getCustomer$ will return `null`, in this case x would be able to exercise only one branch, i.e., `customer == null`, hence $E_x = (1,0,0,0)$. Similarly assume, $Ac2.getCustomer$ has not returned `null` and its `Ac2.getAvailableBalance` is 120 (which is higher than 64.35). Thus, the test case y is able to exercise two branches, i.e., `customer != null` and `(account.getAvailableBalance() > withdrawalAmount)`, so $E_y = (1,1,0,0)$. Now, recall the metric in (1) the difference measured between these two cases would be of 1.

Alternatively, in order to obtain the proper coverage information to make use of the CMD measure in ART of AOP, we suggest employing the program's control flow graph of aspect-oriented programs. For this purpose, *aspect-oriented control flow graph* (AOCFG) proposed by Parizi *et.al* [68] (or other similar approaches such as [69]) would be a capable choice to help testers gain coverage-related information. This type of structural modeling and graph embodiment of aspects not only allows obtaining information related to the branch coverage but also a variety of coverage elements such as node, edge, etc. However, further research needs to be done to study the usefulness of these types of coverage information for ART, including coverage of elements in graphs/models used in aspect-oriented modeling.

In summary, the above analysis demonstrates that it is possible to construct more meaningful distance measure (using the idea of coverage information) in compared with the other presented measures for adaptive random testing of aspect-oriented programs. However, it still requires conducting further research to produce a well-suited coverage-based ART technique for aspect-oriented programs and then to proof the effectiveness of the produced technique through experimentation or proper case study.

IV. SUMMARY OF ANALYSES

For the brevity, a summary of the presented perspectives along with the analyses of the distance measure's properties, are presented in Table I.

TABLE I. SUMMARY OF THE DISTANCE MEASURES OF DIFFERENT PERSPECTIVES

Perspective	Distance/difference measure	Source of measurement	Original Paradigm/ Application domain	Applicability to AOP
Category and choice-based	Category and choice distance	Input values	Procedural programs (with numerical inputs)	Base code
Object-based	Object distance	Input values	Object-oriented programs	Base code
Coverage-based	Coverage manhattan distance	Structural information (e.g., branch coverage)	Procedural and object-oriented programs	Base & aspect code

With respect to above table, the first column lists down the reviewed perspectives. The second column gives the original distance measure provided by the corresponding perspectives. The third, presents the source from which the measurement of the given measures are captured. The fourth column lists the programming paradigms/application domains that the given measure were first proposed or applied to. Finally, the fifth column gives the possible applicability of the distance measures in terms of their suitability to adaptive random testing of aspect-oriented programs.

From the table, it can be clearly seen that only one measure, i.e., CMD, has the capability of being adopted to both base and aspect code, generally the whole AO program. Furthermore, the source of measurement used by this measure, it is more fine-grained and desirable compared to the other two measures.

Nevertheless, based on the theoretical analysis and interpretation shown among different perspectives and their

distance measures and the fact that these measures are capable of providing different level of adoptability to AOP (i.e., relative advantages and weakness), at the moment and based on our understanding of these reviewed perspectives, the coverage-based perspective, to be exact the CMD measure, proposed by Zhou [66] shows to be one of the most suited (with respect to the unique characteristics of AO programs) and promising distance measure towards adaptive random testing of the aspect-oriented programs.

V. CONCLUSION AND FUTURE WORK

Research on automated AOP testing is quite young and there is still a way to grow to its maturity. In ambition to advance the work with test automation of AOP and reaching to a plausible maturity, we have performed some preliminary research to investigate the applicability of one of the current automated test generation and selection techniques (i.e., ART) to AOP. The given investigation included the identification and presentation of the three related perspectives (by comparing their enclosed distance measures) on adaptive random testing of AOP and their general limitations and applicability.

As a general conclusion, our study shows that it is possible to apply the ART technique to AOP, however the current distance measures would not be all applicable or sufficient to address the notion of evenly spread of test cases suggested by ART. Two of the measures were intended to be only applicable to base code of AO programs while one was more applicable in nature, having potential of calculating distance between test cases meant for aspect code. Thus, aspect-oriented programs require evolving the discussed measures and/or developing new effective distance measure that can truly represent the notion of evenly spread of test cases with regard to the unique characteristics of these programs.

At last, we believe the work presented in this paper has provided new avenues of exploration within the area of AOP testing. Decidedly, this would be only the initial stage of leveraging a well-known testing technique to AOP; hence, it still requires further research to establish a concrete and useful ART-based technique for AOP in the future.

ACKNOWLEDGMENT

The authors acknowledge the support of the Malaysian Ministry of Higher Education for supporting this research (Fundamental Research Grant Scheme Phase 2/2010 (FRGS/2/2010/SG/UPM/01/2)).

REFERENCES

- [1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-Oriented Programming" in *Proceedings of the 11th European Conference on Object-Oriented Programming* 1997, p. 220–242.
- [2] G. Kiczales, J. Lamping, C. V. Lopes, J. J. Hugunin, E. A. Hilsdale, and C. Boyapati, "Aspect-Oriented Programming," in *United States Patent 6467086*: Xerox Corporation, 2002.
- [3] A. Colyer and A. Clement, "Aspect-Oriented Programming with AspectJ," *IBM systems journal*, vol. 44, p. 301–308, 2005.
- [4] R. T. Alexander, J. M. Bieman, and A. A. Andrews, "Towards the Systematic Testing of Aspect-Oriented Programs," Colorado State University 2004.
- [5] F. C. Ferrari, J. C. Maldonado, and A. Rashid, "Mutation Testing for Aspect-Oriented Programs," *Proceedings of the 1st International Conference on Software Testing, Verification, and Validation*, 2008, p. 52–61.
- [6] M. Harman, F. Islam, T. Xie, and S. Wrapper, "Automated Test Data Generation for Aspect-Oriented Programs," in *Proceedings of the 8th International Conference on Aspect-Oriented Software Development*, Charlottesville, Virginia, USA, 2009, p. 185–196.
- [7] T. Xie, J. Zhao, D. Marinov, and D. Notkin, "Automated Test Generation for AspectJ Programs" in *Proceedings of the 1st Workshop on Testing Aspect-Oriented Programs*, 2005, p. 1–6.
- [8] T. Xie and J. Zhao, "A Framework and Tool Supports for Generating Test Inputs of AspectJ Programs," in *Proceedings of the 5th International Conference on Aspect-Oriented Software Development*, 2006, p. 190–201.
- [9] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive Random Testing," in *Proceedings of the 9th Asian Computing Science Conference*, 2004, p. 320–329.
- [10] R. Hamlet, "Random Testing," *Encyclopedia of software Engineering*, p. 970–978, 1994.
- [11] J. Mayer and C. Schneckenburger, "An Empirical Analysis and Comparison of Random Testing Techniques," in *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, Rio de Janeiro, Brazil, 2006, p. 105–114.
- [12] T. Y. Chen, F.-C. Kuo, and R. G. Merkel, "On the Statistical Properties of the F-measure," in *Proceedings of the 4th International Conference on Quality Software*, 2004, p. 146–153.
- [13] K. P. Chan, T. Y. Chen, and D. Towey, "Good Random Testing," in *Proceedings of the 9th Ada-Europe International Conference on Reliable Software Technologies*, 2004, p. 200–212.
- [14] Y. Liu and H. Zhu, "An Experimental Evaluation of the Reliability of Adaptive Random Testing Methods," in *Proceedings of the 2nd International Conference on Secure System Integration and Reliability Improvement* 2008, p. 24–31.
- [15] T. Y. Chen, F.-C. Kuo, H. Liu, and W. E. Wong, "Does Adaptive Random Testing Deliver a Higher Confidence than Random Testing?," in *Proceedings of the 8th International Conference on Quality Software* 2008, p. 145–154.
- [16] H. Jaygarl, C. K. Chang, and S. Kim, "Practical Extensions of a Randomized Testing Tool," in *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference* 2009, p. 148–153.
- [17] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "ARTOO: Adaptive Random Testing for Object-oriented Software," in *Proceedings of the 30th International Conference on Software Engineering*, Leipzig, Germany, 2008, p. 71–80.
- [18] J. W. Duran and S. C. Ntafos, "An Evaluation of Random Testing," *IEEE Transactions on Software Engineering*, vol. SE-10, p. 438–444, 1984.
- [19] P. S. Loo and W. K. Tsai, "Random testing Revisited," *Information and Software Technology*, vol. 30, p. 402–417, 1988.
- [20] T. Y. Chen, F.-C. Kuo, and H. Liu, "Distributing Test Cases More Evenly in Adaptive Random Testing," *Journal of Systems and Software*, vol. 81, p. 2146–2162, 2008.
- [21] T. Y. Chen and R. G. Merkel, "Quasi-Random Testing," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, Long Beach, CA, USA, 2005, p. 309–312.
- [22] P. M. S. Bueno, W. E. Wong, and M. Jino, "Improving Random Test Sets using the Diversity Oriented Test Data Generation," in *Proceedings of the 2nd International Workshop on Random testing* Atlanta, Georgia: ACM, 2007, p. 10–17.
- [23] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed Automated Random Testing," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, Chicago, IL, USA, 2005, p. 213–223.
- [24] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted Random Testing: Adaptive Random Testing by Exclusion," *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, p. 553–584, 2006.
- [25] S. Xu, "Orderly Random Testing for Both Hardware and Software," in *Proceedings of the 14th IEEE Pacific Rim International Symposium on Dependable*, 2008, p. 160–167.
- [26] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou, "On Favourable Conditions for Adaptive Random Testing," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, p. 805–825, 2007.

- [27] J. Geng and J. Zhang, "A New Method to Solve the "Boundary Effect" of Adaptive Random Testing," in *Proceedings of International Conference on Educational and Information Technology*, 2010, p. 298–302.
- [28] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and S. P. Ng, "Mirror Adaptive Random Testing," *Information and Software Technology*, vol. 46, p. 1001–1010, 2004.
- [29] K. P. Chan, T. Y. Chen, F.-C. Kuo, and D. Towey, "A Revisit of Adaptive Random Testing by Restriction," in *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 2004, p. 78–85.
- [30] T. Y. Chen and D. H. Huang, "Adaptive Random Testing by Localization," in *Proceedings of the 11th Asia-Pacific Software Engineering Conference* 2004, p. 292–298.
- [31] T. Y. Chen, R. G. Merkel, G. Eddy, and P. K. Wong, "Adaptive Random Testing Through Dynamic Partitioning," in *Proceedings of the 4th International Conference on Quality Software*, 2004, p. 79–86.
- [32] F. T. Chan, K. P. Chan, T. Y. Chen, and S. M. Yiu, "Adaptive Random Testing with CG Constraint," in *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 2004, p. 96–99.
- [33] J. Mayer, "Lattice-based Adaptive Random Testing," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering* 2005, p. 333–336.
- [34] T. Y. Chen, D. H. Huang, F.-C. Kuo, R. G. Merkel, and J. Mayer, "Enhanced Lattice-based Adaptive Random Testing," in *Proceedings of the 2009 ACM Symposium on Applied Computing*, Honolulu, Hawaii, 2009, p. 422–429.
- [35] J. Mayer, "Restricted Adaptive Random Testing by Random Partitioning," in *Proceedings of the International Conference on Software Engineering Research and Practice* 2006.
- [36] J. Mayer, "Adaptive Random Testing by Bisection with Restriction," in *Proceedings of the 7th International Conference on Formal Engineering Methods*, 2005, p. 251–263.
- [37] J. Mayer, "Adaptive Random Testing by Bisection and Localization," in *Proceedings of the 5th International Workshop on Formal Approaches to Testing of Software* 2006, p. 72–86.
- [38] J. Mayer, T. Y. Chen, and D. H. Huang, "Adaptive Random Testing Through Iterative Partitioning Revisited," in *Proceedings of the 3rd International Workshop on Software Quality Assurance*, Portland, Oregon, 2006, p. 22–29.
- [39] T. Y. Chen, D. H. Huang, and Z. Q. Zhou, "Adaptive Random Testing Through Iterative Partitioning," in *Proceedings of the 11th International Conference on Reliable Software Technologies*, 2006, p. 155–166.
- [40] F.-C. Kuo, T. Y. Chen, H. Liu, and W. K. Chan, "Enhancing Adaptive Random Testing for Programs with High Dimensional Input Domains or Failure-unrelated Parameters," *Software Quality Journal*, vol. 16, p. 303–327, 2008.
- [41] J. Mayer, "Adaptive Random Testing with Randomly Translated Failure Region," in *Proceedings of the 1st International Workshop on Random Testing*, 2006, p. 70–77.
- [42] T. Y. Chen and R. G. Merkel, "Efficient and Effective Random Testing Using the Voronoi Diagram," in *Proceedings of the 17th Australian Software Engineering Conference* 2006, p. 300–308.
- [43] T. Y. Chen, D. H. Huang, and F.-C. Kuo, "Adaptive Random Testing by Balancing," in *Proceedings of the 2nd International Workshop on Random Testing*, 2007, p. 2–9.
- [44] T. Y. Chen, F.-C. Kuo, and H. Liu, "Distribution Metric Driven Adaptive Random Testing," in *Proceedings of the 7th International Conference on Quality Software*, 2007, p. 274–279.
- [45] T. Y. Chen, F.-C. Kuo, and H. Liu, "Adaptive Random Testing Based on Distribution Metrics," *The Journal of Systems and Software*, vol. 82, p. 1419–1433, 2009.
- [46] T. Y. Chen, F.-C. Kuo, and H. Liu, "Enhancing Adaptive Random Testing through Partitioning by Edge and Centre," in *Proceedings of the 18th Australian Software Engineering Conference*, 2007, p. 265–273.
- [47] T. Y. Chen, F.-C. Kuo, and H. Liu, "Application of a Failure Driven Test Profile in Random Testing," *IEEE Transactions on Reliability*, vol. 58, p. 179–192, 2009.
- [48] A. F. Tappenden and J. Miller, "A Novel Evolutionary Approach for Adaptive Random Testing," *IEEE Transactions on Reliability*, vol. 58, p. 619–633, 2009.
- [49] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive Random Testing: The ART of Test Case Diversity," *Journal of Systems and Software*, vol. 83 p. 60–66, 2010.
- [50] T. Y. Chen and F.-C. Kuo, "Is Adaptive Random Testing Really Better than Random Testing," in *Proceedings of the 1st International Workshop on Random Testing*, Portland, Maine, 2006, p. 64–69.
- [51] C. Csallner and Y. Smaragdakis, "JCrasher: An Automatic Robustness Tester for Java," *Software: Practice and Experience*, vol. 34, p. 1025–1050, 2004.
- [52] C. Oriat, "Jartege: A Tool for Random Generation of Unit Tests for Java Classes," in *Proceedings of the 1st International Conference on the Quality of Software Architectures*, 2005, p. 242–256.
- [53] J. H. Andrews, S. Haldar, Y. Lei, and F. C. H. Li, "Tool Support for Randomized Unit Testing," in *Proceedings of the 1st International Workshop on Random Testing*, Portland, Maine, 2006, p. 36–45.
- [54] B. Meyer, I. Ciupa, A. Leitner, and L. L. Liu, "Automatic Testing of Object-Oriented Software," in *Proceedings of the 33rd International Conference on Current Trends in Theory and Practice of Computer Science*, 2007, p. 114–129.
- [55] R. M. Parizi, A. A. A. Ghani, R. Abdulla, and R. B. Atan, "Towards a Framework for Automated Random Testing of Aspect-oriented Programs," in *Proceedings of the ISCA 18th International Conference on Software Engineering and Data Engineering*, Las Vegas, Nevada, USA, 2009, p. 217–223.
- [56] R. M. Parizi, A. A. A. Ghani, R. Abdulla, and R. B. Atan, "On the Applicability of Random Testing for Aspect-Oriented Programs," *International Journal of Software Engineering and its Applications* vol. 3, p. 1–20, 2009.
- [57] G. Kiczales, E. A. Hilsdale, J. J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An Overview of AspectJ," in *Proceedings of the 15th European Conference on Object-Oriented Programming* 2001, p. 327–353.
- [58] R. Laddad, *AspectJ in Action: Practical Aspect-Oriented Programming*, first ed. Greenwich: Manning Publications Co., 2003.
- [59] T. J. Ostrand and M. J. Balcer, "The Category-partition Method for Specifying and Generating Functional Tests," *Communications of the ACM*, vol. 31, p. 676–686, 1988.
- [60] F.-C. Kuo, "On Adaptive Random Testing," Melbourne, Australia: Swinburne University of Technology, PhD Thesis, 2006.
- [61] R. E. Filman and D. P. Friedman, "Aspect-oriented programming is quantification and obliviousness," in *Proceedings of the Workshop on Advanced Separation of Concerns* 2000.
- [62] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "Object Distance and Its Application to Adaptive Random Testing of Object-oriented Programs," in *Proceedings of the 1st International workshop on Random Testing* Portland, Maine: ACM, 2006, p. 55–63.
- [63] I. Ciupa, A. Pretschner, A. Leitner, M. Oriol, and B. Meyer, "On the Predictability of Random Tests for Object-Oriented Software," in *Proceedings of the 1st International Conference on Software Testing, Verification, and Validation*, 2008, p. 72–81.
- [64] Y. Lin, X. Tang, Y. Chen, and J. Zhao, "A Divergence-Oriented Approach to Adaptive Random Testing of Java Programs," in *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering*, 2009, p. 16–20.
- [65] R. Pawlak, J.-P. Retaille, and L. Seinturier, *Foundations of AOP for J2EE Development*: Apress, 2005.
- [66] Z. Q. Zhou, "Using Coverage Information to Guide Test Case Selection in Adaptive Random Testing," in *Proceedings of the IEEE 34th Annual Computer Software and Applications Conference Workshops* 2010, p. 208–213.
- [67] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, "Adaptive Random Test Case Prioritization," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, 2009, p. 233–244.
- [68] R. M. Parizi and A. A. A. Ghani, "AJcFgraph-AspectJ Control Flow Graph Builder for Aspect-Oriented Software," *International Journal of Computer Science*, vol. 3, p. 170–181, 2008.
- [69] M. L. Bemardi and G. A. Di Lucca, "An Interprocedural Aspect Control Flow Graph to Support the Maintenance of Aspect Oriented Systems," in *Proceedings of the International Conference on Software Maintenance* 2007, p. 435–444.

Devising Mutant Operators for Dynamic Systems Models by Applying the HAZOP Study

Rodrigo Fraxino Araujo
 José Carlos Maldonado
 Márcio Eduardo Delamaro
Instituto de Ciências Mat. e de Computação
Universidade de São Paulo
 São Carlos, Brazil
 {rfaraujo, jcmaldon, delamaro}@icmc.usp.br

Auri Marcelo Rizzo Vincenzi
Instituto de Informática
Universidade Federal de Goiás
 Goiânia, Brazil
 auri@inf.ufg.br

François Delebecque
Met., Alg. et Log. pour l'Automatique
Inst. Nat. de Recherche en Inf. et Aut.
 Rocquencourt, France
 francois.delebecque@inria.fr

Abstract—Embedded systems are increasingly present in many electronic devices. Therefore, it is necessary to use rigorous testing techniques aimed at ensuring that these systems behave as expected. Our contribution is the definition of mutant operators for the context of embedded systems models. We focus on dynamic systems models, specifically on Simulink and Scicos models, which are considered standards in many industrial application domains, such as avionics and automotive control. The HAZOP study was applied to investigate and analyze all the main features of such models, in order that the resulting mutant operators could be systematically generated. We developed a testing environment to support the mutation testing for dynamic system models, which was used to employ the defined mutant operators in a sample application.

Keywords-Simulink, Scicos, HAZOP, mutation testing.

I. INTRODUCTION

Due to the complexity of systems and the ever-increasing needs for shortening time-to-market pressures, the testing task has become even more challenging. A common problem is the testing stage being performed at the end of a project development life cycle. Thus, when faults are found, the cost to fix them is much higher [19].

A possibility to lessen the aforementioned problem is by using precise models that support a system development life cycle. Models are concise and understandable abstractions that capture the decisions of the functions of a system whose semantics are derived from the concepts and theories of a specific domain [18].

In this context, platforms such as ScicosLab/Scicos [12] and Matlab/Simulink [20] are widely used to design and simulate dynamic system models. One of their advantages is the applications analysis at different levels of abstraction. Another benefit is the automatic code generation, which reduces development costs and programming faults. In this paper we will use *dynamic systems* aiming specifically at *Simulink* and *Scicos* systems.

To ensure the reliability of this kind of system, the industry has been investing in an approach known as model based testing [6]. In this approach, it is easier to automate

the testing activity, which includes an automatic generation of test sets. The testing activity can begin to take place in a more abstract level, even before the software is coded. This leads to a more efficient process with significant cost reduction and a final product with higher quality.

In order to support this approach, our goal is to make possible the application of the mutation testing in embedded systems models, or specifically in dynamic systems models. In this paper, we show how a set of mutant operators was defined by the employment of the HAZOP (Hazard and Operability) [14] study to evaluate the features of such models. Some of these mutant operators were implemented in a testing tool that supports the mutation testing for dynamic systems models.

The mutant operators are responsible for determining the testing requirements of a model, that must be satisfied by the choice of an adequate input test set. A reason that ensures the wide usage of the mutation testing is the quality of the resulting final test set, i.e., its proneness to reveal faults [1].

In order to describe our study and the resulting mutant operators, the remainder of this paper is structured as follows. Section II describes dynamic systems models and the HAZOP study. In Section III, we show how the HAZOP study was employed in dynamic systems models. Section IV presents the mutant operators generated by a rigorous analysis of the achieved results. In Section V, a testing tool to support the mutation testing is described along with a sample application regarding the employment of our defined mutant operators. Section VI presents a discussion regarding related work. Section VII concludes with some final remarks and an outlook on future directions.

II. BACKGROUND

Mutation Testing is a testing approach in which the product under test is altered several times, creating a set of alternative products with slight syntactical differences, the so-called mutants. The tester is responsible for choosing test data that show difference in the behavior among the original

product and the mutant products [16]. The test set quality is measured according to its likelihood of revealing faults [9].

The construction of mutant operators must be driven by an analysis of the characteristics of the product under test. A great deal of authors do not employ general guidelines and a rigorous methodology for their definition. The mutant operators are usually a representation of a fault model considering the underlying product [10]. In our case, we are exploring a larger number of mutant operators generated by systematically applying the HAZOP study in dynamic systems models, which may later be minimized by the conduction of experiments.

In the following subsections we present a brief overview of a dynamic system model and of the HAZOP study. We used it to analyze the features of a dynamic system model, making possible to define appropriate mutant operators, that can guide the test data generation process for this sort of model.

A. Dynamic System Model

A dynamic system consists of a set of possible states, together with a rule that determines the present state from a past state. According to Korn [15], dynamic systems relate model-system states to earlier states. Classical physics, for example, predicts continuous changes of quantities such as position, velocity, or voltage with continuous time.

With the increasing complexity of these systems, development tools have become imperative to support their design. Simulink [20] and Scicos [12] are environments for sharing data, designs and specifications, making possible to develop more reliable critical systems and safely generating code. They are widely used within industry due to the large expressiveness of their languages.

The models used by such environments are based on block diagrams. These blocks include a library of sinks, sources, connectors and linear and non-linear components. Models can be hierarchical, which helps to understand the model organization and how the components interacts with each other [20, 12].

Such platforms offer a convenient way to describe systems that evolve according to time. Such systems are mathematically represented by systems of equations, that are differential equations in the case of continuous time systems, difference equations in the case of discrete time systems, and a mix of both in the case of hybrid systems. The simulation of these types of systems is based on numerical algorithms, where the solution of a system of equations, i.e., the semantics of a dynamic system model, is given by the sequence of values representing the temporal functions [7]. The input values can be read from a file or provided by a signal generator, e.g., a sinusoid or a square wave generator.

Figure 1 contains an example of a dynamic system model that is divided into three subsystems [7]. A continuous time subsystem is present in Figure 1a and represents a

braking pedal as a mass-spring-damper mechanical system. A discrete time subsystem is present in Figure 1b and is responsible for detecting when the pressing force is greater than a given threshold to activate the brake. Figure 1c presents the main system, a composition of both subsystems, containing an input, the force, and an output, the detection result.

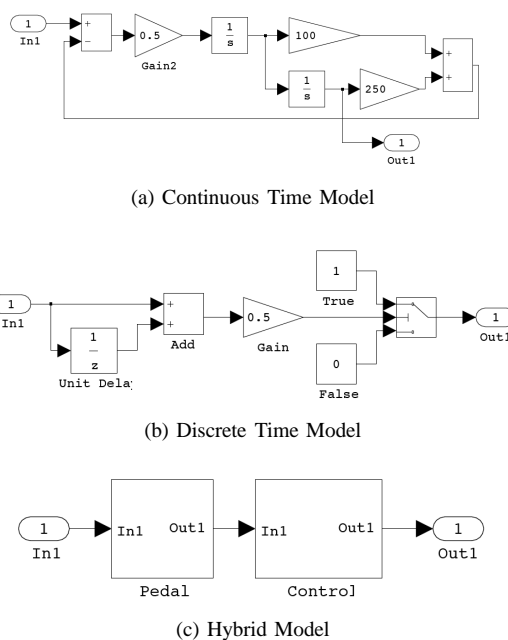


Figure 1: Dynamic Systems Models

























These models are composed by blocks connected by lines (signals). The blocks can be elementary, containing simple operations (as arithmetics, for instance), or subsystems, that contains a composition of elementary blocks. In the models of Figure 1, it is worth emphasizing the *Integrator* and the *UnitDelay* blocks, which introduce the notion of time. When an Integrator is used, the model is called of continuous time, and the operation associated to the block is a mathematical integration over time. A model that uses a UnitDelay is called of discrete time. A mix of both produces a hybrid model, defined as a data flow where the signals are continuous or discrete time functions.

A block worth mentioning is the *Switch*, which contains two data inputs and one control input. The developer must specify how the evaluation of the second input (control) must be performed, in order to redirect the first or third data input to the output. Thereby, this block can be compared to an if-then-else sentence. Table I, adapted from Chapoutot and Martel [7], presents the main blocks of a dynamic system model. VIADINHO

B. HAZOP (Hazard and Operability Study)

Hazard and operability studies (HAZOP) [14] originated in the chemical industry and, thereafter, have been widely

Table I: DYNAMIC SYSTEM BLOCKS

Name	Block	Descript.	Equation
Input		Input	$l_1 = \text{In}(t), \emptyset$
Const.		Constant	$l_1 = c, \emptyset$
Output		Output	$\text{Out}(t) = l_1, \emptyset$
Add		Addition	$l_3 = l_1 + l_2, \emptyset$
Sub		Subtraction	$l_3 = l_1 - l_2, \emptyset$
Product		Multipl.	$l_3 = l_1 * l_2, \emptyset$
Divide		Division	$l_3 = l_1 / l_2, \emptyset$
Gain		Multipl. by Constant	$l_2 = g * l_1, \emptyset$
AND		AND	$l_3 = l_1 \ \&\& \ l_2, \emptyset$
NAND		NAND	$l_3 = !(l_1 \ \&\& \ l_2), \emptyset$
OR		OR	$l_3 = l_1 \ \ l_2, \emptyset$
NOR		NOR	$l_3 = !(l_1 \ \ l_2), \emptyset$
XOR		XOR	$l_3 = (l_1 \ \&\& \ !l_2) \ \ (!l_1 \ \&\& \ l_2), \emptyset$
NOT		NOT	$l_2 = !(l_1), \emptyset$
==		==	$l_3 = (l_1 == l_2), \emptyset$
~=		!=	$l_3 = (l_1 != l_2), \emptyset$
>		>	$l_3 = (l_1 > l_2), \emptyset$
>=		>=	$l_3 = (l_1 >= l_2), \emptyset$
<		<	$l_3 = (l_1 < l_2), \emptyset$
<=		<=	$l_3 = (l_1 <= l_2), \emptyset$
Switch		Conditional Command	$l_4 = \text{if}(\rho(l_2), l_1, l_3), \emptyset$
Integr.		Continuous Time Integration	$l_2(t) = \eta(t), \dot{\eta}(t) = l_1(t)$
Unit Delay		Discrete Time Delay	$l_2(t) = \eta(t), \eta(t+1) = l_1(t)$
Sub System		Subsystem	$l_2 = f(l_1), \emptyset$

applied in different contexts to assess varying sorts of systems. The main purpose of such studies is to systematically examine the behavior of the underlying system in order to determine deviations and hazards that might arise as well as potential related problems. They are currently used in several areas for qualitative risk analysis [3].

The first step in the HAZOP study consists in identifying entities and attributes of the system under examination by means of an analysis of its description. For instance, taking a software system into consideration, such a description can be the software control flow. The next step is to apply a number of predetermined *guidewords* to system attributes in order to investigate possible deviations and determine possible causes and consequences [13].

The role of these guidewords is to act as mnemonics. After structurally applying each of them to attributes of the system under examination, it is possible to focus on a certain sort of anomalous behavior and ponder over it. Thus, this method provides additional insight into potential deviations. However, matching a guideword with an attribute requires interpretation. Depending on the context, guidewords may have more than one interpretation. For instance, *MORE* applied to a data value attribute can be interpreted as *greater*, i.e., yielding a greater value than it should be. Similarly, applying *MORE* to bit rate attributes can be interpreted as *higher*. Moreover, guidewords may be meaningless in certain contexts, demanding the creation of additional guidewords.

III. HAZOP IN A DYNAMIC SYSTEM MODEL

The testing activity is typically applied taking in consideration source code, platform independent intermediate representations or machine-specific code. However, several researches propose its use in a representation at a higher level of abstraction, i.e., models [16]. In our case, we address the testing of Simulink and Scicos models by applying the HAZOP study to the specification of a dynamic system model.

The representation examined is the syntax of the model construction. Attributes are identified for each construct of a dynamic system model, and syntactic deviations are investigated by the employment of *guidewords* to these attributes. For each possible deviation, the cause and consequence of a deviation are examined in order that mutant operators, that result in minor syntactic modifications, can be derived [13].

Table II presents the identified attributes for a dynamic system model. To show how the employment of the HAZOP guidewords to the attributes of a dynamic system model were performed, we present some examples as follows.

In the first example the construct *types* and the attribute *compatibility* affect the blocks *Input* and *Output*. It is possible to apply 2 guidewords:

- AS_WELL_AS. Cause: replacement among compatible types among double, single, int8, uint8, int16, uint16,

Table II: ATTRIBUTES OF A DYNAMIC SYSTEM MODEL

Constructs	Attributes	Related Blocks
Types	Compatibility Intervals	Input Output
Variables	Stored Values	Lines
Constants	Stored Values	Constant
Blocks	Execution Result of Switch Statement	Switch
	Execution Result of Temporal Statement	UnitDelay Integrator
	Interaction Among Subsystems	Subsystem
Expressions	Evaluation Result of Relat. Op.	Relat. Op.
	Evaluation Result of Logic. Op.	Logic. Op.
	Evaluation Result of Arith. Op.	Arith. Op.

int32, uint32 and boolean. Consequence: no loss of information.

- PART_OF. Cause: types with lower capacity can be used as, for instance, single instead of double. Consequence: it is possible to lose information or precision.

In the next example the construct *constants* and the attribute *stored values* affect the block *Constant*. It is possible to apply 3 guidewords:

- MORE. Cause: increase of a numeric value. Consequence: possible incorrect result.
- LESS. Cause: decrease of a numeric value. Consequence: possible incorrect result.
- OTHER_THAN. Cause: replacement among the constants of a model. Consequence: possible incorrect result.

Similar to the aforementioned examples, the *guidewords* were applied to the defined attributes of a dynamic system model, resulting in the analysis of all main blocks for this kind of model. Due to the lack of space, we are not able to present the relations among all guidewords and the defined mutant operators.

IV. MUTANT OPERATORS DEFINITION

A set of mutant operators was derived from the employment of the HAZOP study in a dynamic system model and is presented in this section. It is important to note that not all *guidewords* resulted in a mutant operator, because according to our evaluation, in some occasions the operation would not be significant, or would always result in a faulty model impossible to be simulated.

We decided to keep a conservative approach in the definition of mutant operators, i.e., all coherent mutant operators possible to be derived for this kind of system by the application of the HAZOP study were defined.

Types

Type Replacement Operator

This operator replaces a type with compatible types, and can be applied directly in the Input and Output blocks, which are used in the interaction among systems and subsystems.

Variables

Variable Change Operator

This operator acts in the connections among the blocks of a model, increasing or decreasing the value that is being carried. As it is not possible to know *a priori* which value that is, a possible implementation is to insert an *add* or *subtract* block between the source and destination blocks.

Variable Replacement Operator

This operator acts in the connections among the blocks of a model, replacing the compatible values that are being carried by swapping their connections. For the implementation, special attention must be drawn to the compatibility analysis among the number of inputs and outputs of each block.

Constants

Constant Change Operator

This operator is responsible for increasing or decreasing the value of the constants of a model.

Constant Replacement Operator

This operator replaces the values among the constants of a model.

Blocks

Statement Swap Operator

This operator is responsible for swapping the first and the third input of the Switch block, acting in a way similar to modifying the evaluation result of the blocks condition.

Delay Change Operator

This operator can increase or decrease the delay in which the output of the UnitDelay or the Integrator blocks will be provided to the system.

Subsystem Change Operator

This operator swaps the connections between two subsystems or between a main system and a subsystem aiming to act in the integration of components of a model. Despite being a suboperator of the Variable Replacement Operator (VRO), this operator may be useful if the tester desires to analyze only the interaction among the subsystems of a model.

Block Removal Operator

This operator is responsible for removing each of the blocks of a model, and can be useful to ensure that every block is being used and that a test data exists to force its execution.

Expressions

Relational Op. Replacement Operator

This operator is responsible for the replacement among the relational operators $>$, $>=$, $=$, \sim , $<$ e $<=$.

Arithmetic Op. Replacement Operator

This operator is responsible for the replacement among the blocks Add, Sub, Product, Divide and Gain.

Logical Op. Replacement Operator

This operator is responsible for the replacement among the logical operators AND, OR, NAND, NOR, NOT and XOR.

A. Summary

12 mutant operators were defined by employing the HAZOP study for Simulink-like models and are summarized in Table III. Most of the defined mutant operators deal with the data flow of a Simulink-like model, which is the essence of this type of system. Three mutant operators were defined aiming at dealing with unique features of this kind of model. Although most of the operators deals with modification in a model, in certain cases for their implementations new blocks need to be added and removed. As a result, we consider that they are a complete set, taking into account that along with our analysis, the defined mutant operators force addition, alteration and deleting operations in Simulink-like models.

Table III: MUTANT OPERATORS

Acron.	Description
TRO	Type Replacement Operator
VCO	Variable Change Operator
VRO	Variable Replacement Operator
CCO	Constant Change Operator
CRO	Constant Replacement Operator
SSO	Statement Swap Operator
DCO	Delay Change Operator
SCO	Subsystem Change Operator
BRO	Block Removal Operator
RORO	Relational Op. Replacement Op.
AORO	Arithmetic Op. Replacement Op.
LROO	Logical Op. Replacement Op.

The first one, SSO, aims to swap the inputs of a Switch block, altering the control flow of a system. The second operator, DCO, deals with the temporal characteristics of a system, and acts in the UnitDelay and Integrator blocks. The third one, SCO, operates in the interaction among the subsystems of a model, swapping the blocks connections among them or among a main system and possible subsystems.

Table IV presents the worst case scenario, or the maximum number of mutants to be generated by each mutant operator regarding the model property that is being affected, i.e., input and output ports, blocks or blocks connections (lines). The VRO mutant operator is the most likely to produce a larger number of mutants.

Table IV: NUMBER OF GENERATED MUTANTS

M. Op.	Worst Case Scenario
TRO	(Inputs + Outputs) * Data Types
VCO	Lines * 2
VRO	Lines * (Lines -1)/2
CCO	Constants * 2
CRO	Constants * (Constants-1)/2
SSO	Switches * 2
DCO	Delays * 3
SCO	SSLines * (SSLines -1)/2
BRO	Blocks
RORO	Relat. Op. * 5
AORO	Op. * $2^{Op.Inputs}$ + Gain * 2
LROO	Logic. Op. * 5

V. TESTING TOOL

TeTooDS (Testing Tool for Dynamic Systems) [2] can interpret dynamic systems models, interact with simulation environments such as Scicos or Simulink, and is used to assist in the test data generation task. It was previously developed to provide support for the application of functional criteria, specifically the pairwise approach, in dynamic systems models. This approach ensures that any two possible values, belonging to two different parameters, will be present in at least one test data [11].

We have extended TeTooDS to support mutation testing in dynamic systems models. The first necessary step was the development of a full-blown parser, that provides the information required by the mutant operators to the generation of mutants of a model. These information include input ports, input datatypes, blocks, blocks parameters, connections and output ports. Our parser makes use of the pyparsing module [17], a flexible approach for creating and executing grammars, against the lex/yacc approach or the use of regular expressions. The pyparsing module provides a library of classes that supports building grammars directly into the Python code.

After parsing a Scicos or Simulink model, which is accomplished when a testing project is created in TeTooDS, several options become available to the tester. A possibility is to select which mutant operator will be used for the generation of the mutant models.

The tester can also visualize the mutant models inside TeTooDS: (i) as an image; (ii) as the source code of the model; or (iii) using TeTooDS to call Scicos/Simulink along with the mutant model. It is useful for performing an analysis of equivalent mutant models or to see which mutants are alive or dead.

Test cases can be added by specifying input files that will be read by the dynamic system model during its simulation, together with the specification of which output files should be read by the testing tool when the simulation finishes.

To run the simulation of the main dynamic system model and the generated mutants, TeTooDS provides a default

script that can be used or customized in order that the parameters values, such as start time, stop time and step time, can be configured according to the tester needs. After the simulation finishes, output files are analyzed and the mutation score is updated with the mutants status information.

A. Sample Application

This model represents an electronic regulator which contains a flow regulator, a temperature sensor and a logic controller. The system has three input ports: temperature, temperature lower bound and temperature upper bound. When the temperature is below the lower bound, a valve is closed, i.e., receives a zero value. When the temperature is above the high bound, a valve is opened, receiving a value of 100. When the temperature is between these limits, the valve aperture is calculated by the expression $(5.0/3.0) * (temperature - low_bound)$ [4].

We used all the defined mutant operators, applying one mutation at a time, which resulted in 131 generated mutants. For the execution of the mutants, firstly we selected input data randomly. For the remaining mutants, in order to achieve 100% of mutation score, we manually analyzed each mutant aiming to select a test data that could kill it or mark it as equivalent. Table V shows the number of mutants generated by each operator.

Table V: NUMBER OF MUTANTS

Operator	Mutants	Operator	Mutants
TRO	0	DCO	4
VCO	36	SCO	0
VRO	26	BRO	19
CCO	4	RORO	10
CRO	1	AORO	15
SSO	1	LROO	15

The TRO did not return any mutants, as we used a Scicos model as source and it does not allow the use of several data types. Mutants also were not generated by the SCO operator, as the number of inputs of the subsystems of this particular model are not compatible.

To show the viability of the defined mutant operators, the second step of our case study was to manually generate the C code that corresponds to this particular model. We used Proteum [8] and its 73 mutant operators to generate mutants for the C code, which resulted in 1473 mutants. By applying the test set that was selected to achieve 100% of mutation score in the model, which represents a simulation of the system that is going to be hardware integrated, we could achieve 98.1% in the C code.

Our first intention was to use the code that can be automatically generated by Scicos. Nevertheless, it presents too many unused variables and other pieces of unexecuted code, resulting in a large number of equivalent mutants to be analyzed (up to 100 000 mutants).

We consider that we achieved a high mutation score for this particular model when applying the test set responsible for achieving a full coverage of the model (100%) in the C generated code (98.1%), which encourages the development of a thorough experiment, taking into account all the necessary validity levels. We emphasize that our intention is to assess the feasibility of all mutation operators aiming at possible refinements.

VI. RELATED WORK

The existing literature shows that the mutation criterion is very effective for revealing faults of traditional programs and models. Nonetheless, this criterion has not been widely explored for the context of dynamic systems models.

We are aware of two studies that aim at applying the mutation testing in dynamic systems models. The first one is described by Brillout et al. [5]. They developed a methodology to assess the correctness of Simulink models by automating the test data generation activity. Their objective is to cover the requirements imposed by the mutation testing. In order to generate and optimize the test data, the approach focus on model checking techniques. However, the authors do not clearly present an solution of how to apply the mutation testing, i.e., which mutant operators should be used to generate the testing requirements.

The second study is the one of Zhan and Clark [21]. Despite introducing a testing framework for Simulink models and focusing on the mutation testing, the approach presents a few limitations. The authors make use of a random test data generator and try to improve the test set by the use of dynamic analysis and simulated annealing methods, in order to satisfy the constraints imposed by their mutant operators. We consider as a drawback of their approach the low number of defined mutant operators, i.e., *add*, *multiply* and *assign*. In our approach, we have tried to overcome such issue by performing a systematic analysis of a dynamic system model in order to define a complete set of mutant operators for this context, that includes the ones defined by Zhan and Clark.

VII. FINAL REMARKS AND FUTURE WORK

We address the testing of Simulink and Scicos models. Dealing with these models entails properly concerning their domain specific language, which is geared towards code generation, and also present specific features, as temporal and combinatorial characteristics.

The employment of the HAZOP study to derive mutant operators for a particular type of system can produce different syntactic variations, which can assist in finding possible faults of a system. In this paper we presented the solutions that the authors consider appropriate for dynamic systems models.

One of the advantages of the HAZOP study is that the set of mutant operators can be more complete than those

generated based only on the experience of faults of a developer, since the language constructs are analyzed.

Future work also includes the definition of a method for the automatic generation of test data for dynamic systems models, that aims at satisfying the mutation test requirements. Longer term future work includes the conclusion of an integrated testing environment that can assist in the automation of the testing activity for dynamic systems models.

VIII. ACKNOWLEDGMENT

The authors would like to thank the financial support provided by CNPq (grant number 141976/2008-0). We are also thankful to Vinicius Durelli, who proofread and commented on drafts of this paper.

REFERENCES

- [1] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *27th ICSE*, pages 402–411. ACM Press, 2005.
- [2] R. F. Araujo and M. E. Delamaro. TeTooDS - Testing Tool for Dynamic Systems. In *Tools Session – Brazilian Software Engineering Symposium*, Brazil, 2008. SBC.
- [3] J. S. Arendt and D. K. Lorenzo. *Evaluating Process Safety in the Chemical Industry. A user guide to quantitative risk analysis*. AIChE, second edition, 2000.
- [4] M. Blackburn, R. Busser, and A. Nauman. Why model-based test automation is different and what you should know to get started. In *International Conference of Practical Soft. Quality and Testing*. SPC, 2004.
- [5] A. Brillout, M. He, Nannan afend Mazzucchi, D. Kroening, M. Purandare, P. Rümmer, and G. Weissenbacher. Mutation-based test case generation for simulink models. In *Proceedings of the 8th international conference on Formal methods for components and objects*, FMCO'09, pages 208–227, Berlin, Heidelberg, 2010. Springer-Verlag.
- [6] M. Broy, B. Jonsson, J. Katoen, M. Leucker, and A. Pretschner, editors. *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.
- [7] A. Chapoutot and M. Martel. Abstract simulation: A static analysis of simulink models. In *ICISS '09: Proceedings of the 2009 International Conference on Embedded Software and Systems*, pages 83–92, Washington, DC, USA, 2009. IEEE Computer Society.
- [8] M. E. Delamaro and J. C. Maldonado. Proteum – a tool for the assessment of test adequacy for c programs. In *Proceedings of the Conference on Performability in Computing Systems (PCS 96)*, pages 79–95, 1996.
- [9] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection help for the practicing programmer. *IEEE Computer*, 11(4):34–41, Apr. 1978.
- [10] F. Ferrari, J. Maldonado, and A. Rashid. Mutation testing for aspect-oriented programs. In *Software Testing, Verification, and Validation*, pages 52–61, april 2008.
- [11] M. Grindal, J. Offutt, and S. F. Andler. Combination testing strategies - a survey. *Software Testing, Verification and Reliability*, 15(3):167–199, 2005.
- [12] INRIA Rocquencourt. Scicos, 2011. Available at <http://www.scicos.org>.
- [13] S. Kim, J. A. Clark, and J. A. McDermid. The rigorous generation of java mutation operators using hazop. In *Proceedings of the 12th International Conference on Software and Systems Engineering and their Applications (ICSSEA'99)*, 1999.
- [14] T. Kletz. *Hazop and Hazan: Identifying and Assessing Process Industry Hazards*. CRC Press, fourth edition, 1999.
- [15] G. A. Korn. *Advanced Dynamic-system Simulation: Model-replication Techniques and Monte Carlo Simulation*. Wiley-Interscience, 2007.
- [16] A. Mathur. *Foundations of Software Testing*. Pearson Education, 2008.
- [17] P. McGuire. Pyparsing, 2011m. Available at <http://pyparsing.wikispaces.com>.
- [18] B. Meenakshi, A. Bhatnagar, and S. Roy. Tool for translating Simulink models into input language of a model checker. In Z. Liu and J. He, editors, *ICFEM*, volume 4260 of *LNCS*, pages 606–620. Springer, 2006.
- [19] W. Perry. *Effective methods for software testing, third edition*. John Wiley & Sons, Inc., 2006.
- [20] The Mathworks Inc. MATLAB and Simulink, 2011. Available at <http://www.mathworks.com>.
- [21] Y. Zhan and J. A. Clark. A search-based framework for automatic testing of matlab/simulink models. *Journal of Systems and Software*, 81(2):262–285, 2008.

A Static Robustness Grid Using MISRA C2 Language Rules

Mohammad Abdallah
 School of Engineering and
 Computing Sciences
 Durham University
 Durham, UK
 m.m.a.abdallah@dur.ac.uk

Malcolm Munro
 School of Engineering and
 Computing Sciences
 Durham University
 Durham, UK
 malcolm.munro@dur.ac.uk

Keith Gallagher
 Department of Computer Sciences
 Florida Institute of Technology
 Florida, USA
 kgallagher@fit.edu

Abstract—Program robustness is the ability of software to behave correctly under stress. Measuring program robustness allows programmers to find the program’s vulnerable points, repair them, and avoid similar mistakes in the future. In this paper, a *Robustness Grid* will be introduced as a program robustness measuring technique. A Robustness Grid is a table that contains rules classified into categories, with respect to a program’s function names and calculates robustness degree. The Motor Industry Software Reliability Association (MISRA) rules will be used as the basis for the robustness measurement mechanism. In the Robustness Grid, for every MISRA rule a score will be given to a function every time it satisfies or breaches a rule. The Robustness Grid shows how much each part of the program is robust, and assists developers to measure and evaluate robustness degree for each part of a program.

Keywords-Robustness; Robustness Grid; MISRA C2.

I. INTRODUCTION

Robustness is required in critical programs where failures could cause problems [1]. Robustness is an important factor in any program development process. The IEEE defines robustness as “*The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions*” [2].

In this definition, there are three main aspects; the correct program response, the input data, and system environment. Program response means that the system should respond rationally [3], but not necessarily correctly. It should not fail to reply or react illogically. The input data is one of the factors that affect the robustness of the program. A robust program can continue to operate correctly despite the introduction of invalid input [4].

The environment where the program is run is contained in hardware, other software systems, and the humans that run the program. These factors also affect the program robustness. It is this aspect of robustness that this study is concerned with.

Static measures of software robustness complement robustness testing. Robustness testing is a “*testing*

methodology to detect vulnerabilities of a component under unexpected inputs or in a stressful environment.” [5]

The objective is to evaluate the robustness features of imperative programs from the perspective of programmers and maintainers. Thus it will give an assessment of the program vulnerabilities, in order to help improve and certify the robustness of existing programs.

The Robustness Grid is developed as a measurement tool and is a numeric representation of the robustness degree for each function, and for the program in total. The Robustness Grid certifies C program robustness through applications of MISRA C2 guidelines.

There are different standards that the programmers are advised to follow during writing a C program to produce a robust program. However, these standards are not widely used to measure the program robustness after the program has been written.

This study will contribute a Robustness Grid using a number of robust features. The MISRA C2 language rules will be used as a measurement of the robustness features. The Robustness Grid will provide the robustness degree for the program, and each function it includes, as a numeric value. Thus the Robustness Degree will show the degree of satisfaction that a program has according standards of robustness.

In section 2, MISRA C language rules are presented. Section 3 overviews the existing research in Robustness Grid technique. In Section 4 the Robustness Grid Calculations concepts are listed. Section 5 presents the related work in Robustness measurement. Finally, in Conclusions future research is highlighted.

II. MISRA C2

The Motor Industry Software Reliability Association (MISRA) has published a standard set of rules for C and C++ “*to provide assistance to the automotive industry in the application and creation within vehicle systems of safe and reliable software*” [6]. MISRA C 1998 rules (“MISRA C1”) were published in 1998 and were followed by technical clarification document in 2000. In 2004, MISRA published a second version of MISRA C rules (MISRA C2) to address some technical and logical problems, and for further technical clarification. In MISRA C2 the rules

are rephrased to be more sensible, accurate and comprehensive.

MISRA C2 rules are classified into two types: Required (122 rules) and Advisory (20 rules). Required rules are obligatory and must be followed by developers to create safe programs. Advisory rules are necessary but not as important as the Required rules; however a developer should follow the advisories in order to build a safe program. In addition, MISRA C2 has 21 categories that consider different programming processes, coding styles, and programming syntax.

The MISRA categories cover all C language common programming issues. The MISRA categories start with Environment category, which describes the optimum environment for C programs. Then Language extensions category, where it has headlines for writing comments through programming. Documentation category contains general rules for documentation process.

The syntax format concerns, problems and advice is covered and discussed in the rest of the categories. An example of a MISRA C2 rule is rule 8.1:

Rule 8.1 (required) Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call. [4]

“X.y” is the MISRA rule numbering method and means this is rule 1 (“y”) in category 8 (“X”) (Declarations and definitions). “required” means it the rule is an obligatory rule.

III. ROBUSTNESS GRID

Measuring software robustness needs to examine features in order to produce a relative scale that calculates the robustness degree for functions, and the entire program.

A. Robustness Features and Robustness Degree

Before discussing the Robustness Grid, some terms should be clarified: Robustness Features and Robustness Degree.

Robustness Features are characteristics that affect software robustness, such as code syntax [7]. Robustness Features in this study are divided into two groups depending on their source. Robustness Language Features certify the robustness degree of code syntax and coding style. Second, User Functional Requirements features certify the robustness degree of the service that program provides, how it reacts to input, and how the system responds [8].

Robustness Degree is a scale of a program robustness features satisfaction, expressed as a percentage.

MISRA C rules are divided to several Categories as described in following section. These categories will be used to create the Robustness Grid.

B. Robustness Grid

The *Robustness Grid* is a table showing the robustness degree of every function in a program and for the entire program. Then the robustness features satisfaction percentage will be calculated cumulatively in each

category, function, and whole program. The values highlight the vulnerable points (low percentage score) of the functions and program. *TABLE II* shows an example of the Robustness Grid. Each category in the Robustness Grid is independent, so a function could score a high marks in one category and score low marks in another.

The Robustness Grid has two parts: the *static part* which contains the MISRA C2 rules; here, there is no need to understand the code functionality because only the program code will be certified. The second part is the *dynamic part*, which contains User Functional Requirements. In this paper, only the static part will be discussed.

1) Rules selection method and conditions:

In this study, some assumptions and conditions are applied to programs to be certified by the Robustness Grid:

1. The program must be compileable by a compiler that satisfies the MISRA C2 environment rules.
2. Programs should satisfy MISRA C2 rules number 1.1 and 14.2 which means the program must satisfy the ISO Standards [9].

The total number of MISRA C2 rules, after applying Robustness Grid assumptions and conditions is 100 in 6 Categories.

2) Rule categorization method:

The Robustness Grid (*TABLE II*) is a table that classifies MISRA C2 rules into 6 different Categories; each Category has a set of related rules:

TABLE I. ROBUSTNESS CATEGORY CONSTRUCTIONS

Category	Constructs
0	The rules that considers type definition, arithmetic statements.
1	Rules that consider control statements (if, for, while ...etc).
2	The rules that consider function structure.
3	The rules that consider arrays, pointers, and data structure (union, struct, enum ...).
4	The rule that consider header files and the pre-processor
5	All MISRA C2 advisory rules.

If a rule is in more than one Category, it will be classified under the highest Category. If a single line of code is considered by more than one Category, it will be certified against each Category, individually.

IV. ROBUSTNESS GRID CALCULATIONS

Certifying program robustness using the Robustness Grid uses the following procedure:

1. The program must be able to be compiled by the gcc compiler.
2. Pre-processor code lines are considered as part of the function main, unless it related to a particular function.

- Rules which are not applicable for all functions in a program are removed from Robustness Grid in order to save space.

After the rules have been selected, and program eligibility is satisfied, the Robustness Grid is built for the program. In the Robustness Grid, the calculations that measure the Robustness Degree for the functions and for the program is novel and introduced here for the first time.

The Robustness Grid building process is as follows:

- Each statement in the program is assessed against all the selected MISRA C2 rules.
- All selected rules will be put in their categories depending on the categorisation method defined above.
- Each rule has the applied status next to it, showing whether it is satisfied (+), violated (-), or not applicable (0).
- Program Statements will be grouped by their function.
- For each function the status of all rules is listed.
- The Robustness Grid calculations are made for each function (FACS), category (ACD), and for the entire program (WPCS).

The *SwapAdd.c* program is a simple example program that will be used to illustrate how the Robustness Grid is applied. The *SwapAdd.c* program, shown in Fig 1, is a C program with three functions, *main*, *swap*, and *incr*. The *swap* function exchanges two pointers and *incr* function increments its first parameter by the value in its second parameter, and *main* calls both functions.

In program *SwapAdd.c*, *incr*, *swap*, and *main* are the program functions. In the Robustness Grid the numbers under each function are the rule states; a positive number (+n) means the rule has been satisfied *n* times in the function. Negative numbers (-n) means the rule has been broken *n* times in the function. Zero means the rule is not applicable to the code.

The robustness degree can be calculated as follows:

- Function Category Satisfaction (FCS):** For Category *n* the number of times a rule has been satisfied divided by the number of times the rule has been applied, expressed as percentage.
- Program All Categories Satisfaction (PACS):** For Category *n*, a count of all the times a rule has been satisfied for all the program's functions divided by the number of times the rule has been applied in all program functions, expressed as percentage.
- All Categories (between 0 and *n*) **Accumulative Robustness Degree (ACD):** Number of times rules are satisfied in categories (0 - *n*) divided by number of times rules are applied in categories (0 - *n*), expressed as percentage.
- Function All Categories Satisfaction (FACS):** Number of times rules are satisfied in all categories divided by number of times rules are applied in all categories, expressed as percentage.
- Whole Program Categories Satisfaction (WPCS):** For all program functions: Count of all times rules

been satisfied divided by all times that rules been applicable as a percentage.

```
#include <stdio.h>
#define LAST 10
void incr(int *num, int i);
void swap(int *a, int *b);
int main(){
    int i, sum = 0, *a = 12,*b = 13;
    for ( i = 1; i <= LAST; i++ ) {
        incr(&sum, i);}
    printf("sum = %d\n", sum);
    swap (&a,&b);
    return 0;
}
void incr(int *num, int i) {
    *num = *num + i;
}
void swap(int *a, int *b) {
    int temp= *a;
    *a= *b;
    *b= temp;
    printf ("pointer a is:%d\n",*a);
    printf ("pointer b is:%d\n",*b);}
```

Figure 1. SwapAdd.c Program

The result of analysis as shown in TABLE II shows that the *SwapAdd.c* program satisfied the robustness features by 75.5%. To improve the robustness of the program the category 5 rules should be examined because they have the smallest PACS ratings. It also shows that *swap* should be examined because has the smallest FCS value.

This static Robustness Grid is still produced manually, which is a limitation of this study. The automation for the Grid will be done by using the semantic part of C language.

V. RELATED WORK

Critical programs must be robust to avoid the problems that could be caused by failures [10]. The C Language standards were introduced to avoid the code misinterpretation, misuse, or misunderstanding. The IEEE has the ISO/IEC 9899:1999 standard [9], which is used later by MISRA to produce MISRA C1 and C2. This in turn led to Jones producing "The New C Standard: An Economic and Cultural Commentary" [10]. The LDRA Company uses MISRA C rules in addition to 800 rules that it created to assess programs [11]. Other C standards such as "C programming language Coding guideline" [12] are less frequently used.

Measuring the application of language standard to a program is one program robustness measurement technique. Several techniques have been tried to measure program robustness. Software measurement means estimates the cost, determine the quality, or predict the maintainability [13]. Arup and Daniel [14] presented features such as portability to evaluate some existing benchmarks of Unix systems. As a result they built a hierarchy structured benchmark to identify robustness issues that have not been detected before. Behdis and Shokat [15] introduced a theoretical foundation for a robust matrices that reduce the uncertainty in distributed system. Arne et al. [16] used some robustness criteria such as input data rate, and CPU clock rate to create a multi-dimensional robustness matrices and use them to measure the robustness of a system.

A Robustness Hierarchy is a relative scale to find the robustness characteristics that needs to be added to programs. A Robustness Hierarchy is a technique used to build a robust program. The Hierarchy starts with non-robust program as first step then adds robust features before reaching a robust program in the highest level of the Hierarchy [7].

All the previous software measurement techniques do not give the developer a fully detailed set of measurements. Nor do they specify the parts of the program that need to be modified to raise its quality. Thus the focus of this study is to give the programmer a full description for all the robustness features and the degrees to which they are satisfied.

The Robustness Grid allows the developer to specify the code lines that need to be modified to improve the program Robustness Degree.

VI. CONCLUSION AND FUTURE WORK

A Robustness Grid has been defined and it has been shown how it can work as an assessment tool. This means that every function in a program can be certified using MISRA C2 rules through the calculation of a robustness degree.

The Robustness Degree show the MISRA C2 rules that have been followed and satisfied by the program and the rules that have been violated. The Robustness Degree gives an indication as to where the developer or maintainer should do some code changes to improve the robustness of the program.

The calculation of the Robustness Degree can be considered as simplistic in that is based on percentages of rules that are passed or failed. It does not fall into the trap of allowing positive and negative values to cancel each other out. All rules are treated with the same weight. It is

clear that for any particular program this is not necessarily so. In the future work, a dynamic robustness features will be introduced to make the measurement more accurate and reliable by giving weights to the important statements in the program. Thus in the Robustness Grid, each static rule will be weighted by the Dynamic rules to highlight the different level of importance of the static rules.

REFERECES

- [1] G.M. Weinberg, Kill That Code!, Infosystems, 1983, pp. 48-49.
- [2] IEEE, IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, IEEE Computer Soc, 1990.
- [3] S.D. Gribble, Robustness in complex systems, Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, 2001, pp. 21-26.
- [4] L.L. Pullum, Software fault tolerance techniques and implementation, Artech House, Inc., 2001.
- [5] L. Bin, L. Xuandong, L. Zhiming, M. Charles, and S. Volker, Robustness testing for software components, Elsevier North-Holland, Inc., 2009, pp. 879-897.
- [6] M.I.S.R. Association, MISRA website, last access <retrieved: 7, 2011>.
- [7] M. Abdallah, M. Munro, and K. Gallagher, Certifying software robustness using program slicing, 2010 IEEE International Conference on Software Maintenance, Timisoara, Romania, 2010, pp. 1-2.
- [8] I. Sommerville, Software Engineering, Addison-Wesley, 2006.
- [9] ISO/IEC, International Standard ISO/IEC 9899, International Organization for Standardization, 1999.
- [10] D.M. Jones, The New C Standard: A Cultural and Economic Commentary, Addison-Wesley Professional, 2003.
- [11] LDRA, LDRA Test Suite, last access <retrieved: 7, 2011>.
- [12] E. Laroche, C programming language coding guidelines, last access <retrieved: 7, 2011>.
- [13] N.E. Fenton, and S.L. Pfleeger, Software Metrics, A Rigorous and Practical Approach, PWS Publishing Company, 1997.
- [14] A. Mukherjee, and D.P. Siewiorek, Measuring Software Dependability by Robustness Benchmarking. IEEE Transactions of Software Engineering 23 (1994) 94-148.
- [15] B. Eslamnour, and S. Ali, Measuring robustness of computing systems. Simulation Modelling Practice and Theory 17 (2009) 1457-1467.
- [16] A. Hamann, R. Racu, and R. Ernst, Methods for multi-dimensional robustness optimization in complex embedded systems, Proceedings of the 7th ACM & IEEE international conference on Embedded software, ACM, Salzburg, Austria, 2007, pp. 104-113.

TABLE II. SWAPADD.C ROBUSTNESS DEGREE

Category	C2 Rules	incr	FCS%	swap	FCS%	main	FCS%	PACS%
Category 0	4.1& 7.1	0	3/3 = 100%	+2	5/7 = 71.4%	+1	6/8 = 75%	14/18 = 77.8%
	4.2	+2		+2		+4		
	5.1	+1		+1		+1		
	5.2	0		-2		-2		
Category 1	12.2	+1	4/5 = 80%	0	5/5 = 100%	-1	8/11 = 72.7%	17/21 = 81%
	13.1	+1		+3		+4		
	13.4	0		0		+1		
	13.5	0		0		-2		
	13.6	0		0		+1		
	14.7	+1		+1		+1		
	17.1	-1		0		0		
17.5	+1	+1	+1	+1				
ACD (0 – 1)			87.5%		83.3%		73.7%	79.5%
Category 2	8.1	+1	10/12 = 83.3%	+1	8/13 = 61.5%	0	9/12 = 75%	27/37 = 73%
	8.2	+1		+1		+1		
	8.3	+1		+1		+1		
	8.6	+1		+1		+1		
	8.11	-1		-1		-2		
	14.8	0		0		+1		
	16.1	+1		+1/-2		+3		
	16.2	+1		0		0		
	16.3	+2		+2		0		
	16.4	+1/-1		-2		0		
	16.5	0		0		-1		
	16.8	0		0		+1		
16.9	+1	+1	+1	+1				
ACD (0 – 2)			85%		72%		74.2%	76.3%
Category 3	16.7	+1	100%	+2	100%	0	0	100%
ACD (0 – 3)			85.7%		74.1%		74.2%	77.2%
Category 4	19.6	0	2/2 = 100%	0	2/2 = 100%	+1	4/4 = 100%	8/8 = 100%
	20.1	+1		+1		+1		
	20.2	+1		+1		+1		
	20.9	0		0		+1		
ACD (0 – 4)			87		75.9		77.1	79.3
Category 5	5.7	-1	1/2 = 50%	-2	1/3 = 33.3%	-3	3/6 = 50%	5/11 = 45.5%
	19.1	0		0		+1		
	19.2	0		0		+1		
	19.7	+1		+1		+1		
FACS			84%		71.9%		73.2%	WPCS 75.5%

A Specifications-Based Mutation Engine for Testing Programs in C#

Andreas S. Andreou

Department of Electrical Engineering and Information
Technology,
Cyprus University of Technology
Limassol, Cyprus
email: andreas.andreou@cut.ac.cy

Pantelis Stylianos Yiasemis

Department of Electrical Engineering and Information
Technology,
Cyprus University of Technology
Limassol, Cyprus
e-mail: pm.yiasemis@edu.cut.ac.cy

Abstract—This paper presents a simple and efficient engine which produces mutations of source code written in C#. The novelty of this engine is that it produces mutations that do not contradict with the specifications of the program. The latter are described by a set of pre- and post-conditions and invariants. The engine comprises two parts, a static analysis and syntactic verification component and a mutation generation component. Preliminary experiments showed that the proposed engine is more efficient than a simple mutations generator in terms of producing only valid mutations according to the specifications posed, thus saving time and effort during testing activities.

Keywords-mutation testing; mutation engine; specifications;

I. INTRODUCTION

Technology advancements nowadays lead to the automation of a large number of activities within the software development process. The exploitation of computing power drives the need for producing better, faster and more reliable software systems. Nevertheless, the aforementioned targets increase software complexity and size, making this need hard to be satisfied. The competition in the software development market pushes companies to increase their productivity, developing software in tighter time limits usually sacrificing the quality of the resulting software.

One of the most significant reasons for the inadequate quality control in software development is the lack of efficient software testing. The latter is a way for verifying the correctness and appropriateness of a software system, or, alternatively, for ensuring that a program meets its specifications ([1], [2]). Software testing is not a simple process; on the contrary, it consumes a large percentage of the time and budget of the whole development process. In some cases it even surpasses the time needed for the creation of the software product. Its main purpose is to reveal and locate faults so as to assist developers improving the functional behavior of the system under development.

Software testing consists of two main processes, the identification of faults (testing) and their correction (debugging). Identifying faults is the most time consuming process as it can take up to 95% of the time of software testing. Having this in mind, we can safely conclude that there will be a constant need to develop tools that will assist in accelerating and automating the testing process, guiding developers to locate and debug faults faster and more efficiently.

The aim of the present paper is to introduce a mutation engine for source code written in C#, which is the basic element of a novel mutation testing technique that takes into consideration the specifications of the program for creating only valid mutants. The engine is implemented in Visual Studio 2010 and consists of two components: The first offers the ability to validate the grammatical correctness of the source code and provides a form of statistical analysis for exporting useful information that can be used to process/modify the source code. The second involves the production of mutations of the original source code and facilitates the identification of faults, as well as the assessment of the quality of test data.

The rest of the paper is structured as follows: Section II describes briefly the basic concepts that form the necessary technical background of this work. Section III presents the mutation engine, its architecture and key elements ruling the generation of mutations, as along with a brief demonstration of the supporting software tool. Section IV describes a set of preliminary experiments and the corresponding results that indicate the correctness and efficiency of the proposed approach. Finally, Section V concludes the paper and suggests some steps for future work.

II. TECHNICAL BACKGROUND

According to McMinn [3], three different kinds of software testing techniques exist. These are White Box Testing (WBT), Black Box Testing (BBT) and the mixing of the two called Gray Box Testing (GBT). Each of these three techniques offers its own advantages and disadvantages, differing on the way test cases are created and executed. In BBT the test cases are created based on the functions and specifications of the system under testing without the need for actual knowledge of the source code. WBT requires that the tester needs to have full access to the source code and know exactly the way it works. Advantages of this method are that it can locate coincidental correctness, this is the case where the final result is correct but the way it is calculated is not. Moreover, all possible paths of code execution may potentially be tested offering the ability to identify errors or/and locate parts of dead code, that is, parts that are never executed.

Different techniques have been proposed for WBT making use of the structure of the source code or the sequence of execution, giving birth to static code analysis and testing for the former and dynamic testing for the latter. We concentrate on dynamic testing where the actual flow of execution drives

test data production. One such technique that has gain serious interest among the research community is Mutation Testing (MT).

MT is a relatively new technique introduced by DeMillo et al. [4] and Hamlet [5], which is based on performing replacements in code statements through certain operators that correspond to specific types of errors, producing the so-called mutant programs; the latter are then used to assist in producing or/and assessing the quality of test data as regards revealing the errors in the mutants [6].

The general idea behind MT is that the faults being injected correspond to common errors made by programmers. This means that the mutants are slightly altered versions of programs which are very close to their correct form. Each fault is actually a single change of the initial version of the program, pretty much the same as a slight change (mutation) in living species causing a different form of life. The quality of a produced set of test cases is assessed by executing all the mutants and checking whether the injected faults have been detected by the set or not.

There are quite a few ways to represent code and provide the means for better understanding and management of the source code. Most of them use graphs or/and binary trees that are able to depict graphically how the program actually works. The Control Flow Graph (CFG) is one such way of graphically representing the possible execution paths. Each of its nodes usually corresponds to a single line of code, while the arcs connecting nodes represent the flow of execution. CFG may be used as the cornerstone of static analysis, where its construction and traversing offers the ability to identify and store information about the type of statements present in the source code and the details concerning the alternative courses of execution. A fine example is the BPAS framework introduced by Sofokleous and Andreou [7] for automatically testing Java programs. More to that, CFG may drive the generation of test data by providing the means to construct an objective function for optimization algorithms to satisfy (e.g. by evolution, like Michael et al. [8]).

During the last years the Visual Studio (VS) platform [9] has been constantly evolving becoming one of the most wide spread platforms used today in the software industry. This is partly due to the fact that it provides to developers the ability to create a number of different types of applications, like window-apps, web-apps, services, classes etc. The wide acceptance of VS has driven the development of a number of third party tools and plug-ins that enhance the platform with even more functionality, making development of special-purpose applications simpler and easier. The aforementioned advantages of VS2010 led us to investigate its use for software testing, and more specifically for developing a new mutation testing tool.

Code Contracts (CC) are offered by VS2010 as the means to encode specifications [10]. CC may consist of pre-conditions, post-conditions and invariants. Their aim is to improve the testing process during runtime checking and static contract verification, as well as to assist in documentation generation.

The mutation engine introduced in this paper is partly based on the aforementioned concepts. More specifically, it

utilizes CFG and static analysis as in [7] to extract the information needed for analyzing and describing adequately the source code under investigation. Moreover, it employs CC to embed the specifications required so that the program functions properly and static analysis (contract verification) in order to guide the production of meaningful mutant programs, that is, programs that do not violate their original specifications. The engine targets at offering the means for automatic, time-preserving software testing.

III. MUTATION ENGINE

A. Architecture

As previously mentioned, the mutation engine was implemented in the VS2010 platform. The selection of VS2010 was made partly because it is a relatively newly introduced platform, meaning that the components developed may be used as a backbone for future tools and studies based on this platform, without facing any incompatibility issues compared to the use of older platforms. Also, to the best of our knowledge, at present no other such system exists. The engine was specifically designed to work with the C# programming language, but with minor changes and additions the support of the rest of the programming languages VS2010 platform offers may be enabled as well.

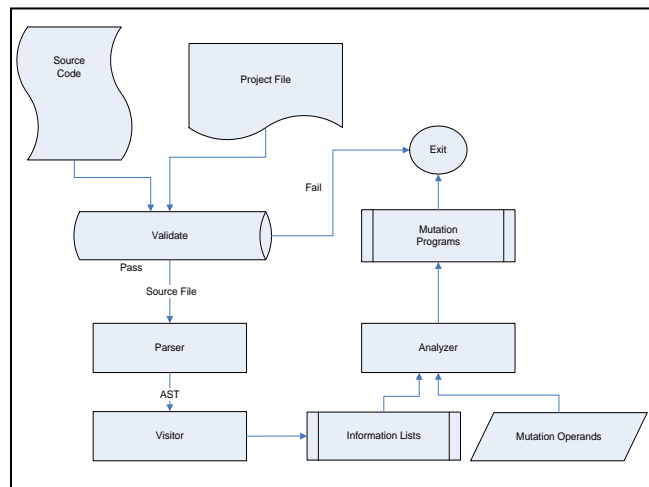


Figure 1. The mutation engine architecture

The architecture of the proposed mutation engine is depicted graphically in Figure 1 where three major components enable the execution of the engine’s stages. The first is a source code validation component, which compiles the source code and presents the erroneous lines in code if such exist. This component takes as input a source code file (.cs), or an executable file (.exe), or a dynamic link library file (.dll), as well as the project file (.csproj). The project file is needed to provide the component with information for references in libraries and files that the source code must use and are part of the program. Validation includes compiling the source code and making sure that no syntactic or other compilation errors exist so as to proceed with the second stage of the engine which is the production of mutations. Otherwise the engine terminates.

The second component performs statistical analysis of the source code without the need of an executable form of the program under testing. By statistical analysis we mean exporting the useful information from the source code as regards the structure of the program. This component takes as input the source code file and uses the class AbstractSourceTree (AST) of SharpDevelop [11] to model the abstract syntax tree of the code. While compiling a source code file, a binary tree is created, each node of which represents a line of code. Traversing this binary tree, once the tree is completed, offers access to any part of the source code.

Analyzing the statistical component described above we can see that it consists of two sub-components, the *Parser* and the *Visitor*. The *Parser* analyses the source code and creates the AST as mentioned earlier. After it finishes, the *Visitor* passes through the tree collecting useful information, while giving the opportunity to the user to make changes and additions to the information stored. The implementation of the *Visitor* utilized the AbstractAstVisitor class of SharpDevelop, with some minor additions to help accessing all the nodes of the AST, both at the high and the low level characteristics of the programming language. The *Visitor* recursively visits each node and stores in stack-form lists all the information identified according to the node's type. In the experiments described in the next section thirteen such lists were created; nevertheless, the way the *Visitor* is structured enables the addition of any new lists or the modification of existing ones in a quite easy and straightforward manner.

The third and final component is actually the heart of the mutations production. This component analyses the information stored in the lists created by the *Visitor* so as to identify the structure and content of the source code, and creates mutated programs by applying a number of predefined operators to the initial program. These mutators are responsible for creating a number of different variations of the initial source code based on the rules each of them represents without breaching the grammatical correctness of the resulting program.

Mutations are performed at the method level via operators that are usually of arithmetic, relational, logical form, and at the class level with operators applied to a class or a number of classes and usually refer to changing calls to methods or changing the access modifiers of the class characteristics (public, private, friendly etc.). The operators supported by the proposed mutation engine are the following:

Arithmetic

- AOR_{BA} – arithmetic operations replacement (binary, assignment)
- AOR_S – arithmetic operations replacement (shortcut)
- AOI_S – arithmetic operations insertion (shortcut)
- AOI_U – arithmetic operations insertion (unary)
- AOI_A – arithmetic operations insertion (assignment)
- AOD_S – arithmetic operations deletion (shortcut)
- AOD_U – arithmetic operations deletion (unary)
- AOD_A – arithmetic operations deletion (assignment)

Relational

- ROR – relational operations replacement

Conditional

- COR – conditional operations replacement
- COI – conditional operations insertion
- COD – conditional operations deletion

Logical

- LOR – logical operations replacement
- LOI – logical operations insertion
- LOIA – logical operations insertion (assignment)
- LOD – logical operations deletion
- LODA – logical operations deletion (assignment)

Shift

- SOR – shift operations replacement
- SOIA – shift operations insertion (assignment)
- SODA – shift operations deletion (assignment)

Replacement

- PR – parameter replacement
- LVR – local variable replacement

B. Specification-Based Mutations

The number of possible mutated programs for a certain case-study may be quite large depending on the type and number of statements in the source code. Therefore, when testing is based on mutations processing time may substantially increase as it is proportional to the number of mutants processed. This is a significant problem that may hinder the use of mutation testing in certain cases. Thus, there is a need to minimize mutation testing execution time. This is feasible taking into account the fact that a considerable number of useless mutations may be observed as the changes made to the code correspond to invalid forms of executions for that particular program as these are determined by the program's specifications. Therefore, we need to take these specifications into consideration when producing the mutants. This is exactly what we do via the Code Contracts supported in VS2010. Additionally, this feature is enhanced by ruling out mutation cases that have syntactical errors and are practically of no use.

The following example demonstrates how mutations are driven by the specifications inserted via CC, where class Test includes methods *Foo* and *Goo* and uses CC to express two pre-conditions (denoted by *Contact.Requires*) and one post-condition (denoted by *Contact.Ensures*):

```
public class Test {
    private int Foo(int a, int b) {
        Contract.Requires(a > b);
        Contract.Requires(b > 0);
        Contract.Ensures(Contract.Result<int>() > 0);
        ...
        return (a / b);
    }
    ...
    private void Goo( ) {
        int x, y;
        ...
        x = y + 10;
        int result = Foo ( x , y ) }
}
```

In *Goo* the assignment of *x* affects the values with which *Foo* is called. The first pre-condition requires that $x > y$. The

engine normally would perform operation replacement substituting '+' with '-', '/', '%' and '*'. Due to the pre-condition the engine will drop the first three replacements and use only the last one as it is the only replacement that will still satisfy the pre-condition. The same applies for $b > 0$, where any arithmetic replacement should not set b equal or less than zero. Therefore, a sort of "thinking" before producing a certain mutation is implemented in the engine which enables the production only of valid mutants thus ensuring that the minimum possible time and effort will be spent in the subsequent analysis and testing activities.

C. The software tool

A dedicated software tool was developed to support the whole process. An example scenario is given below to demonstrate its operation: A source code file and the project file of the program tested are given as input to the system. The project file and all the references to other files or libraries are automatically located and linked, and the source code file is compiled through the validation component. In the case of compilation errors a pop up window is presented to the user with the corresponding information (Figure 2) and the process is terminated. If there are just warnings, the user is again informed, but the system now continues to the next step. Statistical Analysis of the source code is executed next resulting the creation of the AST. The visitor component then passes through the binary tree and creates the lists that store the information found in the source code. Lastly, the third component takes as input the lists created earlier by the visitor and a set of selected mutators, applies these operators and returns the resulting mutated programs (Figure 3).

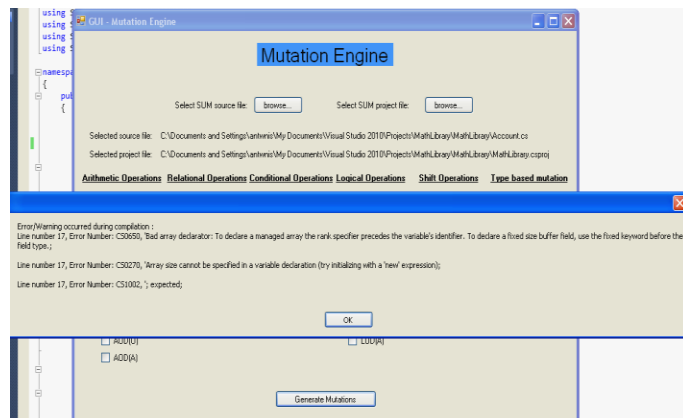


Figure 2. Execution : Errors in compilation

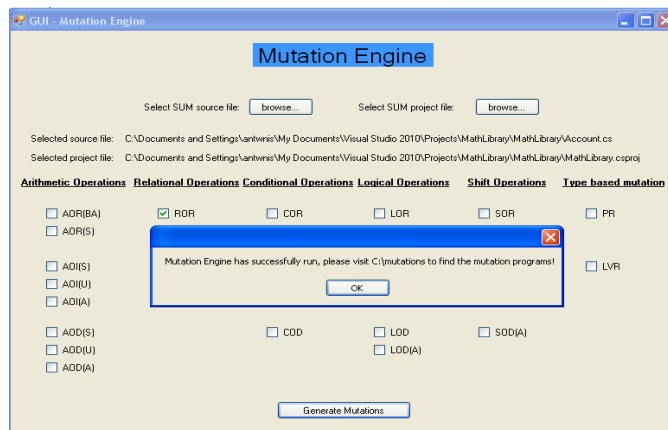


Figure 3. Execution : Mutations successfully produced

IV. EXPERIMENTAL RESULTS

A series of preliminary experiments was conducted to assess the correctness and efficiency of the proposed testing approach. The aim here was twofold: First, to demonstrate that the proposed engine works as it is supposed to, that is, it is able to produce correctly a number of mutations to be used for testing by performing atomic changes to the source code in hand according to a selected operator. Second, to assess whether the incorporation of specifications in the way mutations are produced indeed improves its performance by allowing only certain types of mutations to be executed and thus bounding the computational burden for revealing faults.

The first experiment is involved with assessing the quality (adequacy) of test cases to identify faults in a benchmark program via the use of the proposed approach. The second deals with fault detection using two sample programs with injected faults and producing mutants for detecting those faults. The final experiment compares the number of mutations produced with a standard mutation process to that of a specifications-driven production so as to assess the improvement in time performance. The experiments are analyzed below:

A. Test-Data Quality Assessment

This experiment used as benchmark the well-known triangle classification program listed below, which was tested against certain test data presented in Table I.

```
int triang(int i, int j, int k) {
    if ((i <= 0) || (j <= 0) || (k <= 0))
        return 4;
    int tri = 0;
    if (i==j) tri+=1;
    if (i==k) tri+=2;
    if (j==k) tri+=3;
    if (tri==0) {
        if ((i+j==k) || (j+k==i) || (i+k==j))
            tri=4;
        else tri=1;}
    else {
        if (tri>3) tri=3;
        else {
```

```

if ((tri==1) && (i+j>k))    tri=2;
else {
    if ((tri==2) && (i+k>j)) tri=2;
    else {
        if ((tri==3) && (j+k>i)) tri = 2;
        else tri = 4; } } }
return tri; } }

```

TABLE I. TEST DATA THAT COVER ALL POSSIBLE OUTPUTS OF THE TRIANGLE CLASSIFICATION PROGRAM (TCP)

i	j	k	Result
2	2	2	equilateral
0	1	2	not a triangle
3	3	1	isosceles
3	4	2	scalene

Using the values of Table I for the three variables it seems at first that we have tested adequately the TCP. Nevertheless, if we employ the mutation engine proposed we may conclude that the aforementioned set of test data is of low quality as at least one atomic change produces an error that is not recognized by the set. Indeed, the engine produced several mutations of which the one below passed the set as it successfully yields an identical result as the original program:

```

if ((i <= 0) || (j < 0) || (k <= 0))

```

This simple code mutation suggests that indeed the proposed engine is able to assess the quality of a set of data to adequately test a given program.

B. Fault Detection

This set of experiments investigated the ability of the mutation engine to reveal errors that were injected in the initial source code of two programs, the first finds the maximum number between four integers, while the second implements division of two integer numbers and it is controlled by specifications expressed with code contracts.

In the first example, three faults were inserted in the code below, one relational, one parameter replacement and one unary.

```

public class FindMax {
    public int getMax(int num1, int num2, int num3, int num4) {
        int max = 0;
        if (num1 > num2)    max = num1;
        else max = num3;
/** should have been max = num2 **//
        if (max < num3) {
            max = num3;
            if (max > num4)    max = num3; }
/** condition should have been (max < num4) **//
        else {
            if (max < num4)    max = num4; }
            return -max; } }
/** should have been return max **//

```

The engine applied a series of mutators, of which operators ROR, PR and AOD_U were actually the ones that

revealed the injected errors. More specifically, ROR replaced relational operation ‘>’ with ‘<’, ‘>=’, ‘<=’, ‘_’ and ‘!=’ capturing the proper behavior. PR performed every possible combination of parameter replacement among (num1, num2, num3 and num4) resulting in the correct identification of presenting the error because of the use of num2 instead of num3. Finally, AOD_U successfully located the error in the last line after removing the minus sign.

The second example below employs CC with three pre-conditions, one post-condition and one invariant, and involves two errors inserted in class CompareParadigm that cannot be traced by the static analyzer in VS2010.

```

class CompareParadigm {
    int num,den;

    public CompareParadigm(int numerator, int denominator) {
        Contract.Requires(0 < denominator);
        Contract.Requires(0 <= numerator);
        Contract.Requires(numerator>denominator);
        this.num += numerator;
/** should have been this.num = numerator **//
        this.den = denominator; }

    [ContractInvariantMethod]
    private void ObjectInvariant() {
        Contract.Invariant(this.den > 0);
        Contract.Invariant(this.num >= 0); }

    public int ToInt() {
        Contract.Ensures(Contract.Result<int>()>=0);
        return this.num * this.den; } }
/** should have been this.num / this.den **//

```

The engine was once again capable of bringing these errors to light using the arithmetic operation replacement (AOR_{BA}) and arithmetic operations deletion (AOD_A) mutators.

C. Normal vs Specifications-Based Mutations Production

As mentioned earlier, a sort of “intelligence” was embedded in the engine that eliminates all mutants that violate the pre-conditions, post-conditions or invariants set for a program. Using class CompareParadigm listed earlier, we will compare the number of mutations produced by the mutation engine with the use of specifications to that of a normal (typical) mutations generator (in this case the engine with the CC disabled). Table II lists the mutations produced according the operator used. One may easily notice that a 58% reduction to the mutants was achieved by the “intelligent” engine, which resulted in 16 mutated programs compared to 38 produced without taking into consideration the specs. This is indeed a remarkable saving of effort and time with just a small part of code consisting of less than 20 statements. Therefore, we can safely argue that in cases of large programs the computational burden will be considerably eased, preserving at the same time the effectiveness and efficiency of the testing process.

TABLE II. MUTATED PROGRAMS CREATED BY THE ENGINE WITH (SPECS-BASED) AND WITHOUT THE USE OF SPECIFICATIONS (NORMAL)

Operator	Number of Mutations	
	<i>Specs-based</i>	<i>Normal</i>
AORBA	5	8
AOIS	7	10
AOIU	0	6
LOI	2	6
PR	2	3
LVR	0	5
Total	16	38

V. CONCLUSION AND FUTURE WORK

Software testing is an important, though complex, area of software development that aims at increasing the quality and reliability of software systems. Automatic software testing approaches are increasingly popular among researchers that attempt to handle the aforementioned complexity and lead to faster and cheaper software development with high quality standards.

Mutation testing is a technique that produces different versions of a program under study which differ slightly from the original one and uses these versions either to identify faults or assess the adequacy of a given set of test cases. In this context, the present paper proposed a simple, yet efficient mutation engine, which uses a number of mutation operators that can be applied at the method level and incorporating a sort of intelligence to generate only valid mutants based on the program's specifications. The engine is developed in the Visual Studio 2010 platform and utilized Code Contracts to represent the specifications that must be satisfied with pre-conditions, post-conditions and invariants.

The engine is supported by a dedicated software tool consisting of two main parts. The first part verifies the syntactical correctness of the source code and proper linking with the appropriate libraries, and provides statistical analysis of the source code, using grammatical analysis and producing the Abstract Source Tree representation of the source code. The second part uses the information gathered from the previous part and generates mutations using specific operators and obeying to the rules imposed by the encoded specifications.

A series of experiments was conducted that showed that the mutation engine constitutes a tool that may efficiently be used for identifying faults in the code and for assisting to the creation of the proper set of test data. The incorporation of the specification-based concepts can significantly improve performance by reducing the number of mutants processed, thus saving time and effort.

Future work will involve extending the proposed engine to include more class-level mutators, as well as investigating

the potential of supporting other programming languages under the .Net framework. Moreover, we plan to integrate our tools with tools offered by the VS2010, like the PEX, which is responsible for unit testing and UModel, which assists in creating UML diagrams. This integration will enable the formation of a complete testing environment with dynamic user interaction, both at the flow of control level and at the diagrammatical. Finally, our efforts will concentrate on evaluating the engine on a more systematic basis using sample programs of different size and complexity and assessing various parameters like the time for creating and processing mutations, the type of mutators used, the nature of the errors induced, etc. This systematic investigation will also address scalability issues and more specifically our future experimental evaluation will include code from large-sized, real-life software projects.

REFERENCES

- [1] C. Kaner, J.H. Falk, H.Q. Nguyen, *Testing Computer Software*, John Wiley & Sons Inc., New York, NY, USA, 1999.
- [2] Bertolino, "Software testing research: achievements, challenges, dreams", Proc. 29th International Conference on Software Engineering (ICSE 2007): Future of Software Engineering (FOSE'07), Minneapolis, MN, USA, 2007, pp. 85-103.
- [3] P. McMinn, "Search-based Software Test Data Generation: A Survey", *Software Testing, Verification and Reliability* Vol. 14(2), 2004, pp.105-156.
- [4] R.A. DeMillo, R.J. Lipton and F.G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer", *IEEE Computer* Vol. 11(4), 1978, pp. 34-41.
- [5] R.G. Hamlet, "Testing Programs with the Aid of a Compiler", *IEEE Transactions on Software Engineering*, Vol. 3(4), 1997, pp. 279-290.
- [6] "Mutation Testing Repository", <http://www.dcs.kcl.ac.uk/pg/jiayue/repository/>, [accessed 10 May 2011]
- [7] A.A. Sofokleous and A.S. Andreou, "Automatic, Evolutionary Test Data Generation for Dynamic Software Testing", *Journal of Systems and Software*, Vol. 81(11), 2008, pp. 1883-1898.
- [8] C.C. Michael, G. McGraw and M.A. Schatz, "Generating software test data by evolution", *IEEE Transactions on Software Engineering* (12), 2001, pp. 1085-1110.
- [9] "Visual Studio 2010", (2009) <http://www.microsoft.com/visualstudio/en-us/products/2010-editions>, [accessed 18 May 2011]
- [10] "Code Contracts User Manual", (2010), Microsoft Corporation, <http://research.microsoft.com/en-us/projects/contracts/userdoc.pdf> [accessed 20 May 2011]
- [11] "SharpCode", (2009), <http://www.icsharpcode.net/opensource/sd/>, [accessed 17 May 2011]

Component-based Software System Dependency Metrics based on Component Information Flow Measurements

Majdi Abdellatief^{ab}, Abu Bakar Md Sultan^a, Abdul Azim Abd Ghani^a, Marzanah A.Jabar^a

^aDepartment of Information System, Faculty of Computer Science & Information Technology,
University Putra Malaysia, 43400 Serdang, Selangor, Malaysia

^bMehareeba Technical College, Technical Education Corporation, 2081 Khartoum, Sudan

Khwaja24@yahoo.com, {abakar, azim, marzanah}@fsktm.upm.edu.my

Abstract-The motivation of this paper is that the measurement based on the flow of information connecting software components can be used to evaluate component-based software system dependency. The ability to measure system dependency implies the capability to locate weakness in the system design and to determine the level of software quality. In this paper, dependency between components is considered as a major factor affecting the structural design of Component-based software System (CBSS). Two sets of metrics namely, Component Information Flow Metrics and Component Coupling Metrics are proposed based on the concept of Component Information Flow from CBSS designer's point of view. We also discuss the motivation for and possible uses of system level metrics and component level metrics. Initial results from our on-going empirical evaluation indicate that the proposed metrics are very intuitive.

Keywords-Component-based software system; Software metric; Dependency; Information flow.

I. INTRODUCTION

In Component-based development (CBD) paradigm, Component-based software system (CBSS) are developed using a set of independent components which work together. Some of these components may be developed in-house, while others may be third-party components, without source code [1]. Nowadays, this development methodology has become one of the predominant software engineering solutions for the design of a large and a complex system [2].

Analysis of CBSS dependencies is an important part of software research for understandability [3], testability [4], maintainability [5] and reusability [6][7] of a component-based system. Thus, dependency metrics could have a real impact on the quality of the system delivered to the user. If valid dependency metrics could be identified, they could provide the information required by developers, testers and maintainers to understand the system, identify the critical components, evaluate the impact of change in one component on the other components and even to support the future evolution of the CBSS when adding, removing and modifying some components. It is difficult to perform such tasks without understanding potential component dependencies [8]. In addition, a large and complex CBSS should be evaluated early at the specification phase, to avoid faults, poor interaction among components and failure of one component which could lead to a total system failure [9][10].

Previous research conducted in CBSS metrics concentrated on one of two major areas. Many research papers [11][12][13], focused on measuring the reusability of software components, while others [2][10][14][15][16],

focus on measuring the interaction complexity of integrated components. In the past, only a few papers based on graph theory addressed the evaluation of CBSS dependency [8][10][17][18][19]. However, there has been no theoretical or empirical validation conducted for the proposed metrics. In this paper, interface dependency is considered to be the main dependency affecting CBSSs. Interface dependency exists as relationships among different functionalities and parameters of software components. For example, when one interface relies on other to obtain functionalities necessary for its own tasks. However, if the components produced by component providers only include specifications of the interfaces [19][20], the interface specification does not supply adequate information for analysis of integrated CBSS dependency. Thus, in CBSSs, due to the black box nature and the separation of interface specification from its implementation, the analysis of information flows will be quite difficult using the traditional information flow techniques. Therefore, we first proposed a new method named Component Information Flow (CIF) to analyze the information flows into a component, out of a component and between components. We believe that the CIF is a more suitable and practical basis for characterizing and evaluating CBSS for several reasons. First, often the component's internal structure is not available. Second, the elements of CIF could be directly determined at design phase. Third, the availability of metric values early in the design phase allows the CBSS structure to be corrected with the least cost. Fourth, as seen in the subsections of this paper, it's based on standard Information flow [21], which is considered more sensitive than other measurements.

Based on the concept of CIF, we also propose two sets of metrics, namely, Component Information Flow Metrics and Component Coupling Metrics that represent the CBSS designer's point of view (they are also relevant to testers and maintainers). The proposed metrics depict details about the quality of a structure design at three levels, entire CBSS level, component level and interface level. For each level they concern with the way in which components or interfaces connect.

This paper is organized as follows: Section II describes research methodology. Section III illustrates component-based information flow definitions and concepts. Section IV provides the definition of the metrics and their description. Section V applies our proposed metrics in a small scale example and discusses the results. The conclusion and direction for future work are in Section VI.

II. RESEARCH METHODOLOGY

The metrics are derived in the following steps:

1. Conducting systematic mapping study on existing CBSS metrics and metrics validation techniques.
2. Defining information flow for CBSS.
3. Defining a new dependency metrics for CBSS specification.
4. Application of the proposed metrics in a small scale example.

In step 1, a systematic mapping study of the values for various metrics was carried out by the authors of this paper and the limitations of the current research were drawn from them in “unpublished” [22]. The third author suggested step 2. The planning, data collection and reporting of steps 2 and 3 were performed by the first author with respect to the context defined in Section III. Each step and its content was checked and reviewed by the rest of authors independently and carefully. In case there is ambiguity point, a negotiation took place. Particularly, step 2 was investigated many times since it is considered as the core of this study. Step 4 was conducted by all of the authors as stated in Section V.

III. COMPONENT INFORMATION FLOW CONCEPTS AND CONTEXT

To provide a context for this Section and the next Section, we need a background of software component and CBSS specification method. Components definition adopted in this study clearly fall under Szyperski’s definition [1]. The CBSS structural specification method used is that of Cheesman and Daniels [23]. Our measurement approach assumes that the proposed approach is generally applicable to developments using any of the technology standards such as Sun’s EJB, Microsoft’s COM+ and CORBA Models.

A. Software Component Concept

We visualize software component concepts from the perspective of component developers and CBSS designers. Figure 1 provides a simplified model of a component such that a specification defines the functionality and behaviour of a component which is composed of an interface part and a body part. The specification and interface are visible to CBSS designers, whereas the specification, interface and body are visible to component developers.

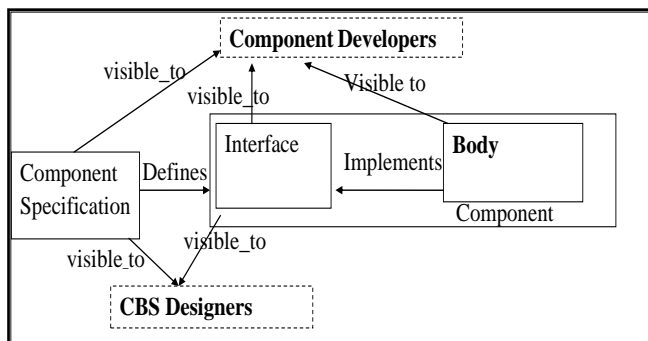


Figure 1. Simplified component model

The interface definition includes a collection of one or more operations to specify the functionality and behaviour identified in the specification. The body of the system implements the external methods and any other internal methods that are required to provide the functionality and behaviour identified in the specification. Metrics may be derived from the specification, interface or body but only metrics derived from the interface and specification can be used by CBSS designers.

B. Definition of Component Information Flow

This subsection describes the mechanisms for deriving the various types of component information flow based on the above assumptions.

The separation of interface from implementation is a core principle of component based development. That is, the functionality specified in the interface could be implemented in different applications by different programming languages. Therefore, it is important to view interfaces and their specifications separately from any specific component that may implement or use such interfaces. To explain this view, it suffices to consider the interface of a component to define the component’s access point [24]. These access points allow clients of a component, usually components themselves, to access the functions provided by the component. Normally, a component could have multiple access points corresponding to different functions provided in the interface [1].

In Figure 2, we depict this view from an interface perspective. This model focuses on what the interface must do to fulfill the client’s information required without considering how this will be accomplished. With respect to the proposed model in Fig 3, for any component in CBSSs, two boundaries are considered: (1) Interface boundary which separates the provider interface from a client interface. The client might be a user, a required interface or an engineering device. (2) The body boundary which separates the provider interface from its implementation.

Component Information Flow (CIF) is characterized by two types of flows, Inter-component flow and Intra-component flow. In the Inter-component flow, the provider interface communicates with client to exchange information by In-flows and Out flows. Thus, the information flows across the interface boundary. The In-flow carries information from a client to a provider interface through the

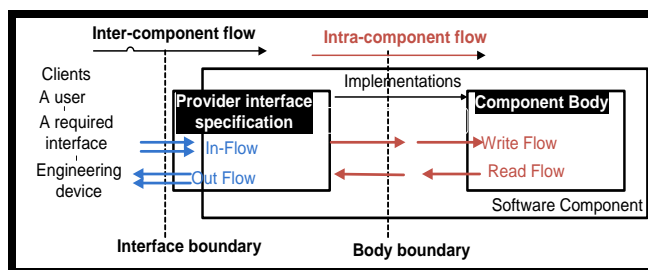


Figure 2. Generic model of component information flow

list of in-parameters. The Out flow carries information from a provider interface to a client through the list of out-parameters. In the Intra-component flow, it is assumed that the data structure is used (i.e., a component body) to store and retrieve the information needed by the provider interface, represented by Read flow and Write flow. Thus, the information flows across the body boundary. In other words, an intra-component flow takes place when an interface retrieves data from or updates a data structure.

An important characteristic of the CIF described above is that the knowledge essential to build the complete flow structure can be established from a simple analysis of a UML requirements specification. The UML modeling technique describes the component specification, the component interaction diagram and the interface specification to design the intended CBSS. Component specifications name the interfaces that a component adhering to the specification must implement. An interface specification consists of a set of operation specifications. An interface specification has to specify how the inputs, outputs, and component object state are related, and what the effect of calling the operation has on that relationship [23]. An operation specifies an individual action that an interface will perform for the client. Each action shows one or more types of information flow (i.e., In-flow, Out flow, Read flow or Write flow), where each type of information flow shows one possible execution flow. Thus, for each interface we can identify all potential flows from the interface specification. To facilitate the mapping of CIF to a complete flows structure, we describe a template for CIF analysis and data collection as shown in Table 1.

To understand the relationship between components and make the concept of CIF clear, consider an example presented in Figure 3, which shows three components, A, B, C and their relationship to each other. This example is purely from the specification perspective. It is assumed that some functionality required by component "A" is implemented by "B" and "C. We depict the information flow among components as a result of methods calling and events firing as Inter-component flow, and the information flow inside the components to update or retrieve from component store as Intra-component flow. The information flow from component "A" to "B" or "A" to "C" can be represented by a set of direct inter-flows plus a set of intra-flows, whereas the information flow from "B" to "C" can be represented by a set of indirect inter-flows plus a set of intra-flows.

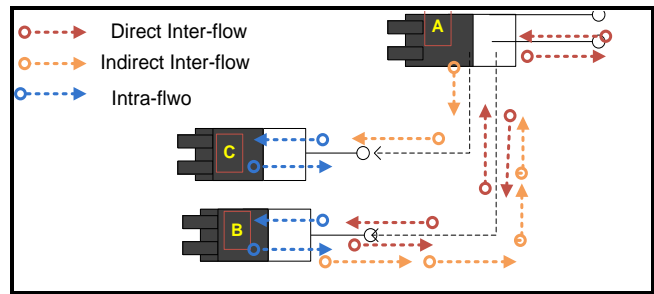


Figure 3. An example of component information flow

The following definitions describe precisely the terms and the four types of information flow presented informally above. These four types of flow identify the logical flow of information between components. The reader should refer to Figure 3 to understand definitions 1, 2 and 4, and Figure 3 to understand definition 4, 5, 6 and 7.

Definition 1: Information flow is the set of messages streaming across the boundaries which define a particular communication between two components based on the logical representation of the interface specification.

Definition 2: There is an Intra-component flow of information from component "B" to component "A" if a component "B" implements some functionality of component "A".

Definition 3: There is an Inter-component flow of information from component "A" to component "B" if one or more of the following conditions hold:

- 1) If a component "A" invokes a component "B" and passes information to it; or component "B" returns a result to a component "A" (termed direct inter-component flow).
- 2) If a component "A" invokes both a component "B" and a component "C" passing output values from "B" to "C" (termed indirect inter-component flow).

Definition 4: In-flow is an inter-component flow type and carries information provided or passed from a client entity to a provider interface.

Definition 5: Out flow is an inter-component flow type and carries information returned from a provider interface to a client entity.

TABLE 1. TEMPLATE FOR COMPONENT INFORMATION FLOW ANALYSIS AND DATA COLLECTION

Interfaces	operations	Operation Description	Information Flow Types	Source of Information Flow	Destination of Information Flow
Each component can consists of one or more interfaces	Each interface can consists of one or more operations	Each operation could be described as a set of messages with respect to the definitions information flow (i, e., definitions 1, 4, 5, 6, and 7)	In-flow	Client interface	Provider Interface
			Out flow	Provider interfaces	Client interface
			Read flow	Provider Interface	Component store
			Write Flow	Provider interface	Component store

Definition 6: Read flow is an intra-component flow type and carries information retrieved from a component store to a provider interface.

Definition 7: Write flow is an intra-component flow type and carries information from a provider interface to update component store.

C. CIF supports:

- a variety of software architectures from simple stand-alone application to large distributed software based on OSI 7 layers or J2EE n-tiers. Therefore, almost any kind of CBSS structure can be analyzed and evaluated.
- all stages of the software life cycle. Analysis can be carried out as early in the requirement specifications or as late in the life cycle as necessary.
- a defined measurement unit. An elementary unit of CIF defined by us is a base flow type (i.e., in-flow, out flow, read flow and write flow)

IV. DEFINITION OF DEPENDENCY METRICS

We use measurement based on the flow of information to evaluate and manage dependencies between components in the CBSS. Particularly, we use the following metrics to characterize the effect of dependency on the structure design of CBSS.

A. Component Coupling Metrics

In our literature survey, we found inconsistencies in the definition of coupling in the literature [6][7][25][26][27][28]. There were several different definitions of coupling, depending on the measurement goal and entity being measured (i.e., inheritance coupling, messages passing coupling or data abstraction coupling) [29]. Thus, the coupling attribute has been defined, measured and interpreted in various ways. Xia [27] studied this ambiguity of coupling concept and redefined it based on its essence. We adopted his definition here. “Component coupling of *m* is the impact-dependence of components to *m*”. The impact-dependence of X2 to X1 means that when X1 is modified, there will be an impact on X2. For example, when changing component X1 in Figure 4, we only need to consider how component X2 will be affected. Component X2 returns F1 and F2 to component X1. F1 and F2 are out-flows of component X2 and in-flows of component X1 which will influence component X1 when component X2 is changed. But when X1 is modified, F1 and F2 have no impact on X2. Therefore, the right definition should consider only the out flow of X1 for its coupling. Another important source which could influence the change in X1 is the number of distinct components receiving the out flows [30]. For example, an impact on a component that depends on one component is not the equivalent to a component that depends on three components, even if both components receive the same number of out flows.

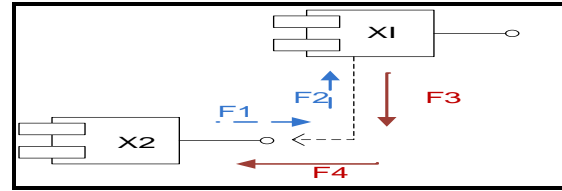


Figure 4. The impact of component modification

Assumption 1: The more the spread of inter-flow from a component, the larger the context of its interface operations and the more the external information required to test and maintain the components.

Accordingly, we defined coupling metrics as

$$\text{Interface Coupling (IC)} = n \times \sum_{i=1}^p OF_i$$

where

p = number of operations in an interface

OF_i = number of out flows in each operation (i)

n = the number of other component to which an interface is coupled

$$\text{Component Coupling (CC)} = \sum_{i=1}^p IC_i$$

where

IC_i = interface coupling

p = the number of interfaces in a component.

$$\text{CBSS coupling} = \sum_{i=1}^n CC_i$$

where

CC_i = component coupling

n = the number of components in the system.

This definition consistent with the study by Kitchenham and Likman [31], which indicated that all the information flow metrics studied, except for informational fan-in, appear to act as indicators of future problems.

B. Component Information Flow Metrics

We adopted the definition of information flow proposed by Ince and Shepperd [32] which is considered to be a more sophisticated metric than the original information flow proposed by Henry and Kafura [21]. The aim of this metric is to predict a critical components. A critical component is one that is more likely to contain errors during testing, faults during operation and is more likely to be costly after faults are found [33]. If a critical component is identified early, then a CBSS designer can take appropriate action to reduce the potential problem, such as redesigning critical components or allocating additional test resources.

Fan-in and fan-out are defined with respect to individual interface as follows:

Definition 8: Fan-in of an interface “I” is the sum of inter-flows into an interface “I” plus the number of intra-flows which an interface “I” retrieves.

Definition 9: Fan-out of an interface “I” is the sum of inter-flows from an interface “I” plus the number of intra-flows which an interface “I” updates.

$$\text{Interface Information Flow (IIF)} = (\text{Fan-in} * \text{Fan-out})^2$$

The following is a step by step guide to derive the information flow metrics values for a CBSS:

1. For each interface in a component, calculate the Interface Information Flow (IIF) value of that interface using the formula below:

$$\text{Interface Information Flow (IIF)} = (\text{Fan-in} * \text{Fan-out})^2$$

2. For each component in a CBSS, sum the Interface Information Flow (IIF) values for all interfaces in that component. We will term this the Component Information Flow (CIF).

$$\text{Component Information Flow (CIF)} = \sum_{i=1}^p (\text{IIF}_i)$$

where

p = the number of interfaces in a component

3. Sum the Component Information flow (CIF) values for all components in a CBSS. We will term this the (CBSIF).

$$\text{CBS Information Flow (CBSIF)} = \sum_{i=1}^n (\text{CIF}_i)$$

where

n = the number of components in a CBSS

Kitchenham [31], Shepperd [34] and Lanza [35] have shown that the multidimensional metrics are a more effective approach in understanding, assessing and identifying problem components than any method based on a single metric. Therefore, we grouped the set of metrics to characterize and evaluate different levels of design as follows:

1. Dependency Structures of Interface (DSI)

To characterize and evaluate the dependency behavior of the interfaces we can rank the interfaces according to the Interface Coupling metrics (IC) and Interface Information Flow metrics (IIF) in a scatter plot

2. Dependency Structures of Component (DSC)

To characterize and evaluate the dependency behavior of the components we can rank components according to the Component Coupling metric (CC) and the Component Interface Information Flow metric (CIF) in a scatter plot.

3. Dependency Structures of CBSS (DS-CBSS)

To characterize and evaluate the dependency behavior of the CBSSs we can rank the CBSSs according to the CBSIF and CBSS coupling in a scatter plot.

DSI and DSC represent component level metrics while DS-CBSS represents CBSS level metrics. For CBSS level metrics, CBSS designers should compare different compositions of the same system with respect to testing and

maintenance. For component level metrics, CBSS designers should compare different component of the same system with respect to reusability of component.

V. INCORPORATING THE METRICS INTO WEB-BASED CBSS APPLICATION

To study the usefulness of our metrics, we applied them to assess the structure design of Hotel Management System (HMS) which is used in [23] as well as in [36]. Other researchers such as Mahmood and Lai [14] use a similar approach. The choice of HMS was even better since it developed according to [23], which is a good example of Szyperski’s CBSS specification methodology. Figure 5 shows HMS architecture used in the study. The HMS is a web based application that allows a user to search, reserve a hotel room and checks the availability of rooms and prices or cancels his reservation at any time. (Full details of the application can be found at [37]).

In the context of HMS the goals of the application were:

- To explain and demonstrate the capabilities of our proposed metrics and to help software engineering community gain a deep understanding of their definition and application context.
- To investigate whether the metrics results yielded intuitive information to characterize and evaluate the CBSS dependency.

A. Data Collection

Data collection was done by manual inspection of the HMS specification (i.e., components specification, interfaces specification and interaction diagrams). The CIF analysis was performed for each component in the HMS using template defines in Table 1. The following quantitative data was collected:

- The number of inter-component flows.
- The number of intra-component flows.
- The number of components.
- The number of interfaces in each component.
- The number of operations in each interface.

This information was tabulated and analyzed using Excel program. We discarded billing component from the study because we did not find enough information about it is specification. The Data were primarily collected by the first author and checked by the second and third authors independently to help avoiding bias and error. In the event of a disagreement, a negotiation took place. The results were reviewed and discussed in a formal meeting by the authors of this paper.

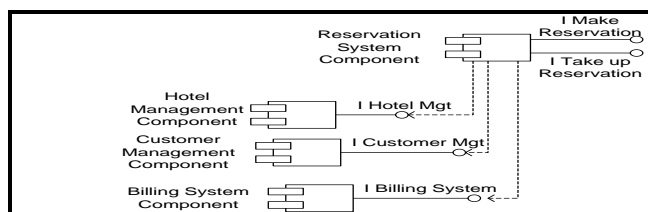


Figure 5. HMS architecture

B. Data analysis

Given the goal of producing components which have a better dependency and with respect to the concept of coupling and information flow complexity, we should interpret the coupling metric and the information flow metric in isolation to verify their functionality, since they reflect the behavior of components based on different concepts, goals and definitions. This claim should, as we understand it, not be interpreted outside the context of metrics hypothesis. Obviously, the coupling metrics reflect the behavior of components in terms of a one directional relationship (i.e., the number of inter-flows out of the component), which in turn assesses the component’s impact on the overall system. Whereas, the information flow complexity metrics reflects the behavior of components in terms of bi-directional relationship (i.e., fan-in and fan-out), which assesses the amount of information flowing to and from other components of the system.

The component dependency might be characterized as better, if the component has relatively low values of both coupling metric and information flow metric, which in turn indicates lower CBSS maintenance time and cost.

C. Result and discussion

When changing the reservation system component, we need to consider how both the hotel management component and customer management component will be affected. Whereas, when modifying either the hotel management component or customer management component, we only need to consider how the reservation system component will be influenced. According to the component coupling metric results shown in Figure 6, the coupling of reservation system component is quite high compared with hotel management and customer management components. This means that the reservation system component depends strongly on the customer management component and hotel management components. Usually, high Coupling refers to a more elusive problem [38][39]. Any changes made to a highly coupled component would probably require changes to many other components in the design. Consequently, in the future, understandability, maintainability and reusability of the reservation system component is likely to be quite difficult. The customer management component has the lowest coupling degree which means it’s the easiest to modify and reuse.

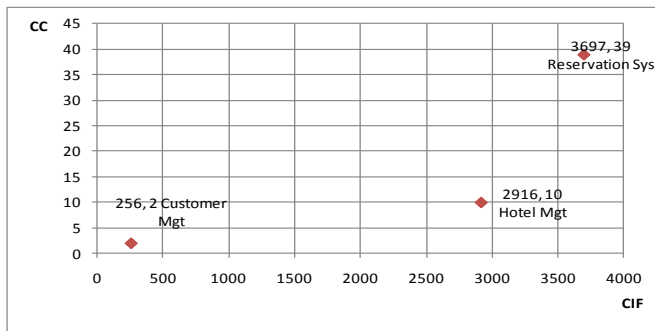


Figure 6. Dependency Structures of Components

In addition, it is interesting to note that the CIF metric values are consistent with the component coupling metric values. Empirical studies in the literature confirm that a high value of information flow measure can reveal three potential problem areas: component which possibly lack functionality, component with stress point (which means a change to it could affect other component in its environment) and/or an inadequate refinement [21].

As shown in Figure 7, in the case of IC metric, the “I make reservation” (IMR) and “I take reservation” (ITR) interfaces indicate highly coupled interfaces. Therefore, it is recommended to investigate IMR and ITR interfaces in terms of the number of other component to which each interface is coupled. The underlying theory of this metrics is that an interface should have a low coupling with other interfaces in a system. The high values of IC metric might mean that the responsibilities of their operations are not clearly defined, which in turn means that the understandability and testability of those interfaces in isolation is very hard, significantly lowering design quality. In contrast, the “I Hotel” and “I Customer” interfaces show lower coupling degree which means they can be easily tested and maintained.

The IIF metric shows interesting results when looking at the total level of information flow. The results show that “I Hotel” interface and IMR interface have relatively high values. The high value of “I Hotel” interface is due to large number of operations exposed by the “I Hotel” interface. This implies that the “I Hotel interface” and IMR interface should be redesigned or investigated by an expert.

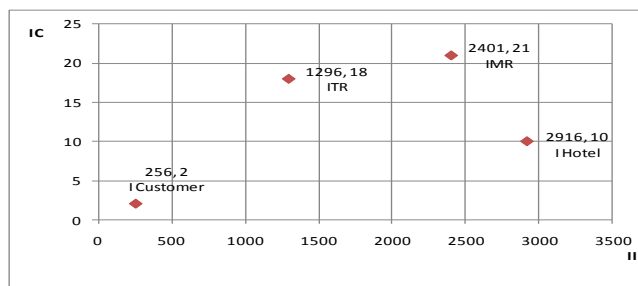


Figure 7. Dependency Structures of Interfaces

VI. CONCLUSION AND DIRECTIONS FOR FUTURE RESEARCH

In this paper, first, we proposed a method named CIF for analyzing information flow in CBSSs. We believe that the CIF is very useful, much easier to collect earlier in the lifecycle, and is a practical basis for evaluating CBSS.

Second, we proposed two sets of metrics which characterize and evaluates the dependency between components, so that CBSS designers can identify critical components in terms of error-proneness and evaluate the impact of the change on the whole CBSS in terms of the difficulty of making a corrective change, which in turn allows designers to target components that need to be revised to improve the quality of the design.

Overall, we believe that our propose metrics can become a very useful tool in help monitoring, managing and controlling test cost estimation, quality estimation and complexity analysis. The component level metrics can be used to identify complex components and/or critical components. Complex and/or critical components assembly would potentially take longer time to develop and test than a simple one. Therefore, developers, tester and maintainers with better experience and more money should be used to integrate and test critical components. For a software tester, complex components require substantial testing effort [2]. The metrics could be used as the basis of a coverage measure of testing for each component (i.e. testers should as a minimum cover all input and output flows). There are also coverage measures that can be based on combinatorial testing of the inputs. Components produced by component providers only include specifications of the interfaces. This imposes difficulties on sufficient testing of an integrated CBSS [40]. For testing such components, we need techniques that do not require the source code and instead relay mainly on the specification of system [20][41]. We believe that the CIF analysis is very useful for this purpose.

The system level metrics might be suitable for effort estimation. In particular, the CBSS metrics should be related to testing costs (since testing requires activating the information flows to confirm the functional and non-function requirements have been met). They might be used to estimate minimal set of test cases that must be run when one component is modified.

This paper represents only the beginning of the research that should be undertaken to explore this approach. So we invite researchers to comment on whether the new approach we proposed captures the real essence of component information flow or if there are areas that are left out.

ACKNOWLEDGMENT

We would like to thank Barbara Kitchenham for her ideas, comments, suggestions and support as we prepare this paper.

REFERENCES

- [1] C. Szyperski, *Component Software: Beyond Object Oriented Programming*, Second Editioned, Addison Wesley, New York, 2002,
- [2] L. Narasimhan and B. Hendradjaya, "Some theoretical considerations for a suite of metrics for the integration of software components," *Information Sciences*, vol.177, 2007, pp. 844-64.
- [3] A. De Lucia, A.R. Fasolino and M. Munro, "Understanding function behaviors through program slicing," *wpc*, 1996, pp. 9.
- [4] S. Bates and S. Horwitz, "Incremental program testing using program dependence graphs," *Proc. Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ACM, 1993, pp.384-396
- [5] K.B. Gallagher and J.R. Lyle, "Using program slicing in software maintenance," *Software Engineering*, *IEEE Transactions on*, vol.17, 1991, pp. 751-61.
- [6] G. Gui and P.D. Scott, "Measuring Software Component Reusability by Coupling and Cohesion Metrics," *Journal of Computers*, vol.4, 2009, pp. 797-805.
- [7] G. Gui and P. Scott, "Ranking reusability of software components using coupling metrics," *Journal of Systems and Softwar*, *Journal of Systems and Software*, vol.80, 2007, pp. 1450-9.
- [8] B. Li, "Managing dependencies in component-based systems based on matrix model," *Proc. Proceedings Of Net. Object. Days*, Citeseer, 2003, pp.22-25
- [9] J. Gorman, "OO Design Principles & Metrics," Online verfügbar unter <http://www.parlezuml.com/metrics/OO%20Design%20Principles%20&%20Metrics.pdf>, zuletzt geprüft am, vol.15, 2006, pp. 2009.
- [10] N.S. Gill and Balkishan, "Dependency and interaction oriented complexity metrics of component-based systems," *SIGSOFT Softw. Eng. Notes*, vol.33, 2008, pp. 1-5., <http://doi.acm.org/10.1145/1350802.1350810>.
- [11] M.A.S. Boxall and S. Araban, "Interface Metrics for Reusability Analysis of Components," *Proc. Proceedings of the 2004 Australian Software Engineering Conference*, IEEE Computer Society, 2004, pp.40
- [12] H. Washizaki, H. Yamamoto and Y. Fukazawa, "A Metrics Suite for Measuring Reusability of Software Components," *Proc. Proceedings of the 9th International Symposium on Software Metrics*, IEEE Computer Society, 2003, pp.211
- [13] O.P. Rotaru and M. Dobre, "Reusability metrics for software components," *Proc. Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications*, IEEE Computer Society, 2005, pp.24-1
- [14] S. Mahmood and R. Lai, "A complexity measure for UML component-based system specification," *Software: Practice and Experience*, vol.38, 2008, pp. 117-34.
- [15] N. Salman, "Complexity Metrics AS Predictors of Maintainability and Integrability of Software components," *Journal of Arts and Sciences*, 2006,
- [16] L. Kharb and R. Singh, "Complexity metrics for component-oriented software systems," *SIGSOFT Softw. Eng. Notes*, vol.33, 2008, pp. 1-3., <http://doi.acm.org/10.1145/1350802.1350811>.
- [17] A. Sharma, P.S. Grover and R. Kumar, "Dependency analysis for component-based software systems," *SIGSOFT Softw. Eng. Notes*, vol.34, 2009, pp. 1-6., <http://doi.acm.org/10.1145/1543405.1543424>.
- [18] S.M. Alhazbi, "Measuring the complexity of component-based system architecture," *Proc. Information and Communication Technologies: From Theory to Applications*, 2004. *Proceedings. 2004 International Conference on*, 2004, pp.593-594
- [19] M.E.R.V.M.S. Dias and D.J. Richardson, "Describing Dependencies in Component Access Points," *Proc. Proceedings of the 4th Workshop on Component Based Software Engineering*, 23rd International Conference on Software Engineering, 2001,
- [20] S.D. Cesare, M. Lycett and R.D. Macredie, *Development of Component-based Information System*, Prentice Hall of India, New Delhi, 2006,
- [21] S. Henry and D. Kafura, "Software Structure Metrics Based on Information Flow," *IEEE Trans. Softw. Eng.*, vol.7, 1981, pp. 510-8., <http://dx.doi.org/10.1109/TSE.1981.231113>.
- [22] M. Abdellatif, A.b.M. Sultan, A.A. Abdul Ghani and M. Jabar, "A mapping Study to Investigate Component-based System Metrics,"
- [23] J. Cheesman and J. Daniels, *UML Components: A Simple process for Specifying Component Based Software*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2001,
- [24] I. Crnkovic, B. Hnich, T. Jonsson and Z. Kiziltan, "Specification, implementation, and deployment of components," *Commun ACM*, vol.45, 2002, pp. 35-40.
- [25] M.M. Pickard and B.D. Carter, "A field study of the relationship of information flow and maintainability of COBOL programs," *Information and Software Technology*, vol.37, 1995, pp. 195-202.
- [26] E.B. Allen, T.M. Khoshgoftaar and Y. Chen, "Measuring coupling and cohesion of software modules: an information-theory approach," *Proc. metrics*, Published by the IEEE Computer Society, 2001, pp.124
- [27] F. Xia, "On the concept of coupling, its modeling and measurement," *Journal of Systems and Software*, vol. 50 pp. 75-84. 2000.

- [28] W. Khlif, N. Zaaboub and H. Ben-Abdallah, "Coupling metrics for business process modeling," WSEAS Transactions on Computers, vol.9, 2010, pp. 31-41.
- [29] L. Sallie, "Object-oriented metrics that predict maintainability," J.Syst.Software, vol.23, 1993, pp. 111-22.
- [30] L.C. Briand, S. Morasca and V.R. Basili, " Measuring and assessing maintainability at the end of high level design," Proc. Software Maintenance, 1993. CSM-93, Proceedings., Conference on, IEEE, 1993, pp.88-87
- [31] B.A. Kitchenham and S.J. Linkman, "Design metrics in practice," Information and Software Technology, vol.32, 1990, pp. 304-10.
- [32] D. Ince C. and M. Shepperd J., " An empirical and theoretical analysis of information flow-based system design metrics," Proc. 2nd European Software Engineering Conf, Springer Verlag, 1989,
- [33] K. El-Emam, "A methodology for validating software product metrics," 2010,
- [34] M. Shepperd, "Measurement of structure and size of software designs," Information and Software Technology, vol.34, 1992, pp. 756-62.
- [35] M. Lanza and R. Marinescu, Object-Oriented Metrics in Practice: Using software Metrics to Characterize, Evaluate, and improve the Design of Object-Oriented Systems, Springer, Berlin Heidelberg - Germany, 2006,
- [36] M. Heisel and J. Souquières, "Adding Features to Component-Based Systems," Objects, Agents, and Features, vol. 2975 pp. 25-36. 2004.
- [37] "<http://www.umlcomponents.com>," August/8/ 2011.
- [38] L. Briand, S. Morasca and V.R. Basili, "Defining and validating high-level design metrics," pp. 31. 1994.
- [39] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Trans. Softw. Eng., vol.20, 1994, pp. 476-93., <http://dx.doi.org/10.1109/32.295895>.
- [40] Y. Wu, M.H. Chen and J. Offutt, "UML-based integration testing for component-based software," COTS-Based Software Systems, 2003, pp. 251-60.
- [41] E.J. Weyuker, "Testing component-based software: A cautionary tale," Software, IEEE, vol.15, 1998, pp. 54-9.

Module Interactions for Model-Driven Engineering of Complex Behaviour of Autonomous Robots

Vladimir Estivill-Castro

School of ICT / IIS

Griffith University, Nathan Campus

Brisbane, Australia

Email: v.estivill-castro@griffith.edu.au

René Hexel

School of ICT / IIS

Griffith University, Nathan Campus

Brisbane, Australia

Email: r.hexel@griffith.edu.au

Abstract—In this paper, we describe a model-driven engineering approach that enables the complete description, validation, verification and deployment of behaviour to autonomous robots, directly, and automatically from the models. This realises the promises and benefits of model-driven engineering, such as platform-independent development and behaviour traceability. However, such a top-down approach of modelling by finite-state machines and sub-machines creates a conceptual challenge to the behaviour designer due to the complex interaction of independent modules. Simply finding which modules are necessary for other modules can be a challenge. We also describe here our solution to this. Interestingly, our approach goes in the opposite direction of Object Oriented Software Engineering as currently represented by the Unified Modeling Language and corresponding software processes. That is, typically, the static models are derived first (and in particular class diagrams), while dynamic modelling follows later with behaviour diagrams and interactions diagrams. We actually start with the description of behaviour in finite state machines and we complement this by static information provided by logics that describe concepts and by our dependencies diagrams that show static dependencies between modules.

Index Terms—Automation of Software Design and Implementation. Software Modeling. Model-Driven Engineering. Visual Modeling.

I. INTRODUCTION

Model-driven engineering raises the level of abstraction in software engineering so that engineers no longer have to be concerned with programming language details or the specifics of execution platforms. We show here an approach where executable software is generated automatically from models. We show that we can easily adapt to new platforms and behaviour requirements and illustrate this with the development of the complex software that constitutes the RoboCup challenge. The Mi-Pal team, qualified for RoboCup-2011, uses this approach to compose the programs that constitute the behaviour and execute on the humanoid autonomous robot platform.

We aim at systems at higher levels of abstraction. Our first toolset for a higher level of abstraction are logics, and in particular logics that emulate common reasoning. We argue for logics that describe a context by iterative refinement and are natural and analogous to how humans describe a context, starting from the most general case, then proving extensions or refinements. Similarly, our second tool is behaviour captured by a hierarchy of finite state machines (FSMs). This enables

iterative refinement, describing the most general behaviour, which is then refined by a finite state sub-machine (sub-FSM).

For this reason, we use models at different levels of abstraction. From a high-level, platform independent model, it is possible to generate a working program without manual intervention. We describe this approach but we focus here on the technologies and infrastructure to facilitate design, verification and validation of inter-module communication. Other research publications expand on the details and technologies that have enabled this approach. In particular, we have discussed [1][2] the advantages of using non-monotonic reasoning to express in logic what otherwise becomes laborious and error-prone in an imperative programming language. For example, sanity checks on the landmarks reported by a vision system significantly benefit from their abstraction into logic rules. In fact, logic and iterative refinement are common in expressing and describing a concept. The *off-side rule* in soccer is an example that starts with “Usually a player is not off-side” (a default situation); then progressively some exceptions are presented. For example, “Unless two opponent players are between [a player] and the opponents’ goal line”, but then exceptions of the exception continue, forming the definition [3].

Modelling by FSMs, where the labels for transitions can be statements in a logic that demand proof, has been contrasted with plain FSMs, Petri nets, and Behavior trees (relevant behaviour modelling techniques in software engineering) using the very prominent example of modelling the behaviour of a microwave oven [4]. Our approach produces smaller models, clarifies requirements and we can generate implementations for diverse platforms and programming languages, e.g., the same models can generate code in Java for a Lego Mindstorm (www.youtube.com/watch?v=iEkCHqSfMco) as well as C++ for a Nao (www.youtube.com/watch?v=Dm3SP3q9_VE). The modelling of a microwave is a classical example in the literature of software engineering [5][4] as well as model-checking [6, Page 39] as the safety feature of *disabling radiation when the door is open* is an analogous requirement to the famous case of faulty software on the Therac-25 radiation machine that caused harm to patients [7, Page 2].

We have illustrated [8] the power of non-monotonic logic to describe and complement the descriptions of Behavior

trees and of fine state machines for requirements engineering. Further illustrations [9] show the benefits of this idea in the context of embedded systems and robots. The software engineering architecture and the software design patterns that support our model-driven engineering are based on a whiteboard architecture [10][11]. This offers a cognitive architecture [12] or a *working memory* as well as a publisher-subscriber pattern for module communication, analogous to what others have called a repository architecture [5], or Data-Distribution Service [13]. Our whiteboard architecture is complementary to Aldebaran's inter-module communication and messaging architecture in the Red-Documentation.

Our interest for high-level modelling is that RoboCup, and in particular the Standard Platform League is an important benchmark for the deployment of legged robots in human environments (with RoboCup@Home also promoting this in the home or office). Therefore, there is a clear overlap with concerns in the field of software engineering, such as reliability, safety, human-computer interaction, requirements engineering, platform independence, composability, distribution, simplicity, and most importantly, model-driven engineering.

However, a challenging aspect of our approach is to model the interactions between modules, and to have a tool that enables the display of modules dependencies as behaviour designers integrate the behaviours of a complex system. Because we had shown an equivalence between FSMs and Behavior trees [8], we could translate our models to tools like BECCIE [14] that capture some of the module interactions, and this was sufficient for the already mentioned example of the micro-wave oven [9]. However, BECCIE's limitations do not enable this to scale further. Here we illustrate the new tools we have developed to achieve this.

The rest of this paper is structured as follows: Section II exemplifies the approach used. Section III shows how module interactions are modelled and what the consequences are for complex behaviour and iterative refinement. The paper is concluded with a discussion in Section V.

II. MODEL-DRIVEN ENGINEERING

We present a case study in the context of the SPL for RoboCup-2011 to illustrate our model-driven engineering approach, considering the FSM that playing robots are supposed to conform to. The model for this appears on page 7 of the SPL rules, and essentially indicates that the league's game controller would emit UDP packets (or a manual push of the chest button) for the playing robots to update their state. As in any requirements engineering scenario, the rules are under-specified and ambiguous – more seriously the actual SPL game controller (server) does not follow nor enforce the specified transitions. For example, Figure 2a and Figure 2b show the current activities for the state INITIAL and for the state READY (both corresponding to a state of the behaviour required by the competition). An `OnEntry` activity is to post (to the whiteboard) the message type `NaoMotionPlayer` (whose listener is `gunaomotion` with the message content `play_get_up_anywhere`, which is a pre-loaded motion

that stands up the Nao). Also in this state and also `OnEntry`, we post message type `LEDS` whose listener is `gunaoleds` to turn the ChestBoard off.

However, we do model our `guGameController` FSM for the behaviour executed by our robots for participation in RoboCup 2011. Figure 1 is produced with *Qfsm* (`qfsm.sf.net`), a graphical tool for designing FSMs. This produces XML files that our own tool, `qfsm2gu`, translates into to ASCII files. These files contain the *transition table* of the FSM and *activities* for each state of the automaton. Our FSMs are interpreted by our `gubehaviourinterpreter` module that, e.g., for `guGameController.fsm` reads the transition table from the file `TguGameController.txt` (transition files always start with the letter T), and the activities from `AguGameController.txt` (activity files start with A).

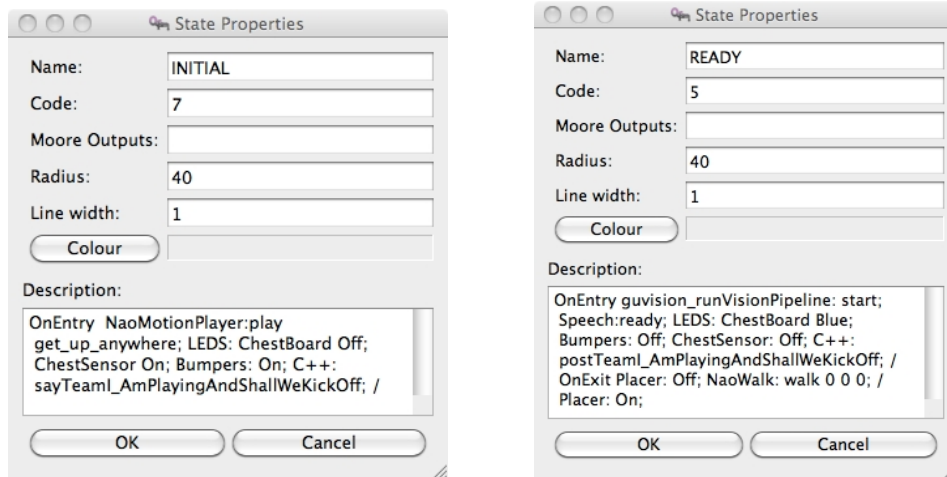
A. The semantics of our finite state machines

There are some important aspects of the interpreter of FSMs that represent behaviour. First, the transitions out of a state are not evaluated simultaneously, but they are evaluated in reverse order of their appearance in the transition file. Importantly, this liberates the behaviour designer of the concern of ensuring that only one transition can fire at any one time. In a sense, this provides a priority relation between the transitions and can be specified explicitly with *Qfsm* in the output field of a transition.

Second, the label of a transition is a query to an expert to make a proclamation about the truth value of that label. The interpreter will halt, waiting for a response on the *whiteboard* for this particular message type that indicates this proposition requires proof, typically by a logic inference engine – `gucdlmodule` that implements *Propositional Clausal Defeasible Logic* [15] (but, we have an implementation for standard prolog as well using `gnuprolog`). However, many times, the question is directly related to a sensor. That is, the best expert to ascertain the truth value of the transition label is a wrapper for a sensor (providing information about anything external to the system). For example, in the `guGameController.fsm` of Figure 1, a label `UDPSaysRedKickOff` is a query, but is answered by `guUDPReceiver`, which is the actual module connecting to the league's UDP server that can assert if the league's game controller is now broadcasting that the red team is to kick-off. There is a special label `TRUE` that always fires and causes a state transition.

It is important to highlight that the behaviour interpreter, the logic engine, and many of our modules are developed to conform to the POSIX standard (and therefore not only execute on the Nao but also, e.g., Linux, and MacOS). This enables module simulation, developing and testing independently of the platform. In particular, one can impersonate an expert by using our `testcdl` module and a FSM is oblivious to this.

We can use the example of `guGameController.fsm` to stress which of our modules provide the interface between the whiteboard and the Nao platform. In addition to `guUDPReceiver`, the following modules must run: `gunaobuttonsensor` for button-press events and



(a) The activities of INITIAL. (b) The activities of READY.
 Fig. 2. Display of activities in two states of `guGameController.fsm` using `qfsm`.

`gupositionsensor` to detect if (and which way) the robot has fallen.

Other modules are *actuators* that send a message or produce and effect on the environment external to the system. Actuators are typically subscribers through the whiteboard to postings by FSMs. It is important to understand that a state has essentially two types of *activities*, postings to the whiteboard or execution of some C++ code. The possibility to integrate C++ code means that any behaviour that we do not represent as FSMs can also be integrated into our modelling. The *activities* in our state machines are classified into three different execution steps (following very much the conventions of state machines for modelling Object Oriented Systems in OMT [16], UML, and may other standards for state machines).

- **On Entry:** These activities are executed at least once, and always just once and before any other activity upon arriving at the state.
- **On Exit:** These activities are executed at least once, and always just once and after any other activity upon leaving the state.
- **Internal activities:** These activities may not be executed at all. They are executed once, every time the entire set of leaving transitions has been tested (against the corresponding expert) and determined no transition fires.

Evaluation of leaving transitions and execution of internal activities is repeated until a transition fires that moves the machine to a new state.

Actuators that listen to messages posted by the `guGameController` state machine include the following.

- `gunaoleds`: The interface to illuminate Nao’s ears, face, feet and the chest button.
- `gunaospeechmodule`: The robot speaks to identify itself.
- `gunaomotion`: The interface to actions like to get up if the robot is lying down.

In the C++ code, there is a method named `sayTeamI_AmPlayingAndShallWeKickOff`. This

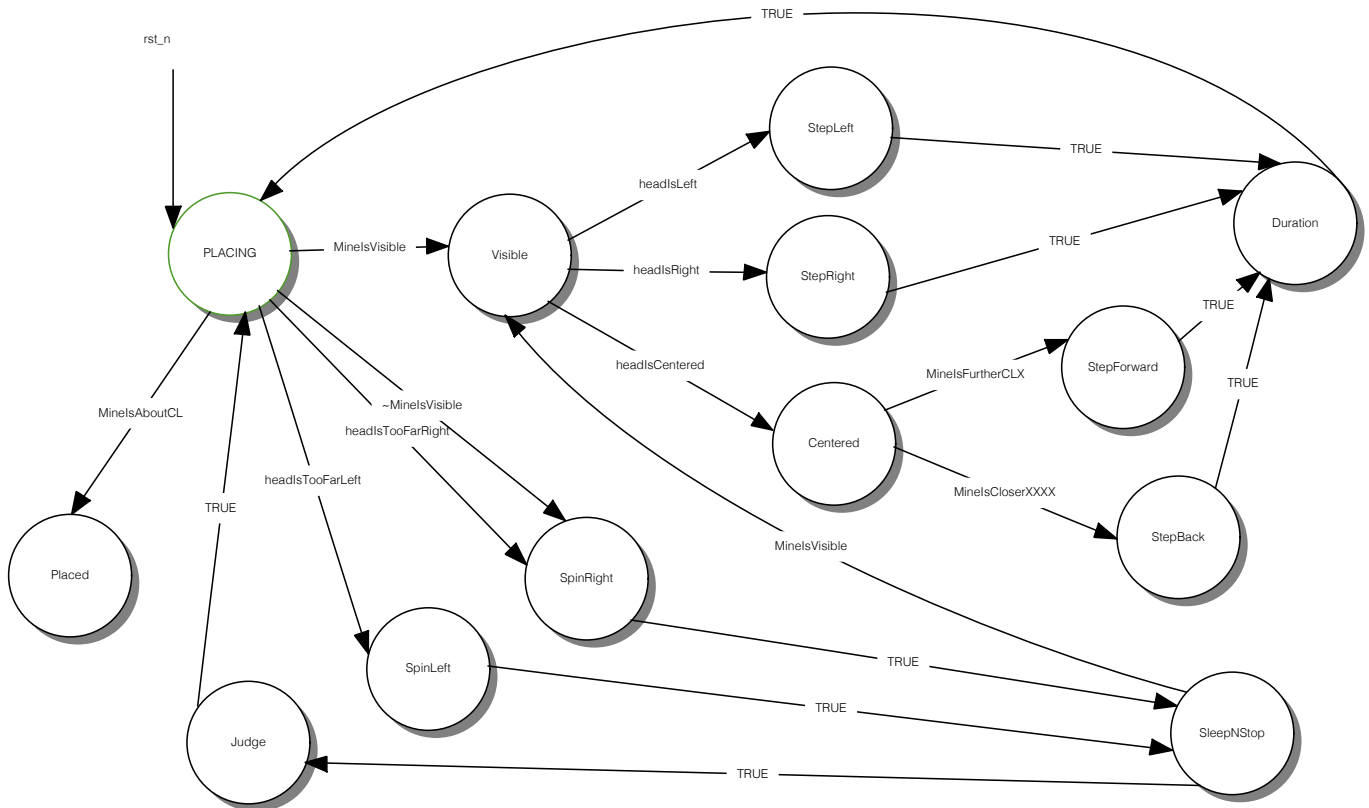
routine uses C++ variables that record the integer number (player number) and the team as red or blue. This could also be modelled by states, but the state machine would basically be a clone of itself for playing red and for playing blue. Thus, this illustrates that sometimes clarity (and generality) is achieved with some algorithmic C++ code (rather than duplicating all the states). The values of the borrowed code are initially supplied on the command line but are updated by the `guGameController` state machine as the event from the league game controller demands via UDP.

B. The abstraction power of sub-machines

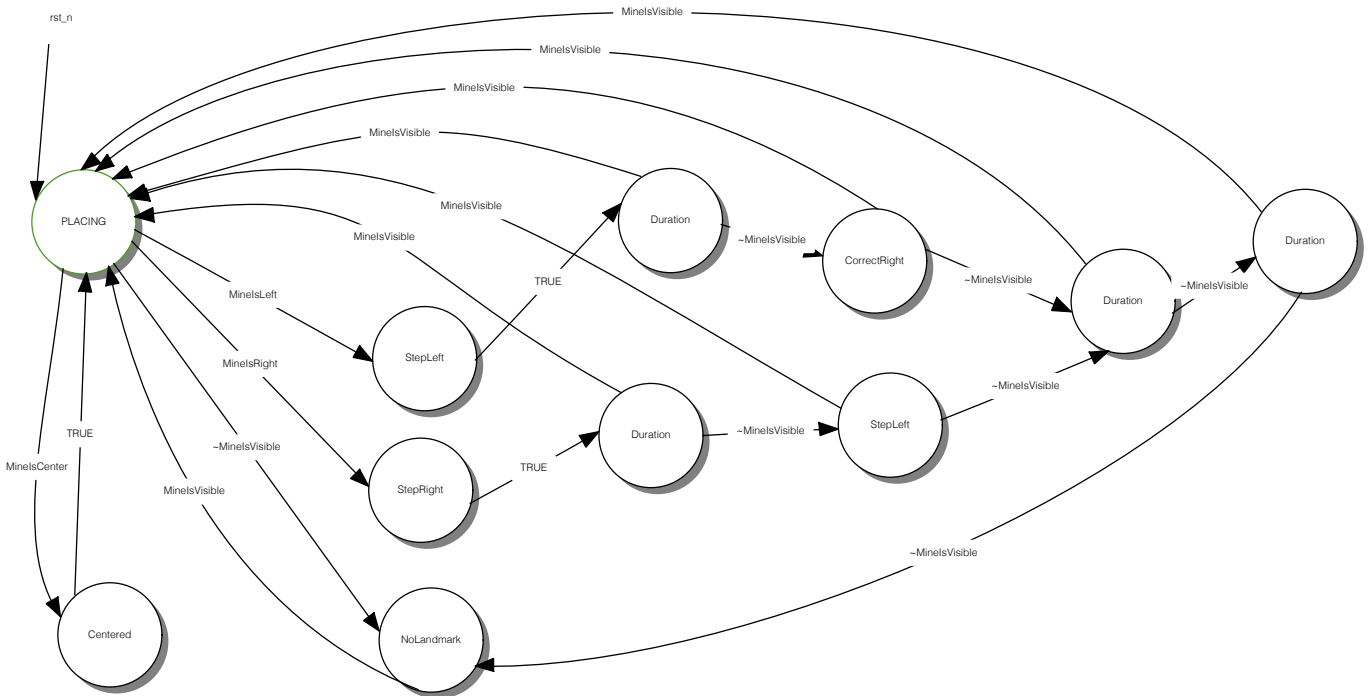
While everything that is required for the SPL in the INITIAL state is defined in the corresponding state of the `guGameController.fsm`, this is not the case for the READY state. There are many things that are done directly here in the `OnEntry` section, such as starting the vision pipeline in the module `gunaovision`. Also the ChestBoard LED is set to blue, and the sensitivity of the buttons is turned off (otherwise, the feet bumpers sense events just by walking).

So, how to achieve the behaviour that in state READY the robot is to find its correct position within the field before the state SET? The posting of the message with a type corresponding to the name of a sub-FSM starts a previously dormant automaton. In this case `Placer: On` (the message content is `On`). The `OnExit` activity is a posting of `Placer: Off` that makes this sub-machine dormant.

Sub-machines are a sub-class (in the C++ and object-oriented sense) of FSMs with the additional feature that they can be suspended or resumed. The `Placer.fsm` sub-machine (Fig. 3a) uses an implementation of a Kalman filter for localisation inside our module `gulocalizationfilter`. It uses walks from `gunaomotion` to walk until it is 150 cm from its own goal. The localisation module listens to he whiteboard for postings by `gunaovision` of landmark sightings (for its internal sensor model) and also to the walk commands for its internal motion model. `Placer.fsm` uses



(a) The Placer.fsm sub-machine.



(b) The GoalTracker.fsm sub-machine.

Fig. 3. Qfsm model of two READY-state sub-FSMs of guGameController.fsm.

gunaosensor (and Naophysical) to know if the head angle relative to the body is pointing straight or sideways (in order to post suitable commands to motion; the step to move

in the direction of a landmark may be to walk forward or to walk to the side or even to spin). The GoalTracker uses only the gulocalizationfilter filtering to post

(provider) and consumer (listener) message types, the modules are completely analogous.

Our models can now be generated completely automatically from the dictionary of messages, which in turn is generated directly from the code (in fact it is part of the code). The result is a series of diagrams that show dependencies for each module as well as overall module dependencies. For example, Figure 4 shows the module concerned at the centre of the diagram. Those modules that are suppliers to `guGameController.fsm` are shown in the upper row of modules, and the green arrows show that these are experts, queried in labels of the FSM, about the change of state. In this case, `guGameController.fsm` will be making blocking calls to these modules requesting they make a proclamation on a particular proposition (evaluating to `true` or `false`) that labels the transition. The bottom row of modules are those to whom the module `guGameController.fsm` will be posting a message. The black arrows indicate also this is a non-blocking interaction while the direction of the arrow indicates who is the provider of a message and who is the listener. In this illustration we have chosen a faulty version of a FSM that posts a message not recognised in the dictionary, i.e., no listening module has been found. Therefore, we see the word “unknown” in red as a destination of a message. This warns the behaviour designer that there is a fault in the current design of interactions of the software, at least with the respect of the behaviour specified by this FSM.

Discussion

What additional advantages besides the correctness of module interactions does this provide? The behaviour designer can now configure particular testing, verification and validation plans, and the corresponding script can be generated automatically. For example, by looking at the corresponding diagram for a module, and indicating associated modules, a particular script can be rapidly configured for testing the chosen module on a particular platform. The script will only start those modules necessary for interaction and support of the module under scrutiny, and therefore significant resources of compilation, porting to the platform, and test configuration are saved. Let's recall the importance of testing [5, Chapter 7] and in particular testing automation and early validation; the sooner we verify a change and test that we have not introduced a fault or broken the current functionality the better. This leads to more traceability, to more reliability and to more robustness in the software process and the product itself.

Why not use UML's collaboration diagrams or UML's sequence diagrams (or some other sort of UML interaction diagram)? Simply because such UML diagrams are used to model the dynamic behaviour of the system. They represent a particular trace of execution. The order and time of message passing is the principal aspect. Our FSMs are already the dynamic model. In fact, our proposed diagrams here represent static information; they are a static model of the software on-board of the robot. This is precisely why they are so useful in configuring versions and identifying the modules that together

integrate a module. Thus, the diagrams here are in fact more analogous to UML composite structures [17]. In fact, it is trivial to convert the dependency information on whiteboard message suppliers and listeners to corresponding ports therein. However, this would not capture the fact that the responsibility for such compositions are factored out from the individual modules (as we already mentioned, our software architecture is actually a repository architecture in the terminology of Sommerville [5, Chapter 6] or whiteboard architecture [10][11]).

IV. OTHER ASPECTS

Some features in our approach that enable further powerful, high-level control on the behaviours for the robots are

- 1) to dynamically load a behaviour (a FSM) at any time and not only at start-up, and
- 2) to dynamically modify vision pipelines, so the camera feed (upper or lower camera) is adjusted, based on FSM context.

We mentioned that the FSMs (or sub-machines) that model our behaviour are in fact encoded as two tables: the transitions table and the actions table. The capability to read, parse and have an internal representation of the FSM is not only used at start-up time, but can be used on demand. In the example discussed earlier regarding the model of the Game Controller, the robot can, during execution, re-load the transition table from the file `TguGameController.txt` and the activities from `AguGameController.txt`. Once the corresponding parsing and internal representation are ready for the interpreter, this refreshed behaviour can take over. This parsing and re-building of the internal representation is not a CPU-intensive operation. The grammar of the transition table and the activities table is very straightforward and the internal representation is not particularly different from a graph representation of the FSM as the diagrams we have been displaying. Namely, our class `fsmMachine` that represents a behaviour model is a vector of `fsmStates`. An object of the class `fsmState` has a `stateID`, `stateName`, a vector of `fsmTransitions` and an `fsmActivity` object. An `fsmActivity` object has postings and/or callbacks for each of three possibilities: `OnEntry`, `OnExit` and `Internal`. An object of the class `fsmTransition` can hold an expression to evaluate.

Granted, this parsing must be combined with the facilities that enable sub-machines. That is, sub-machines can be paused (and therefore become dormant), and later be resumed from their initial state. Therefore, a dormant sub-machine can be re-loaded without the need to halt the whole robot. Moreover, re-loading a sub-machine can be part of a behaviour. Therefore, this opens the door to the possibility of the robot learning or adapting its behaviour while operating, by simply modifying the behaviour model during execution (however, such a learning behaviour is not implemented yet).

Once the concept of a model being able to be loaded during runtime and not only during start-up is available it is not difficult to see that a linear software architecture, such as a pipeline (also known as a pipe and filter architecture [5,

Section 6.3.4]), can easily be modified and adapted with specific commands during runtime. This is what enables the second aspect mentioned above.

The advantages provided by these facilities are many. For example, they can be used as a powerful mechanism for a faster and more reliable software development cycle for the robot (and, in general, for embedded systems). To illustrate this, it is enough to consider what the testing of a behaviour demands if the robot needs to be shut down every time a new behaviour is loaded. Typically, re-booting a robot such as the Nao is quite time consuming, and requires placing the robot in a safe position, e.g., to physically prevent the robot from falling. The boot process is slow, because it is not only the operating system that needs to be loaded, but also all the middleware that enables the hardware subsystems, and any other modules that the behaviour uses and that are part of the system as a whole. As we alluded earlier, in the case of playing robotic soccer, these include many modules for motion, vision, sonar, actuators, etc. In general, which modules are required for a behaviour is determined by our new diagrams illustrating module dependencies. Dynamically loading a behaviour (or a sub-behaviour as a sub-machine) enables iterative refinement and testing of new behaviour, without the lengthy delay of re-booting the robot for every single modification of the behaviour model. This facilities and speeds up the testing of every behaviour. The more a behaviour is tested, the more reliable it becomes.

V. CONCLUSION

We have described our model-driven engineering approach to software development. We can completely develop the behaviour of autonomous humanoids robots through models that consist of

- 1) models for logics that describe the domain knowledge and the declarative part of the system,
- 2) models for the action part of the system, that are visualised by finite state machines, or state diagrams.

However, understanding the interactions, the service available, and the request that will be made to service providers needs validation and visualisation. We have described the mechanisms to obtain such diagrams and the benefits they provide to software development.

Nevertheless, there are also some aspects of our infrastructure that constitute immediate targets for further work;

- to expand even further the vocabulary of messages the behaviour interpreter can use when requesting a proof so we can use other inference engines,
- to add priorities to the messages on the whiteboard, so we can have a subsumption architecture, and
- to add a planning module (so we can apply the infrastructure to other environments besides soccer, that demand more planning and are less reactive).

ACKNOWLEDGEMENTS

The authors would like to thank Andrew Rock and David Billington for fruitful discussions and collaboration in the

conceptual idea of model-driven engineering of behaviour of autonomous robots.

The authors also thank Carl Lusty, Steven Kuok, and Vitor Bottazzi who helped significantly in the programming of many of the modules and tools used in the practical illustration of this approach, which is the large software environment and system that is the code for the RoboCup Standard Platform League.

REFERENCES

- [1] D. Billington, V. Estivill-Castro, R. Hexel, and A. Rock, "Non-monotonic reasoning for localisation in robocup," in *Australasian Conference on Robotics and Automation*, C. Sammut, Ed. Sydney: Australian Robotics and Automation Association, December 5-6 2005.
- [2] —, "Using temporal consistency to improve robot localisation," in *RoboCup 2006: Robot Soccer World Cup X*, ser. Lecture Notes in Computer Science, G. Lakemeyer, E. Sklar, D. G. Sorrenti, and T. Takahashi, Eds., vol. 4434. Springer, 2006, pp. 232–244.
- [3] —, "Chapter 3: Non-monotonic reasoning on board a sony AIBO," in *Robotic Soccer*, P. Lima, Ed. Vienna, Austria: I-Tech Education and Publishing, 2007, pp. 45–70.
- [4] —, "Non-monotonic reasoning for requirements engineering," in *Proceedings of the 5th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*. Athens, Greece: SciTePress (Portugal), 22-24 July 2010, pp. 68–77.
- [5] I. Sommerville, *Software engineering (9th ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2010.
- [6] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT Press, 2001.
- [7] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT Press, 2008.
- [8] D. Billington, V. Estivill-Castro, R. Hexel, and A. Rock, "Plausible logic facilitates engineering the behavior of autonomous robots," in *IASTED Conference on Software Engineering (SE 2010)*, R. Fox and W. Golubski, Eds. Anaheim, USA: ACTA Press, February 16 - 18 2010, pp. 41–48, location: Innsbruck, Austria.
- [9] —, "Modelling behaviour requirements for automatic interpretation, simulation and deployment," in *SIMPAR 2010 Second International Conference on Simulation, Modeling and Programming for Autonomous Robots*, ser. Lecture Notes in Computer Science, N. Ando, S. Balakirsky, T. Hemker, M. Reggiani, and O. von Stryk, Eds., vol. 6472. Darmstadt, Germany: Springer, November 15th-18th 2010, pp. 204–216.
- [10] B. Hayes-Roth, "A blackboard architecture for control," in *Distributed Artificial Intelligence*, A. H. Bond and L. Gasser, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988, pp. 505–540.
- [11] D. Billington, V. Estivill-Castro, R. Hexel, and A. Rock, "Architecture for hybrid robotic behavior," in *Hybrid Artificial Intelligence Systems, 4th International Conference, HAIS 2009, Salamanca, Spain, June 10-12, 2009. Proceedings*, ser. Lecture Notes in Computer Science, E. Corchado, X. Wu, E. Oja, Á. Herrero, and B. Baruque, Eds., vol. 5572. Springer, 2009, pp. 145–156.
- [12] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, Inc., 2002.
- [13] C. H. Wu, W. H. Ip, and C. Y. Chan, "Real-time distributed vision-based network system for logistics applications," *Int. J. Intell. Syst. Technol. Appl.*, vol. 6, pp. 309–322, March 2009. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1521389.1521397>
- [14] L. Wen and R. G. Dromey, "From requirements change to design change: A formal path," in *2nd International Conference on Software Engineering and Formal Methods (SEFM 2004)*. Beijing, China: IEEE Computer Society, 28-30 September 2004, pp. 104–113.
- [15] D. Billington, "Propositional clausal defeasible logic," in *Proceedings of the 11th European conference on Logics in Artificial Intelligence*, ser. JELIA '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 34–47. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87803-2_5
- [16] J. Rumbaugh, M. R. Blaha, W. Lorensen, F. Eddy, and W. Premerlani, *Object-Oriented Modelling and Design*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1991.
- [17] *OMG Unified Modeling Language (OMG UML), Superstructure, V2.3*. Object Management Group, May 2010, ch. 9, Composite Structures, pp. 167–198.

Case Study for a Quality-Oriented Service Design Process

Michael Gebhart, Suad Sejdovic, Sebastian Abeck

Research Group Cooperation & Management

Karlsruhe Institute of Technology (KIT)

Karlsruhe, Germany

{gebhart | sejdovic | abeck} @kit.edu

Abstract—Due to the usage of distributed information, such as sensor information, geographical information systems are designed according to service-oriented principles. Thus, the development of new solutions within this context requires a design of necessary services. These services have to follow certain quality attributes that have evolved as important for services, such as loose coupling and autonomy. In this paper, a quality-oriented design process is considered and its applicability and effectiveness are shown within the Personalized Environmental Service Configuration and Delivery Orchestration project of the European Commission.

Keywords—service; design; quality; geographical information system; case study

I. INTRODUCTION

Today, geographical information systems use distributed information, such as sensor information, that measures environmental data, such as air temperature, or presume precipitation. This information is provided by public authorities or private sectors in form of services [8]. The geographical information system acts as service consumer, thus sends requests to the services and receives according responses. Additionally, functionality of the geographical information system can also be provided in form of services in order to enable the realization of systems at a higher level.

Accordingly, the development of such geographical information systems requires a design of necessary services in order to support the usage of distributed information and the provision of functionality that bases on this information. The design of services consists of two elementary phases, the identification and the specification [1, 2, 9, 10, 11, 25]. During the identification phase service candidates as proposals for services and their dependencies are formalized [5, 6]. Each service candidate includes a set of operation candidates that represent preliminary operations. A dependency between service candidates describes that a service requires another service for fulfilling its functionality. Within the specification phase, the final specifications of the services are created. Each specification constitutes a so-called service design and consists of a specification of the service interface and the realizing service component. The service interface describes provided and required operations, message and data types, interacting roles and the interaction protocol [7]. The specification of the service component determines the services provided by the

realizing component and the services required for fulfilling the provided functionality. Additionally, the internal behavior in form of a composition of own functionality and functionality provided by other services is formalized.

For services several quality attributes have been identified that should be fulfilled in order to attain goals that are associated with the application of service-orientation, such as an increased flexibility [5, 6, 12, 14, 15, 20, 30], reusability [5, 21], or maintainability [19] of provided functionality. Wide-spread quality attributes that support these goals are a unique categorization, loose coupling, discoverability, and autonomy [2, 6, 13, 16, 17, 18, 19]. Since these goals are also important for geographical information systems, the quality attributes should be considered when designing new services in the context of a geographical information system. This requires a quality-oriented service design process when developing a service-oriented geographical information system.

In the context of the project Personalized Environmental Service Configuration and Delivery Orchestration (PESCaDO) [3, 4] of the European Commission, a service-oriented geographical information system has to be developed in cooperation with the Fraunhofer Institute of Optics, System Technologies and Image Exploitation. This system enables getting personalized information regarding the personal profile and environmental conditions. Since the services should fulfill quality attributes, such as loose coupling, a quality-oriented service design process has to be applied. For this purpose, the design process created by the authors of this paper as introduced in [1] has been applied. This design process includes a transfer of artifacts of the business analysis phase into artifacts of the design phase and considers a certain set of quality attributes. In this case, the quality attributes of a unique categorization, loose coupling, discoverability and autonomy are regarded using the quality indicators as introduced in [2]. This case study shows how to apply the design process for a geographical information system of a real world project and demonstrates the applicability and effectiveness of the design process.

The paper is structured as follows: Section 2 introduces the PESCaDO project and the considered service design process. In Section 3, the design process is performed in order to design the necessary services for PESCaDO. In this context, the artifacts of the design phase are systematically derived and revised subsequently. Section 4 concludes the paper and offers suggestions for future research.

II. FUNDAMENTALS

In the following, the PESCaDO project and the considered scenario of this project are introduced. Additionally, the quality-oriented service design process that is applied for designing the required services is described.

A. Personalized Environmental Service Configuration and Delivery Orchestration

Nowadays, more and more people are aware of the influence that environmental conditions can have on the quality of their life. Since each individual has the need for specific information about the environment that is affecting him and his life, information personalization plays a major role.

The PESCaDO project of the European Commission [3, 4] takes up this issue and aims at developing a platform for getting personalized information regarding the personal profile, such as health status, mode of presentation or language of an individual, and also takes into consideration the intention of the individual. PESCaDO covers the discovery of services providing the data, their orchestration, the processing of the data and the delivery of the gained information. In terms of reusability, technology independence and the flexible usage of existing functionalities, a service-oriented approach should be pursued [5, 6, 12, 14, 15]. The resulting services are expected to consider the quality attributes of a unique categorization, loose coupling, discoverability, and autonomy. These attributes are chosen, because they can be evaluated during design time [1, 2]. Quality attributes, such as statelessness, require implementation information.

Within a first prototype, the data access functionality has to be developed. One special requirement is the semantic support for accessing environmental data. Thus, the system has to be capable to identify all related data sources for a requested phenomenon like temperature. For this purpose, it has to be able to extend a single requested phenomenon by other related ones. For example, if the system has identified the phenomenon "Pollen" as relevant, it also will have to retrieve information about more specific phenomena, like "Birch Pollen". For achieving this goal, the system uses a knowledge base, which contains a related ontology. The focus in the development of the first prototype lies on the extension of the requested phenomenon and accessing the related data in the background.

B. Quality-Oriented Service Design Process

The quality-oriented service design process, which is illustrated in Figure 1, starts with the business analysis phase that yields artifacts that constitute the input for the service design phase. The primary goal of this phase is the identification and modeling of the considered business use cases and the realizing business processes [9, 10]. The artifacts use terms as introduced within the domain model for a common understanding. The business processes can consider already existing services in order to increase the reuse of functionality. This means, that the activities within the business process are aligned with the operations of existing services regarding their granularity and names.

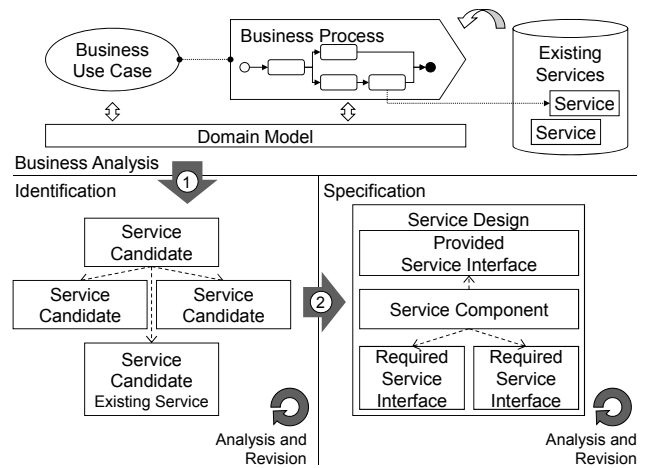


Figure 1. Quality-oriented service design process.

Within the service design phase, two activities have to be performed. In a first step service candidates are systematically identified by using the modeled business processes of the analysis phase. Afterwards, these service candidates are analyzed and revised according to the quality attributes unique categorization, loose coupling, discoverability, and autonomy [1]. In a second step the service specification is performed. The service specification uses the identified and revised service candidates as input and defines service design, i.e., the service interfaces and service components. After a systematic derivation, the service designs are revised with regard to the previously mentioned quality attributes. This additional revision is necessary as service designs include more information than service candidates.

III. CASE STUDY FOR A QUALITY-ORIENTED SERVICE DESIGN PROCESS

Within PESCaDO the business use case for getting an observation has to be considered. The business use case can be modeled using use case diagrams of the Unified Modeling Language (UML) [34]. Furthermore, the UML profile for business modeling as introduced by IBM [22, 23] can be applied with its adapted notation for use case diagrams as shown in the following figure.

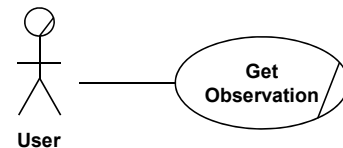


Figure 2. Considered business use case.

For the derivation of service candidates, especially the internal behavior of the business use case is required. This behavior is represented by a business process and can be modeled using the Business Process Model and Notation (BPMN) [31]. Figure 3 shows the business process as main artifact for deriving service candidates as first step of the service design phase.

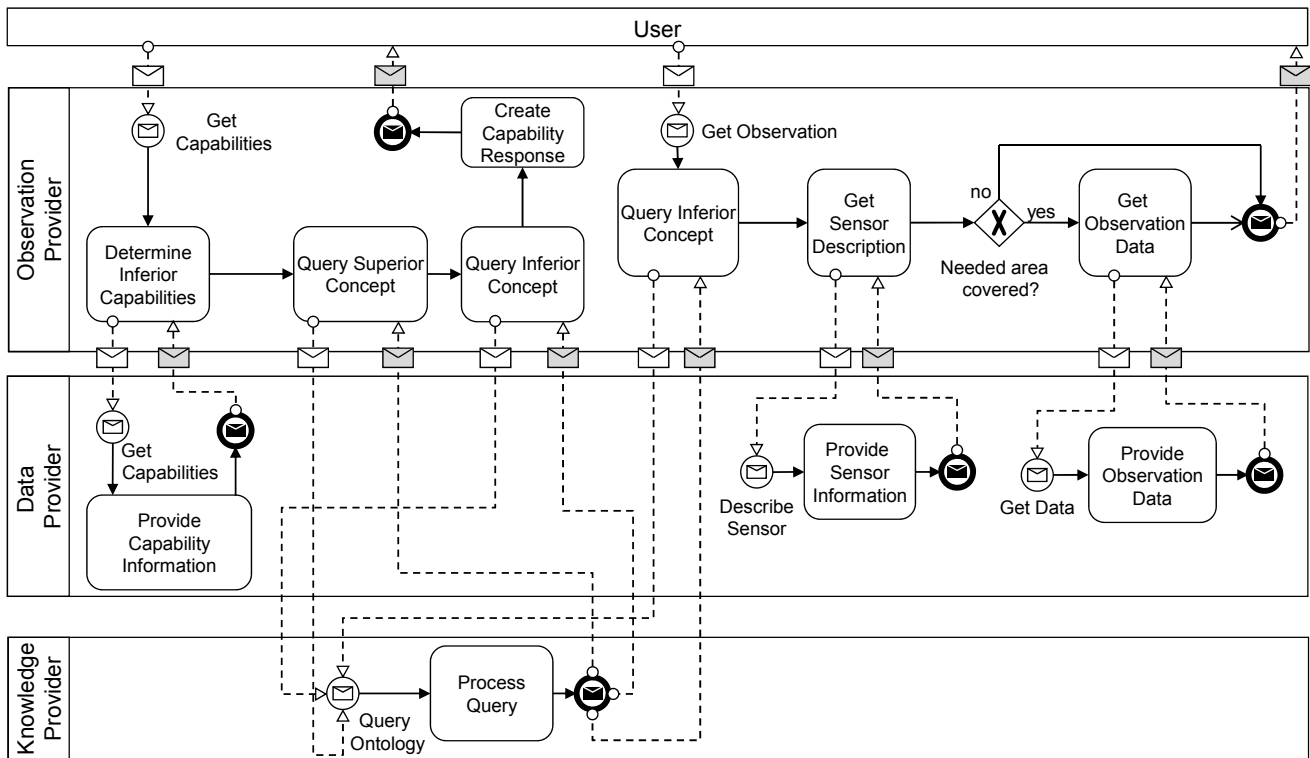


Figure 3. Considered business process.

Each term within the business use case and business process bases on a common domain model for avoiding ambiguity and misunderstandings. This domain model can be described using an ontology based on the OWL 2 Web Ontology Language (OWL) [32, 33]. It determines the concepts and their relations within the considered domain.

A. Identification

For the derivation of service candidates each pool within the BPMN business process is transformed into one capability element of the Service oriented architecture modeling language (SoaML), for this element represents a collection of capabilities that corresponds to the understanding of service candidates. Each capability element contains operations that represent operation candidates as preliminary operations of the service [7, 24, 26]. Figure 4 shows the derived service candidates.

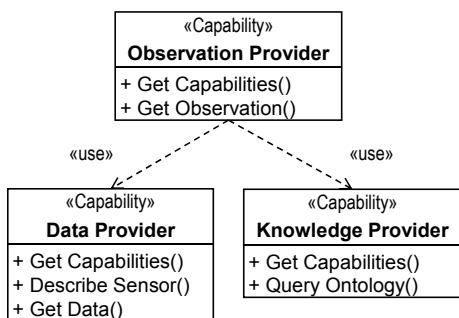


Figure 4. Derived service candidates.

The operation candidates within the service candidates, i.e., capability elements, are derived from the business process and its contained message start events. The usage dependencies are determined by means of the interaction between the pools. The names of the service candidates and operation candidates are taken from the business process.

In a next step, the service candidates have to be analyzed and revised with regard to the quality attribute unique categorization, loose coupling, autonomy and discoverability using the quality indicators introduced in [2].

1) *Unique Categorization*: According to Erl [5, 6, 28, 29], business-related and technical functionality should be divided. This quality indicator is fulfilled because all services only provide business-related functionality. Similarly, agnostic and non-agnostic functionality should be separated. Also this quality indicator is fulfilled, for all services only provide agnostic functionality, which is not specific for certain business processes. Another quality indicator for the unique categorization addresses the sovereignty of data. If a service manages a business entity, it should be explicitly managing this business entity for ensuring consistent and clear responsibility [5, 6, 12]. Within the business process there are two types of data: ontology data and observation data. The former are accessed by the knowledge provider and the latter by the data provider, which is why this quality indicator is fulfilled optimally. The last quality indicator for a unique categorization describes that the operations within one service should use common business entities. The data provider and knowledge provider only operate on ontology

data or observation data. However, the observation provider uses both observation data and ontology data, which may result in a split of these two operation candidates into two separate service candidates. Since the ontology data describes the observation data in more detailed, the ontology data does not represent an own business entity. Thus, the operation candidates can be grouped within one service candidate. As result, the derived service candidates best fulfill the quality indicators for a unique categorization.

2) *Loose Coupling*: According to Josuttis [15], long-running operations should be able to be invoked asynchronously. Since there are no long-running operations, respectively operations candidates, within the derived service candidates, this quality indicator does not have to be considered. Additionally, the parameters within the operations should be preferably simple types if they are used across several services. Complex types that are used within several services should be avoided. Since during the identification phase the parameters are not defined, this quality indicator can not be determined. Instead, this quality indicator will be considered during the specification phase. A further quality indicator describes that the operations should be abstract [5, 6, 15, 17]. This means that they should hide implementation details. The operation candidates are on a high-level of abstraction, which is why this quality indicator is fulfilled. Also if there is an state-changing operation, a compensating operation should be provided [17]. Since there is no data written or created, there is no state-changing operation.

3) *Discoverability*: The discoverability is only of interest during the specification phase, when the names of services and operations are finally determined. During the identification phase the artifacts are only preliminarily named.

4) *Autonomy*: One quality indicator for the autonomy of services focuses on the direct dependencies between services [5], which should be minimal for a maximum autonomy. Within the derived service candidates, the only service candidate with dependencies is the observation provider. However, due to the requirement of using distributed functionality, this quality indicator can not be improved. Another quality indicator addresses the overlapping of functionality [5, 28]. Services should have a certain functional scope. Since the service candidates do not have any overlapping functionality.

As result, the derived service candidates optimally fulfill the quality indicators for the considered quality attributes and thus do not have to be further revised.

B. Specification

The subsequent phase, the specification phase, focuses on the creation of service designs. A service design consists of a service interface, which describes the service from an external point of view, and a service component, which performs the provided functionality [2]. First, the service candidates of the identification phase are used to generate preliminary service designs that can be further revised in order to fulfill the desired quality attributes. Figure 5 shows the derived service interface for the Observation Provider.

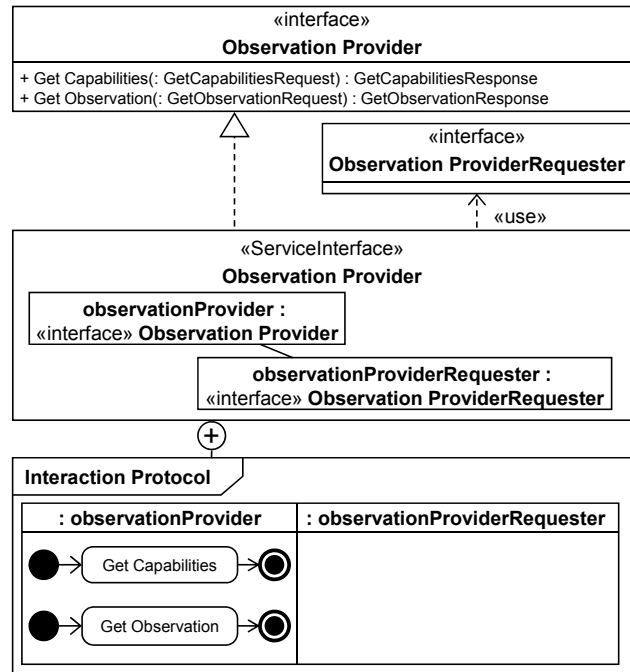


Figure 5. Derived service interface.

The service interface is formalized using the ServiceInterface modeling element of SoaML [7]. A service interface includes operations provided by the service and operations that have to be provided by the service consumer in order to receive callbacks. In SoaML these aspects are modeled using UML interfaces that are associated with the ServiceInterface element by generalizations and usage dependencies. Additionally, it defines the participating roles and an interaction protocol, which determines the possible orders of operation calls that result in valid results. Latter is modeled using a UML Activity that is added as ownedBehavior. The derivation of a service interface from service candidates transforms the operation candidates into provided operations. Also the name of the service candidate is used for the name of the service interface. Additionally, messages, roles and the interaction protocol are added systematically.

The service component includes provided services, services that are required to fulfill the functionality, and the internal behavior of the component in form of a flow of operation calls. The service component is represented by a Participant in SoaML. A Participant can be an organization, a system or a component within a system. It contains ServicePoints for provided services and RequestPoints for required services. Each ServicePoint and RequestPoint is typed by the describing ServiceInterface element. In Figure 6, the service component for the Observation Provider is shown. The name of the service component is directly derived from the name of the service candidate. The internal behavior is added as ownedBehavior in form of a UML Activity. It will be illustrated in context of the subsequent revision phase.

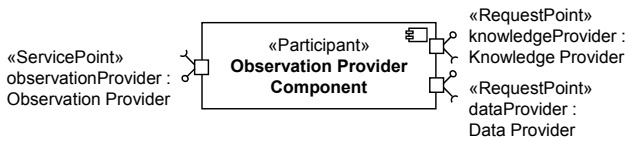


Figure 6. Derived service component.

In a next step, the subsequent analysis and revision phases can be performed considering the quality attributes unique categorization, loose coupling, discoverability, and autonomy.

1) *Unique Categorization*: Since the quality indicators that influence the unique categorization have already been optimal on basis of service candidates and the service designs were derived from these service candidates, the unique categorization is also optimal on basis of service designs. Thus, there is no revision required.

2) *Loose Coupling*: In contrast to the identification phase, during the specification phase, the parameters are formalized. For geographical information systems, standard data types, such as the Keyhole markup language (KML) [35], exist. Also within PESCaDO, standardized data types are expected to be used. Since complex types that are used across several services should be avoided, the data types are modeled within single UML packages for each service design. This ensures that changing data types does not necessarily affect other services. The infrastructure, for instance in form of an enterprise service bus, can handle the transformation between similar data types. The other quality indicators are still optimal, for the affecting artifacts have not changed during the specification phase.

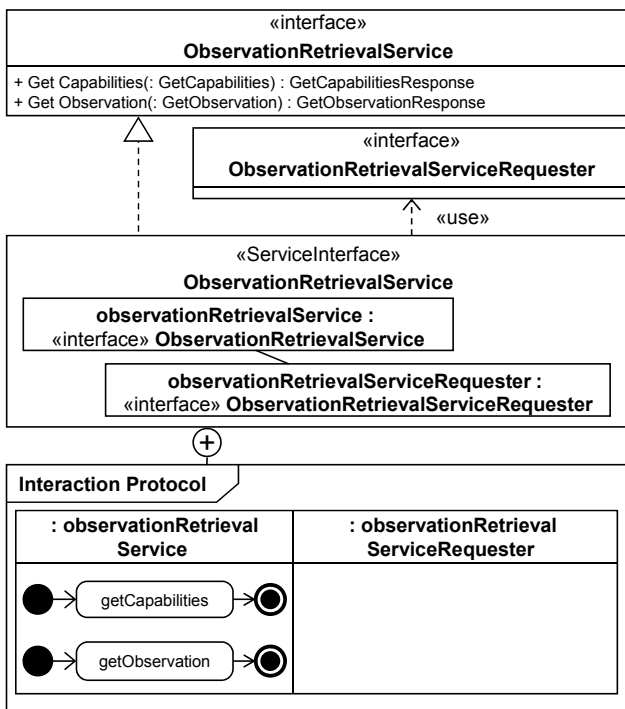


Figure 7. Revised service interface.

3) *Discoverability*: During the specification phase, the final names of the services and data types are determined. According to Josuttis [15] and Maier et al. [17], the names of the visible artifacts should be functionally named. Additionally, the names should follow naming conventions. Thus, during the specification phase, the names of the artifacts should be inspected in detail. Exemplarily naming conventions are the usage of the english language and beginning operation names with a lower-case character. In Figure 7, a revised service interface is shown that considers the naming conventions of the PESCaDO project. Additionally, the service has been renamed regarding its actual functionality for improving its discoverability.

This revision also affects the service component that uses this service interface. Figure 8 shows the revised service component of the Observation Provider. The service component and the ServicePoints and RequestPoints have been adapted to the revised service interfaces and the naming conventions for PESCaDO. Additionally, the internal behavior of the service component for one of the provided operations is shown.

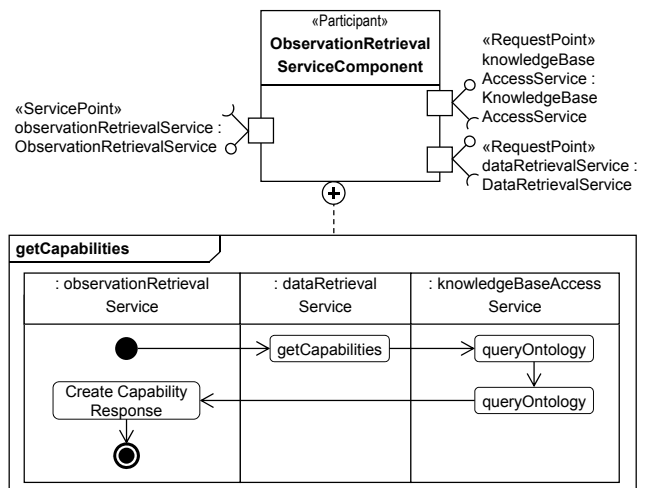


Figure 8. Revised service component.

4) *Autonomy*: Since the autonomy has already been optimized during the identification phase, there is no revision necessary regarding this quality attribute.

By finishing the revision of the initial service designs, the specification phase ends. The results for developing a prototype for the PESCaDO project are three revised service specifications, which now can serve as an input for the implementation phase [27]. The service designs have been revised that the resulting services optimally fulfill the chosen quality attributes of a unique categorization, loose coupling, discoverability and autonomy.

IV. CONCLUSION AND OUTLOOK

In this paper, we applied a quality-oriented service design process to the Personalized Environmental Service Configuration and Delivery Orchestration project of the European Commission. The design process enabled the

systematic derivation and revision in order to gain service designs that fulfill both the functional requirements and the quality attributes of a unique categorization, loose coupling, discoverability and autonomy. The service designs result in services that support the strategic goals that are associated with service-oriented architectures, such as an increased flexibility and maintainability. Due to the application on a concrete scenario, the usage of the design process in terms of its applicability and effectiveness for real-world projects is demonstrated.

The case study also showed shortcomings of the service design process that are expected to be solved in the future: The used quality indicators that were derived from common and wide-spread descriptions of quality attributes use terms that are not exactly defined. For example, the meaning of agnostic functionality is not clear. The IT architect has to interpret these terms in order to determine the quality indicators and the quality attributes. This may result in wrong measures.

Thus, this case study showed the applicability and effectiveness of the service design process. However, in the future, we plan to further refine the definitions of terms used within the quality indicators and quality attributes to reduce ambiguities, thus increase the correctness of the results. Additionally, we plan to apply the design process on further scenarios.

REFERENCES

- [1] M. Gebhart, M. Baumgartner, and S. Abeck, "Supporting service design decisions", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 76-81.
- [2] M. Gebhart, M. Baumgartner, S. Oehlert, M. Bliersch, and S. Abeck, "Evaluation of service designs based on soaml", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 7-13.
- [3] The PESCaDO Consortium, "Service-based infrastructure for user-oriented environmental information delivery", EnviroInfo, 2010.
- [4] Fraunhofer Institute of Optronics, System Technologies and Image Exploitation, "D8.3 Specification of the pescado architecture", Version 1.0, 2010.
- [5] T. Erl, *Service-Oriented Architecture – Concepts, Technology, and Design*, Pearson Education, 2006. ISBN 0-13-185858-0.
- [6] T. Erl, *SOA – Principles of Service Design*, Prentice Hall, 2008. ISBN 978-0-13-234482-1.
- [7] OMG, "Service oriented architecture modeling language (SoaML) – specification for the uml profile and metamodel for services (UPMS)", Version 1.0 Beta 1, 2009.
- [8] The European Parliament and the Council of the European Union, "INSPIRE", Directive 2007/2/EC, 2007.
- [9] IBM, "RUP for service-oriented modeling and architecture", IBM Developer Works, http://www.ibm.com/developerworks/rational/downloads/06/rmc_soma/, 2006. [accessed: June 04, 2011]
- [10] U. Wahli, L. Ackerman, A. Di Bari, G. Hodgkinson, A. Kesterton, L. Olson, and B. Portier, "Building soa solutions using the rational sdp", IBM Redbook, 2007.
- [11] A. Arsanjani, "Service-oriented modeling and architecture – how to identify, specify, and realize services for your soa", IBM Developer Works, <http://www.ibm.com/developerworks/library/ws-soa-design1>, 2004. [accessed: June 04, 2011]
- [12] G. Engels, A. Hess, B. Humm, O. Juwig, M. Lohmann, J.-P. Richter, M. Voß, and J. Willkomm, *Quasar Enterprise*, dpunkt.verlag, 2008. ISBN 978-3-89864-506-5.
- [13] A. Erradi, S. Anand, and N. Kulkarni, "SOAF: An architectural framework for service definition and realization", 2006.
- [14] R. Reussner and W. Hasselbring, *Handbuch der Software-Architektur*, dpunkt.verlag, 2006. ISBN 978-3898643726.
- [15] N. Josuttis, *SOA in der Praxis – System-Design für verteilte Geschäftsprozesse*, dpunkt.verlag, 2008. ISBN 978-3898644761.
- [16] B. Maier, H. Normann, B. Trops, C. Utschig-Utschig, and T. Winterberg, "Die soa-service-kategorienmatrix", SOA-Spezial, Software & Support Verlag, 2009.
- [17] B. Maier, H. Normann, B. Trops, C. Utschig-Utschig, and T. Winterberg, "Was macht einen guten public service aus?", SOA-Spezial, Software & Support Verlag, 2009.
- [18] M. Pereplechikov, C. Ryan, K. Frampton, and H. Schmidt, "Formalising service-oriented design", *Journal of Software*, Volume 3, February 2008.
- [19] M. Pereplechikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling metrics for predicting maintainability in service-Oriented design", *Australian Software Engineering Conference (ASWEC 2007)*, 2007.
- [20] M. Hirzalla, J. Cleland-Huang, and A. Arsanjani, "A metrics suite for evaluating flexibility and complexity in service oriented architecture", *ICSOC 2008*, 2008.
- [21] S. W. Choi and S. D. Kimi, "A quality model for evaluating reusability of services in soa", *10th IEEE Conference on E-Commerce Technology and the Fifth Conference on Enterprise Computing, E-Commerce and E-Services*, 2008.
- [22] S. Johnston, "Rational uml profile for business modeling", IBM Developer Works, <http://www.ibm.com/developerworks/rational/library/5167.html>, 2004. [accessed: June 04, 2011]
- [23] J. Heumann, "Introduction to business modeling using the unified modeling language (UML)", IBM Developer Works, <http://www.ibm.com/developerworks/rational/library/360.html>, 2003. [accessed: June 04, 2011]
- [24] J. Amsden, "Modeling with soaml, the service-oriented architecture modeling language – part 1 – service identification", IBM Developer Works, <http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/index.html>, 2010. [accessed: January 04, 2011]
- [25] P. Kroll and P. Kruchten, *The Rational Unified Process Made Easy, a Practitioner's Guide to the RUP*, Addison-Wesley, 2003.
- [26] M. Gebhart and S. Abeck, "Rule-based service modeling", *The Fourth International Conference on Software Engineering Advances (ICSEA 2009)*, Porto, Portugal, September 2009, pp. 271-276.
- [27] P. Hoyer, M. Gebhart, I. Pansa, S. Link, A. Dikanski, and S. Abeck, "A model-driven development approach for service-oriented integration scenarios", 2009.
- [28] T. Erl, *SOA – Design Patterns*, Prentice Hall, 2008. ISBN 978-0-13-613516-6.
- [29] T. Erl, *Web Service Contract Design & Versioning for SOA*, Prentice Hall, 2008. ISBN 978-0-13-613517-3.
- [30] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA – Service-Oriented Architecture Best Practices*, 2005. ISBN 0-13-146575-9.
- [31] OMG, "Business process model and notation (BPMN)", Version 2.0 Beta 1, 2009.
- [32] W3C, "OWL 2 web ontology language (OWL)", W3C Recommendation, 2009.
- [33] M. Horridge, "A practical guide to building owl ontologies using protégé 4 and co-ode tools", <http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/>, Version 1.2, 2009. [accessed: January 04, 2011]
- [34] OMG, "Unified modeling language (UML), superstructure", Version 2.2, 2009.
- [35] OGC, "Keyhole markup language (KML)", <http://www.opengeospatial.org/standards/kml/>, Version 2.2, 2008. [accessed: June 04, 2011]

Meta-Model for Global Software Development to Support Portability and Interoperability in Global Software Development

Bugra Mehmet Yildiz, Bedir Tekinerdogan
 Department of Computer Engineering
 Bilkent University
 Ankara, Turkey
 {bugra, bedir}@cs.bilkent.edu.tr

Abstract— Global Software Development (GSD) considers the coordinated activity of software development that is not localized and central but geographically distributed. To support coordination among sites, usually it is aimed to adopt the same development and execution platform. Unfortunately, adopting a single platform might not be always possible due to technical or organizational constraints of the different sites in GSD projects. As such, very often GSD projects have to cope with portability and interoperability problems. To address these problems we propose to apply model-driven architecture design (MDA) approach. For this we present a common meta-model of GSD that we have derived from a systematic domain analysis process. The meta-model enhances the understanding of GSD, is used to define platform independent models of GSD architecture, and transform platform independent models to platform specific models.

Keywords-Global Software Development, Architecture Modeling, Model-Driven Development

I. INTRODUCTION

Global Software Development (GSD) is a software development approach that can be considered as the coordinated activity of software development that is not localized and central but geographically distributed. In principle, GSD can be considered as the realization of outsourcing. The reason behind this globalization of software development stems from clear business goals such as reducing cost of development, solving local IT skills shortage, and supporting outsourcing and offshoring [1]. There is ample reason that these factors will be even stronger in the future, and as such, we will face a further globalization of software development [6].

One of the challenging issues in setting up global software development is the interoperability among the distributed sites [13][14]. Interoperability is defined as the ability of two or more systems or components to exchange information and to use the information that has been exchanged [7]. Although it is aimed to adopt the same platforms in global software development projects, this might not be always possible due to technical or organizational constraints. As such, different sites might run on different operating system platforms, use different component language platforms, or adopt a different middleware platform. Further, due to the continuous evolution of project requirements, the platforms on different sites might also need to evolve. Portability of the existing

software to a new platform is not easy for even a single site development project; in the case of global software development projects this is even a much harder problem. Altogether, both the portability and interoperability problems will impede the adoption of a global software development approach.

Portability to different platforms and interoperability among different sites working on different platforms have been mainly addressed in the model-driven software development approaches. In this context, Model Driven Architecture (MDA) is a framework defined by the Object Management Group (OMG) that separates the platform specific concerns from platform independent concerns to improve the reusability, portability and interoperability of software systems [12]. To this end, MDA separates Platform Independent Models (PIMs) from Platform Specific Models (PSMs). The PIM is a model that abstracts from any implementation technology or platform. The PIM is transformed into one or more PSMs, which include the platform specific details. Finally the PSM is transformed to code providing the implementation details. Obviously by separating the platform specific concerns and providing mechanisms to compose these concerns afterwards in the code MDA provides a clean separation of concerns and as such the systems are better reusable easier to port to different platforms and have increased interoperability.

We present the model-transformation pattern for transforming the global platform independent model to the local platform specific models. An important part of the model transformation is the common GSD meta-model. We describe both the abstract syntax and the concrete syntax of the meta-model. The abstract syntax is defined using the UML notation; the concrete syntax is specific for the parts of the meta-model. The meta-model enhances the understanding of GSD, and supports the model transformation for solving portability and interoperability problems.

The remainder of the paper is structured as follows. Section II provides some background on GSD. Section III describes the meta-model for GSD and Section IV describes the related work. Finally, Section VI concludes the paper.

II. GLOBAL SOFTWARE DEVELOPMENT

A GSD architecture usually consists of several nodes, or sites, on which different teams are working to develop a part of the system. The teams could include development teams,

testing team, management team, etc. Usually, each site will also be responsible for following a particular process. In addition, each site might have its own local data storage. Overall we can identify four important key concerns in designing GSD:

Development - the software development activities typically using a software development process. This includes activities such as requirements analysis, design, implementation and testing. Each PDS will address typically a subset of these activities.

Communication – communication mechanisms within and across sites. Typically the different sites need to adopt a common communication protocol.

Coordination – coordination of the activities within and across sites to develop the software according to the requirements. Coordination will be necessary to align the workflows and schedules of the different sites. An important goal could be to optimize the development using appropriate coordination mechanisms.

Control – systematic control mechanisms for analyzing, monitoring and guiding the development activities. This does not only include controlling whether the functional requirements are performed but also which and to what extent quality requirements are addressed.

In fact each of these concerns requires further in-depth investigation and has also been broadly discussed in the GSD community.

To realize multi-site development is not a trivial task. In particular if the different sites are working on different platforms the interoperability problems must be resolved.

Figure 1 shows the transformation pattern for mapping a global platform independent model to local platform specific models. The platform independent model can be considered the same across multiple development sites. If needed the local sites can keep working on different platforms. In that case the alignment and the interoperability can be achieved by defining transformation patterns, which map the local platform models to the global platform independent models, and vice versa. To support the model transformation a proper definition of the GSD meta-model is necessary. We discuss this in the next section.

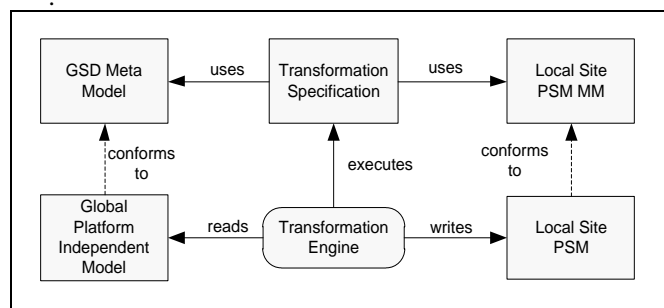


Figure 1. Model-Transformation pattern for mapping GSD PIM to local PSM

III. META-MODEL FOR GLOBAL SOFTWARE DEVELOPMENT

Meta-models define the language for the models. In both software language engineering [9] and model-driven development domains [2], a meta-model should have the following two key elements:

Abstract Syntax: Captures the concepts provided by the language and relationships between these concepts.

Concrete Syntax: Defines the notation that facilitates the presentation and construction of models in that language.

Based on the literature of GSD, we have defined a meta-model for GSD that defines the concepts and their relations to enhance the understanding of GSD and support the model transformation. Since the meta-model is quite large and we aim the modeling of different concerns of GSD, we have decomposed meta-model into six meta-model units. Each of these meta-model units includes semantically close entities and address different concerns. These units are *Deployment*, *Process*, *Data*, *Communication*, *Tool* and *Migration*. Each unit includes abstract syntax representing GSD elements and their relations and visual concrete syntax for visualization of these elements.

A. Deployment Unit

Deployment Unit concerns the deployment of the teams to different sites. The abstract and concrete syntax of this unit are shown in Figure 2.

Team is the primary essential entity in Deployment and also in the whole meta-model and is defined as a group of persons that work together to achieve a particular goal. A *Team* may be organized in a temporary way that it will be dismissed after its function is complete. *Team* is allocated at a particular *Site*. *Site* may to a country, city or a building where a *Team* works at. Location attribute determines where *Site* is placed in the world. Time zone shows the local time of *Site*. *Teams* may belong to different types of *Organizations*, such as commercial organizations, subcontractors or non-profitable organizations such as open source communities. *Teams* can be from different countries and depending on the society they are in, they may have different *Social Cultures*. Like *Social Culture*, *Team*'s background including work experience, the time that members work together, their habits are captured by *Work Culture* entity. *Expertise Area*, *Team* and *Site* can be further decomposed into sub-parts. For example, a Software *Team* may consist of sub-Teams each responsible for Design, Implementation, Testing and Integration.

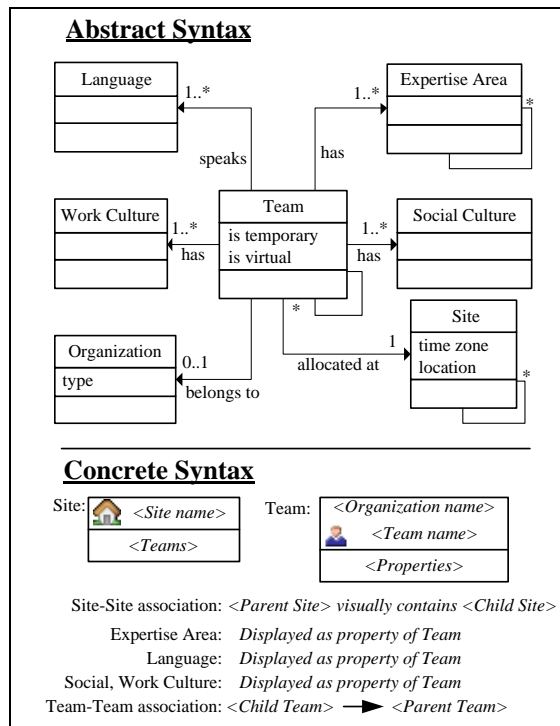


Figure 2. Deployment Unit: Abstract and Concrete Syntax

B. Process Unit

Process Unit concerns the different kind of processes in GSD. The abstract and concrete syntax of this unit are shown in Figure 4.

Process is defined as a planned set of activities that aims to provide some service. Teams participate in Process in order to provide some service. Service is defined with Function. A Function can be any service during software development process that requires some Expertise Areas such as software development, architecture design, business management, requirements elicitation and so on. Coordination is also a Function that should be provided for coordinating several Teams' activities. A Process consumes or uses several different Data Entities and also creates other Data Entities for providing targeted Functions. For supporting activities defined in Process, Process concept is further specialized into Workflow, Business Process and Development Process (not shown in figure).

C. Data Unit

Data Unit is for representing ownership and physical deployment of software development data. The abstract and concrete syntaxes are shown in Figure 4.

Data Entity is the fundamental entity of this unit. It represents any piece of data: digital, textual or informal piece of information such as notes taken by developers, telephone calls that are usually not recorded. Data Entity has size whose unit is defined by size type; for example, a 120-page report, 6 minutes of voice record, 2 gigabyte of digital data. Creation date and last update date show the history of Data Entity. Data Entity has Actual Type where Actual Format can be one of predefined formats (video, sound, text, picture

and complex-Data Entity) or some designer defined format. If Data Entity is digital, then in addition to Actual Format, it has a Digital Format. Data Entity may be implemented in one or more Languages.

Data Entity is stored in Data Storage. Data Storage corresponds to any object in real world that can store information. For example, some textual document is stored in paper form, or it is stored in a voice record, or it is stored digitally in the format of some text editor. Data Storage has ability to store some Actual Types and if it can store digital data, then it can support some Digital Types also. A Data Storage instance is owned by one or more Teams and it can be located in one Site or may be distributed over several Sites like distributed databases.

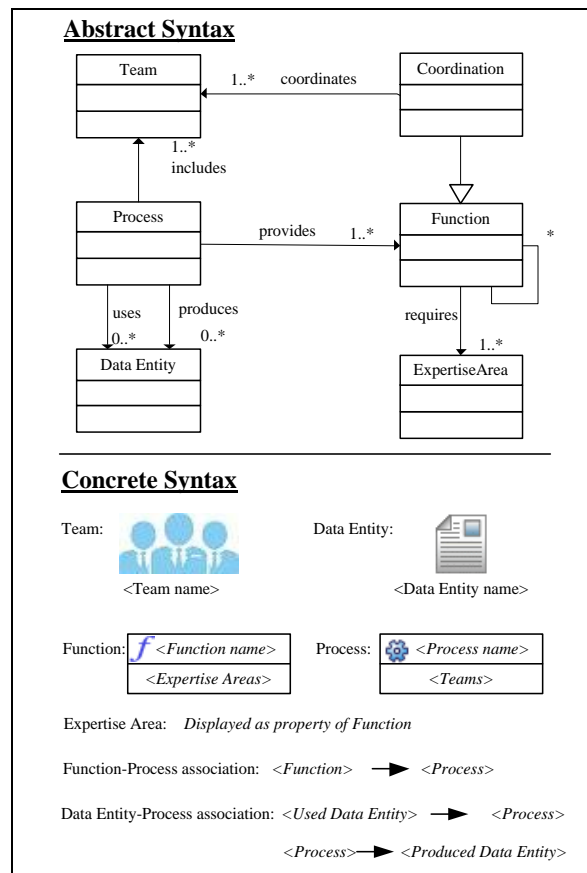


Figure 3. Process Unit: Abstract and Concrete Syntax

D. Communication Unit

Communication Unit focuses on the representation of both formal and informal communication activities between Teams. The abstract and concrete syntaxes are shown in Figure 5.

Communication is done over Communication Platform in the context of Process and it can be an instance of sudden/event based communication activity like a telephone call or a continuous communication channel such as a discussion forum. Type attribute is for representing in which way Communication takes place such as email, phone call,

face-to-face chat and so on. *Suggested time period* is an important attribute for GSD since Teams work in different time zones, some *Communication* channels can be used effectively in a defined time period. For example, phone calls should be done during the hours when both sides are in or around their work hours.

Communication has two sides, which are caller and receiver. Generally speaking, caller starts communication and receiver is the one who is called by caller. For example, an email sender is classified as caller and receiver is the one who receives email. Sometimes, there can be multiple callers such as video conferences or there can be multiple receivers such as discussion forums. It is also possible that caller and receiver are the same such as a planned meeting. For all cases, caller and receivers are considered as *Teams* in this unit. While *Teams* communicate, one or more *Data Entities* are carried in the context of *Communication*.

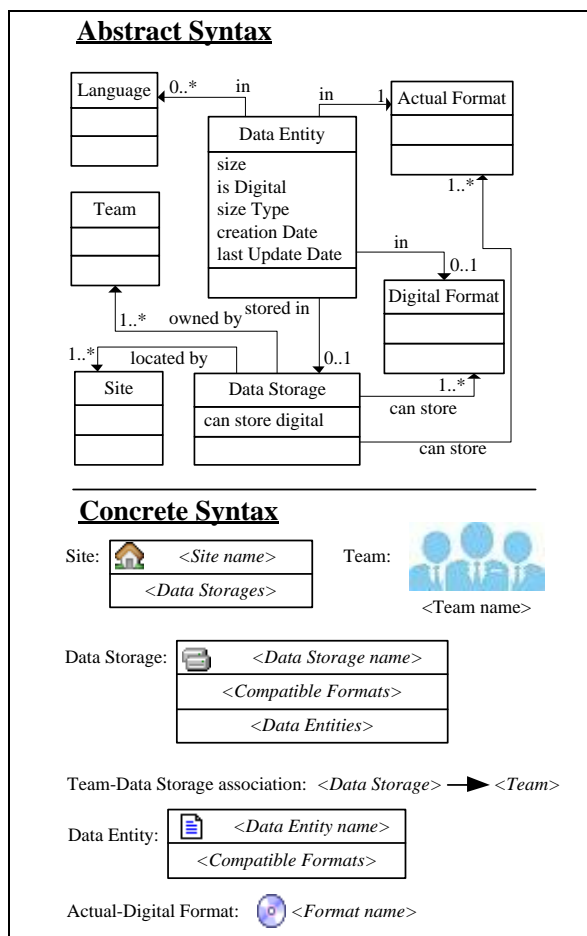


Figure 4. Data Unit: Abstract and Concrete Syntax

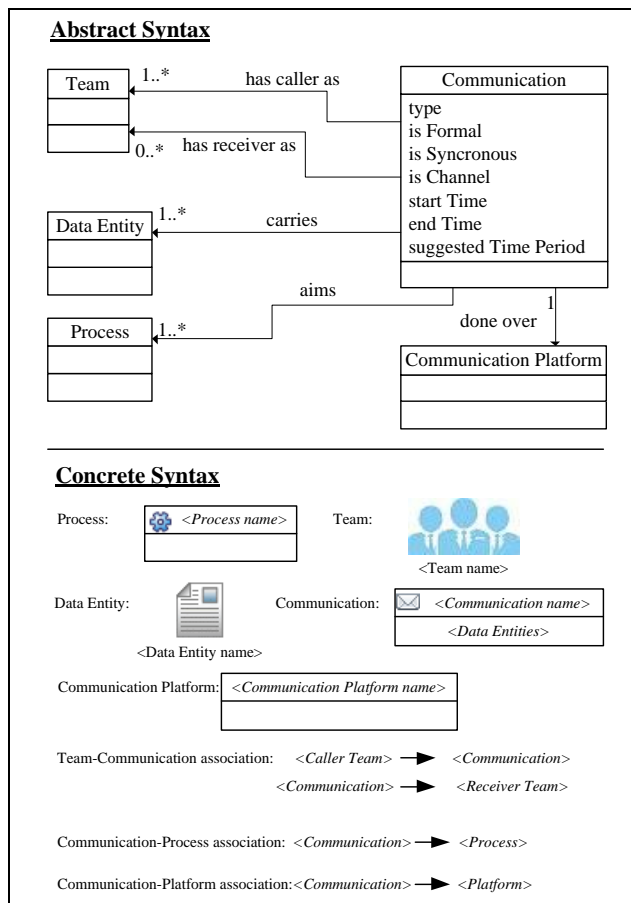


Figure 5. Communication Unit: Abstract and Concrete Syntax

E. Tool Unit

Tool Unit captures details of tools used by *Teams* for communication and providing *Functions*. The abstract and concrete syntax are shown in Figure 6.

Tool is compatible with one or more *Actual Format* and *Digital Format*. Platform is the set of *Tools* used by *Teams* for communication or providing some functions. Depending on the purpose, the platform is defined as *Function Platform* or *Communication Platform*.

F. Migration Unit

Migration Unit concerns the migration and traveling of *Teams* during GSD activities. These travels are especially needed in the first and final phases of the projects to ease and support coordination and integration. The abstract and concrete syntax are shown in Figure 6.

Migration is executed by one or more *Teams* from *Site* to *Site* at a particular date. In a *Migration*, *Teams* may carry *Data Storage* such as documents, digital data containers and so on. *Migration* is executed in the context of *Process*.

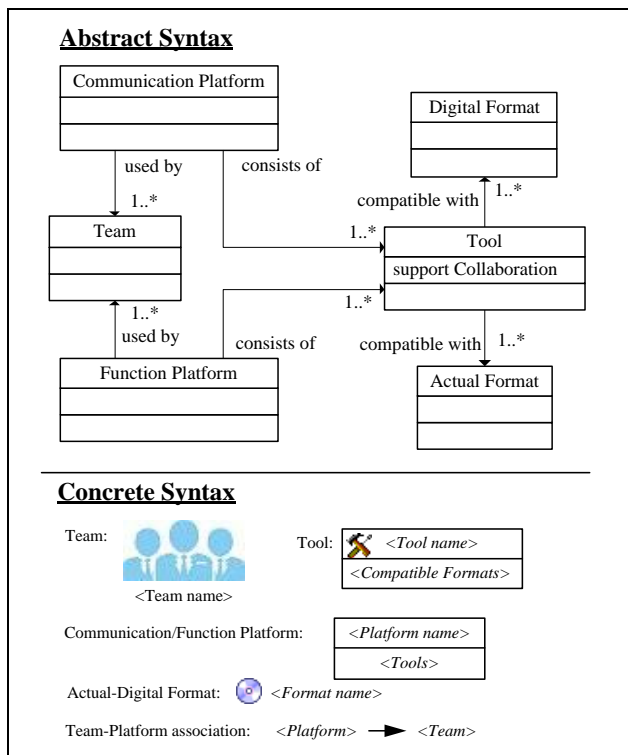


Figure 6. Tool Unit: Abstract and Concrete Syntax

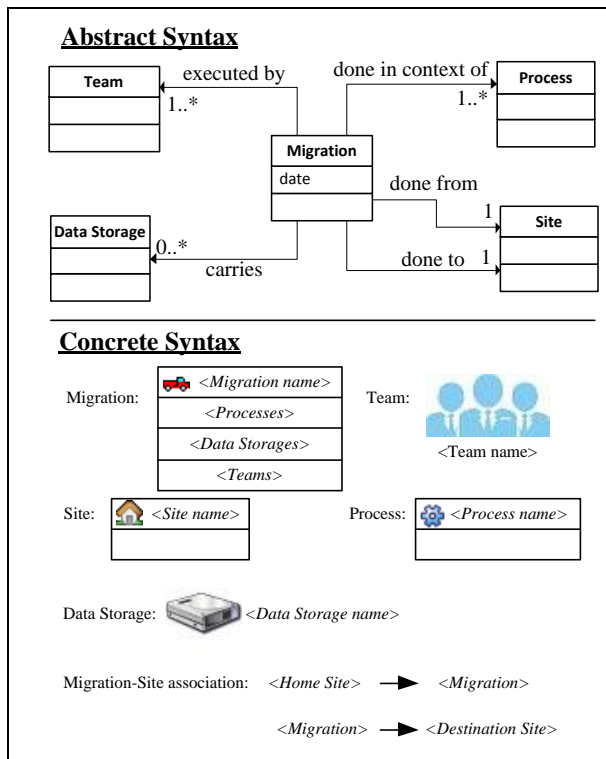


Figure 7. Migration Unit: Abstract and Concrete Syntax

G. Example Case

As an example case, consider a GSD environment with 5 Sites. Company A operates in United States. Customer relations and requirements management jobs are done in New York while software architecture is designed in Los Angeles. Company B is hired as subcontractor for developing software and testing, which is located in Pekin, China. Moving from this case definition and Deployment meta-model unit, the model in Figure 9 can be drawn.

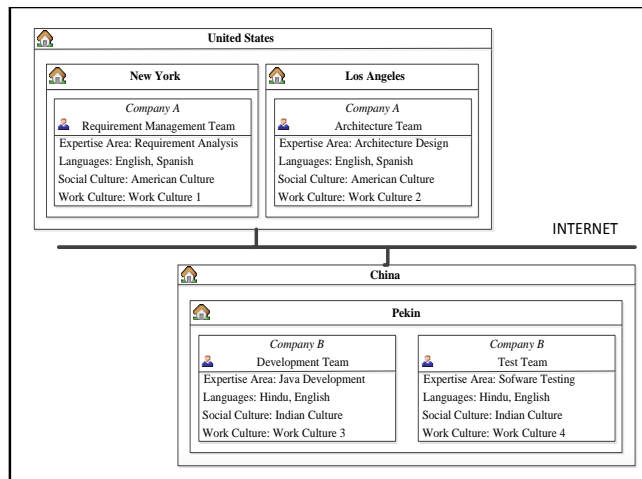


Figure 9. Example Case Model

IV. RELATED WORK

Notably, architecting in GSD has not been widely addressed. The key research focus in the GSD community seems to have been in particular related to tackling the problems related to communication, coordination and control concerns. Clerk et al. [4] report on the use of so-called architectural rules to tackle the GSD concerns. Architectural rules are defined as “principles and statements about the software architecture that must be complied with throughout the organization”. They have defined four challenges in GSD: time difference and geographical distance, culture, team communication and collaboration, and work distribution. For each of these challenges they list possible solutions and describe to what extent these solutions can be expressed as architectural rules. The work of Clerk et al. aims to shed light on what kind of architectural rules are necessary to guide the GSD. We consider our work complementary to this work. In our work the design actions that relate to the expected answers of questions are defined as design actions.

Tool support has been named as one of the important challenges for GSD since it requires making software development tools and environments more collaborative [13]. Booch and Brown [3] have introduced the vision for Collaborative Development Environment (CDE), which is defined as “a virtual space wherein all the stakeholders of the project – even if distributed by time or distance – may negotiate, brainstorm, discuss, share knowledge, and generally labor together to carry out some task, most often to

create an executable deliverable and its supporting artifacts". A number of efforts have been carried out to support the idea of CDEs. Whitehead [13] has presented a survey on existing collaboration support tools in software engineering. Whitehead distinguishes among four broad categories of tool support to support collaboration in software engineering: Model-based collaboration tools for representing the adopted models; Process support tools for representing software development process; Awareness tools for informing developers about the ongoing work of others and to avoid conflicts; Collaboration infrastructure to support data and control integration and likewise support interoperability. Despite the clear need and benefits of the existing CDE tools, it appears that most of the work on CDE has focused on the (social) collaboration concern and less on the (technical) development part. Further the tools that address development primarily focus on collaborative coding and relatively little attention has been paid to architecture design. There seems to be a general agreement that more research is needed in this domain. Our approach and the meta-model definition can be considered as part of the efforts for enhancing CDE for design of GSDs.

Maciel et al. [10] present a domain-specific architecture (DSA) defining middleware services to provide interoperability in collaborative environments. Similar to our approach they define a platform independent model that is independent of platform specific models. In their approach the reference architecture (PIM) is based on MDA's UML Profile for Enterprise Distributed Object Computing (EDOC) [11] and the viewpoints defined in RM-ODP (Open Distributed Processing-Reference Model) are adopted [8]. In our approach we do not use a general purpose architecture framework such as RM-ODP but adopt a meta-model based on a domain analysis of the GSD literature.

V. CONCLUSION AND FUTURE WORK

Different challenges have been identified to set up a Global Software Development environment. Our literature study on GSD showed that in particular the challenges of communication, coordination, and control of GSD is addressed in the GSD community but less focus has been provided on the modeling, documentation and analysis of architecture for GSD. One of the key technical problems in GSD projects is the evolution of platforms on different sites and the need for interoperability among different sites. A close analysis of the literature shows that the application of MDSG has not been explicitly addressed, neither in the GSD community nor in the MDSG community. In this paper we have provided a general transformation pattern for mapping a global platform independent model to the platform specific models at local sites. Portability can be supported by defining transformation definition that map the new platform models to the global platform independent models and vice versa. Interoperability is supported due to the common model, global platform independent model that conforms to the meta-model that we have defined in the paper. The meta-model aimed to support the portability and interoperability in GSD but also enhances the

understandability and communication about GSD. In our future work we plan to define domain specific languages for the six units of the GSD meta-model. For this we will use the Eclipse Modeling Framework [5] and develop the corresponding tool support for realizing the automatic or semi-automatic model transformations in GSD projects.

REFERENCES

- [1] P. J. Agerfalk, B. Fitzgerald, H. H. Olsson, and E. O'Conchúir, "Benefits of Global Software Development: The Known and Unknown," in International Conference on Software Process, ICSP 2008. 2008. Leipzig, Germany.; Springer Berlin / Heidelberg
- [2] J. Bézuvin. On the Unification Power of Models. *Software and System Modeling (SoSym)* 4(2):171-188, 2005.
- [3] G. Booch and A. Brown. Collaborative Development Environments. *Advances in Computers* Vol. 59, Academic Press, August, 2003.
- [4] V. Clerc, P. Lago and H. van Vliet, "Global Software Development: Are Architectural Rules the Answer?" Proc. of the 2nd International Conference on Global Software Engineering, pp. 225-234. IEEE Computer Society Press, Los Alamitos, 2007.
- [5] Eclipse Modeling Framework, <http://www.eclipse.org/modeling/emf/>, accessed: July 2011.
- [6] J. D. Herbsleb, Global Software Engineering: The Future of Socio-technical Coordination, 2007 Future of Software Engineering, p.188-198, May 23-25, 2007
- [7] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.
- [8] ISO-2004. Use of UML for ODP system specification. Working Draft. ISSO/IEC JTC1/SC7.
- [9] A. Kleppe. Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley Longman Publishing Co., Inc., Boston, 2009.
- [10] R. S. P. Maciel, C. G. Ferraz, and N. S. Rosa, "An MDA domain specific architecture to provide interoperability among collaborative environments," in Proceedings of the 19th Brazilian Symposium on Software Engineering (SBES '05), pp. 1-16, Uberlandia, Brazil, October 2005.
- [11] OMG EDOC. UML Profile for Enterprise Distributed Object Computing Specification. OMG Adopted Specification (ptc/02-02-05), 2002.
- [12] OMG Model Driven Architecture, OMG Model Driven Architecture. <http://www.omg.org/mda/>. Accessed in June 1, 2011.
- [13] J. Whitehead, Collaboration in Software Engineering: A Roadmap, In FOSE '07: 2007 Future of Software Engineering, pp. 214-225, 2007.
- [14] I. S. Wiese and E. H. M. Huzita, "IMART: An Interoperability Model for Artifacts of Distributed Software Development Environments," Global Software Engineering, 2006. ICGSE '06. International Conference on , vol., no., pp. 255-256, Oct. 2006.

A New Approach to Software Development Process with Formal Modeling of Behavior Based on Visualization

Abbas Rasoolzadegan, Ahmad Abdollahzadeh Barfouroush
Information Technology and Computer Engineering Faculty
Amirkabir University of Technology (Tehran Polytechnic)
{rasoolzadegan, ahmad}@aut.ac.ir

Abstract—This work investigates the advantages and limitations of various modeling methods. Despite of their advantages, due to some limitations of each modeling method, using only one of them as the sole approach will not ensure high quality software. This work proposes a new feasible approach to improve the software development process by integrating semi-formal and formal modeling methods. In this approach, software is initially modeled using the formal specification language Object-Z. The formal models, produced by Object-Z, are formally refined to ensure correctness. Then, software behavior is extracted and visualized in specific intervals using UML. Applying design patterns to the visualized models increases reusability and flexibility. The newly improved models are then re-formalized. Such an iterative and evolutionary process continues until developing the software with the desired quality. This paper proposes a new approach to develop reliable, yet flexible software.

Keywords—Formalization; visualization; design patterns; formal modeling methods; semi-formal modeling methods.

I. INTRODUCTION

Requirements engineering (RE) plays a crucial role in software development cycle. Studies show that the major causes of most software projects failure are imprecise and incomprehensive understanding, elicitation, specification, analysis, validation, and verification of software requirements during software development process [3]. Moreover, mainstream software development, with its recurring practice of trial and error, already suffers from its premature insistence on code and program testing. The problem is that code is expensive; it has too much detail, and is not at the right level of abstraction to help thinking about the problem and design of its solution [1].

The increasing importance of requirements engineering and need for further abstraction leads to increasing use of models during software development cycle, in general, and throughout RE process, in special. Models can be used at different phases of a software life-cycle, ranging from requirements (more abstract) to detailed design (more concrete). It also gives a basis for a stepwise approach to software development: abstract models are refined into more concrete ones in a stepwise manner, where each step carries some design decisions. This is known as model refinement [3].

Models and modeling play a crucial role in software development cycle. In software engineering, models are used to describe both the problem (requirements) and the solution (design) in order to gain a better understanding of the issues involved. Once a model has been constructed it can be analyzed to uncover flaws and expose fundamental issues [23]. This role of models cannot possibly be assumed by code. The idea is not new, but there is a recent trend towards more use of models in mainstream circles of software engineering. This is the goal of MDSE [19], which tries to alleviate the complexity of software development by using models. Model transformation has a key role in MDSE. A model transformation takes as input a model conforming to a given meta-model and produces as output another model conforming to a given meta-model. One of the characteristics of a model transformation is that it is also a model, i.e. it conforms to a given meta-model.

There are two reasons for against-our-expectation behavior of the software [25]: either there are shortcomings or omissions in the original specification, or the software does not conform to its specification. These two issues result from the following causes: 1) incomplete, ambiguous, and inconsistent requirements specification, 2) imprecise and imperfect verification of the specification and design which in turn lead to incomplete and untimely discovery of the software's errors during the development cycle. These problems arise from the weaknesses of informal and semi-formal modeling methods (SFMMs) in specification and verification of the software requirements.

This paper investigates the advantages and shortcomings of SFMMs and formal modeling methods (FMMs) by surveying the literature [1][5][13][25]. Reference [26] has already investigated the advantages and disadvantages of SFMMs and FMMs, empirically, by specifying the multi-lift system case study. The most important conclusion is that each modeling method has some unique advantages and limitations. Using only one of them as the sole approach leads not to satisfy all required aspects of software quality such as reliability, flexibility, reusability, scalability, and so on [30]. Combination of these methods is necessary to successfully understand, analyze, specify, validate, and verify requirements, problems, and solutions. Although, there are several valuable attempts to integrate these

methods to utilize unique advantages of both formal and semi-formal modeling methods, there is a long way ahead to achieve the promised goals.

This paper proposes a new approach to enhance the software development process. This work emphasizes on the software behavior rather than its structure. In the proposed approach, the formalism plays the key role, i.e., the structure and behavior of the software is initially modeled using a suitable formal modeling language (such as Object-Z). These formal models, along with formal refinement [3] ensure correctness and reliability. Then, with an iterative and evolutionary approach and in specific intervals, software behavior is extracted from formal models to be visualized in a semi-formal modeling language (such as UML). Visualized behavior increases and facilitates the interactions among project stakeholders (such as analyzers and designers), who are not, necessarily, familiar enough with complex mathematical concepts of formal methods. This also provides the possibility of applying design patterns on visualized behavior to improve its flexibility, reusability, and scalability. So, potential shortcomings and inconsistencies of the software behavior are identified and, consequently, required changes are applied and a newly improved version of the formal behavior is produced. The improved models are then re-formalized. The proposed approach is a step towards development of correct, reliable [6], flexible, reusable, and scalable software through enabling the construction of formal models from semi-formal ones (formalizing) and vice versa (visualization) during an iterative and evolutionary approach. References [26] and [27] present a case study in order to show the proposal applicability.

A detailed study regarding visualization and formalization is given in [1]. All related works are just a step in the right direction, but much more is yet to be done. The most frequently adopted approach is to define transformations between the visual and formal models [1][2][4][7][11][12][14][18][20][23][24]. However, a significant problem with these suggested approaches is that the transformation itself is often described imprecisely, with the result that the overall transformation task may be imprecise and incomplete. Consequently, the confidence the developer may have in the models is reduced, making the transformation approach unreliable.

The rest of this paper is organized as follows: Section 2 presents the motivation of the work by describing the reasons of integrating SFMMs and FMMs and its importance. The advantages and limitations of semi-formal and formal modeling methods are also investigated according to the literature review in this section. Section 3 defines the problem to be solved by the proposed approach. Finally, Section 4 discusses future work and draws conclusions.

II. MOTIVATION

This section describes the motivation of this paper via elaborating the benefits and limitations of SFMMs and FMMs according to the literature review.

A. Semi-formal Modeling Methods

SFMMs consist of a development method and a collection of notations for modeling software systems. UML is a unification of semi-formal modeling notations [23][31]. In summary, the main strengths of semi-formal techniques are as follows:

- Semi-formal notations are graphical, making them appealing, intuitive, and easy to be adopted. They are good at describing particular aspects of systems, abstracting away from details, and giving a good overall picture of what is being described. Sometimes they do not require a great deal of expertise to be understood. So they provide a good medium for discussions with clients.
- SFMMs are more than just a notation. They provide step-by-step guidance on how to approach problems. They encourage problem decomposition, which helps to reduce complexity.

Lack of a sound mathematical basis is the major weakness of SFMMs. They do not have a formal semantics. There are several problems related to their semantics:

- Either they are defined informally and vaguely using natural language, or they are defined through meta-modeling using some meta-language that is not precisely defined.
- Developers tailor the interpretation of diagrams to the problem at hand informally, tacitly, and sometimes unconsciously. This constitutes a source of confusion and ambiguity. Such misinterpretations might be even greater if the specification volume is large or development team crosses national and cultural boundaries [5].

These limitations lead to lack of means for mechanical analysis. They can also make the understanding more apparent than real; All is too easy and superficial, and the specifier is never confronted with the relevant issues. As a result, semi-formal methods cannot produce a precise, complete, and consistent specification. Specification plays a vital role in producing reliable software. Design and subsequent implementation is based upon the specification. Misunderstandings in the specification lead to the delivery of final applications that do not match user requirements. Moreover, testing is always carried out with respect to requirements as laid down in the specification. If the specification document is in any way ambiguous it is open to interpretation, and hence misinterpretation, making testing a rather inexact science.

Next section shows how the formal methods help in covering the weaknesses of SFMMs in specification, validation, and verification.

B. Formal Modeling Methods

FMMs are inspired by the way mature engineering disciplines build their artifacts: based on prediction and calculation with sound mathematical theories. Formal methods are utilized in all phases of software development process. FMMs, using formal languages such as Object-Z [7], provide the software with a precise, unambiguous, and abstract specification. In the next steps, required details are added to the initial abstract specification through an evolutionary process, including some design steps towards the final program. Accordingly, the initial formal specification is gradually refined. The refinement process will proceed until the generation of the final code [3]. Certain notations of formal methods support the notion of formal refinement. Formal refinement ensures that these refinements and transformations are correct. The correctness of a refinement is demonstrated through mathematical proof [23]. The benefits of using the formal modeling techniques have been recognized as follow:

- Formal modeling helps to gain a deep understanding of the system and its domain. It encourages the specifier to be abstract, yet rigorous and precise, forcing the modeler to ask all sorts of questions.
- Formal modeling clarifies the customer's vague ideas, revealing ambiguities, inconsistencies, and incompleteness in the requirements [23].
- The analysis of formal models can be used to support verification and validation. In verification, a formal model can be proved or checked for the satisfaction of desired properties, and that a refined design or implementation satisfies its specification. In validation, a requirements model can be checked against its requirements for white-box system testing either through animation or proof, and for black-box system testing by generating test cases from the model.

Although the increased rigor, precision and means of calculation that formal techniques offer seems indisputable [22], formal methods have not been taken up by industry. To explain this, many reasons have been hypothesized, education being one of them. So, FMMs have been embraced only in domains where reliability is absolutely crucial, such as safety-critical, security-critical, and high integrity systems [5]. Some other recognized shortcomings of FMMs are given below:

- Formal methods are notorious for being hard. Substantial efforts are required for formal modeling and verification. They are only effectively usable by highly-skilled experts.
- Most formal methods are suited to describe particular aspects of systems, but usually not all aspects. The problem occurs when all aspects need to be modeled.
- Formal methods provide a notation to write models and approaches to analyze them. However, software engineering practices require further support: guidelines, approaches to modeling, and patterns.

- The large variety of formal methods makes the choice of a particular one difficult.
- Most formal methods have little automated support beyond type-checking; developers are usually left the onus of performing proofs, which demand too much time and expertise for practical application.
- Practitioners need to be trained, and, since there is not much experience in using formal methods, the costs associated with their use are high. They also require an investment of time and money in specification, before any code is written.

The main conclusion is that FMMs and SFMMs have some advantages and limitations. Using only one of them as the sole approach leads not to satisfy all required aspects of software quality. This paper advocates an approach to building a framework for rigorous MDSE based on combining UML as a semi-formal language with Object-Z as a formal modeling language. SFMMs are supplemented with FMMs to introduce rigor in the development and to sweeten formal methods usage with diagrams.

III. PROBLEM DEFINITION

The problem to be investigated by this work is defined in this section. Solving this problem is a step towards developing high quality software. To do so, a new approach based on integrating Object-Z, as a formal, and UML, as a semi-formal modeling language, is proposed.

Using FMMs as the sole approach to software development leads to reliable software but with the following issues:

1. There are different interpretations of the initial informal requirements by customer and development team. There is also possibility of changing requirements during software development. These issues end to production of a software in contrary with the initial requirements. Fig. 1 illustrates this problem. There are two reasons for such an incorrect result: 1) there is no possibility of proving a perfect match between actual informal requirements and initial formal specification (δT_1), 2) it is difficult to do validation in the interval δT_2 because of the trouble in understanding the formal models. So formal methods, certainly brings us to a result that conforms to the initial formal specification (because of formal refinements), however, it does not necessarily conform to the actual informal requirements.

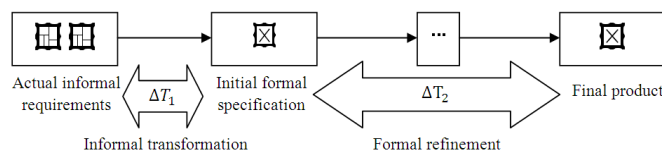


Figure 1. Imprecise interpretation of customer requirements

Visualization is an approach to solve the first problem, which leads to facilitate requirements validation in the interval δT_1 [15]. However, prototyping [16] is a better

solution for requirements validation. To do so, the formal specification should be transformed so that its new form can be executed or animated [16][32].

2. Even assuming that the initial formal specification exactly represents the actual informal functional requirements of the customer, we still do not reach the software with good enough quality of non-functional requirements such as reusability, flexibility, scalability, and extendibility. There are two reasons for such an unexpected result: 1) difficulty in utilizing the heuristic and narrative techniques of software engineering such as design patterns in the interval δT_2 , 2) inability of development team members such as analyzers and designers in understanding complex mathematical concepts of formal languages.

This work aims to solve the second problem. To do so, a new approach is suggested to improve software development process by combining Object-Z and UML to achieve high quality models of specification and design. In other words, this work proposes a new approach to develop high quality software through model transformation between Object-Z and UML. Fig. 2 illustrates a schematic view of the new proposed approach. Visualization facilitates understanding of the formal models and subsequently provides possibility of interaction with stakeholders, who are not necessarily familiar enough with complex mathematical concepts of formalism. It also simplifies using the narrative techniques of software engineering such as design patterns during software development process.

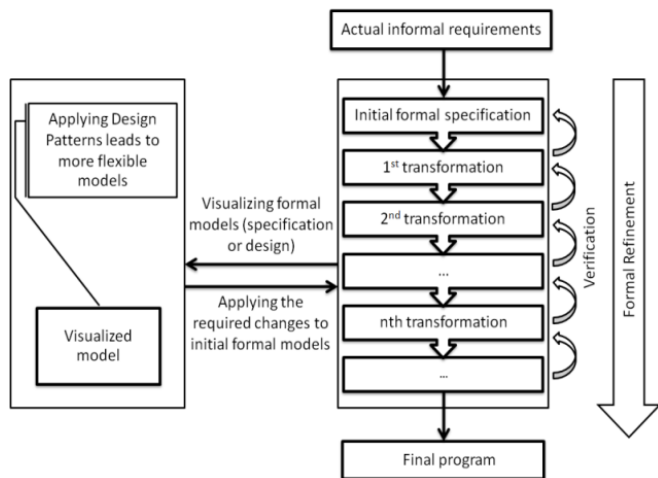


Figure 2. A schematic view of the proposed approach

As illustrated in Fig. 2, the *initial formal specification* is produced as the first artifact, according to the informal requirements of the stakeholders, using Object-Z. The initial formal specification is then refined using several transformations. Details of design are gradually added to the initial formal specification during transformations referred to as formal refinement. Formal refinement ensures correctness and reliability of the produced artifacts. In time

of reviewing the artifacts from the aspect of behavioral design patterns, the last refined formal artifact is visualized in a dominant semi-formal modeling language, i.e., UML. UML diagrams make it possible to revise the structure and behavior of the software from the view points of design patterns. The visualized model is then gradually revised using behavioral design patterns. Such a revision improves the flexibility and reusability of the visual models. The last revised visual model is then re-formalized in Object-Z. Repeatedly, the more required details of design or even implantation are augmented to the formal model using formal refinement. Such an iterative and evolutionary process continues until achieving a final product with the desired quality.

Software includes two aspects: structure (static) and behavior (dynamic) [16][21]. The proposed approach concentrates on software behavior. It facilitates analyzing and validating the behavioral aspect of formal models of software by visualization. Visualization prepares an appropriate ground to use heuristic and narrative principles of software engineering such as behavioral design patterns during software development process. So, the potential shortcomings and inconsistencies of the behavioral aspect of these models are identified. This improves the process of gradual augmentation of design decisions to the initial formal specification. Such an improvement leads to more flexibility, reusability, and scalability in developing software.

Design patterns are high level building blocks that promote elegance in software by ordering proven and timeless solutions to common problems in software design. Applying design patterns in software design has important effects on software quality metrics such as flexibility, reusability, scalability, and robustness [9][22][28][29][33]. There are three types of design patterns, including structural, creational, and behavioral patterns [8][9]. According to the above-mentioned goal of this work, we focus on the behavioral patterns (such as mediator, observer, and state) which shift your focus away from flow of control to let you concentrate just on the way objects are interconnected.

Object-oriented design encourages the distribution of behavior among objects to increase software reusability and flexibility. An important issue here is how peer objects know about each other. Peers could maintain explicit references to each other, but that would increase their coupling. Though distributing software into many objects generally enhances reusability and flexibility, proliferating interconnections tend to reduce reusability again. Moreover, it can be difficult to change the software behavior in any significant way, since behavior is distributed among many objects. Such a difficulty decreases the flexibility again. As a result, you may be forced to define many subclasses to customize the software behavior. The mediator pattern avoids this by introducing a mediator object between peers. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their

interaction independently. In this respect, we attempt to propose a systematic approach to improve the quality of formal design from the viewpoint of the mediator design pattern. That is, a formal design, in Object-Z, is received as an input, and then behavior of this formal design is abstractly visualized, in UML, as an output. Indeed, there is a focus on visualizing those aspects of the software behavior that are prone to revising from the viewpoint of the mediator pattern. Moreover, this approach, after full implementation, will automatically explore and recognize the suitable times in order to review the software behavior from the view point of mediator pattern throughout the software development process.

Moreover, software distribution into a collection of cooperating classes requires maintaining consistency among related objects. You don't want to achieve consistency by making the classes tightly coupled, because that reduces their reusability and flexibility. Observer pattern define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. In short, the required activities to visualizing the software behavior (by focus on those aspects of behavior that are required for revision from the viewpoint of observer pattern) include: 1) systematic elicitation of the objects that their states are dependent on each other, 2) visualizing the discovered objects as appropriate candidates for review, as well as 3) automatic proposing of the suitable times to review the software behavior from the viewpoint of observer design pattern.

Strategy pattern define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it. We use the strategy pattern when: 1) many related classes differ only in their behavior. Strategies provide a way to configure a class with one of many behaviors, 2) you need different variants of an algorithm. Strategies can be used when these variants are implemented as a class hierarchy of algorithms, and 3) a class defines many behaviors, and these appear as multiple conditional statements in its operations. Instead of many conditionals, move related conditional branches into their own Strategy class. So the Strategy pattern increases the flexibility through defining families of related algorithms, preventing subclassing, and eliminating conditional statements. Summarily, the required activities to visualize the software behavior from the viewpoint of strategy pattern include: 1) systematic discovery and elicitation of the classes that have several behaviors, 2) visualizing the discovered classes as appropriate candidates for review, as well as 3) automatic proposing of suitable times for software behavior review from the viewpoint of the strategy design pattern.

In all above-mentioned revision processes, the required changes, revealed after visualization, are re-formalized and thus the primary formal models are improved from the view point of behavioral design patterns. Software behavior is visualized from the required aspects using the suitable diagrams of UML such as class diagram [15][16]. Class

diagram makes it possible to revise the structure and behavior of the software from the view points of design patterns

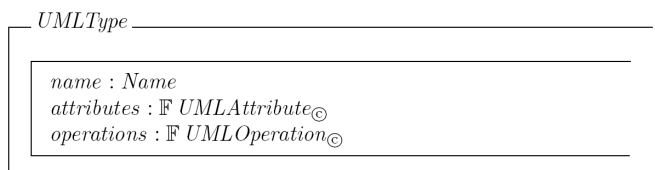
There has been an evolution in the way of transforming the models [10][17]. In model transformation, the most important issue is how to preserve the semantic and the syntactic structure of model elements. To do so, this work tends to propose a formal bidirectional meta-model-based transformation between UML and Object-Z. To do so, a meta-model should be formally defined for Object-Z in a similar architecture to which the UML meta-model is defined [11]. Then these meta-models will be used to define a systematic transformation between the two languages at the meta-level. In this way, we can provide a precise, consistent, and complete transformation between the two languages preserving the semantics and the syntactic structure of models presented in both languages. Since UML and Object-Z share basic object-oriented concepts, an attempt to create a systematic transformation between the two languages seems sound. Proposing such a meta-model-based mechanism is left for future work. In the following subsections, as an instance, we show how a common construct between UML and Object-Z such as class can be formally defined at the meta-level in a unified format using Object-Z [11]. Then a formal rule is presented to transform class construct from UML to Object-Z based on the formal definitions of class in UML and Object-Z at meta-level.

A. Formal definition of UML class

A UML class has a name, attributes, and operations. An attribute has a name, a visibility, a type, and a multiplicity. An operation has a name, a visibility, and parameters. Each parameter of an operation has a name and a given type. Prior to formalizing classes, we define a given set, *Name*, from which the names of all classes, attributes, operations, operation parameters, associations, and roles are drawn:

[*Name*]

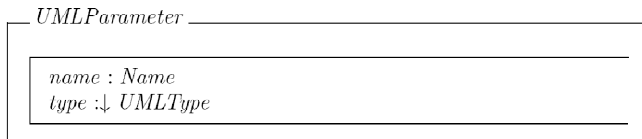
The class *UMLType*, as an Object-Z class, is a meta-type, from which all possible types in UML such as object types, basic types (integer and string), and so on can be derived. Each type has a name and contains a collection of its own features: attributes and operations. Thus, a circled *c* which models a containment relationship in Object-Z is attached to the types of *attributes* and *operations*.



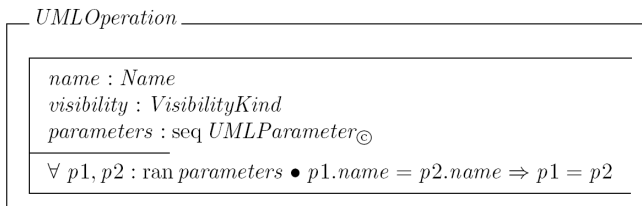
Attributes and parameters are also defined as follows. Variable *multiplicity* in *UMLAttribute* describes the possible number of data values for the attribute that may be held by

an instance. Visibility in UML can be private, public, or protected.

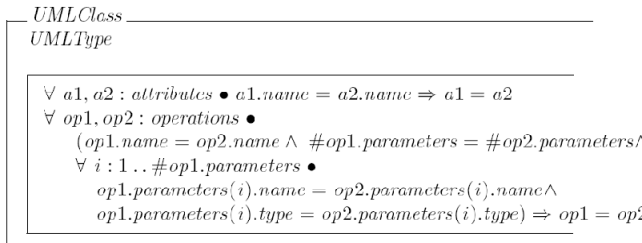
$VisibilityKind ::= private \mid public \mid protected$



Within an operation, parameter names should be unique.

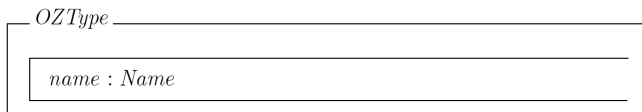


With these classes, an Object-Z class *UMLClass* is defined as follows. Since a class is a type, it inherits from *UMLType*. Attribute names defined in a class should be different and operations should have different signatures. The class invariant formalizes these properties.



B. Formal definition of Object-Z class

First, the semantics of type *Name* is extended to include the names of all classes, attributes, operations, and operation parameters in Object-Z. The following Object-Z class *OZType* is a formal description of metaclass *OZType*. In the metamodel, *OZType* is an abstract class from which all possible types in Object-Z can be derived.



The Object-Z class *OZAttribute* is a formal description of attributes. Each attribute has a name, a type, and a multiplicity constraining the number of values that the attribute may hold. It also has an attribute, *relationship*, to represent whether this attribute models a relationship

between objects. Like UML, relationships between objects can be common reference relationships, shared, or unshared containment relationships. For this, we define an enumeration type, *RelationshipKind*, which can have *relNone*, *reference*, *sharedContainment*, and *unsharedContainment* as its values. The value *relNone* represents pure attributes of a class. When an attribute models a relationship, the attribute *navigability* represents the direction of the relationship (although the navigability of a relationship is modeled implicitly in Object-Z). Visibility in Object-Z can be public or private.

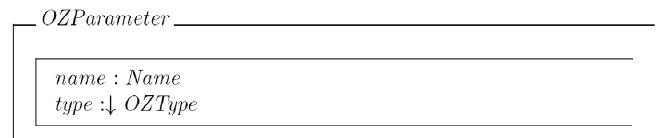
$VisibilityKind ::= private \mid public$

$RelationshipKind ::= relNone \mid reference \mid$
 $sharedContainment \mid unsharedContainment$

$NavigabilityKind ::= navNone \mid bi \mid one$



We formalize *OZParameter* and *OZOperation* in the same way as *OZAttribute*.



Now we are in the position to formalize Object-Z classes. An Object-Z class named *OZClass* is a formal description for classes in Object-Z. Since classes are a kind of type, *OZClass* inherits from *OZType*. The attribute *superclass* maintains inheritance information of classes. Each class has its own attributes and operations defining static and dynamic behaviors of its instances. Circular inheritance is not allowed. Attribute and operation names should be unique within a class. These properties are specified in the predicate of *OZClass*. Functions *directSuperclass* and *allSuperclass* return direct superclass of a class and all inherited *superclasses* of a class, respectively.

$\begin{aligned} \text{directSuperclass} &: \text{OZClass} \rightarrow \mathbb{P} \text{OZClass} \\ \text{allSuperclass} &: \text{OZClass} \rightarrow \mathbb{P} \text{OZClass} \end{aligned}$		
$\forall oc : \text{OZClass} \bullet$ $\begin{aligned} \text{directSuperclass}(oc) &= oc.\text{superclass} \\ \text{allSuperclass}(oc) &= \text{directSuperclass}(oc) \cup \\ &(\cup \{sco : \text{directSuperclass}(oc) \bullet \text{allSuperclass}(sco)\}) \end{aligned}$		
$\frac{\text{OZClass}}{\text{OZType}}$		
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;"> $\begin{aligned} \text{superclass} &: \mathbb{F} \text{OZClass} \\ \text{attributes} &: \mathbb{F} \text{OZAttribute}_{\odot} \\ \text{operations} &: \mathbb{F} \text{OZOperation}_{\odot} \end{aligned}$ </td> </tr> <tr> <td style="padding: 5px;"> $\begin{aligned} \text{self} &\notin \text{allSuperclass}(\text{self}) \\ \forall a1, a2 : \text{attributes} \bullet a1.\text{name} = a2.\text{name} &\Rightarrow a1 = a2 \\ \forall op1, op2 : \text{operations} \bullet op1.\text{name} = op2.\text{name} &\Rightarrow op1 = op2 \end{aligned}$ </td> </tr> </table>	$\begin{aligned} \text{superclass} &: \mathbb{F} \text{OZClass} \\ \text{attributes} &: \mathbb{F} \text{OZAttribute}_{\odot} \\ \text{operations} &: \mathbb{F} \text{OZOperation}_{\odot} \end{aligned}$	$\begin{aligned} \text{self} &\notin \text{allSuperclass}(\text{self}) \\ \forall a1, a2 : \text{attributes} \bullet a1.\text{name} = a2.\text{name} &\Rightarrow a1 = a2 \\ \forall op1, op2 : \text{operations} \bullet op1.\text{name} = op2.\text{name} &\Rightarrow op1 = op2 \end{aligned}$
$\begin{aligned} \text{superclass} &: \mathbb{F} \text{OZClass} \\ \text{attributes} &: \mathbb{F} \text{OZAttribute}_{\odot} \\ \text{operations} &: \mathbb{F} \text{OZOperation}_{\odot} \end{aligned}$		
$\begin{aligned} \text{self} &\notin \text{allSuperclass}(\text{self}) \\ \forall a1, a2 : \text{attributes} \bullet a1.\text{name} = a2.\text{name} &\Rightarrow a1 = a2 \\ \forall op1, op2 : \text{operations} \bullet op1.\text{name} = op2.\text{name} &\Rightarrow op1 = op2 \end{aligned}$		

C. Formal transformation rule for class

As illustrated in Fig. 3, a formal description for mapping a UML class to an Object-Z class is given by function *mapUMLClassToOZ* that takes a UML class and returns the corresponding Object-Z class. The UML class name is used as the Object-Z class name. All attributes of the UML class are declared as attributes in the state schema of the corresponding Object-Z class. Also, each operation in the UML class is translated to an operation schema. In UML, types of attributes are a language-dependent specification of the implementation types and may be suppressed. Types of attributes in Object-Z are language-independent specification types and cannot be omitted. Operations parameters are similar. Detailed transformation rules regarding attribute types and operation parameter types are not provided. Instead, an abstract function, *convType* is defined that maps a UML type to an Object-Z type.

$\text{convType} : \downarrow \text{UMLType} \rightarrow \downarrow \text{OZType}$
$\text{mapUMLClassToOZ} : \text{UMLClass} \rightarrow \mathbb{P} \text{OZClass}$
$\forall uc : \text{UMLClass} \bullet$ $\begin{aligned} \text{mapUMLClassToOZ}(uc) &= \{oc : \text{OZClass} \mid uc.\text{name} = oc.\text{name} \wedge \\ &\forall ua : uc.\text{attributes} \bullet \\ &\quad \exists oa : oc.\text{attributes} \bullet \\ &\quad \quad oa.\text{name} = ua.\text{name} \wedge oa.\text{type} = \text{convType}(ua.\text{type}) \wedge \\ &\quad \quad oa.\text{visibility} = ua.\text{visibility} \wedge oa.\text{multiplicity} = ua.\text{multiplicity} \wedge \\ &\quad \quad oa.\text{relationship} = \text{relNone} \wedge oa.\text{navigability} = \text{navNone} \\ &\forall uo : uc.\text{operations} \bullet \\ &\quad \exists oo : oc.\text{operations} \bullet \\ &\quad \quad oo.\text{name} = uo.\text{name} \wedge oo.\text{visibility} = uo.\text{visibility} \\ &\forall up : \text{ran } uo.\text{parameters} \bullet \\ &\quad \exists op : \text{ran } oo.\text{parameters} \bullet \\ &\quad \quad op.\text{name} = up.\text{name} \wedge op.\text{type} = \text{convType}(up.\text{type})\} \end{aligned}$

Figure 3. Formal transformation rule for class

Visibility and multiplicity features are mapped to those of Object-Z.

An appropriate evaluation method helps determine the overall effects of the new approach in relation to promised objectives. This method also includes any recommendations for improvement. As previously mentioned, the major goal of introducing the new approach is to improve the process of formal modeling (including specification and design) of software behavior based on visualization. So we should measure the capability of the suggested approach in satisfying the expected goals. Evaluation criteria of the proposed approach include: 1) correspondence percentage between visual and formal models transformed to each other by the proposed meta-model based transformation method, 2) the amount of increasing the quality (such as flexibility, reusability, and scalability) of the developed software using the proposed method. As we intend to propose a meta-model-based transformation approach, a formal and systematic transformation between the two languages will be defined at the meta-level. So we can prove the correctness, precision, and completeness of the transformation mathematically. In addition, to demonstrate the proposed approach, a high quality multi-lift system as a non-trivial case study will be developed using the proposed approach.

IV. CONCLUSION AND FUTURE WORK

Although, the widespread use of SFMMs in mainstream software development provides the possibility of developing flexible, reusable, and scalable software, it does not lead to software reliable enough for safety-critical purposes. Their semantics are not well defined. FMMs have precise semantics, allowing for unambiguous models of systems to be specified and designed. However, their use has not been widely adopted due to the mathematical nature of the languages.

Investigation of integrated methods has taught us many things: (a) visual modeling notations and formal methods can coexist within the same development and complement each other when developing software models, (b) this coexistence is useful and provides many benefits, and (c) formalization of diagrammatic languages, like UML, and visualization of formal models, like Object-Z, is far from trivial.

This work proposes a new approach for integrating visual and formal models to ensure achieving more flexible, reusable, scalable, yet reliable software. To do so, we propose a precise mechanism to transform graphical models into formal specifications and vice versa. This work intends to present a meta-model-based transformation between UML and Object-Z. The two languages will be defined in terms of their meta-models, and a systematic transformation between the models will be provided at the meta-level. As a result, we provide a precise, consistent, and complete transformation between visual models in UML and formal models in Object-Z. Visualizing the formal models of the software behavior prepares an appropriate ground to revise them from the viewpoints of design patterns. Although, this paper draws the path towards solving the defined problem and achieving the promised goals, proposing the meta-model-based transformation is left for future work.

REFERENCES

- [1] N. Amálio, Generative frameworks for rigorous model-driven development, PhD thesis, Dept. of Computer Science, University of York, 2006.
- [2] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "UML2Alloy: A Challenging Model Transformation", Proc. ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS), pp. 436-450, 2007.
- [3] D. Björner, Software Engineering 3: Domains, Requirements, and Software Design, Springer, 2006.
- [4] F. Bouquet, F. Dadeau, and J. Gros Lambert, "Checking JML specifications with B machines", Proc. ZB 2005, LNCS, vol. 3455, Springer, pp. 434-453, 2005.
- [5] Q. Charatan and A. Kans, Formal Software Development: From VDM to Java, Palgrave Macmillan, 2004.
- [6] R. N. Charette, "Why software fails", IEEE Spectrum, vol. 42(9), pp. 42-49, 2005.
- [7] R. Duke and G. Rose, Formal Object-Oriented Specification Using Object-Z, MacMillan Press, 2000.
- [8] E. Freeman, E. Freeman, and B. Kathy Sierra, Head First Design Patterns. O'Reilly Media, First edition, 2004.
- [9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Pattern: Elements of Reusable Object-Oriented Software, Addison-Wesley Publishing Company, Fifth printing, 1995.
- [10] M. Kessentini, M. Wimmer, H. Sahraoui, and M. Boukadoum, "Generating Transformation Rules from Examples for Behavioral Models", Proc. Second International Workshop on Behavior Modeling: Foundation and Applications, Paris, France, 2010.
- [11] S. Kim and D. Carrington, "A formal meta-modeling approach to a transformation between the UML state machine and Object-Z", Proc. ICFEM 2002, LNCS, vol. 2495, Springer, pp. 548-560, 2002.
- [12] H. Miao, L. Liu, and L. Li, "Formalizing UML models with Object-Z", Proc. ICFEM2002, Springer-Verlag, pp. 523-534, 2002.
- [13] J. Bowen and M. Hinchey, "Seven more myths of formal methods", IEEE Software, vol. 12 (4), pp. 34-41, 1995.
- [14] I. Poernomo, "Proofs-as-model-transformations" LNCS, vol. 5063, pp. 214-228, 2008.
- [15] F. Polack, "SAZ: SSADM version 4 and Z", Proc. Software Specification Methods: an overview using a case study, Springer, pp. 21-38, 2001.
- [16] I. Porres, "Modeling and Analyzing Software Behavior in UML", PhD thesis, Department of Computer Science, Abo Akademi University, Finland, 2001.
- [17] R. Pressman, Software Engineering: A Practitioner's Approach, 7th edition, McGraw Hill, 2009.
- [18] S. K. Rahimi, "Specification of UML Model Transformations", Proc. Third International Conference on Software Testing, Verification and Validation, pp. 323-326, Paris, 2010.
- [19] R. Razali, C. Snook, M. Poppleton, and P. Garratt, "Usability Assessment of a UML-based Formal Modeling Method Using Cognitive Dimensions Framework", Human Technology, 2008.
- [20] D. C. Schmidt, "Model-driven engineering", IEEE Computer, 39 (2), pp. 25-31, 2006.
- [21] C. Snook and M. Butler, "UML-B: Formal modeling and design aided by UML", ACM Trans. Softw. Eng. Methodol, vol. 15 (1), pp. 92-122, 2006.
- [22] I. Sommerville, Software Engineering, 8th edition, Addison Wesley, June 4, 2006.
- [23] S. Stepney, F. Polack, and I. Toyn, "Patterns to guide practical refactoring: examples targeting promotion in Z", Proc. ZB 2003, Finland, LNCS, vol. 2651 of, Springer, pp. 20-39, 2003.
- [24] J. R. Williams, Automatic Formalization of UML to Z, MSc Thesis, Department of Computer Science, University of York, 2009.
- [25] J. Ludewig, "Models in software engineering – an introduction", Software and Systems Modeling, vol. 2(1), Springer-Verlag, 2003.
- [26] A. Rasoolzadegan and A. Abdollahzadeh, Specifying a Parallel, Distributed, Real-Time, and Embedded System: Multi-Lift System Case Study, Technical Report, Information Technology and Computer Engineering Faculty, Amirkabir University of Technology, Tehran, Iran, 2011: <http://ceit.aut.ac.ir/~86131901/Publications.htm>.
- [27] A. Rasoolzadegan, A. Abdollahzadeh, "Empirical Evaluation of Modeling Languages Using Multi-Lift System Case Study", Proc. MSV'11: The 8th annual International Conference on Modeling, Simulation and Visualization Methods, Las Vegas, Nevada, USA, 2011.
- [28] S. Blazy, F. Gervais, and R. Laleau, "Reuse of specification patterns with the B method". Proc. ZB 2003, Turku, Finland, LNCS, vol. 2651, Springer, pp. 40-57, 2003.
- [29] A. Flores, R. Moore, and L. Reynoso, "A formal model of object-oriented design and GoF design patterns", Proc. FME 2001, LNCS, vol. 2021, pp. 223-241, Springer, 2001.
- [30] A. H. Eden and T. Mens, "Measuring Software Flexibility", IEE Software, vol. 153(3), pp. 113-126. London, UK: The Institution of Engineering and Technology, 2006.
- [31] OMG, Object Constraint Language (OCL). version 2.0, Object Management Group, 2006: <http://www.uml.org>.
- [32] H. Liang, J. Song Dong, J. Sun, and W. Wong, "Software monitoring through formal specification animation", Innovations in Systems and Software Eng., vol. 5(4), pp. 231-241, 2009.
- [33] S. Kim and D. Carrington, "A rigorous foundation for pattern-based design models", Proc. ZB 2005, LNCS, vol. 3455, Springer, pp. 242-261, 2005.

Non-Functional Requirements for Business Processes in the Context of Service-Oriented Architectures

Oliver Charles

*agilTech Information Technologies GmbH
Am Krebsgraben 15
D-78048 Villingen-Schwenningen, Germany
oliver.charles@agiltech.de*

Bernhard Hollunder

*Furtwangen University of Applied Sciences
Robert-Gerwig-Platz 1
D-78120 Furtwangen, Germany
hollunder@hs-furtwangen.de*

Abstract—We present novel concepts to formalize and apply non-functional requirements (NFRs) for business processes in the context of Service-Oriented Architectures (SOAs). Today, popular languages for modeling business processes do not support the specification of NFRs in a systematic manner. However, there is a strong demand to explicitly address such requirements when designing and deploying software systems. In this paper, we elaborate an extension for BPMN (Business Process Model and Notation) towards the modeling of NFRs. A key feature is the tool independent representation of NFRs, which will be achieved by applying the widely used WS-Policy standard. Our approach also covers the mapping of the specified NFRs to the technical level represented by BPEL (Business Process Execution Language). For the monitoring of NFRs we exploit techniques from Complex Event Processing (CEP). A key characteristic of our solution is its coherence: from NFRs modeling at design level to their technical enforcement and dynamic validation during execution. The feasibility of our approach has been demonstrated by a proof of concept implementation based on NetBeans, Glassfish ESB, IEP as CEP implementation, and the BPEL Service Engine.

Keywords—Non-functional requirements, Business process, BPMN, BPEL, SOA, WS-Policy, Web services, Quality of service

I. INTRODUCTION

When introducing a Service-Oriented Architecture (SOA) for some enterprise, the definition of appropriate business processes as well as services plays a crucial role. A business process can be viewed as a well-defined sequence of activities to achieve a particular business goal. In order to exchange data with back-end systems (e.g., ERP systems, specific business applications and database systems), business processes typically use course-granular services, which hide the technical details of the services' implementation. Today, services are often realized with the Web services technology. In other words, a business process within a SOA composes a set of Web services in such a way that higher business goals will be obtained.

When employing Web services in the area of so-called mission critical business applications, "pure" Web services are not sufficient. This is because in such an environment non-functional requirements (NFRs) such as message reliability, confidentiality, availability and performance must be addressed. The importance of NFRs for Web services has been stressed elsewhere (see e.g., [1] or [7]). There are proven standards such as WS-SecurityPolicy [2] bringing selected NFRs to Web services.

As Web services are composed by business processes, the interaction of NFRs at service level on the one hand and at process level on the other hand must be clearly defined. Hence, it is crucial to assign – explicitly or implicitly – NFRs to business processes such as time and resource consumption, auditability and scalability (e.g., as described by Adam and Doerr in [3]).

In the past, there has been much work on modeling functional requirements of business processes. The most prominent approaches used in SOA infrastructures are the *Business Process Model and Notation* (BPMN) and the *Business Process Execution Language* (BPEL). While BPMN primarily focuses on the graphical representation of business processes, BPEL tackles technical aspects such as the mapping to Web services to be invoked during process execution. It should be noted that there is broad tool support, for an overview see e.g., [4].

Currently, there is only very limited support for specifying NFRs for business processes, though. In fact, the BPMN and BPEL do not provide language features for including NFRs features. As a consequence, when transforming a process model into an executable format the application developer must pollute the business logic with mechanisms for realizing the desired NFRs. This approach, however, would strongly limit the reusability and adaptability, if the solution should be deployed in an environment where different sets of NFRs must be supported.

In this paper, we present a novel approach for formalizing NFRs for business processes that overcomes these deficits. Special focus lies on its coherence, because we not only cover the modeling of NFRs at design level, but also their technical enforcement and their dynamic validation during execution. Our approach comprises the following aspects:

- Modeling of NFRs with BPMN and BPEL by exploiting standard extension mechanisms.
- Enforcement strategy for NFRs based on Web service handlers.
- Usage of standards such as WS-Policy to formalize NFRs at the technical level.
- Static and dynamic validation of NFRs.
- Tool support and proof of concept.

The paper is structured as follows. The next section will give a short introduction to the underlying technologies required to understand our approach. Related work will be

discussed in Section three, followed by a detailed description of our solution. Section five will cover the proof of concept implementation. Conclusions and open issues are part of the final section.

II. FOUNDATIONS

This section briefly introduces the most important concepts and techniques as required for the understanding of our approach. We start with *Business Process Management*, which is the general area our results apply to. Then we provide background information to *Non-Functional Requirements*, followed by a short review of *WS-Policy*, which is a well-known and widely used standard for formalizing NFRs for Web Services and SOAs.

Due to limited space, this paper does not give an introduction to BPMN and BPEL. Hence, we assume an understanding of the basic concepts of these technologies.

A. Business Process Management

Business Process Management (BPM) includes concepts, methods, and techniques to support the design, implementation, enactment, monitoring, and strategy alignment of business processes. In the context of SOAs, BPM focuses on how business processes can be automated using SOA infrastructure elements. The target is not only a high automation of processes, but also to enable development and management to react in a flexible and agile manner on changing business or technical requirements.

BPM covers the following topics:

- Strategy phase
- Design phase
- Execution phase
- Monitoring phase.

As the name of the first phase indicates, the main focus is the elaboration of the mid- to long-term alignment of an enterprise and how IT can be leveraged to automate and optimize business processes. Having defined the strategic goals, in the design phase the identified business processes are brought to “IT-level”. This includes a proper description from which an implementation will be derived. The usage of graphical modeling languages – in particular BPMN and BPEL – is not only advantageous for the domain experts, but also helps bridging the gap to the implementation level. While the execution phase is concerned with the usage of the implemented business processes by clients, the goal of the monitoring phase is to receive data regarding the runtime behavior such as identification of bottlenecks, quantity of invoked processes, and performance analysis.

As already mentioned in the introduction, our solution considers NFRs at the design, implementation, and monitoring level. That is the reason why we term it *coherent*.

B. Non-Functional Requirements

In system and software engineering there are mainly two categories of requirements: *functional* and *non-functional requirements*. A functional requirement describes a specific business or technical functionality of a system in terms of the input/output behavior. In contrast, a non-functional require-

ment addresses a quality of service (QoS) attribute of the implementation. In software engineering, there was (and still is) much research on NFRs for software systems. Standardization organizations such as ISO have identified manifold aspects (see e.g., ISO/IEC 9126 [6], which is superseded by ISO/IEC 25000 [5]).

There are several publications that consider NFRs in the specific context of SOA, e.g., by O’Brien, Merson, and Bass [7]. OASIS [1] gives a classification of different types of NFRs (which are called quality factors). Besides others, the following topics are covered:

- duration and response time
- throughput
- availability and reliability
- standard conformance
- observability
- security aspects such as confidentiality, authentication, authorization, integrity, and non-repudiation
- pricing and accounting
- robustness.

Let us make some remarks. Even though we can find in the literature characterizations of NFRs, there are often differences regarding their exact meaning and definition. Some of them can be described by a formula; e.g., response time, duration, and availability. The behavior of other NFRs such as integrity can be defined in terms of functions for digital signature. Robustness is an example for an NFR that has diverse facets such as error tolerance, often described as the ability to deal with erroneous input. A business process, for example, should not crash or run into an inconsistent state if it is called with invalid parameter values.

C. WS-Policy

WS-Policy [8] is a specification of the W3C and provides a policy language to formally describe “properties of a behavior” of services. A WS-Policy description is a collection of so-called assertions. A single assertion may represent a capability, a requirement or a constraint and has an XML representation. An example for an assertion is

```
<Performance max_runtime_minutes="15"/>
```

which formalizes a condition for the runtime behavior of a particular business process.

WS-Policy introduces operators to form policies, which are basically sets of assertions. Policies can be attached via the WS-PolicyAttachment [9] specification to other entities such as a BPEL process description and a Web service’s WSDL. We will come back to this issue when introducing our solution.

III. RELATED WORK

In [10], Pavlovski and Zou present an approach to model NFRs for business processes in a graphical manner. They introduce extensions for BPMN, the enforcement on the technical level (e.g., in BPEL) has not been elaborated, though. For the modeling of NFRs, Zou and Pavlovski propose two extensions of BPMN: i) an “operating condition” artifact and ii) a “control case” artifact. With an

operating condition artifact, a business process modeler should be able to connect NFRs such as security, performance or availability to activities of the BPMN process model. The use of the control case artifact is optional and is introduced to refine an operating condition artifact. From a more technical point of view, a control case artifact is a reference to a table containing detailed information about the modeled NFRs.

The approach of Rodriguez et al. [11] also tackles the modeling of NFRs within BPMN. However, their solution is restricted to the modeling of security requirements. They do not extend the standardized artifacts of BPMN, but rather implement new Business Process Diagram (BPD) core elements. In this context it is described how to extend the BPD meta-model towards the coverage of security issues. The mapping of “security-enhanced” process models to the technical level (as in the approach in [10]) is not addressed.

Tai et al. [12] explain a new idea about how transactional behavior can be modeled as NFRs within BPEL. To express this with XML, the authors use WS-Policy [8] in combination with WS-PolicyAttachment [9]. They directly attach WS-Policy descriptions to selected BPEL elements within the process document. Proposed elements are for example `<partnerLink>` or `<scope>`. To enforce the attached WS-Policy descriptions, Tai et al. assume a coordination middleware, which executes the BPEL-process taking into account the NFRs.

Charfi et al. present in [13] another approach to model non-functional requirements with BPEL. Their approach is based on well-known standards and specifications such as WS-Policy, WS-PolicyAttachment and XPath. It has to be mentioned that their approach is not a completely new one but a combination of the mentioned standards.

To sum up, there are several approaches that extend process models towards NFRs. However, they either focus on BPMN or BPEL. As we will see in the next section, our solution – beside other features – includes the mapping from BPMN to BPEL.

IV. THE OVERALL ARCHITECTURE

A. Modeling NFRs in BPMN

BPMN does not provide explicit language constructs for modeling NFRs. Basically, there are two options to overcome this limitation: i) introducing new language features optimized for modeling NFRs, and ii) applying existing artifacts in a specific way. A disadvantage of the first alternative would be missing support by existing BPMN tools. Therefore, we pursue the second approach.

A so-called text annotation is a standard artifact of BPMN, which allows one to attach auxiliary information to model elements. The following figure gives an example:



Figure 1. QoS artifact for BPMN.

At the left hand side there is some business process activity. In order to impose NFRs for this activity, we assign a text artifact. In this approach, we distinguish between arbitrary text annotations and those, which formalize NFRs. The latter are called “QoS artifacts” and are text artifacts with a particular content and specific syntax.

In our approach, we support the following syntax: The prefix “QoS” indicating a QoS artifact is followed by a category name, which specifies a particular NFR. In the previous example, we impose a performance restriction to the modeled activity. Finally, a set of attribute/value-pairs define the specific properties for the NFR.

The content of a QoS artifact is a text with some well-defined structure, which will be mapped to XML. In order to support syntax checking, we have defined XML schemas for the supported NFRs. Due to lack of space, we omit the description of the schemas.

We have defined a library comprising well-known QoS artifacts. Each QoS artifact comes with a modeling manual describing its meaning, formalizing the required syntax by means of an XML schema, and optional modeling examples. It should be noted that the set of predefined QoS artifacts could be extended by additional NFRs basically by defining its XML schema.

To sum up, our NFRs modeling approach is a light-weight solution, which reuses standard artifacts supported by BPMN tools. As a consequence, process models can be exchanged between different tools without losing NFRs model information. The usage of XML schemas not only specifies the specific syntax but also allows the automated validation. Last but not least, the QoS artifacts library can be reused in different settings.

B. Modeling NFRs in BPEL

As mentioned above, BPEL does not support the modeling of NFRs in a direct manner. In [13] it has been shown how to overcome this limitation by applying the standards WS-Policy, WS-PolicyAttachment and XPath. The main idea is to link BPEL process elements to a WS-Policy description. Such a description contains WS-Policy assertions formalizing NFRs (see Section II-C). A well-known set of assertions for the security domain has been introduced in [2].

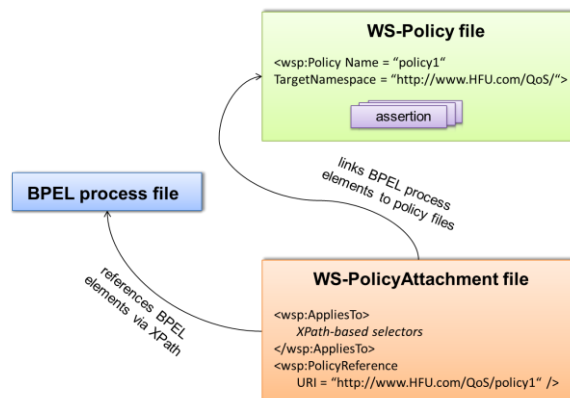


Figure 2. Assigning WS-Policy to BPEL.

Figure 2 depicts the linkage between the BPEL process and the policy description. A WS-PolicyAttachment file contains an <AppliesTo> entry referring to the BPEL element to which the WS-Policy description should be applied. The latter is linked via the <PolicyReference> element, which is also introduced by the WS-PolicyAttachment specification. As we apply XPath for selecting the targets, this approach exclusively uses well-known and widely supported specifications.

This concept clearly separates i) the logic of the business process and ii) the required NFRs. As a consequence, both parts of the overall application can evolve independently from each other, which has a positive effect on maintainability, reusability and adaptability of the solution. As an example, consider a WS-Policy file that formalizes a particular set of NFRs. The policy can be applied to several business applications. As a consequence, this not only increases reusability of the required “NFRs patterns” but also guarantees conformance to corporate compliance rules.

C. Transformation of NFRs – From BPMN to BPEL

Having described how to represent NFRs within BPEL, we are now able to consider the mapping from QoS artifacts in a BPMN model to WS-Policy descriptions for BPEL. It should be noted that we do not consider the general transformation rules mapping BPMN elements to BPEL elements, because they are part of most BPMN/BPEL modeling tools.

To map NFRs we proceed as follows: For each QoS artifact, we create both a WS-PolicyAttachment file as well as a WS-Policy file. The assertions contained in the policy description correspond to the NFRs of the QoS artifacts. These assertions in turn have references to the XML schema definition and the modeling manual, respectively. After all WS-Policy documents have been created, they are used by the corresponding WS-PolicyAttachment files to link the required policies to BPEL process elements as already described.

D. Enforcement of NFRs

This section is concerned with the question how the modeled NFRs can be enforced. Basically, we observe that there are two targets to which the modeled NFRs will be applied: i) the business process itself, and ii) the communication between a BPEL service and an underlying Web service. From a modeling perspective, we use the following convention: if the category name of a QoS artifact starts with “WSComm_”, the latter target is meant, otherwise the NFR applies to the business process.

If a policy relates to the business process itself, which means that the described prefix is not set by the BPMN modeler, the Web service developer has to extend the Web service’s application logic, i. e., the source code.

If a policy relates to the service communication, the Web service developer has the responsibility to enforce the NFRs with the help of interceptors (also called handlers), which can be installed in SOA infrastructures and manipulate the outgoing and incoming messages. Details can be found e.g., in [14].

In order to enforce a specified behavior, typically an appropriate WS-Policy description will be attached to the Web service’s WSDL as well as to BPEL process elements (see Figure 3). This policy may for example specify that the invoker (e.g., the BPEL service) must encrypt the parameter values passed to the Web service, which in turn is able to decrypt these values. For standard NFRs (such as security and reliable messaging) Web services frameworks typically provide respective handlers. For other NFRs such as accounting and resource consumption specific handlers must be configured.

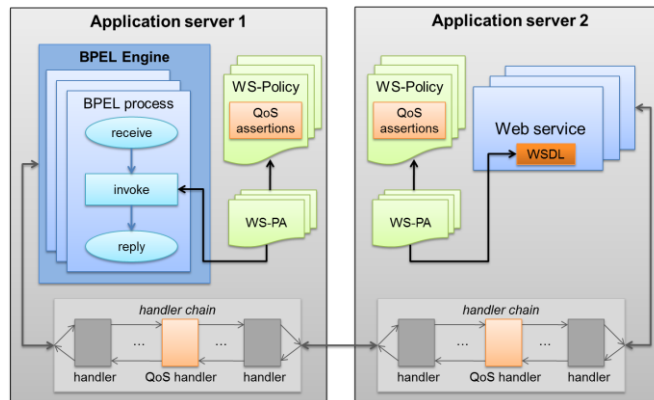


Figure 3. QoS enforcement through handlers.

To support several NFRs, all the required handlers must be installed. This can be achieved by using so-called handler chains supported by Web services frameworks. Before a request is delivered to the service implementation, each handler will be invoked.

E. Validation of NFRs

Our architecture also includes components for validating NFRs. We distinguish between static and dynamic validation. During the static validation process, the NFRs contained in a BPMN process diagram will be checked against the WS-Policy descriptions of the underlying Web services implementations. Static validation can be automated by applying WS-Policy compatibility algorithms such as WS-Policy intersection [8] and semantic policy differencing [15].

Performance is an example of an NFR where dynamic validation must be applied. As the actual execution time of a process depends on factors, which are not determinable *a priori* (e.g., server consumption, network latency and user interaction), a monitoring system is required in order to continuously observe the infrastructure. To provide a monitoring system with the required data, so-called sensor components (such as JMX and NFRs handlers, see [16]) can be installed in SOA infrastructures.

This system will inform, for instance, the system administrator if some NFRs are violated. Depending on the severity of the violation (e.g., leakage of sensible data) actions may be immediately performed such as shutting down a service or a server.

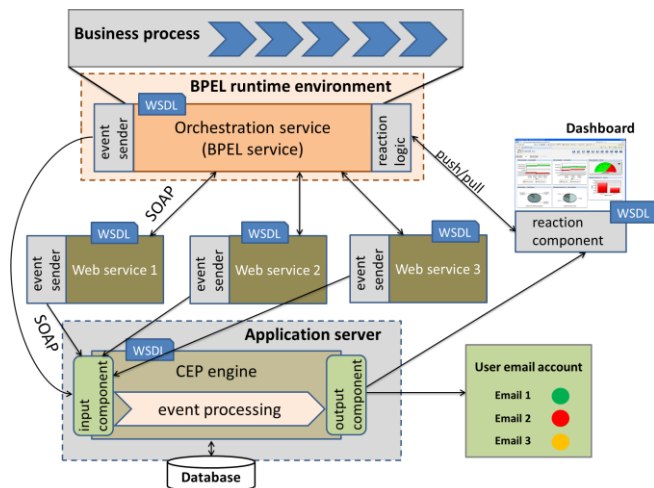


Figure 4. Dynamic validation with CEP [17].

Complex Event Processing (CEP) [18] has been introduced as a technology to find correlated data items in a continuous flow of data. The data items to be selected are specified by patterns defined, for instance, with the Continuous Query Language (CQL). It turned out that the conditions, which indicate a violation of an NFR during execution, can be appropriately defined as CEP patterns. Figure 4 illustrates the integration of an abstract CEP engine in a BPEL/Web services environment. We have identified the following components:

- input component
- output component
- CEP engine
- reaction component
- event senders.

The input component receives events from the BPEL engine and the Web services, respectively. The so-called event senders, which are specific implementation of the above mentioned sensor components, inform the CEP engine about significant actions in the business application (e.g., transition within the business process, Web service invocation, passing of non-encrypted sensible data, etc.). Subsequently, the input component passes the received events to the CEP engine. As soon as the CEP engine detects data items that match a CEP pattern, a new (complex) event will be created. The output component has the responsibility to pass it to a user (e.g., via SMTP) or to the reaction component, which in turn will inform the orchestration service, or to a management system (via Web service invocation) about the violation of an NFR.

V. PROOF OF CONCEPT

The overall architecture presented in the previous section is quite generic and can be instantiated in different ways. In order to show the feasibility of our approach we have developed a proof of concept implementation based on

NetBeans IDE and Glassfish ESB. This combination comprises the following tool set:

- *BPMN/BPEL designer* to model business processes.
- *BPEL runtime environment* for executing BPEL processes.
- *Web services* development, deployment and runtime environment.
- *Intelligent Event Processing (IEP)* service engine as implementation of a CEP engine.

The BPMN/BPEL designer allows the graphical modeling of business processes according to the BPMN and BPEL languages. It should be noted that only those BPMN elements are supported by the tool, which can be mapped to the XML BPEL process file.

One of these elements is the documentation artifact, analogous to the common BPMN text artifact that allows the attachment of comments to elements in a BPEL process. These comments are transformed to the common BPEL tag <documentation> within the underlying XML BPEL process file. To avoid this intrusion, we extended the BPMN/BPEL designer by a new QoS artifact (see Figure 5) with which it is possible to implement our introduced transformation process as described above.

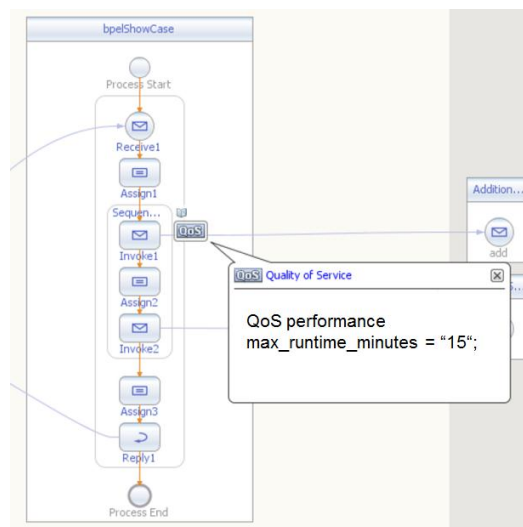


Figure 5. QoS artifact in the BPMN/BPEL designer.

With the NetBeans composite application display it is possible for a Web service developer to attach handlers via the context menu not only to the BPEL service but also to the Web services, which are invoked. This enables an easy configuration of handlers required for enforcing the defined NFRs.

The IEP service engine comes with a graphical modeling language for selecting, transforming and aggregating events. This modeling language also provides predefined types for output components, e.g., datasets, database tables and dashboard formats. It is also possible to generate WSDL interfaces for the input components and their implementations as Web services, which can be used by the event senders.

Hence, IEP enables the validation of a business process during its execution.

Independent of IEP, it is also possible to make theoretical commitments before process execution. For example, a Web service developer wants to check if the modeled runtime of a business activity complies with the modeled runtime of the Web services. Therefore, the WS-Policy assertion of the business activity has to be checked against the sum of runtime assertions of the Web services to be invoked. Unfortunately, this functionality is not provided yet by NetBeans and Glassfish ESB, respectively, so that this check has to be accomplished manually.

VI. CONCLUSIONS AND FUTURE WORK

In software engineering, there has already been much work on non-functional requirements. This is motivated by the fact that nearly all deployed application systems must not only fulfill the desired business logic, but should also guarantee aspects such as robustness, scalability, security, performance and reliability. Although NFRs should be especially considered when designing applications according to the SOA principle, there is currently only partial support – both from a conceptual as well as technical point of view.

In our work, we have presented a coherent concept for formalizing, applying, enforcing, and monitoring NFRs for business processes. A driving force of our solution is the commitment to well-known standards and widely used technologies such as BPMN, BPEL, WS-Policy, CEP, and others. As a consequence, the conceptual framework of our solution can be instantiated in several ways based on existing tools such as NetBeans and Glassfish ESB.

This demonstrates the high impact of our results on software engineering practice. Specifically, our approach is a further step towards improving the development of business application with well-defined NFRs. We support the well-known separation of concerns principle by flexibly attaching NFRs to business processes.

Our work can be extended in several ways. In order to leverage our solution, further NFRs should be formalized. This includes the definition of the required QoS artifacts for BPMN and their mapping to corresponding WS-Policy assertions. To disseminate our approach in software engineering practice, additional proof of concept implementations would be quite helpful; especially an instantiation with the Visual-Studio IDE and the .NET technology.

ACKNOWLEDGMENT

We would like to thank Markus Schalk for the extensive support during the elaboration of the architecture and the proof of concept. This work has been partly supported by the German Ministry of Education and Research (BMBF) under research contract 17N0709.

REFERENCES

- [1] OASIS, Web Services Quality Factors 1.0 (07/2010), Retrieved April 16, 2011, from <http://docs.oasis-open.org/ws-qm/ws-qf>
- [2] OASIS, WS-SecurityPolicy 1.3 (03/2009), Retrieved October 11, 2010, from <http://docs.oasis-open.org/ws-sx/ws-securitypolicy>
- [3] S. Adam and J. Doerr, "Towards Early Consideration of Non-Functional Requirements at the Business Process Level", Proceedings of International Conference on Information Resources Management, pp. 227-230, 2007.
- [4] OMG, Object Management Group / Business Process Management Initiative, Retrieved March 03, 2011, from <http://www.bpmn.org>
- [5] ISO/IEC, "Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE", ISO/IEC 25000, 2005.
- [6] ISO/IEC, "Software engineering – Product quality", ISO/IEC 9126-1 to 9126-4, 2001-2004.
- [7] L. O'Brien, P. Merson and L. Bass, "Quality Attributes for Service-Oriented Architectures", in Proceedings of International Workshop on Systems Development in SOA Environments (SDSOA '07), pp. 1-5, 2007.
- [8] W3C, Web Services Policy 1.5 (09/2007) – Framework, Retrieved April 03, 2011, from <http://www.w3.org/TR/ws-policy>
- [9] W3C, Web Services Policy 1.5 (09/2007) – Attachment, Retrieved April 03, 2011, from <http://www.w3.org/TR/ws-policy-attach>
- [10] C. Pavlovski and J. Zou, "Non-Functional Requirements in Business Process Modeling", in Proceedings of the Asia-Pacific Conference on Conceptual Modelling, pp. 103-112, 2008.
- [11] A. Rodriguez, E. Fernández-Medina and M. Piattini, "A BPMN Extension for the Modeling of Security Requirements in Business Processes, IECE Trans. Inf. & Syst, pp. 745-752, 2007.
- [12] S. Tai, R. Khalaf, and T. Mikalsen, "Composition of Coordinated Web Services", Middleware, pp. 294-310, 2004.
- [13] A. Charfi, R. Khalaf and N. Mukhi, "QoS-Aware Web Service Compositions Using Non-intrusive Policy Attachment to BPEL", in Proceedings of the 5th International Conference on Service-Oriented Computing, pp. 582-593, 2007.
- [14] E. Hewitt, Java SOA Cookbook: SOA Implementation Recipes, Tips, and Techniques, O'Reilly, 2009.
- [15] B. Hollunder, "Domain-Specific Processing of Policies or: WS-Policy Intersection Revisited," Proceedings of the 7th IEEE International Conference on Web Services (ICWS), pp. 246-253, 2009.
- [16] A. Wahl, A. Al-Moayed, and B. Hollunder, "An Architecture to Measure QoS Compliance in SOA Infrastructures", Proceedings of the Second International Conferences on Advanced Service Computing (Service Computation 2010), pp. 27-33, 2010.
- [17] O. Charles, M. Schalk, and B. Hollunder, "CEP meets SOA", OBJEKTSpektrum: Vol 5, pp. 28-32, 2010.
- [18] D. Luckham, "The Power of Events", Addison Wesley, 2007.

A Framework for Adapting Service-oriented Applications based on Functional/Extra-functional Requirements Tradeoffs

Raffaella Mirandola
Politecnico di Milano
 DEI, Milano, Italy
 mirandola@elet.polimi.it

Pasqualina Potena
Univ. degli Studi di Bergamo
 DIIMM, Dalmine (BG), Italy
 pasqualina.potena@unibg.it

Elvinia Riccobene
Univ. degli Studi di Milano
 DTI, Crema (CR), Italy
 elvinia.riccobene@unimi.it

Patrizia Scandurra
Univ. degli Studi di Bergamo
 DIIMM, Dalmine (BG), Italy
 patrizia.scandurra@unibg.it

Abstract—This paper introduces an adaptation framework for service-oriented applications based on trade-offs between functional and extra-functional (e.g., availability, performance, and adaptation cost) requirements. The framework relies on an optimization method for adaptation space exploration based on the combined use of meta-heuristic search techniques and of functional and extra-functional patterns (e.g., architectural design patterns and tactics). A formal service-oriented component model, called SCA-ASM, is also adopted for the specification and functional analysis of service-oriented applications. Through a sample application, we exemplify the methodology with emphasis on the use of extra-functional patterns.

Keywords—Service-oriented applications; software adaptation and evolution; extra-functional adaptation patterns.

I. INTRODUCTION

Service-oriented applications are playing an important role in several application domains (e.g., health care, defense and aerospace) since they offer advanced and flexible functionalities in widely distributed environments by composing, possibly on demand, different types of services. These applications may require dynamic adaptation to changing user needs, system intrusions or faults, changing operational environment and resources. Foundational theories and notations are required to support the engineering of such applications. Also required are techniques for monitoring and evaluating the behavior and performance of these applications, fully integrated in a software engineering process that reflects the *closed-loop paradigm* (e.g., the MAPE-K loop in the context of autonomic computing) [1] are required.

Extra-functional properties of services are often specified as quality of service (QoS) constraints and their values are dynamic [2]. For example, the system response time depends on environmental factors among which input data, server load, and network latency. The adaptation decisions for implementing the single changes should be triggered whenever unsatisfactory behaviors and values are reported by monitoring modules or required by the user (or “system designer” or “system maintainer”), and the right trade off among the functional requirements, software qualities and the adaptation cost should be considered. A decision, for example, taken for modifying the dynamic of a service may

be good for the satisfaction of the system reliability, but at the same time it may require a high adaptation cost for adapting the interfaces of services [3].

This paper presents an adaptation framework based on functional and extra-functional requirements trade offs. It is based on a formal *service-oriented component model*, named SCA-ASM [4], for the specification and analysis of service-oriented applications, and on a runtime *optimization method* for adaptation exploration that uses a mixed approach of metaheuristic search techniques [5], of functional and extra-functional adaptation patterns, such as architectural design patterns and tactics, or also software actions defined by the maintainer based on his/her experience. The adopted optimization approach enhances the one defined in [3] by taking into account also functional issues that allow, among other things, to relax the independence assumption between adaptation actions for different adaptation requirements.

According to the *design for adaptability* vision in [6], our framework supports both evolution (at re-design time) and self-adaptation (at run time). The second form of adaptation regards temporary modification (such as the re-execution of an unavailable service or a substitution of an unsuitable service) permitting to respond to changes in the requirements and/or in the application context. However, when changes regard critical aspects and should be applied permanently to the system, they should be considered as evolution steps, and therefore fast answers are not essential since adaptation strategies are evaluated and carried out at (re-)design time.

This paper is organized as follows. Section II reports related works. Section III provides background on SCA-ASM. Section IV describes our adaptation methodology. Section V presents the overall architecture of our framework. Section VI exemplifies our methodology through a sample application. Finally, Section VII sketches some future work.

II. RELATED WORK

A survey on adaptation approaches and frameworks can be found in [1]. Most of them typically adapt a system by adopting different service selection policies, varying system parametrization or exploiting the inherent redundancy of the Service-Oriented Architecture (SOA) environment.

Some frameworks exist for the dynamic generation of a service composition, but usually they adapt a system only in a reactive way, after the adaptation request triggered by a user. They support the service selection with respect to a service composition defined by the user (e.g., the VRESCO runtime environment [7]) or choose the service composition, which they have generated together with a finite set of other candidates, that better fulfill the required quality (e.g., [8]).

With respect to the state-of-art, our work is the first framework (to the best of our knowledge) that supports the adaptation of service-oriented applications (including both static and dynamic aspects) at runtime and at re-design time. It uses a mixed approach of metaheuristic search techniques, whose effectiveness and efficiency has been already demonstrated for supporting the service selection activity at runtime [2], and of functional and extra-functional adaptation patterns [9]. Existing approaches typically do not take into account functional aspects and assume that a component is functional equivalent to its alternatives [10]. Concerning design solutions, existing approaches (e.g., [9] and [11]) do not quantify or predict the impact of the adoption of one or more solutions on the system quality and functionality. As opposite, we address such a problem.

III. BACKGROUND ON SCA-ASM

SCA-ASM [4] [12] is a formal and executable modeling language based on: (i) the open standard *Service Component Architecture* (SCA) [13] for heterogeneous service assembly, and (ii) the *Abstract State Machines* (ASM) formal method [14], which is an extension of FSMs where *states* are arbitrary complex data (multi-sorted first-order structures) and the *transition relation* is specified by *rules* describing how functions change from one state to the next. The SCA-ASM formalism is able to model service interactions, orchestration, compensations, and services internal behavior.

An SCA assembly (or composition) of service-oriented components can be graphically produced using the Eclipse-based SCA Composite Designer (an inner module of the SCA tools), and also stored or exchanged in terms of an XML-based file that is then used by the SCA runtime to instantiate and execute the system. The ASM formalism complements the structural description of the SCA assembly with a formal and executable behavioral description of the assembled components. Figure 1 shows an example of an SCA assembly of a stock trading application (better described in Sect. VI), while Figure 2 shows an ASM fragment of the `OrderDeliveryComponent` component behavior.

IV. THE ADAPTATION METHODOLOGY

An SCA assembly can be adapted through the following *actions*: adding/removing components, component services, references, properties, reference-service wires and promotion wires (component interactions); changing a component implementation (but keeping its shape); changing component

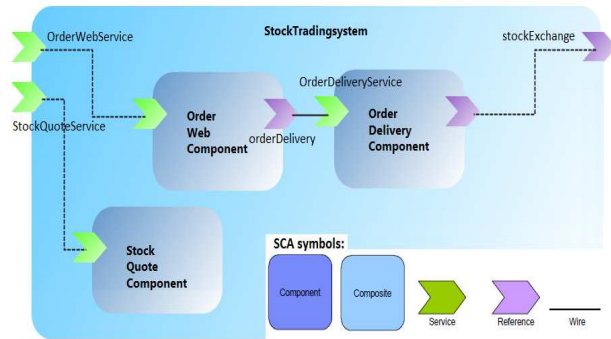


Figure 1. Stock Trading System

```

module OrderDeliveryComponent
  // @Provided service interface
  import OrderDeliveryService
  // @Required service interface
  import StockExchangeService
  ... // Other module imports
  signature: // ASM function declarations
  // @Reference to the external stock exchange system
  shared stockExchange: Agent -> StockExchange
  // @Backref back reference to the requester
  shared client: Agent -> Agent
  // Other function declarations for internal computation
  controlled order: Agent -> Order
  definitions:
  // ASM rules for the provided service operations
  @Service
  rule r_place($client in Agent, $o in Order) = ... // to place buy or sell orders
  ...
  // ASM rule for the component's agent behavior
  rule r_OrderDeliveryComponent =
  seq
    r_wreceive(client(self), "place", order(self))
    // direct service invocation
    r_place(client(self), order(self))
    r_wreply(client(self), "place", order(self))
  endseq
  // constructor rule
  rule r_init($agent in OrderDelivery) = $ ... // do initialization (if any)
  
```

Figure 2. ASM module of the `OrderDeliveryComponent`

properties values; changing SCA domains (components re-deployment). It is also possible to change the component interaction style in synchronous/asynchronous, stateful or not, unidirectional or bidirectional. See [13] for details.

Actions can be combined into an *adaptation plan*, which is a set of actions modifying the static and dynamic parts of a system architecture to address a certain requirement. Adaptation plans may differ for adaptation cost and/or for the system quality achieved after their application.

The proposed *adaptation process* starts from a set of initial SCA-ASM assemblies (initial *candidates* or *population*) fulfilling the existing/new functional requirements. It proceeds iteratively till stop criteria are satisfied. Currently, we use a predefined number of iterations to determine the end of the search. More sophisticated stop criteria could use convergence detection and stop when the global optimum is probably reached. At each iteration step, new candidates are generated from the initial population (whose size depends on

the specific search technique) by two subprocesses executed in parallel: (i) *metaheuristic search* by applying user adaptation plans, service selection and service re-deployment; (ii) *functional/extra-functional patterns application* by exploiting architectural design patterns and tactics. Then the functional and quality analyses of the resulting candidates are performed together with an assessment of the adaptation costs. In case of self-adaptation, SCA-ASM assemblies are automatically selected as solutions according to predefined selection criteria (e.g., cost minimization). In case of evolution, the solution can be more accurately selected also considering a possible feed-back from the user [6].

Metaheuristic search techniques. Several metaheuristics [5] with different characteristics could be adopted depending on the problem: for example, considering the system reliability, a possible heuristic is to regard as increasing the whole reliability of the system when the reliability of the most used components increases. As remarked in [10], there exist design options for which we have no prior knowledge on how they affect the extra-functional property of a particular system. To this extent, undirected operations could be performed (e.g., random choices or exhaustive evaluation of all neighboring candidates).

Architectural design patterns and tactics. Architectural patterns are templates for concrete software architectures. They are adopted to embody functional requirements and, in particular, to enable self-adaptability by introducing sensors/effectors components (e.g., Microkernel pattern, Reflection pattern, Interception pattern) [1]. To build new design solutions embodying extra-functional requirements, we adopt architectural tactics [9], which are reusable architectural building blocks that provide a generic solution to issues pertaining to quality attributes.

V. THE ADAPTATION FRAMEWORK

Figure 3 shows the main modules of the framework. The core of the framework is an optimization approach (implemented by the *Reasoner* module) that adapts (through the *Executor* module) an SCA-ASM assembly (developed by the *System Model Creator*) of a service-oriented application with respect to the functional requirements, system qualities, and adaptation costs. Adaptation actions can be triggered automatically (after receiving alerts from the *Monitor* module) and/or by the user (through the *User Requests Manager* module that also interacts with the *Monitor* module to figure out internal or context changes). An *Analysier* assists during the adaptation process for functional and quality analysis purposes. A description of each module follows.

System Model Creator and Executor. This module consists of two sub-modules (the *creator* and the *executor*) and relies on the integration of the SCA tools and runtime platforms (like Tuscany, FraSCAti, etc.) with the ASM toolset ASMETA [15], to graphically model, compose, analyze, deploy, execute, and introspect service-oriented applications.

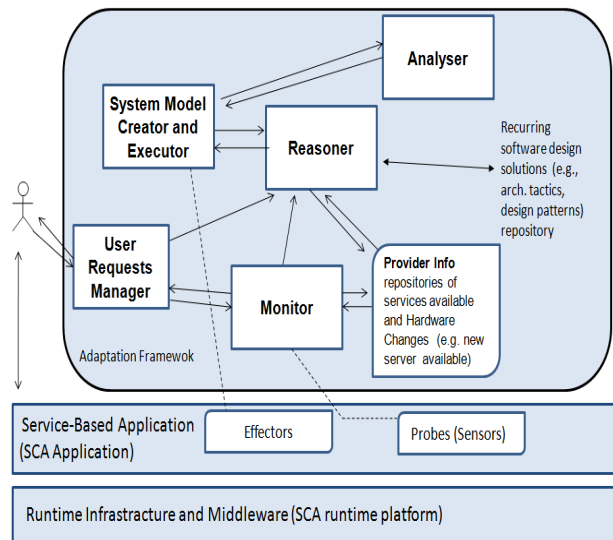


Figure 3. The Adaptation Framework

An SCA-ASM model (or assembly) of an application can be produced from scratch, or generated from an existing system implementation. Analysis techniques can be employed to assure consistency between the architecture and the implementation. Another feature of the *System Model Creator* is allowing, by exploiting the SCA Policy Framework [13], the designer to specify for components necessary metadata annotations. These are useful for providing metrics that can be extracted from the model for non-functional analysis purposes, and for representing policies that can be guarantee by the runtime platform. It also allows the application of design patterns and tactics to an SCA-ASM assembly, leading to a chain of adaptation actions. To guarantee the functional correctness of the resulting assembly and that changes claimed by the adaptation actions do not compromise the satisfaction of existing functional requirements, an interaction with the *Functional and Quality Analyser* is required. Different adaptation actions of the SCA assembly may be enacted manually (as suggested by the *User Request Manager*) or automatically (by the *Reasoner*).

The *System Model Executor* implements the adaptation actions suggested by the *Reasoner*. Through the use of effectors, changes applied at model level must be related to the underlying mechanisms and runtime infrastructure. To this extent, guidelines of existing approaches supporting dynamic service invocation and of the ones for dynamically adapting the system behavior could be exploited. In the case of SCA, mechanisms like introspection and reconfiguration, for managing and enacting self-adaptation [16] are applied.

User Requests Manager. It allows users to make adaptation requests by providing appropriate adaptation plans. It assures that plans of different adaptation requirements are independent between each other, i.e., changes claimed by a plan do not compromise the application of other plans.

Reasoner. It is activated after receiving either adaptation requests from the user or alerts from the *Monitor*. By using an optimization approach, it produces a set of software adaptation actions. Through the help of the *System Model Creator*, it generates the new system architecture model, i.e., the new SCA-ASM assembly model including both structural and behavioral aspects. The adaptation space exploration process implemented by the reasoner is iterative and is based on the combined use of meta-heuristic search techniques and of functional and extra-functional adaptation patterns (i.e. architectural design patterns and tactics). A detailed description of the optimization method and related techniques are out of the scope of this paper.

Monitor. It controls the system at runtime through the use of probes (sensors). It may trigger self-adaptation when detecting relevant context and internal changes or an evolution cycle of the system for introducing important and permanent changes [6]. For implementing such a module, several monitoring approaches exist in the literature (see, e.g., [1]). The monitor can also continually measure the services' QoS attributes. The providers can improve the estimate of the services' non-functional properties by monitoring them.

Functional and Quality Analyzer. It consists into a set of external tools that can be invoked for different analysis purposes. Essentially two sub-modules can be identified: one for the *functional*, the other one for the *non-functional analysis*. The *functional analyzer* is linked with ASMETA [15], a set of tool for the ASMs. It is invoked when a preliminary analyses of the functional requirements satisfiability of the SCA-ASM assembly would be performed by easier techniques as simulation or scenario-based validation. Later, heavier formal verification techniques (as model checking) can be exploited when more complex functional properties [17] must be proved to guarantee behavioral system correctness. The functional analyzer is also invoked when correctness must be proved upon a refinement step of the SCA-ASM assembly due to adaptation actions. Techniques for checking correctness of model refinement as supported by the ASMETA tool-set. The *non-functional analyzer* exploits external tools for performance and reliability analysis like *qnetworks* [18] and *LQNSolver* [19]. The system qualities (e.g., performance and reliability) and the adaptation costs are predicted exploiting the SCA-ASM model of the system. Examples of adaptation costs can be found in [3]. Considering quality analysis, different approaches/strategies can be used depending on several factors due mainly to the use of our framework for evolution (at re-design time) or self-adaptation (at run time). If permanent changes, for example, are requested or a safe-critical service has to be adapted, precise (often expensive) analysis must be performed (e.g., see [20] for performance analysis). As opposite, if runtime changes are claimed and these require, for example, only the adaptation of parameters without using more sophisticated analysis, faster approaches must

be adopted allowing a prompt run-time adaptation (see, e.g., techniques for estimation of quality at runtime, such as [21]).

VI. THE STS CASE STUDY

We describes the adaptation methodology by a sample application from the Stock Trading System (STS) in [9]. Figure 1 shows the SCA assembly of the STS. Briefly, an STS user, through the *OrderWebComponent* interacting with the *OrderDeliveryComponent*, can check the current price of stocks, placing buy or sell orders and reviewing traded stock volume. Moreover, he/she can know stock quote information through the *StockQuoteComponent*. STS interacts also with the external Stock Exchange system, which we do not model.

Figure 2 shows a fragment of the ASM (abstract) model for the *OrderDeliveryComponent* behavior. The main service of this component (the rule *r_place* annotated with *@service*) is to place buy or sell orders when requested (see the blocking *receive* action and the *reply* action preceding and following, respectively, the service invocation within the component's main rule *r_OrderDeliveryComponent*). The ASM definition for the provided and required interfaces of the *OrderDeliveryComponent* are reported in Figure 4. They are ASM modules containing only declarations of business agent types (the subdomains *OrderDelivery* and *StockExchange* of the predefined ASM Agent domain) and of business functions (parameterized ASM out functions) used as temporary locations to store service computation results.

```

module OrderDeliveryService
import ... //Other module imports
signature:
// the domain defines the type of the provider component's agent
domain OrderDelivery subsetof Agent
// business function value
out place: Prod(Agent,Order) -> Order

//@Remotable
module StockExchangeService
import ... //Other module imports
signature:
domain StockExchange subsetof Agent
out sendOrder: Prod(Agent,Order) -> Rule

```

Figure 4. ASM modules of the *OrderDeliveryComponent* interfaces

Below, we apply to the STS case study some adaptation strategies adopted by our methodology. Specifically, first we describe the application of a simple metaheuristic technique, and then we show the use of some tactics as examples of extra-functional adaptation patterns. Details on the experimental data set used in this case study can be found in [23]. *Metaheuristic search:* Figure 5 shows an example of instantiation of our optimization process by considering, on the STS example, the *steepest-ascent hill-climbing* metaheuristic [5] that tries to adapt the system minimizing

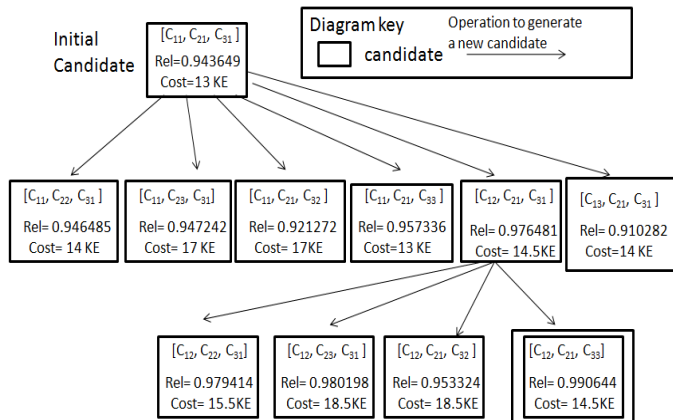


Figure 5. Example of steepest-ascent hill-climbing application

the adaptation costs and assuring a level of system reliability greater than 0.98. The initial candidate is the vector $[C_{11}, C_{21}, C_{31}]$ (see Figure 5), where C_{ij} denotes the j th instance available on the market for the component C_i with C_1 indicating the OrderWebComponent, C_2 the StockQuoteComponent and C_3 the OrderDeliveryComponent. Each vector comes with two parameters: the resulting system reliability and the cost of the solution (predicted using the reliability and cost model used in [22] and reported in Table 1 in [23]). At each iteration step, a set of new candidates is generated by replacing, one at a time, an existing component with one available on the market. The best candidate is then selected as the one improving the system reliability and minimizing the adaptation costs. It becomes the basis for next candidates generation. The process terminates either if no better candidates can be found or the reliability threshold is reached. In our case, the optimization process returns the solution $[C_{12}, C_{21}, C_{33}]$ with reliability equals to 0.990644 and cost equals to 14.5 KE.

Application of extra-functional adaptation patterns: We here show how availability and performance tactics can be used to embody extra-functional requirements of the STS example into its architecture. Let us assume the following extra-functional requirements (taken from [9]):

NFR1. *The STS should be available during the trading time (7:30 AM6:00 PM) from Monday through Friday. If there is no response from the system for 30 s, the STS should notify the administrator.*

NFR2. *The system should be able to process 300 transactions per second, 400,000 transactions per day. A client may place multiple orders of different kinds (e.g., stocks, options, futures), and the orders should be sent to the system within 1 s in the order they were placed.*

To address NFR1 the *Fault Detection Tactic* for the detection and notification of a fault to a monitoring component or to the system administrator can be adopted. Such kind of tactic can be refined into other ones (e.g., *Ping/Echo*,

Heartbeat and *Exception* tactics [9]). As done in [9], we support NFR1 combining *Ping/Echo* and *Heartbeat*.

As in [9], NFR2 is supported combining the *FIFO* (for the Resource Arbitration) and *Introduce Concurrency* (for the Resource Management) tactics. The *FIFO* tactic allows clients to place each type of orders (e.g., stocks, options, futures) to a dedicated queue for immediate processing. Finally, to handle considerable amount of transactions by their kinds within a very short time, as suggested in [9], NFR2 can be also supported by reducing the blocking time of transactions on I/O, which can be realized by the combined use of the *FIFO* and *Introduce Concurrency* tactics (i.e., by concurrent dispatching of the same kind of orders).

Figure 6 shows how it would change the SCA assembly by composing these tactics: the assembly is extended to add the new Queue component (for the *FIFO* tactic) and the Monitor component (for the *Fault Detection Tactic*). The OrderWebComponent is refined for concurrently producing orders to place into the Queue. Similarly, the OrderDeliveryComponent is refined for adding the monitoring functionality and for the concurrent consuming of different kinds of orders placed into the Queue component. Of course, this implies a change of the components shape (i.e. in the required/provided interfaces) and of their behavior. The behavior, for example, of the OrderDeliveryComponent is refined in ASM as shown in the fragment reported in Figure 7: the consuming and sending of different kind of orders (stock, option, or future) are executed in parallel (i.e. concurrently) by the *par* rule.

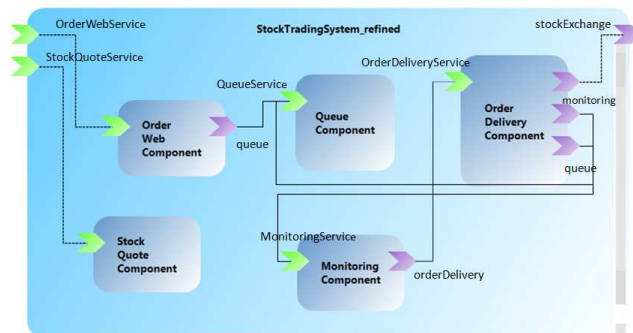


Figure 6. Adapting the STS by applying tactics for NFR1 and NFR2

It is possible to prove that the behavior of the OrderDeliveryComponent in Figure 7 is a correct refinement [14] of that in Figure 2, and, therefore, all initial functional requirements are still guaranteed. Moreover, the impact of the adoption of the tactics should be quantified with respect to the existing system quality. For example, the introduction of new components could decrease the maximum level of reliability. In the STS example, after the embedding of new components into the OrderDeliveryComponent for NFR1, if the probability of failure of the instance available for this component

```

module OrderDeliveryComponent
...
rule r_OrderDeliveryComponent=
... seq
  par //Queue consuming
    r_wsndreceive[client(self),"dispatch","Stock",stockorder(self)]
    r_wsndreceive[client(self),"dispatch","Option",optionorder(self)]
    r_wsndreceive[client(self),"dispatch","Future",futureorder(self)]
  endpar
  par //Order sending to the Stock Exchange system
    r_wsnd(stockExchange(self),"sendOrder",(self,stockorder(self)))
    r_wsnd(stockExchange(self),"sendOrder",(self,optionorder(self)))
    r_wsnd(stockExchange(self),"sendOrder",(self,futureorder(self)))
  endpar
endseq ...

```

Figure 7. The refined behavior of the OrderDeliveryComponent

increases (for example, from 0.00006 to 0.0002 [23]), then the reliability of the overall solution will decrease (from 0.990644 to 0.970639 [23]). Therefore, it could happen that the reliability constraint is not satisfied any more (in the example indeed, the system reliability is not greater than 0.98). Note that, also the reliability of the new *Queue* component may contribute to decrease the system reliability.

VII. CONCLUSION AND FUTURE DIRECTIONS

This paper presented an adaptation framework for service-oriented applications that relies on design-for-adaptability principles while supports the closed-loop paradigm. With such a kind of support, a system is able to monitor itself and its context to detect significant changes, decide how to react on the base of functional/non-functional trade offs, and execute such decisions at runtime or at re-design time.

We intend to enhance our framework towards several directions. Currently, we are implementing a prototype to compare different implementations of our optimization process (e.g., with heuristics depending on application domain or quality attributes) on realistic examples. We intend to support the right trade-off between the adaptation overhead (due, e.g., to the frequent execution of the reasoning algorithms) and the accrued benefits of changing the system.

REFERENCES

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 14, pp. 14:1–14:42, 2009.

[2] F. Rosenberg, M.B. Müller, P. Leitner, A. Michlmayr, A. Bouguettaya, and S. Dustdar, "Metaheuristic optimization of large-scale qos-aware service compositions," in *Proc. of the IEEE Int. Conf. on Services Computing*, 2010, pp. 97–104.

[3] R. Mirandola and P. Potena, "Self-adaptation of service based systems based on cost/quality attributes tradeoffs," in *Proc. of WoSS at SYNACS 2010*, pp. 493–501.

[4] E. Riccobene, P. Scandurra, and F. Albani, "A modeling and executable language for designing and prototyping service-oriented applications," to appear in *Proc. of EUROMICRO SEAA 2011*.

[5] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, 2003.

[6] A. Bucchiarone, C. Cappiello, E. Di Nitto, R. Kazhamiakin, V. Mazza, and M. Pistore, "Design for adaptation of service-based applications: Main issues and requirements," in *ICSO/ServiceWave 2009 Workshops*, ser. LNCS, 2010, pp. 467–476.

[7] F. Rosenberg, P. Celikovic, A. Michlmayr, P. Leitner, and S. Dustdar, "An End-to-End Approach for QoS-Aware Service Composition," in *EDOC*, 2009, pp. 151–160.

[8] D. Chiu, S. Deshpande, G. Agrawal, and R. Li, "A Dynamic Approach toward QoS-Aware Service Workflow Composition," in *ICWS*, 2009, pp. 655–662.

[9] S. Kim, D. Kim, L. Lu, and S. Park, "Quality-driven architecture development using architectural tactics," *Journal of Systems and Software*, no. 8, pp. 1211–1231, 2009.

[10] H. K. A. Martens, "Automatic, model-based software performance improvement for component-based software designs," in *Proc. of FESCA 2009*, vol. 253, no. 1, 2009, pp. 77 – 93.

[11] K. Vallidevi and B. Chitra, "Effective self adaptation by integrating adaptive framework with architectural patterns," in *Proc. of A2CWIC 2010*, ACM, pp. 67:1–67:4.

[12] E. Riccobene and P. Scandurra, "Specifying formal executable behavioral models for structural models of service-oriented components," in *Proc. ACT4SOC 2010*, pp. 29–41.

[13] "Service Component Architecture (SCA)" www.osoa.org, 2007. [accessed: May 18, 2010]

[14] E. Börger and R. Stärk, *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 2003.

[15] "The ASMETA tooset," <http://asmeta.sf.net/>, 2006. [accessed: April 26, 2011]

[16] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J. Stefani, "Reconfigurable sca applications with the frascati platform," in *Proc. of Int. Conf. on Services Computing*, IEEE, 2009, pp. 268–275.

[17] C. Attiogbé, P. André, and G. Ardourel, "Checking component composability," in *Software Composition*, ser. LNCS, W. Löwe and M. Südholt, Eds., 2006, pp. 18–33.

[18] M. Marzolla, "The qnetworks toolbox: A software package for queueing networks analysis," in *Proc. ASMTA 2010*, Springer.

[19] G. Franks, P. Maly, M. Woodside, D.C. Petriu, and A. Hubbard, "Layered Queueing Network Solver and Simulator User Manual, LQN software documentation," 2006.

[20] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Trans. Software Eng.*, no. 5, pp. 295–310, 2004.

[21] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *Proc. of ICSE'09*, pp. 111–121.

[22] V. Cortellessa, F. Marinelli, and P. Potena, "An optimization framework for "build-or-buy" decisions in software architecture," *Computers & OR*, vol. 35, no. 10, pp. 3090–3106, 2008.

[23] R. Mirandola, P. Potena, E. Riccobene, and P. Scandurra, "A framework for adapting service-oriented applications based on functional/extra-functional requirements tradeoffs: the Stock Trading System case study," TR Univ. of Bergamo (Italy), <http://cs.unibg.it/potena/AdaptationFramework/TRExpResults.pdf>, 2011. [accessed: July 21, 2011]

PSW: A Framework-based Tool Integration Solution for Global Collaborative Software Development

Juho Eskeli
VTT Technical Research Center of Finland
Oulu, Finland
Juho.eskeli@vtt.fi

Carmen Polcaro
Innovalia Association
Bilbao, Spain
cpolcaro@innovalia.org

Jon Maurologoitia
CBT Communication Engineering
Getxo, Spain
jmaurologoitia@cbt.es

Abstract—The market of solutions for collaborative and distributed software development offers currently a wide range of tools that support specific tasks involved in these kind of projects. Several solutions aim to support the whole development process in a single tool or via groups of tools by providing distributed teams the possibility to share and connect information and to use common interfaces. Nonetheless, every one of them includes some disadvantages that lessen their value for companies that use them across their distributed development projects. In this paper the authors will highlight relevant issues associated with collaborative and distributed software development projects. Prisma Workbench will be presented as the framework to overcome many of these issues and to provide a compelling option for teams to integrate their existing tools into a complete collaborative solution. Currently Prisma Workbench is being tested by the partners involved in the ITEA2 PRISMA Project and some of the first feedback will be presented as well.

Keywords—collaborative software development; global software development; collaboration; tools; tool integration

I. INTRODUCTION

Collaborative and distributed software development is currently one of the most common ways of facing the development for many applications that due to its complexity or size require a large team working together [1]. The level of distribution for each group of the team can vary from different departments of the same company located in the same building to the case that several companies' located in completely distant regions of the world participate in a common development. The motivation to adopt this organizational paradigm can vary from case to case: cost reduction, collaboration between reference centres or using this as a way to increase the innovation inside the company [2]. The number of cases that can be found in the industry is enormous [3][4].

A distributed software organization model brings problems to the development process that have to be addressed with specific methodologies or tools. The most

relevant that could be identified as part of the PRISMA Project[5], previously to the development of Prisma Workbench (PSW) [6], are highlighted here:

- **Communication Breakdown:** the barrier of not being able to discuss issues and agree on specific topics face to face leads to delays in the development process.
- **Coordination Breakdown:** can happen in a project where people don't know each other or don't have the possibility to interact continuously to adapt project planning. The chances of the project to go on wrong track are higher and following of planning is difficult.
- **Control Breakdown:** For project managers, having a clear view of the status of a project when the team is distributed in different locations and work in different time zones can be a really challenging task. The level of control that the project manager will have is not as deep as in a non-distributed scenario.
- **Cost of currently available tools:** currently a number of providers offer their commercial solution for collaborative development. The price of implementing these solutions in companies is sometimes an obstacle.
- **Poor interoperability between tools:** in a case where each team is using their own tools, integration between the tools is difficult and most of the times impossible. For this reason manual copying or exporting of data from one tool to another is often needed.
- **Lack of traceability:** during the development project information elements are created which traceability should be maintained throughout the whole process. These elements include e.g., client requests, system requirements, test information, bug reports, and so on. Having no connection between the tools that manage each of these elements makes the traceability maintenance an effort consuming task.

Nowadays, the market of tools that support specific tasks of the development process is very large. In most cases their learning curve is high. Therefore, teams feel reluctant to include a new tool or change the tools that they are currently using as part of their development process although this could sometimes lead to a better integration with the rest of a distributed team.

Another type of tools, which will be discussed in chapter V of this paper, presents a global solution that supports the whole development process. As mentioned before, these solutions include sometimes a price tag that not every company is able to pay, especially in those cases where SME's are involved.

Prisma Workbench, the solution proposed in this paper is a tool integration framework designed for collaborative distributed software development. This framework allows connecting of software development tools to create company specific software development environment instances. In this paper the solution is presented from instance point of view; how it can be used with a particular set of tools. The tool set mentioned consists of tools proposed by the PRISMA project partners.

PSW fills the gaps that exist in the current collaborative software development environments. It allows distributed teams to integrate their own existing tools and link data among them. PSW provides the visibility of how the project is running and what every group is doing to the whole development team as if everybody would be working in the same room.

II. RELATED WORK

Wasserman [7] defines tool integration as follows: 'tool integration is intended to produce complete environments that support the entire software development lifecycle.' In our vision tool integration can be used to provide a consistent software development environment using tools that were not planned to be used together initially. Furthermore, with the help of suitable tool set a notable part of software development lifecycle can be supported. Thus, the vision is not entirely separate of what Application Lifecycle Management (ALM) tools attempt to provide. According to Kääriäinen [8] ALM can be understood as coordination of activities and the management of artefacts such as requirements, test cases, etc. during the lifecycle of a software development project.

Schwaber [9] and Shaw [10] mention that the type of ALM solutions at that time could be divided into single vendor (e.g., IBM Jazz), multi-vendor (e.g., Eclipse, ALF), and single repository approaches. In single vendor approach a vendor has built a framework where other vendors can build integrations. In multi-vendor approach development and direction is driven by open source community (e.g., Eclipse, ALF). In single repository approach all the software lifecycle artefacts are managed in a single place.

According to the previous classification PSW is a multi-vendor platform. Furthermore, it is a framework integration based on tools' own repositories. As described by [11] framework-based integrations attempt to classify tools and

provide integration between tool classes based on vendor-neutral interfaces and mechanisms. Furthermore, the framework-based approach aims to provide an integration environment and common look and feel without limiting the choice of tools [11].

As far as we know our solution is unique because it does not rely on any specific software development tool. Also, in theory the tool set could be extended to support notable parts of software development lifecycle using a suitable tool set. Modelbus [12] is a project of tool integration, but to our understanding the focus is mainly integration of modelling tools and study of model transformations. Also Eclipse Mylyn is advertised as ALM framework [13], but as far as we can tell it seems to focus largely on task management and integration of task / defect management tools.

III. FEATURES

PSW has been developed from ground up based on the experiences achieved from ITEA Merlin[24] and ITEA2 TWINS[25] projects. The previous Eclipse based tool integration has been described in detail in [14]. In case of PSW the main interface is via a web browser. This approach was chosen to decrease dependency on a particular technology/platform (Eclipse) and making it easier to use PSW in day-to-day operations (i.e. lower the barrier of deployment).

The solution proposed is a tool integration framework designed for collaborative distributed software development. In its current form it has been previously presented in [5]. PSW allows connecting of software development tools to create company specific software development environment instances. In this paper the solution is presented from instance point of view; how it can be used with a particular set of tools. The tool set mentioned consists of tools proposed by the PRISMA project partners.

PSW implements a repository neutral integration of tools. This means that the lifecycle data produced during software development process is maintained in separate tools. The benefit of this type of approach is that it has minimum impact on the company's current tool set. The caveat is that integrations to the tools have to be constructed on a per tool basis. However, there is no need to create point-to-point integrations between each of the tools because PSW acts as a hub where tools are connected via its integration interface.

For PSW one of the primary goals has been to make the integration of new tools as easy as possible. To get to this goal the following steps have been taken: designed integration mechanism for simple integration, provide example integrations, and created integration instructions. The integration mechanism has been described in [6]. The example integrations will be described later in this section.

The solution provides visibility of tools data in easy to understand dashboards (see Figure 1 and Figure 2) that can be customized based on the user's preferences. Furthermore, the framework handles user sign-in into the separate tools transparently. The solution also provides the means for the user to create links between different lifecycle items. These links can then be exploited in the reporting to e.g., demonstrate amount of defects in a build. The reporting

solution built into PSW allows users to customize their own reports.

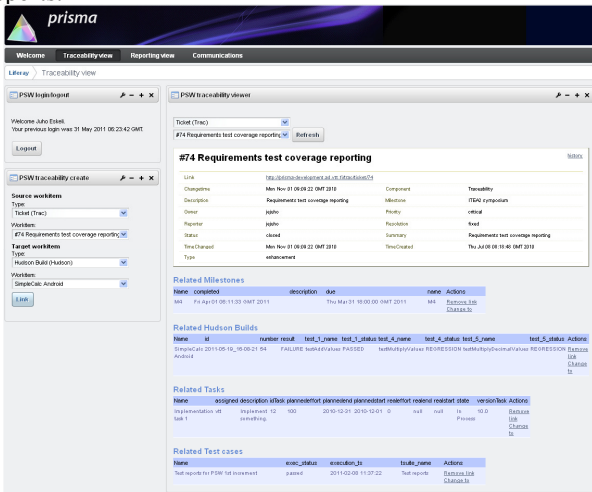


Figure 1. The traceability view showing a requirement and related work products



Figure 2. The reporting view showing a generated graph based on data retrieved from the integrated tools

To support collaborative, distributed development PSW provides means for asynchronous (chat) and synchronous (voice & video) communication with the help of a tool (OpenMeetings). The notifications system provides users up-

to-date information about any important events (e.g., build status) in the project.

Although PSW can be connected to several other commercial tools or custom developed ones, the project team has made a selection of open source solutions that cover the complete development process. By using these solutions, companies will be able to start working together also if currently no tool is used for any of the requirements specification, development or testing tasks. The solutions that have been selected are the following:

- Edgewall Trac[15]: this tool originally developed for bug tracking has also been used a simple requirement management tool. As part of PSW it should be used for requirement management and bug tracking.
- Subversion: this versioning system is one of most popular in the open source community.
- Testlink[16]: This web based test management tool will support your test case and test data management.
- Openmeetings: with Openmeeting companies will be able to host their own audio and video conferencing solution.

IV. BENEFITS

PSW addresses many relevant issues related to collaborative and global software development. Some of these issues were extracted during the research done by the PRISMA Project and have been highlighted in the introduction of this paper. After taking into account the features available in PSW we propose how distributed development process could be dramatically enhanced using PSW:

- Communication enabler: the possibility to organize virtual meetings and link those to other information items such as requirements, test, etc. enables a centralized solution where every group of the team can refer to decisions made any time during the development process.
- Improved team coordination: by sharing the status of key information such as requirements or tests and providing an event log, every member of the team will be informed of what others are doing. This will help them to coordinate their own work according to the planning. In a scenario where groups work in different time zones this log will be sometimes the only reference to achieve this kind of coordination.
- Centralized project management: the dashboards provide information a project manager needs to have for a quick image of how the project is running comparing to the plan. It will also give access to more detailed view of specific tasks. Using only one tool (PSW) for overview will facilitate the continuous control of projects. The virtual meeting functionality will be a key tool for the interaction between project managers, group managers, developers and testers.

- Seamless integration between tools: PSW will enable tools from different vendors, located in distant location to integrate while maintaining their independence. As described in chapter VI, this integration can be done easily through standard REST or WS communication interfaces. The number of manual copying processes between tools to maintain the traceability throughout projects will be reduced and in most cases eliminated
- High level of traceability: One of the main benefits of PSW is the possibility to trace information from different tools as if all of it would be in one tool.
- Low cost of investment: By including PSW in your organisation, every group will still be able to use the same tools as they had done before since they will be integrated instead of being replaced. The investment needed is therefore much lower than in other cases where only tools from the same vendor can be linked.

The research performed as part of the PRISMA Project has included the analysis by the partners of the improvements achieved by using PSW in tasks that were supported before by independent tools or by no tools at all. Since the PRISMA project is still ongoing and will be finished by the end of 2011, only the preliminary results of this analysis can be presented here. Currently PSW is being tested in real distributed software development projects in order to extract the most valuable results. This analysis is being performed using the tools provided by default with PSW and described in chapter III. Some of these tools had already been in use for some time by the partners involved in the project.

The first comment that has been shared after starting this testing phase is that, although using the same tools as before, the information supported by those is not isolated anymore. The tool supported traceability helps every member of the team to have a clear view about how every information artefact is related with the rest.

The centralized reporting tool has been identified by project, development and test managers as one of the best features in order to review the status of the overall project. It is one of the main functionalities where PSW combines data coming from several tools and provides a higher level of information.

Future publications will detail the complete results from this analysis.

V. EXISTING SOLUTIONS AND APPROACHES

As mentioned before, the market offers currently a number of solutions focused on distributed and collaborative environment. As described below, most of them include any restriction due to being closely related with one development technology, provider or business model.

- Jazz: This solution from IBM is targeted to integrate the Rational line of tools which support several phases of the development process. These tools include Rational Requirements Composer, Build

Forge and Quality Manager. Jazz also offers the Open Services for Lifecycle Collaboration (OSLC), an industry initiative to enable interoperability of tools developed by different vendors. Though promising, during the research performed in the PRISMA project, this interoperability was not achieved. Jazz is free to download from its site but currently it would be only useful for distributed teams that use Rational solutions.

- Teamforge: This webportal provided by Collabnet allows the collaboration of developers and IT project managers by providing the tools to plan and coordinate projects following agile methodologies. Collabnet features the management for user stories, source code integration, discussion forums, bug tracking and file and document sharing. Teamforge is licensed as a subscription based service. Although powerful, this solution forces every group of the team to use new tools and follow agile development methodologies which is not always the case in some companies.
- Application Lifecycle Framework (ALF): This Eclipse project proposal has been archived but its goal aimed to provide a logical definition of the overall interoperability business process. This technology handles the exchange of information, the business logic governing the sequencing of tools in support of the application lifecycle.
- Team Foundation Server (TFS): Microsoft offers this collaborative back end solution that can be connected with other Microsoft tools in order to exchange data among them. TFS does not have any user interface, rather it exposes web services which are the connection point between the tools. These include all the Visual Studio solutions but also Microsoft Project, Office or Sharepoint and cover almost the complete development lifecycle. As a disadvantage, teams where no Microsoft development tools are used will not be able to benefit from the TFS integration features.
- SourceForge.net: It claims to be the world's largest open source software development web site. They say that as of February, 2009, more than 230,000 software projects have been registered to use their services by more than 2 million registered users. SourceForge provides the following features for projects: discussion forums, wiki, version control system, file management and other tools more suited to open source projects.

VI. TECHNOLOGY BEHIND PSW

PSW consists of two main components: a server and collection of JSR 286 portlets. The server component integrates tools, implements some basic functions needed by tool integration such as user management, and provides its services to the portlets (or other possible clients). The portlets act as the user interface.

The server is built on top of Apache Tuscany[17], which is a framework for building Service Oriented Architecture (SOA) solutions. The framework takes care of runtime handling (initialization, termination, etc) of services. SOA was selected because it promotes loose-coupling between software components. Loose-coupling is useful because it provides us the freedom to add / remove / change the tools as needed. Yet another reason was because the SOA based approach provides us easy access to the distributed tools.

The integration mechanism of PSW has been described in [6]. A new tool can be integrated by creating a Java class that implements a Java interface definition provided by us. In the interface definition there are specific functions that need to be filled in; i.e. to get all work products (e.g., requirements) from the tool. What happens here is that the integrator creates a glue code that connects the data from the tool to PSW. The actual data from the tools can be fetched via any means supported by the tool, e.g., using REST or WS. Example integrations and guidance are provided to make the integration as easy as possible.

The server also takes care of authenticating the users to the tools. In essence a user's account for the tools is tied to the user's PSW account. Furthermore, it implements a traceability service which can be queried for work product relations and for creating new ones. The traceability mechanism is implemented so that no data is replicated. Instead unique identifiers are used to identify the work products in the tools, and the relations are stored in a relational database, MySQL[18]. The information artefacts are maintained in the original tool repositories.

For improved performance the data from tools has to be temporarily cached, for which Memcached[18] is used. Caching is needed because some of the tool specific queries can take a long time to complete (e.g., due to amount of data, tool location). The cache is updated at definable intervals. During an update the changes in the work products are detected and stored. The changes can then be queried using the notification service and shown in the user interface (i.e. portlets).

The user interface consists of several portlets implemented following the JSR 286^{Error! No se encuentra el origen de la referencia.} standard. The views (e.g., traceability, reporting) are implemented via one or many portlets and use the services provided by the server to produce their output. The portlets have been designed so that minimal or no changes need to be done if the set of tools is changed. The technologies used are Java, JavaServerPages[20] (JSP), and Javascript (jQuery etc.). For current implementation Liferay[21] portal has been chosen to run the portlets since it supports the JSR 286[22] standard. Nonetheless any other platform which support this standard could be used.

The reporting feature is the most recent addition into PSW. It enables users to build their own customized reports. An existing implementation (BIRT[23]) was studied and found promising; however the effort needed to implement custom reports with it in portlets was considered to be too much compared with the result. The reporting feature enables users to filter the data (e.g., from which tools, what type of work products) they use for the reports. Some

rudimentary manipulation of the data can also be performed e.g., addition or grouping of values. Existing traceability information can also be used to create e.g., requirements test coverage report. The plot types supported are currently bar, line, and pie chart. New types can be easily implemented with the library that is responsible for generating the charts. Additionally, the parameters used for creating the report can be stored for further usage, e.g., recurring reports. Reports with data can also be stored, named, and dated for reference. Finally, the reports can be exported in CSV and PDF formats.

VII. CONCLUSION

In this paper the authors have presented relevant issues that development teams face when a distributed organization model is adopted. These issues, which were identified as part of the research of the PRISMA Project, have been the motivation to develop PSW, a solution that allows the integration of a heterogeneous number of tools in order to collaborate and exchange data while maintaining their independence.

Solutions for collaborative software development that are currently available have been described, highlighting the advantages of PSW among them.

PSW features, technology background and benefits have been also thoroughly explained in order to make clear how using this solution in a distributed and collaborative environment could dramatically reduce the impact of this organization model in software development projects.

ACKNOWLEDGMENT

The authors would like to thank the partners involved in the ITEA2 PRISMA Project for their contribution and inspiration.

REFERENCES

- [1] P. Parviainen, J. Eskeli, T. Kynkäänniemi, M. Tihinen, 2008. Merlin Collaboration Handbook - Challenges and Solutions in Global Collaborative Product Development. In Proceedings of ICISOFT (SE/MUSE/GSDCA)2008. pp.339-346
- [2] T. Forbath, P. Brooks A. Dass, A , "Beyond Cost Reduction: Using Collaboration to Increase Innovation in Global Software Development Projects.", 2008. IEEE International Conference on Global Software Engineering.
- [3] M. Bass, J.D. Herbsleb, C. Lescher, "Collaboration in Global Software Development Projects at Siemens: An Experience Report", 2007 , IEEE, International Conference on Global Software Engineering.
- [4] Booz Allen Hamilton, "Globalization of Engineering Services", August 2006 , NASSCOM
- [5] Prisma Project website <http://www.prisma-itea.org/>
- [6] J. Eskeli, J. Maurologoitia, "Global Software Development: Current Challenges And Solutions.", 2011. ICISOFT
- [7] A. Wasserman, "Tool Integration in Software Engineering Environments", Springer-Verlag, Berlin, International Workshop on Environments, pp. 137-149, 1990.
- [8] J. Kääriäinen, "Towards an Application Lifecycle Management Framework", VTT Publications, Dissertation, 103p., 2011.
- [9] C. Schwaber, "The Changing Face of Application Life-Cycle Management", Forrester Research Inc., August 2006.

- [10] K. Shaw, "Application Lifecycle Management for the Enterprise", Serena Software, White Paper, http://www.serena.com/docs/repository/company/serena_alm_2.0_for_t.pdf, April 2007. (available 24.5.2011)
- [11] J. Pederson, "Creating a tool independent system engineering environment", In: IEEE Aerospace Conference, 8 pp., March 2006.
- [12] C. Hein, T. Ritter, and M. Wagner, "Model-driven tool integration with modelbus", In Workshop Future Trends of Model-Driven Development, 2009.
- [13] <http://www.eclipse.org/mylyn/> (read 27.05.2011)
- [14] J. Eskeli & P. Parviainen, "Supporting hardware-related software development with integration of development tools", Proceedings - 5th International Conference on Software Engineering Advances, ICSEA 2010, IEEE Computer Society, pp. 353 – 358, 2010.
- [15] Edgewall <http://trac.edgewall.org/>
- [16] Teamst <http://www.teamst.org/>
- [17] Apache Tuscany <http://tuscany.apache.org/>
- [18] Mysql <http://www.mysql.com/>
- [19] Memcached <http://memcached.org/>
- [20] JSP <http://java.sun.com/products/jsp/>
- [21] Liferay <http://www.liferay.com/>
- [22] JSR286 <http://www.jcp.org/en/jsr/detail?id=286>
- [23] BIRT <http://www.eclipse.org/birt/phoenix/>
- [24] Merlin-project <http://virtual.vtt.fi/virtual/proj1/projects/merlin/icgse.html>
- [25] TWINS-Project <http://www.twins-itea.org/>

Feature-Oriented Programming and Context-Oriented Programming: Comparing Paradigm Characteristics by Example Implementations

Nicolás Cardozo^{*†}, Sebastian Günther[†], Theo D'Hondt[†] and Kim Mens^{*}

^{*}ICTEAM Institute

Université Catholique de Louvain

Louvain-la-Neuve, Belgium

Email: {nicolas.cardozo,kim.mens}@uclouvain.be

[†]Software Languages Lab

Vrije Universiteit Brussel

Brussels, Belgium

Email: {ncardozo,sgunther,tjdondt}@vub.ac.be

Abstract—Software variability can be supported by providing adaptations on top of a program's core behavior. For defining and composing adaptations in a program, different paradigms have been proposed. Two of them are feature-oriented programming and context-oriented programming. This paper compares an exemplar implementation of each paradigm. For the comparison, a common case study is used in which we detail how adaptations are defined, expressed, and composed in each paradigm. Based on the case study, we uncover similarities and differences of each implementation, and derive a set of characteristics that identify each of them. The experiment shows several overlapping similarities between the two implementations, which is an indicator that there is a similar core set of characteristics for each paradigm. This finding brings the two seemingly disjoint research directions together, and can stimulate future research both in the direction of merging features and context as well as to improve the characteristic strengths of each paradigm.

Keywords—feature-oriented programming; context-oriented programming; language paradigms

I. INTRODUCTION

Software variability is an important factor in design and implementation of programs. Software programs are often developed for high customizability, for example to provide individual variants for particular clients. The implementation of such programs consists of core behavior and of different adaptations that add or modify the functionality. Program variability can be realized by using language level abstractions as introduced by different paradigms tailored to express program adaptations. Two such paradigms are feature-oriented programming (FOP) [1] and context-oriented programming (COP) [2].

The FOP paradigm is concerned with identifying functionality in the form of features. A feature is a stakeholder-relevant functionality [3] that can be implemented coarsely as a module or fine-granular as different lines of code scattered over the source code [4]. In FOP, adaptations are provided by features that can be expressed in several ways, for example by annotating the core program, or by defining adaptations as refinements. To yield different program variants, features are composed with the core program. Normally, feature composition is done statically at compile time, but recent approaches also offer runtime composition [5].

The COP paradigm is concerned with runtime behavior modifications in order to provide functionality that is adapted with respect to the execution environment of a program. In most COP implementations, adaptations are defined as first-class entities,

to which context-dependent behavior is associated in a modular fashion. Adaptations are dynamically activated and deactivated at runtime to provide and undo context-dependent behavior [2].

Our objective is to identify the similarities and differences for realizing variability in these two paradigms. To this end, we use the expression product line (EPL) case study, providing an example implementation in each paradigm. For FOP we use rbFeatures, a versatile extension of the Ruby programming language that introduces features as first class entities [6][5][7]. For COP we use Subjective-C [8], a COP implementation for mobile devices that is based on the Objective-C programming language. From a comparison in the expression and implementation of the variability concerns of the EPL case study, we derive a set of characteristics that describe how each paradigm introduces variability.

A clear identification of the core characteristics between the two paradigms is a first result to help in forming a joined research for implementing variability. As we will see, the overlapping set of characteristics is an indicator that the FOP and COP paradigms could be brought together as a hybrid language for software variability.

The paper is organized as follows. We provide background to FOP and COP in Section II. Then in Section III, we provide a side-by-side comparison between the FOP (using rbFeatures) and COP (using Subjective-C) implementations of the case study. We compare both implementations with the help of the Expression Product Line (EPL) case study. Based on the case study, we discuss the similarities and differences of both implementations in Section IV. Sections V and VI respectively present the related work, and the conclusion and future work.

II. BACKGROUND

This section introduces the feature-oriented programming and context-oriented programming paradigms.

A. Feature-Oriented Programming

The concept of features initially emerged with the goal to express distinct functionality that is targeted towards a specific stakeholder [3]. This notion of a feature is called conceptual [6], because it only regards the end-user visible behavior, but not its implementation. How to implement such conceptual features is considered in the feature-oriented programming paradigm. Basically, a program consists of different artifacts that provide

the program's functionality. Features encompass different parts of these artifacts, and are therefore distinguished into coarse-grained and fine-grained features [4]. Coarse-grained features can be represented with conventional mechanisms provided by a programming language, such as modules and packages. These can then be composed conveniently with the program's core behavior. Fine-grained features are more difficult to represent and compose, because they can consist of individual classes, methods, or even parts of method bodies. Related work shows a diversity of FOP implementation approaches [9]. Each approach differentiates how features are represented, expressed, and composed. We distinguish these approaches as follows:

- *Annotations* – These approaches use the existing program source code and mark the occurrences of feature-related source code. One type of annotations are *source code annotations* such as “`#ifdef`” statements in C++, which are native preprocessor directives. Before the program gets compiled, all parts of the source code that do not belong to the current feature configuration are pruned. Then, a program variant is created by compiling the remaining source code [10]. Another option is to use *virtual annotations*. In this case, the source code itself is not annotated, but a suitable intermediate program representation, such as the abstract syntax tree [10]. This approach requires tool support for representing the annotations and for generating a program variant.
- *Modules* – These approaches use the programming language modularization concepts to represent features. Among these approaches are traits in Scala [11], atoms and units in Jiazzi [11], Classboxes [12], CaesarJ [13], and Object Teams/Java [14]. The capabilities of modules constraint the level to which especially fine-grained features can be represented and composed.
- *Refinements* – These approaches separate a program into a fixed base program and extensions that are called refinements. Refinements are added to a program, where they change the behavior and the structure. Typically, these approaches add specific language constructs to express these refinements. Some approaches as the AHEAD tool suite [15], for example, use the keyword `refine` as a language construct, other approaches introduce concepts similar to refinements, such as aspects from aspect-oriented programming [16].

We use `rbFeatures` [6][5][7] as the FOP example language. `rbFeatures` is a versatile, pure language of Ruby, that allows features to be defined as first-class entities, giving a close integration of features and other application code. In order to express which part of the source code belongs to a feature, semantic annotations, called feature containments in `rbFeatures`, are used. Containments consist of a condition and a body. A containment condition is a logical expression determining which features need to be active or inactive in order for the body to be included in the program. The containment body is any piece of code: modules, classes, methods, and even individual lines and characters. `rbFeatures` allows to express the hierarchy and constraints of features with an expressive rule language. A program that is feature-refactored with `rbFeatures` allows both runtime and compile-time composition. At runtime, features can be activated and deactivated to immediately affect the program behavior, even allowing different variants of a program to

exist at runtime [5]. At compile-time, the semantic annotations can be preprocessed to derive a static variant. This is done by pruning source code not define within the configured containments.

B. Context-Oriented Programming

Context-oriented programming paradigm [2] allows software systems to be modularized into behavioral adaptations that can be activated, deactivated and composed at runtime. Adaptations are triggered by changing properties of the execution environment, such as device presence, battery level, or user settings. COP languages typically provide dedicated constructs for the definition of behavior adaptations in a modularized fashion, as well for the composition and execution of such adaptations [2][17]. Program entities in which adaptations are defined are called *layers* [2] or *contexts* [17], which are normally defined as first-class entities of the program. We will refer to them as contexts.

Contexts may specify either behavioral or structural adaptations. The former case focuses in modifying functionality of the program with more suited behavior to particular situations of the execution environment. The later case concerns with providing new entities or adapting existing entities in the program for a particular situation.

Behavioral adaptations are the key concepts of COP. Adaptations rely in the dynamic activation and deactivation of context entities. When a context is activated, its associated behavioral adaptations become available in the current scope of the application. Similarly, whenever contexts are deactivated the behavior adaptations become unavailable to the execution environment, and the observed behavior of the program is restored to its former state. Behavioral adaptations can be associated with more than one context, in such a case, a new context entity is created implicitly, representing the combination of the contexts, to which the adaptation is associated [18]. Combined contexts are made available if and only if all of its components are active.

If not dealt with careful, dynamic activation and deactivation of contexts may lead to unexpected or inconsistent behavior. To manage such situations it is possible to define different dependency relations among contexts [8][19]. Constraints imposed by the dependency relations are verified at runtime when a particular context is to be activated or deactivated.

Contexts are stateful objects, state of variables and objects defined within a context are always preserved between context activations [2][8][20]. Moreover, within a context it is possible to extend the definition of objects already existing in a program by dynamically adding state properties to them. As with behavioral adaptations, these structural adaptations also become available and unavailable as the context in which they are defined is respectively activated or deactivated.

In the remainder of this paper we use Subjective-C [8] a full context-oriented language extension of Objective-C whose design is influenced by the Ambience language [18]. Contexts are defined as first-class entities. Context-dependent behavior adaptations need to be defined as methods in a class. These methods are annotated with the name of the context they belong to. Context-dependent behavior is not accessible by the core program until the context to which they are associated is activated. When a context is activated, a method replacement mechanism (from Objective-C's meta-object protocol) replaces original methods with their context version at runtime. Subjective-C uses a *context manager* to maintain a record

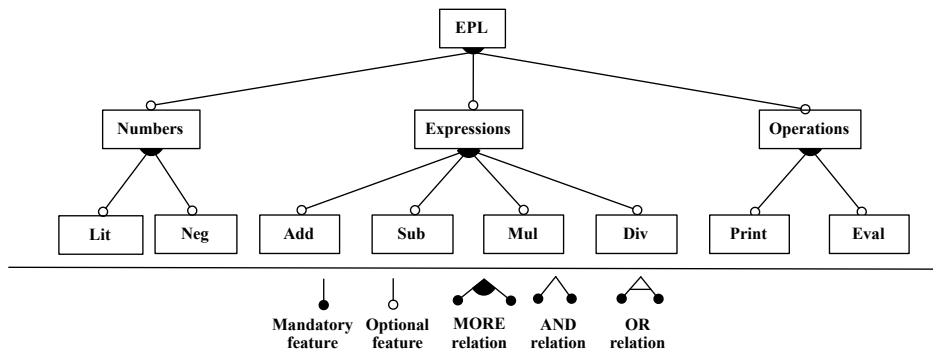


Figure 1: Feature diagram of the Expression Product Line.

of all context objects at runtime, whether they are active or not, and the dependency relations between contexts.

III. CASE STUDY: EXPRESSION PRODUCT LINE

The expression product line (EPL) [11] is a well known case study concerned with finding suitable modularization concepts for representing different types of integers, expressions, and operations over them. All possible program variations of the EPL are shown in Figure 1 – this notation is called a feature diagram, as it depicts the constraints between different features of a program [21]. For the EPL all its features are optional, meaning that they can be build in any combination. We conduct a side-by-side implementation of the case study providing first the rbFeatures example, and subsequently the Subjective-C one. The two implementations are compared based on the principal techniques each uses to realize software adaptations, specifically we are concerned with: (a) The way in which adaptations are declared, (b) The way in which adaptation’s behavior is declared, and (c) The way in which modification to the core program is done.

A. Adaptation Declaration

The implementation of the product line starts with its top down definition. As originally expressed in the case study, expressions like ADD and NEG, as well as the operations for PRINT and EVAL are defined as features, shown in the following snippet for rbFeatures.

```
class Add
  is Feature
end

class Print
  is Feature
end
```

In Subjective-C, LIT, ADD, NEG and the other expression elements are defined as regular (behavior-less) objects, taking advantage of the polymorphic abilities of the language. The PRINT and EVAL operations are defined as contexts providing behavior for expression objects. Operations are declared as named context objects and added to the *context manager*.

```
@interface Add : Exp {
  Exp *left, *right;
}
@end
```

```
SCContext* Print = [[SCGlobalContext alloc]
  initWithName:@"print"];
[[SCContextManager sharedContextManager] addContext:
  Print];
```

B. Behavioral Declaration

Once all adaptations have been defined, the next step is to define the specific behavior added by the features to other objects of the program. We consider enhancing Add expressions with the printing behavior provided by the PRINT adaptation.

In rbFeatures, adaptations are introduced by forming feature containments around a pice of feature-specific code, which can be for example a method declaration. In the following example, the containment condition is the PRINT adaptation, and the containment body is the method declaration. When the PRINT adaptation is activated, a call to the `print` method will behave as shown in the snippet, otherwise the method will return an error message.

```
class Add
  Print.code do
  def print
    Kernel.print(@left.print + " + " + @right.print)
  end
end
```

In Subjective-C, behavioral adaptations are also introduced by adding context-dependent methods within the body of the object that defines it. This is shown in the following snippet.

```
@implementation Add {
  @contexts Print
  - (NSString) print {
    return [NSString stringWithFormat:@"%@" + @"", [left
      print], [right print]];
  }
  @end
}
```

Unlike rbFeatures, in Subjective-C it is not possible to define specific lines within a method as context-dependent. However, this is possible in other COP languages [2][18].

C. Behavioral Modification

Behavior defined for the different EPL expressions and operations is available to the program through the explicit activation of the related feature. For example, in order to have the PRINT

Entity Representation	Annotations	Modules	Refinements	First-Class Entities
Adaptation Constraints	Hierarchy		Rules	
Adaptation Trigger	Internal		External	
Adaptation Activation	Compile-Time		Runtime	
Composition Process	Order-Independent	Order-Dependent	Blocking	Non-Blocking
Adaptation Properties	Stateful		Extensible	Cascading

Figure 2: Morphologic scheme of all implementation characteristics.

adaptation, in `rbFeatures` a call to the `Print.activate` method must be made to activate the adaptation. In Subjective-C the `@activate(Print)` keyword is used to process the context activation.

However, there is a difference in the processing of the two activation messages. In `rbFeatures` the source code enclosed by the feature definition is re-executed, that is, with every activation the code gets redefined and because of changed containment conditions, new behavior is eventually added to the program. Subjective-C, on the other hand, does not re-execute any code. Instead, activation of a context allows its associated methods, variables, objects, and so on, to be visible by the method dispatcher. This is the main reason context-dependent variables are stateful. Whichever the state of a variable is, it remains untouched as long as the context in which the variable is defined is inactive, since it cannot be found by the program.

IV. COMPARISON OF FEATURE-ORIENTED PROGRAMMING AND CONTEXT-ORIENTED PROGRAMMING

In this section, the similarities and differences encountered between our FOP and COP implementations are made explicit. Then we define them as specific characteristics of the implemented paradigm.

We summarize the observed similarities as follows:

- Features and contexts are declared as *first-class entities* of the program.
- Features and contexts add adaptations on top of the core behavior by *annotating source code* at the place where adaptations would normally be defined in.
- Features and contexts can both be *activated and deactivated at runtime*, immediately changing the program behavior.
- Both implementations offer a runtime representation of the *dependencies between adaptations*.

The differences are the following:

- There is *no automatic adaptation* of the dependent features in `rbFeatures`, while Subjective-C uses the dependency relations defined between contexts to automatically activate or deactivate related contexts.
- Feature activation is *externally triggered* by the user in `rbFeatures`, while Subjective-C uses *internal triggers* based on the program state to activate contexts.
- There is *no stateful* composition of adaptations in `rbFeatures`: while instances of objects with feature-dependent behavior retain their state, class variables will be overridden during

the program adaptation. Subjective-C uses a *stateful* representation of contexts. Variables declared in a context cannot be accessed or modified unless the context that defines them is available. Maintaining there state between activations.

- Features can be composed at *compile-time* and *runtime* in `rbFeatures`, while Subjective-C only offers runtime composition.

We use this comparison and related work as the frame of reference for FOP and COP to define a set of characteristics that identify our implementations of each paradigm. These characteristics, also illustrated in Figure 2, are the following ones:

- ENTITY REPRESENTATION [ANNOTATIONS, MODULES, REFINEMENTS, FIRST-CLASS ENTITIES] – Specifies how adaptations are represented. On the one hand are pure annotations that are external to the program and receive their meaning as contexts or features from the processing tool. On the other hand we see first-class entities that are high-level abstractions and can be fully integrated with the program.
- ADAPTATION CONSTRAINTS [HIERARCHY, RULES] – The availability of composition constraints, for example in the form of a hierarchy (a hierarchically higher adaptation is only available if its children are) or rules (arbitrary expressions that state which adaptations need to be active or inactive for a particular adaption to be composed with the program).
- ADAPTATION TRIGGER [INTERNAL, EXTERNAL] – The adaptation process is triggered by an internal signal, like a certain program state upon which it reacts, or by an external signal, for example a change in environment that is detected by a sensor or through a command by the user.
- ADAPTATION ACTIVATION [COMPILE-TIME, RUNTIME] – The adaptation can occur statically at runtime, usually loosing the information about the adaptation and producing a program with fixed behavior, or fully dynamically at runtime.
- COMPOSITION PROCESS [ORDER-DEPENDENT, ORDER-INDEPENDENT, NON-BLOCKING, BLOCKING] – An important difference in the adaptation process is whether the activation order influences the adaptation result, for example when adaptations provide different composition of source code pieces. Furthermore, the adaptation process can block activation of other adaptations during the composition.
- ADAPTATION PROPERTIES [STATEFUL, EXTENSIBLE, CASCADING] – In a stateful adaptation, defined objects and variables retain their states between deactivations and activations.

Extensible means to modify existing adaptations or to add new ones at the program runtime. Finally, cascading denotes the capability that if an adaptation needs to be added or removed from the program, all dependent adaptations are automatically removed or added.

In terms of these characteristics, we can identify our implementations as shown in Figure 3. As we see, there are 6 common characteristics shared between the implementations, and 6 unique ones. Judging from this representation, the main difference between features and contexts is the availability of compile-time composition of program and the availability of stateful, cascading adaptations.

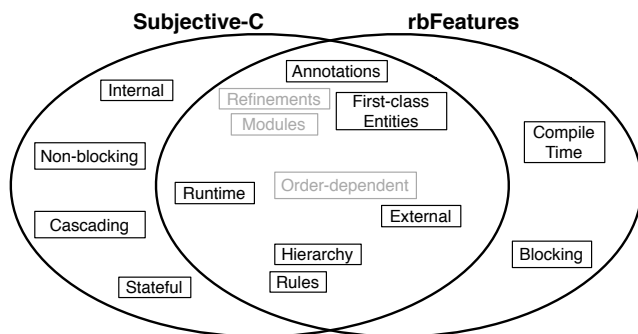


Figure 3: rbFeatures and Subjective-C characteristics.

V. RELATED WORK

To the best of our knowledge, a structured comparison of COP and FOP paradigms as proposed in this paper has not been done. However, several COP ideas are used to build FOP programs and vice versa. We discuss such proposals here.

A first close relation can be seen from the concept of superimposition, which is the process of merging software artifacts by merging their substructures [22]. This mechanism lies at the heart of introducing adaptations of programs, and it is used by several implementations for feature-oriented programming and context-oriented programming.

Context-oriented languages borrow several concepts of feature-oriented programming, at both the implementation and design level. The ContextL [2] COP language uses the concept of mixin-layers [23] normally used as an implementation technique for FOP. Specifically, ContextL uses layers as the main abstraction to define adaptations [24]. Based on the need to express dependencies between layers, and to better control their interaction, a Feature Description Language (FDL) was introduced in ContextL [25] to automatically enforce dependencies between layers.

Additionally, an extension of feature-oriented domain analysis has been used for the design of context-oriented systems, namely Context-Oriented Domain Analysis (CODA) [19]. In this approach, feature diagrams are extended to express resolution strategies whenever there are multiple adaptations available that provide behavior for the same functionality. The CODA approach also introduces *inclusion* and *exclusion* relations between adaptations. The former relation expresses that if an adaptation can be activated all included adaptations are also activated. The later one, expresses

that if an adaptation can be activated, all its excluded adaptations are deactivated.

VI. CONCLUSION AND FUTURE WORK

This paper shows how feature-oriented programming and context-oriented programming paradigms provide closely related strategies for realizing software variability. To understand the differences and similarities between the two paradigms, we implemented a common case study with an FOP language (rbFeatures) and a COP language (Subjective-C). Based on analyzing how behavioral adaptations are expressed and implemented, we derived a set of characteristic properties constituting each paradigm. We found that six characteristics are common in both paradigms, and six are different. In essence, the difference lies in the availability of compile time and/or runtime adaptations and in the stateful transition of the program's behavior.

This contribution helps to clarify the commonalities of the two seemingly disjoint research directions, and can help to stimulate research both towards the merging of features and contexts, as well as to improve the characteristic strength of each paradigm.

In future work, the next step is to extend this study with an in-depth analysis of other FOP and COP languages. We wish to further refine the characteristics, and based on it, it would be possible to think about how FOP and COP can be merged in hybrid languages for variability, for example, by adding stateful representation of features or to add compile-time composition to COP implementations that restrict the amount of runtime contexts deployed in devices.

ACKNOWLEDGEMENTS

This work has been supported by the ICT Impulse Programme of the Brussels Institute for Research and Innovation, and by the Interuniversity Attraction Poles Programme, Belgian State, Belgian Science Policy. We thank the anonymous reviewers for their comments on an earlier version of this paper.

REFERENCES

- [1] C. Prehofer, "Feature-Oriented Programming: A Fresh Look at Objects," in *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP)*, ser. Lecture Notes in Computer Science, M. Aksit and S. Matsuoka, Eds., vol. 1241. Berlin, Heidelberg, Germany: Springer-Verlag, 1997, pp. 419–443.
- [2] P. Costanza and R. Hirschfeld, "Language Constructs for Context-Oriented Programming: An Overview of ContextL," in *Proceedings of the 1st Symposium on Dynamic Languages*. New York, USA: ACM, 2005, pp. 1–10.
- [3] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Carnegie Mellon University, USA, Tech. Rep. CMU/SEI-90-TR-21, 1990.
- [4] C. Kästner, S. Apel, and M. Kuhlemann, "Granularity in Software Product Lines," in *Proceedings of the 30th International Conference on Software Engineering (ICSE)*. New York: ACM, 2008, pp. 311–320.

- [5] S. Günther and S. Sunkle, "Dynamically Adaptable Software Product Lines using Ruby Metaprogramming," in *Proceedings of the 2nd International Workshop on Feature-Oriented Software Development (FOSD)*. New York: ACM, 2010, pp. 80–87.
- [6] S. Günther and S. Sunkle, "Feature-Oriented Programming with Ruby," in *Proceedings of the First International Workshop on Feature-Oriented Software Development (FOSD)*. New York: ACM, 2009, pp. 11–18.
- [7] S. Günther and S. Sunkle, "rbFeatures: Feature-Oriented Programming with Ruby," in *Science of Computer Programming*. Elsevier, 2011, accepted 01.01.2011, in press.
- [8] S. González, N. Cardozo, K. Mens, A. Cádiz, J.-C. Libbrecht, and J. Goffaux, "Subjective-c: Bringing context to mobile platform programming," in *Proceedings of the International Conference on Proceedings of the International Conference on Software Language Engineering*, ser. series-lncs, B. Malloy, S. Staab, and M. van den Brand, Eds., vol. 6563. Eindhoven: Springer, 2011, pp. 246 – 265.
- [9] S. Apel and C. Kästner, "An Overview of Feature-Oriented Software Development," *Journal of Object Technology (JOT)*, vol. 8, no. 5, pp. 49–84, 2009.
- [10] C. Kästner and S. Apel, "Virtual Separation of Concerns – A Second Chance for Preprocessors," *Journal of Object Technology (JOT)*, vol. 8, no. 6, pp. 59–78, Sep. 2009.
- [11] R. E. Lopez-Herrejon, D. Batory, and W. Cook, "Evaluating Support for Features in Advanced Modularization Techniques," in *Proceedings of the 19th European Conference on Object-Oriented Programming (ECOOP)*, ser. Lecture Notes in Computer Science, A. P. Black, Ed., vol. 3586. Berlin, Heidelberg, Germany: Springer-Verlag, 2005, pp. 169–194.
- [12] A. Bergel, S. Ducasse, O. Nierstrasz, and R. Wuyts, "Classboxes: Controlling Visibility of Class Extensions," *Computer Languages, Systems & Structures*, vol. 31, no. 3, pp. 107–126, 2005.
- [13] I. Aracic, V. Gasiunas, M. Mezini, and K. Ostermann, "An Overview of CaesarJ," in *Transactions on Aspect-Oriented Software Development I*, ser. Lecture Notes in Computer Science, A. Rashid and M. Aksit, Eds. Berlin, Heidelberg, Germany: Springer-Verlag, 2006, vol. 3880, pp. 135–173.
- [14] S. Herrmann, "A Precise Model for Contextual Roles: The Programming Language ObjectTeams/Java," *Applied Ontology*, vol. 2, no. 2, pp. 181–207, 2007.
- [15] D. Batory, "Feature-Oriented Programming and the AHEAD Tool Suite," in *Proceedings of the 26th International Conference on Software Engineering (ICSE)*. Washington: IEEE Computer Society, 2004, pp. 702–703.
- [16] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-Oriented Programming," in *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP)*, ser. Lecture Notes in Computer Science, M. Aksit and S. Matsuoka, Eds. Berlin, Heidelberg, Germany: Springer-Verlag, 1997, vol. 1241, pp. 220–242.
- [17] S. González Montesinos, "Programming in ambience: Gearing up for dynamic adaption to context," Ph.D. dissertation, Université Catholique de Louvain, October 2008.
- [18] S. González, K. Mens, and A. Cádiz, "Context-oriented programming with the ambient object system," *Journal of Universal Computer Science*, vol. 14, no. 20, pp. 3307–3332, 2008.
- [19] B. Desmet, J. Vallejos, P. Costanza, W. De Meuter, and T. D'Hondt, "Context-Oriented Domain Analysis," in *Modeling and Using Context, Sixth International and Interdisciplinary Conference on Modeling and Using Context*, August 2007, pp. 178–191.
- [20] S. González, K. Mens, and P. Heymans, "Highly Dynamic Behaviour Adaptability through Prototypes with Subjective Multi-methods," in *Proceedings of the 2007 symposium on Dynamic Languages (DLS)*, ser. DLS '07. New York, NY, USA: ACM, 2007, pp. 77–88.
- [21] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, San Francisco et al.: Addison-Wesley, 2000.
- [22] S. Apel and C. Lengauer, "Superimposition: A Language-Independent Approach to Software Composition," in *Software Composition*, ser. Lecture Notes in Computer Science, C. Pautasso and E. Tanter, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, vol. 4954, pp. 20–35.
- [23] Y. Smaragdakis and D. Batory, "Mixin Layers: An Object-Oriented Implementation Technique for Refinements and Collaboration-Based Designs," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 2, pp. 215–255, 2002.
- [24] B. Desmet, J. Vallejos, and P. Costanza, "Introducing Mixin Layers to Support the Development of Context-Aware Systems," in *3rd European Workshop on Aspects in Software (EWAS)*, ser. Technical Report IAI-TR-2006-6, G. Kneisel, Ed. Universität Bonn, 2006, pp. 23–30.
- [25] P. Costanza and T. D'Hondt, "Feature Descriptions for Context-Oriented Programming," in *Proceedings 12th International Conference for Software Product Lines (SPLCL), 2nd International Workshop on Dynamic Software Product Lines (DSPL)*, S. Thiel and K. Pohl, Eds., vol. 2 (Workshops). Ireland: University of Limerick, 2008, pp. 9–14.

Soft Constraints in Feature Models

Jorge Barreiros

Instituto Superior de Engenharia de Coimbra, Coimbra
Universidade Nova de Lisboa, Lisboa, Portugal
jmsousa@isec.pt

Ana Moreira

CITI/Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, Lisboa, Portugal
amm@di.fct.unl.pt

Abstract—Feature Models represent admissible configurations of products in Software Product Lines. Constraints are used to represent domain specific knowledge, such as requiring or excluding a feature in the presence of another. Configurations failing to conform to these constraints are deemed invalid. However, in many cases useful domain information cannot be expressed comfortably with such forceful, hard constraints. Therefore, we propose the use of softer constraints of less forcing nature. We categorize possible semantics for such constraints, analyze their impact on the feature expression and describe some specific analysis procedures that are unique to the use of soft constraints.

Keywords—Feature Models; Software Product Lines; Soft Constraints; Feature Consistency; Feature Interaction, Semantic Validation

I. INTRODUCTION

In opposition to traditional single system development, Software Product Line (SPL) development is concerned with the creation of families of software products. In SPLs, product variants belonging to the same family are created by specifying a feature configuration, which is then realized by the composition of corresponding artifacts from a common pool of assets (such as requirements documents, design models, code, etc.) [1].

Feature models are frequently used in SPL development for identifying valid product configurations, that is, configurations corresponding to a variant that can be created by an application engineer using the SPL [2]. Feature models identify valid configurations by using a feature tree annotated with additional domain constraints. These can be represented graphically (e.g., linking dependent features with a dependency arrow) or textually, by means of arbitrary cross-tree expressions (Boolean expressions depending on the configuration variables). Feature models can be represented using logic expressions according to well known transformations described in [3, 4]. A feature model expression is obtained by conjoining the feature tree expression with the domain constraints.

An example of a feature model can be found in Fig. 1, where *Sound*, *Keyboard* and *Screen* are mandatory subfeatures of the root feature node *Phone*, while *MP3Player* and *Camera* are optional subfeatures. *Polyphonic* and *Monophonic* are mandatory and alternative subfeatures of the *Sound* feature, and *Monochromatic* and *Polychromatic* are alternative

subfeatures of the *Screen* feature. One domain constraint is represented: the *requires* arrow describes that selection of the *Camera* feature implies the selection of the *Color* feature.

Links such as the one connecting *Camera* and *Color* in Fig. 1 describe *hard constraints*. Any configuration that does not respect this constraint is invalid. It can be the case, however, that domain information is not comfortably representable using such strict constructs. For example, a situation can be considered where the overwhelming majority of configurations do indeed respect a certain restriction, but a few exceptions may exist. In this case, restrictions on admissible configurations cannot be as strict. A simple example will be the case of a default selection for a group of alternative selections: if the parent feature of such group is selected, then the preferred alternative configurations may be suggested.

We propose the use of *soft constraints*, of less forcing nature, in these situations. The concept of soft constraint has been described earlier in the context of probabilistic feature models [5]. Probabilistic feature models extend standard feature models by the addition of “soft” constraints that are associated with a degree of probability. These are often obtained as the result of a feature mining processes. We consider the use of a similar concept in in standard, deterministic feature models. This allows richer semantics to be represented in feature models, with advantages such as enhanced analysis and improved configuration support. An example of such a constraint in Fig. 1 would be “*Sound suggests Polyphonic*”, expressing domain knowledge that indicates the more common sound configuration option. Naturally, soft constraints do not need to be restricted to parent-child features as described: other relations such as “*Monophonic suggests Monochromatic*” can be represented. This type of constraints can be useful for efficiently capturing useful domain information that might be lost otherwise, as it is usually absent in standard feature models. It can be used to good effect for multiple purposes, depending on the specific semantics that are adopted as described later, such as allowing interactive configuration tools to suggest configuration choices to the user.

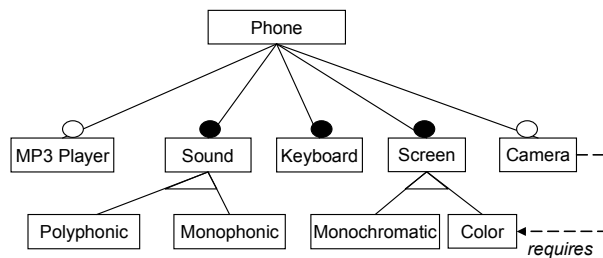


Figure 1. Mobile phone feature model.

Using soft constraints also allows some semantic consistency analysis that would otherwise be impossible, e.g., if a suggested dependency can never be realized in a feature model, then probably something is not right. Conflicting suggestions can also be found (e.g., multiple suggestions that cannot be satisfied simultaneously), highlighting that a trade-off analysis may be in order to compatibilize the inconsistent soft constraints.

The contributions of this work are the categorization of soft constraint semantics, the formalization of the impact (if any) of these constraints on the logic representation of the feature model and the description of automated analysis procedures made possible by the use of soft constraints.

In Section II, we present motivating examples for our work. In Section III, we discuss benefits of the use of soft constraints and propose a categorization of the different types of soft constraints. In Section I, we suggest a formalization and analysis techniques for detecting unsatisfiable and conflicting soft constraints. In Section V we present related work and we conclude in Section VI.

II. MOTIVATION

Consider the example in Fig. 2, adapted from [5], where a feature model is used to describe configuration variability for an automobile vehicle. In this case, hard domain restrictions are used to enforce the selection of manual transmission in sports vehicles and to make sure

that emission control techniques are always used in products destined for markets with stricter environmental legislations. While observance of such constraints is always found in valid products, soft constraints are used to represent relevant relations between features that, while not as critical or universally applicable as the hard constraints, are also important. In this case, it is well known that the USA market tends to favor vehicles with automatic transmission over those with manual transmission, while the converse is true for the European market. Using soft constraints, such information can be readily represented in the feature diagram, bringing in additional semantics that can be used to good effect.

Another example of the use of soft constraints can be found in Fig. 3. In this case, the feature model is used to represent dynamic variability of the runtime behavior of a real-time system. The system should adapt its behavior to conform to variations in its environment. The state of the operation environment is assessed by appropriate sensors and the corresponding features are (de)selected accordingly, with corresponding impact on the runtime behavior as dictated by the constraints. A base control task is to be active at all times, while fan control is only suggested if the temperature is medium, but mandatory if it reaches a high level. A filtering task is suggested if electric noise is detected.

The need to use soft constraints to describe the variability in this scenario is supported by the fact that the suggested (non mandatory) features may not always be selected because of limited resources (e.g., available CPU load). This means that a feature such as *Fan Control* may in fact remain unselected in the presence of its suggestor (i.e., the *Noisy* feature), which cannot be comfortably expressed using only hard constraints.

These examples suggest that soft constraints can be used to good effect in feature models, by allowing the inclusion of important domain information of non-forcing nature.

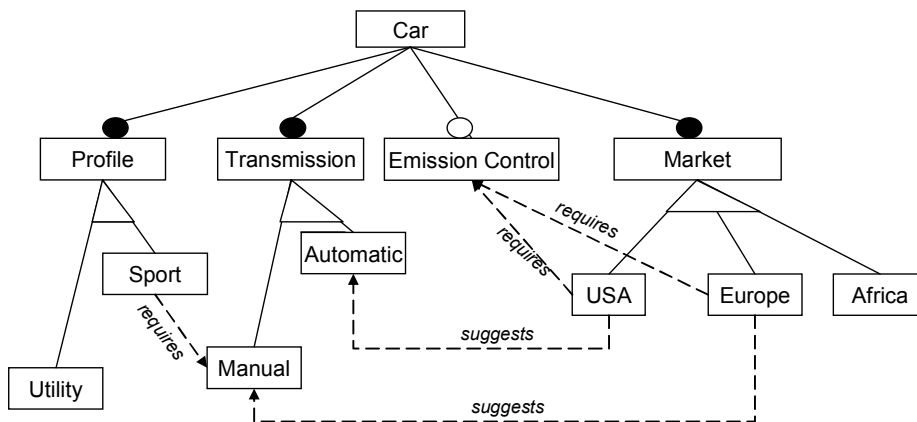


Figure 2. Feature model for car configuration

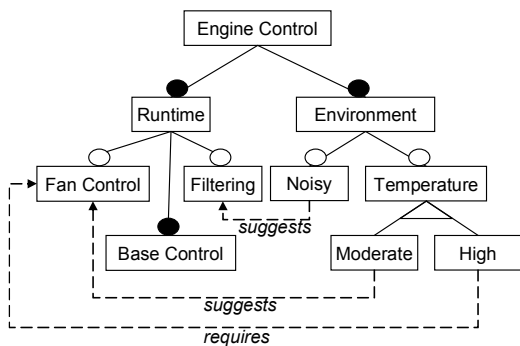


Figure 3. Engine control system

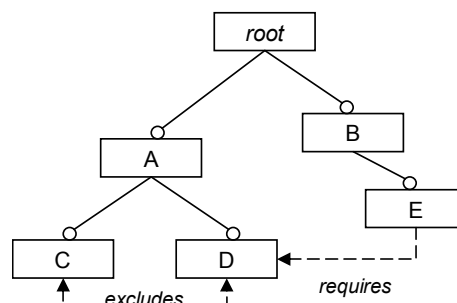


Figure 4. Iterative configuration example

III. SOFT CONSTRAINTS

In this section, we discuss the benefits gained by using soft constraints in feature models and present a categorization of alternative semantics.

A. Benefits

Benefits of soft constraints in feature models include:

- Improved configuration support:** Interactive configuration and completion techniques can assist the configuration of feature models by assessing the liveness of features after each configuration step. Starting from an empty configuration where all features are considered to be unspecified (neither selected or deselected), after a feature is selected or deselected by the user, the liveness of all features is re-evaluated with respect to the partial configuration already defined. Features that are found to be dead (always unselected) in that partial configuration can be safely deselected automatically. Conversely, features that are common to all configurations that include the partial configuration so far specified can be automatically selected. For example, if the developer specifies feature *C* in Fig.4 to be selected, then features *D* and *E* can be automatically deselected by the configuration tool, as no valid configuration including feature *C* will contain either (i.e., both are dead in all configurations where *C* is selected). Similarly, *A* and *root* are common to all such configurations, so they can be selected automatically, leaving only feature *B* unspecified. Interactive configuration and completion tools can use soft constraint information to make configuration suggestions to the user. For example, if “*A* suggests *B*”, the configuration tool can propose the selection of *B* by default whenever *A* is selected and *B* is unspecified. In the case of normative soft constraints, increased restrictions on admissible configurations also help to narrow down the correct configurations. Also, if a valid configuration fails to conform to a large percentage of soft constraints, it can be flagged to the developer as suspicious.

- Improved semantic-oriented consistency checks:** Standard consistency analysis of feature models is concerned with ensuring that valid configurations do exist. If soft constraints are present, it is possible to make sure that configurations are available that verify the suggested dependencies. If that is not the case, this may be a sign that an analysis or modeling error has occurred. For example, if it was actually impossible to configure a car for the European market with manual transmission despite such association being suggested (e.g., because of the unintended side effect of some hard constraints), this would be highly suspicious and should be reported to the developer for additional consideration. This could be the case if hard domain restrictions would make it impossible to select a configuration where both such features are selected.
- Controlled generalization of feature models:** A generalization of a feature model is a transformation that increases the number of admissible configurations, making sure that previously valid configurations remain valid. In some cases, soft constraints can be used as a mechanism for controlled generalization of feature models. For example, if it was found, after creating the feature model in Fig. 2, that it should actually be possible, under certain circumstances, to produce vehicles without emission control for the USA market, the hard restriction that forbids such products from being created could be transformed into an equivalent soft constraint. This would have the benefit of preserving important domain information while accommodating the need to allow for spurious “rogue” configurations.

B. Semantics and Categorization

Soft constraints can be interpreted according to different semantics, from unassuming configuration suggestions (e.g., describing a predominant configuration as in [5]) to stricter impositions that must be enforced if possible (i.e., a feature must be selected if possible). According to the adopted interpretation, different types of analysis and interpretations may be possible. Therefore, we must consider the possible semantics. These can be broadly categorized in two different categories:

- **Annotational:** A soft constraint with an annotational semantics does not impose any additional restriction when added to a feature model. Its main purpose is to embed domain information in the feature model to assist the configuration automation and semantic consistency checking. The validity of any specific product configuration is never influenced by the presence of an annotational soft constraint.
- **Normative:** A normative soft constraint must be considered when assessing the validity of a product configuration. These constraints represent configuration information that may potentially condition the validity of some configurations. A normative soft constraint must be satisfied if possible, but can be ignored otherwise. The concept of “possible satisfaction” is, generally, always dependent on the characteristics of the feature model and is also potentially dependent on domain-specific information (external to what is represented on the feature model: see below). A normative soft constraint may change the validity of a configuration (with respect to the unconstrained feature model), but it may never cause a feature model to become inconsistent. Normative constraints can be interpreted informally as meaning “requires-if-possible”, “may-require”, “require-if-does-not-make-configuration-invalid” or some other similar formulation.

Applying normative constraints entails the need to assess the “possibility” of selecting a specific feature. The topology of the feature model and cross-tree-constraints is always a decisive factor in making that assessment (i.e., it cannot be reasonably considered “possible” to select a feature when doing so would generate an invalid configuration). However, it may be the case that the feature model information is not sufficient to assess the possibility of selecting a feature: in this case, external factors, not represented in the feature model would come

into play. This suggests the following characterization of normative constraints:

- **Internal:** The feature model holds all the information required to assess selection possibility.
- **External:** The information in the feature model alone is not sufficient for assessing possibility of selection. External factors come into play.

In the example of Fig. 2, if the soft constraints are interpreted under annotational semantics, then any configuration that upholds the hard constraints is considered valid, regardless of complying or not with the soft constraints. On the other hand, if an (internal) normative semantic is considered, the following interpretation holds: “If the *USA* feature is selected, then the *Automatic* feature must be selected, unless doing so would generate an invalid configuration”. That is, a normative soft constraint should be interpreted as a hard constraint, unless doing so would turn an otherwise valid configuration into invalid. In Fig. 3, a potential example of external normative soft constraints is represented: in this case, the *Fan Control* feature should always be selected if the *Moderate* heat feature is selected, unless that is not possible, according to domain information that is not necessarily integrated in the feature model. For example, knowing that the implementations of the *Base Control*, *Fan Control* and *Filtering* features compete for a limited resource (CPU load), assessing of the possibility of including the *Fan Control* feature must be conducted with respect to external information. It is out of the scope of this work to discuss how such external information would be obtained or retrieved – as examples, an oracle could be used to provide the required information or a domain specific ontology could be queried.

Table I presents a summary of the characterization of hard and soft constraints.

TABLE I. SOFT AND HARD CONSTRAINTS CHARACTERIZATION

Nature	Subtype	Description	Affects FM consistency?	Affects config validity?	Semantics
Hard		<i>A requires B</i>	Yes	Yes	$A \Rightarrow B$
		<i>A excludes B</i>	Yes	Yes	$A \Rightarrow \neg B$
Soft	Normative	<i>A may-require B</i>	No	Yes	Equivalent hard restriction should be upheld unless doing so would make the configuration invalid.
		<i>A may-exclude B</i>	No	Yes	May be further categorized as “external” or “internal”
	Annotational	<i>A encourages B</i>	No	No	Measure of belief concerning the correlation between the configuration of both features.
		<i>A discourages B</i>	No	No	

IV. SOFT CONSTRAINT ANALYSIS

In this section, we present some formalization and analysis techniques specific for feature models with soft constraints. Although we propose a specific terminology for each different type of soft constraints in Table I, in the remaining text we use a link labeled “suggests” to indicate either “mayRequire” or “encourages” when the distinction is not important. For economy of space, exclusion-oriented constraints are not specifically discussed, but most results apply with minimal, usually obvious, adaptations.

A. Feature Expression for Normative Soft Constraints

Internal normative soft constraints may change the assessment of the validity of configurations with respect to the unconstrained feature model. This results in a change of the model expression when a new soft constraint is introduced in an existing feature model. The effect of inserting an internal normative soft constraint (A suggests B) results in a new feature model expression defined by:

$$F_S(A, B, \dots) = F(A, B, \dots) \wedge ((A \Rightarrow B) \vee \neg F(A, \neg B, \dots)) \quad (1)$$

where F is the feature model expression without the soft constraint and F_S is the resulting feature model expression.

An advantage of using internal normative soft constraints is that standard feature model techniques apply normally, e.g., satisfiability-based techniques are commonly applied to the analysis of feature model expressions [6], for tasks such as finding dead features. This can be also done in a feature model annotated with soft constraints by considering the relevant F_S .

Equation (1) can be applied iteratively with respect to all soft constraints to obtain the feature expression corresponding to a feature model with multiple soft constraints. However, as described in Section IV.C, conflicting constraints may warrant additional care.

B. Unsatisfiable Constraints

Soft constraints can be used to include meaningful domain information in the feature model. One of the benefits this provides is the possibility of verifying if the feature model admits the existence of solutions that satisfy these soft constraints. That is, verifying if the feature model is *semantically* consistent with well known domain properties represented by soft constraints. If that is not the case, it is almost certainly an indication that an analysis error has been made and the feature diagram should be evaluated. This is not the same problem as the standard consistency assessment of a feature model as in that case we are only concerned with ensuring that at least one valid configuration exists. Consider the example in Fig. 5; in this case, because B and C are alternative features, it is not possible to find any configuration that conforms to the soft constraint suggestion. If the soft constraint represents a well known domain property, then it can be reasonably assumed that an analysis error has been made and that a re-evaluation of the feature model or the soft constraint might be advisable.

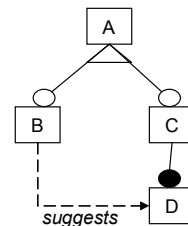


Figure 5. Unsatisfiable soft constraint

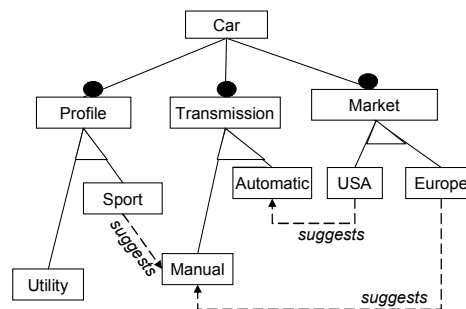


Figure 6. Conflicting soft constraints example

Unsatisfiable soft constraints can be identified by assessing the unsatisfiability of:

$$\neg(F_S(A, B, \dots) \Leftrightarrow F(A, B, \dots)) \quad (2)$$

Where F and F_S are defined as in (1). Unsatisfiability of (2) is indicative of an unsatisfiable soft constraint. Unsatisfiable constraint analysis can be performed not only with respect to normative constraints but also annotational ones. This is one of the advantages of including annotational soft constraints in feature models. Although these do not actually change the feature expression in any way, the same equations can be used for the purpose of constraint satisfiability analysis.

C. Conflicting Soft Constraints

Consider that, in the example of Fig. 2, after constructing the feature model, the developer finds that, although unusual, in some cases it may be necessary to allow configurations with the *Sport* profile and *Automatic* transmission. One way to handle this situation is to reduce the strength of the hard domain constraint that imposes *Manual* transmission for *Sport* vehicles by transforming it into a corresponding soft constraint. A partial representation of the resulting feature model is found in Fig.6.

It can be observed that simultaneous selection of the *USA* and *Sport* features will entail conflicting suggestions of transmission configuration. In such a situation, we describe the corresponding constraints to be conflicting. It is worth noting that this model is not inherently wrong as would be the case if hard constraints were involved.

The following procedure can be used to determine if soft constraints (A→B) and (C→D) will conflict when added to in a consistent feature model with expression $F(A, B, \dots)$:

1. Verify the satisfiability of $F(A, B, \dots) \wedge A \wedge C$. If it is not satisfiable, then no conflict exists.
2. If that is not the case, verify the satisfiability of $F(A, B, \dots) \wedge (A \Rightarrow B) \wedge (C \Rightarrow D)$. If it is not satisfiable, then a conflict exists.

When conflicting soft constraints are to be applied to a feature model, the order by which (1) is iterated to obtain the feature expression, as described in Section IV.A, is relevant to the outcome. Assuming all conflicting suggestions are of equal force, this is not desired and the following process should be used instead:

1. Identify all groups of conflicting soft constraints.
2. Iterate over all groups of conflicting soft constraints and compute:

$$F_{s,n}(A, B, \dots) = F_{s,n-1}(A, B, \dots) \wedge \bigvee_i ((A_i \Rightarrow B_i) \vee \neg F(A_i, \neg B_i, \dots))$$

with $F_{s,0}(A, B, \dots) = F(A, B, \dots)$

This will create a feature expression where all conflicting suggestions are integrated. No preference is given to any suggestion over other, that is, in the example of Fig. 6, configurations with $\{Sport, Manual, USA\}$ are just as admissible as $\{Sport, Automatic, USA\}$. If an interactive configuration tool was being used, $\{USA, Sport\}$ were selected and both *Automatic* and *Manual* were unspecified, both of these features could be presented as configuration suggestions. Nevertheless, in some situations it may be desirable to perform a trade-off analysis and prioritize the relative importance of soft constraints. This would be the case if, for example, the *Sport* feature was a dominating factor on the choice of transmission. In this case, rather than following the process outlined above, (1) should be used instead, in order of the desired priority. That is, first consider the effect of the *Sport* feature on the feature model and only then compute the effect of the *USA* feature (on the previously computed feature model). This would allow for disambiguation of the suggestions represented by the soft constraints.

V. RELATED WORK

In [5], probabilistic feature models are described that use soft constraints as descriptions of features that have high probabilities of being concurrently selected in the same configuration. Probabilistic feature models and corresponding samples spaces are suited to represent feature models obtained through feature mining processes. The fundamental purpose of probabilistic soft constraints in that context is to represent the results of the mining process. According to the classification in Section III.B, probabilistic soft constraints are inherently annotational, and as such do not affect the validity of any specific configuration, as is the case of the normalizing soft constraints we describe and analyze. We envision the use of soft constraints more as a fundamental construct of feature models, rather than being an auxiliary artifact.

“Encourages” and “discourages” constraints have been proposed for feature models in [7]. However, no precise semantics have been provided, precluding automated analysis and reasoning as described in our work.

In [8], fuzzy logic is applied to related feature configurations to customer profiles. Fuzzy logic is a powerful tool for handling uncertainty. Nevertheless, normative semantics may be difficult to include in such an approach.

VI. CONCLUSIONS

We presented an exploratory analysis of the use of soft constraints in feature models. Possible semantics were specified and specific analysis techniques described. We found that soft constraints are useful in a diversity of contexts and offer the possibility of bringing additional important domain information to the feature model.

Future work includes application of soft constraints to well known industrial and academic case studies. Our prototype tool will be integrated with configuration tools providing enhanced configuration support.

VII. ACKNOWLEDGMENTS

This work has been partially supported by the Portuguese Government through the PROTEC program grant SFRH/PROTEC/49834/2009 and by the Portuguese research centre CITI through the grant PEst-OE/EEI/UI0527/2011.

REFERENCES

- [1] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*: Addison-Wesley, 2001.
- [2] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*: Addison-Wesley Professional, 2000.
- [3] D. S. Batory, "Feature Models, Grammars, and Propositional Formulas," in *Software Product Lines, 9th International Conference, SPLC 2005* Rennes, France, 2005, pp. 7-20.
- [4] K. Czarnecki and A. Wasowski, "Feature Diagrams and Logics: There and Back Again," in *11th International Software Product Line Conference (SPLC)* Kyoto, 2007, pp. 23-34.
- [5] K. Czarnecki, S. She, and A. Wasowski, "Sample Spaces and Feature Models: There and Back Again," in *Software Product Lines, 12th International Conference, SPLC* Limerick, Ireland, 2008, pp. 22-31.
- [6] M. Mendonça, A. Wasowski, and K. Czarnecki, "SAT-based analysis of feature models is easy," in *Software Product Lines, 13th International Conference, SPLC 2009*, San Francisco, California, USA, 2009, pp. 231-240.
- [7] H. Wada, J. Suzuki, and K. Oba, "A feature modeling support for non-functional constraints in service oriented architecture.," *IEEE Computer Society*, pp. 187-195, 2007.
- [8] S. Robak and A. Pieczynski, "Employment of fuzzy logic in feature diagrams to model variability in software families.," in *10th IEEE International Conference on Engineering of Computer-Based Systems (ECBS 2003)* Huntsville, AL, USA, 2003, pp. 305-311.

Feature Modeling of Software as a Service Domain to Support Application Architecture Design

Karahan Öztürk

Department of Computer Engineering,
Middle East Technical University
Ankara, Turkey
e-mail: karahanozturk@gmail.com

Bedir Tekinerdogan

Department of Computer Engineering
Bilkent University
Ankara, Turkey
e-mail: bedir@cs.bilkent.edu.tr

Abstract—Cloud computing is an emerging computing paradigm that has gained broad interest in the industry. SaaS architectures vary widely according to the application category and number of tenants. To define a proper SaaS architecture it is important to have a proper understanding of the domain. Based on our extensive domain analysis approaches, we provide a feature model for SaaS that depicts the design space and represents the common and variant parts of SaaS architectures. The feature model enhances the understanding of SaaS systems, and supports the architect in designing the SaaS application architectures.

Keywords- modeling, service, architecture, design, SaaS

I. INTRODUCTION

Cloud computing is an emerging computing paradigm that has gained broad interest [6][19]. Unlike traditional enterprise applications that rely on the infrastructure and services provided and controlled within an enterprise, cloud computing is based on services that are hosted on providers over the Internet. The services that are hosted by cloud computing approach can be broadly divided into three categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service and Software-as-a-Service (SaaS). In this paper we will focus on the Software as a Service context [18]. SaaS is a web-based, on-demand distribution model where the software is hosted and updated on a central site and does not reside on client computers [1][3]. With SaaS, software applications are rented from a provider as opposed to purchased for enterprise installation and deployment. Similar to the general benefits of cloud computing the SaaS approach yields benefits such as reduced cost, faster-time-to-market and enhanced scalability.

An appropriate SaaS architecture design will play a fundamental role in supporting the cloud computing goals [13][4]. Based on the literature we can derive the basic components required for SaaS. However, while designing particular applications one may derive various different application design alternatives [1] for the same SaaS architecture specification. Each design alternative may meet different functional and nonfunctional requirements. It is important to know the possible design so that a viable realization can be selected.

To enhance the understanding of SaaS systems and support the architect in designing SaaS architectures we propose

defining a feature model for SaaS architectures. A feature model is the result of a domain analysis process whereby the common and variant properties of a domain or product are elicited and modeled [15]. In addition, the feature model identifies the constraints on the legal combinations of features and as such, a feature model defines the feasible models in the domain. The feature model has been derived after an extensive literature study to SaaS architectures. This included basically a systematic literature study on cloud computing in general and software as a service architectures in particular. It should be noted that we could not put all the references in this paper due to space limitations. Based on a commonality and variability analysis of the selected papers the common and variant features of SaaS were derived.

The remainder of the paper is organized as follows. Section II presents SaaS architecture for which a feature model will be defined. Section III presents the family feature model for SaaS. Section IV presents an example illustrating the derivation of application architecture based on application feature model. Finally section V concludes the paper.

II. SOFTWARE AS A SERVICE ARCHITECTURE

SaaS has been widely discussed in the literature and various definitions have been provided. In general when describing SaaS, no specific application architecture is prescribed but rather the general components and structure is defined. Based on the literature we have defined the reference architecture for SaaS as given in Figure 1 [3][13][18][6]. Besides of the theoretical papers we have also looked at documentation of reference architectures as defined by SaaS vendors such as Intel [18], Sun [19] and Oracle [10].

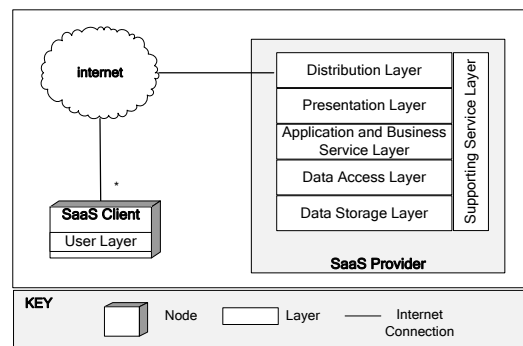


Figure 1. SaaS Reference Architecture

In principle, SaaS has a multi-tier architecture with multiple thin clients. In Figure 1 the multiplicity of the client nodes is shown through the asterisk symbol (*). In SaaS systems the thin clients rent and access the software functionality from providers on the internet. As such the cloud client includes only one layer User Layer which usually includes a web browser and/or the functionality to access the web services of the providers. This includes, for example, data integration and presentation. The SaaS providers usually include the layers of Distribution Layer, Presentation Layer, Business Service Layer, Application Service Layer, Data Access Layer, Data Storage Layer and Supporting Service Layer.

Distribution Layer defines the functionality for load balancing and routing. *Presentation Layer* represents the formatted data to the users and adapts the user interactions. The *Application and Business Service Layer* represents services such as identity management, application integration services, and communication services. *Data Access Layer* represents the functionality for accessing the database through a database management system. *Data Storage Layer* includes the databases. Finally, the *Supporting Service Layer* includes functionality that supports the horizontal layers and may include functionality such as monitoring, billing, additional security services, and fault management. Each of these layers can be further decomposed into sub-layers.

Although Figure 1 describes the common layers for SaaS reference architecture, it deliberately does not commit on specific *application architecture*. For example, the number of clients, the allocation of the layers to different nodes, and the allocation of the data storage to nodes is not defined in the reference architecture. Yet, while designing SaaS for a particular context we need to commit on several issues and make explicit design decisions that define the application architecture. Naturally, every application context has its own requirements and likewise these requirements will shape the SaaS application architecture in different ways. That is, based on the SaaS reference architecture we might derive multiple application architectures.

III. FEATURE MODEL OF SaaS

To support the architect in designing an appropriate SaaS application architecture a proper understanding of the SaaS domain is necessary. In this section we define the SaaS feature model that represents the overall SaaS domain. Figure 2 shows the conceptual model representing the relation between feature model and SaaS architecture.

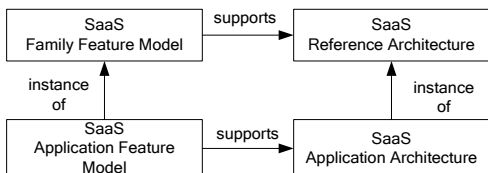


Figure 2. Conceptual model representing relation between feature model and SaaS architecture

We distinguish between *family feature model* and *application feature model*. The family feature model represents the features of the overall SaaS domain, whereas the

application feature model represents the features for a particular SaaS project. The application feature model is derived from the family feature model. The features in the feature model typically refer to the architectural elements in the SaaS architecture. As discussed in the previous section we also distinguish between SaaS reference architecture and SaaS application architecture. For designing the SaaS application architecture first the required features need to be selected from the family feature model resulting in the application feature model. The application feature model will be used to support the design of the SaaS application architecture. In the following we will elaborate on the family feature model.

A. Top-Level Feature Model

The top level feature diagram of SaaS that we have derived is shown in Figure 3. The key part represents the different types of features including *optional*, *mandatory*, *alternative*, and *or features* [15]. Note that the features in Figure 3 denote the layers in the SaaS reference architecture as defined in Figure 1. All the layers except the Support Layer have been denoted as mandatory features. The Support Layer is defined as optional since it might not always be provided in all SaaS applications. Each of these layers (features) can be further decomposed into sub-layers.

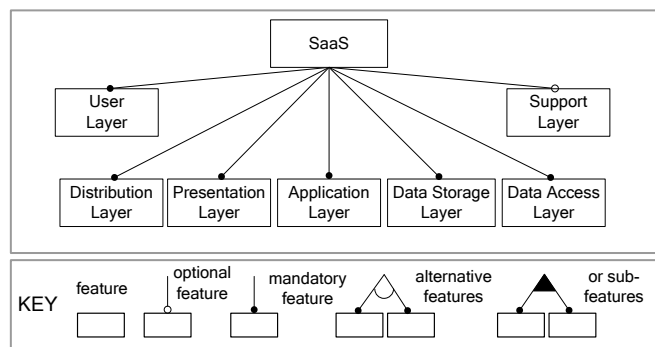


Figure 3. Top-Level Feature Model

B. User Layer

User layer is the displaying layer that renders the output to the end user and interacts with the user to gather input. This layer is the only part that the user can see. In principle the user layer might include a *Web Browser* or Rich Internet Application (RIA), or both of these (or features). RIA is especially used on mobile platforms.

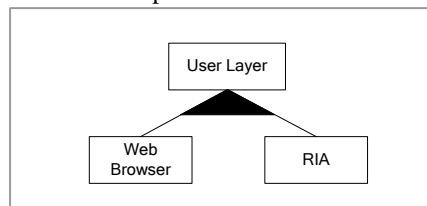


Figure 4. Feature Diagram for User Layer

C. Distribution Layer

Figure 5 shows the features for the distribution layer feature. This layer is the intermediate layer between the

internet and the SaaS application. The main concerns of the layer are scalability, availability and security. The mandatory features of this layer are load balancers and firewalls [11].

A firewall inspects the traffic and allows/denies packets. In addition to this, firewalls provide more features like intrusion detecting, virtual private network (VPN) and even virus checking. The distribution layer can have a single firewall or a firewall farm. A firewall farm is a group of connected firewalls that can control and balance the network traffic.

Load balancers divide the amount of workload across two or more computers to optimize resource utilization and increase response time. Load balancers are also capable of detecting the failure of servers and firewalls and repartitioning the traffic. Load balancers have the mandatory features of *Type* and *Strategy*, and an optional feature *Load Balancer.Firewall*. There are two types of load balancers, *hardware based* and *software based*. Load balancing strategies decide how to distribute requests to target devices. *Passive* load balancing strategies use already defined strategies regardless the run time conditions of the environment. Some of the most used passive strategies are *Round Robin*, *Failover*, *Random* and *Weighted Random*. *Dynamic* load balancing strategies are aware of information of the targets and likewise route the requests based on traffic patterns. Some of the most used passive strategies are *Fastest Response Time*, *Least Busy*, *Transfer Throughput*, *IP Sticky* and *Cookie Sticky*.

The optional *Load Balancer.Firewall* can be used as firewall by providing both packet filtering and stateful inspection. Using load balancer as a firewall can be an effective solution for security according to network traffic and cost requirements. This feature excludes the “*Distribution Layer.Firewall*” feature.

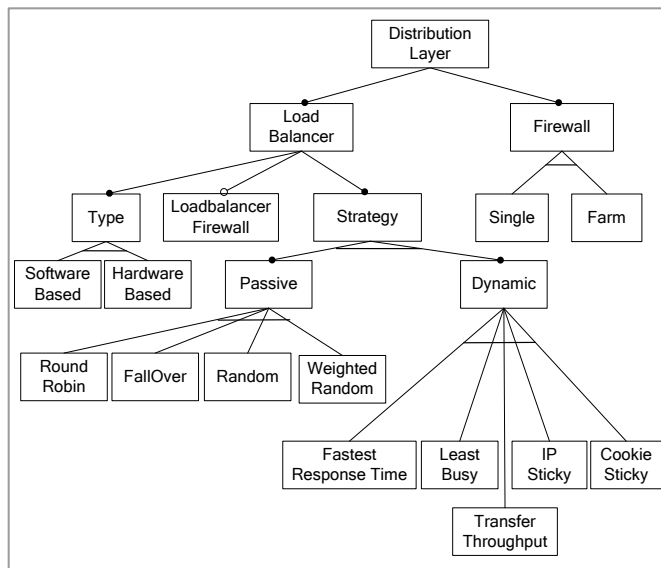


Figure 5. Feature Diagram for Distribution Layer

D. Presentation Layer

Figure 6 presents the presentation layer feature. The presentation layer consists of components that serve to present data to the end user. This layer provides processes that adapt the display and interaction for the client access. It

communicates with application layer and is used to present data to the user.

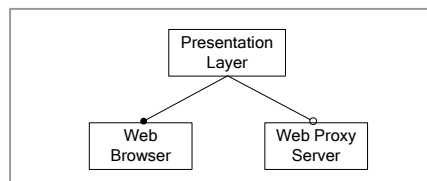


Figure 6. Feature Diagram for Presentation Layer

The presentation layer feature includes two subfeatures, the mandatory *Web Server* and optional *Web Proxy Server* features. A web server handles HTTP requests from clients. The response to this request is usually an HTML page over HTTP. Web servers deal with static content and delegate the dynamic content requests to other applications or redirect the requests. *Web Proxy Server* can be used to increase the performance of the web servers and presentation layer, caching web contents and reducing load is performed by web proxy servers. Web proxy servers can also be used for reformatting the presentation for special purposes as well for mobile platforms.

E. Application Layer

Figure 7 shows the feature diagram for Application Layer, which is the core layer of the SaaS architecture. Business logic and main functionalities, Identity Management, orchestration, service management, metadata management, communication, and integration are provided by this layer.

Especially in the enterprise area, SaaS platforms are usually built on SOA technologies and web services. *Application Server*, *Integration*, *Metadata Management*, *Identity Management* and *Communication* are mandatory features for the application layer. In case of using SOA, some other features – *ESB*, *Orchestration*, *Business Rules Engine*, are used in this layer. In the following subsections we describe these features in more detail.

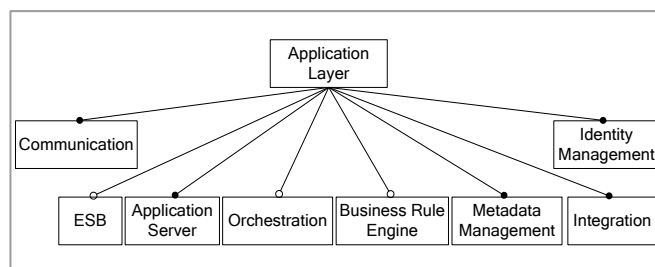


Figure 7. Feature Diagram for Application Layer

- **Application Server**

An application server is a server program that handles all application operations between users and an organization's backend business applications or databases. The application server's mission is to take care of the business logic in a multi-tier architecture. The business logic includes usually the functions that the software performs on the data. Application servers are assigned for specific tasks, defined by business needs. Its basic job is to retrieve, handle, process and present

data to the user interface, and process any input data whether queries or updates, including any validation and verification and security checks that need to be performed.

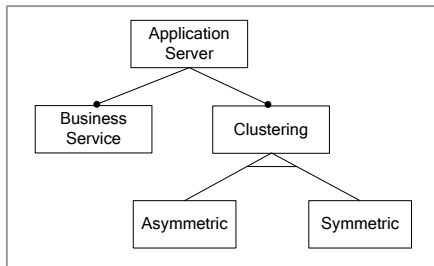


Figure 8. Feature Diagram for Application Server

SaaS applications have to have continuous uptime. Users around the world can access the application anytime. Application failure means customer and monetary loss. The application should be prevented from single point of failure. In addition to availability issues, there are performance and scalability capabilities to overcome for SaaS applications. By combining more than one computer and make it as a unified virtual resource can solve these problems. This technique is called server clustering. There are two techniques for server clustering: *asymmetric* and *symmetric*. In asymmetric clusters, a standby server exists to take control in case of another server gets of failure. In symmetric clusters, every server in the cluster do actual job. The first technique provides more available and fault tolerant system but the latter is more cost-effective.

- *ESB*

When we are talking about SaaS applications and service oriented architecture, the requirement is providing an infrastructure for services to communicate, interact, and transform messages. Enterprise Service Bus (ESB) is a platform for integrating services and provides enterprise messaging system. Using an ESB system does not mean implementing a service oriented architecture but they are highly related and ESB facilitates SOA.

- *Orchestration*

Orchestration is a critical mission in SOA environment. A lot of tasks should be organized to perform a process. Orchestration provides the management, coordination and arrangement of the services. BPEL is, for example, an orchestration language that defines business processes. Some simple tasks may be performed by ESB but more complex business processes could be defined by BPEL. To interpret and execute BPEL a BPEL engine is needed.

- *Metadata Management*

SaaS has a single instance, multi-tenant architecture. Sharing the same instance to many customers brings the problem of customization. In SaaS architecture, customization is done using metadata. Metadata is not only about customization (e.g. UI preferences), it is also intended to provide configuration of business logic to meet customers need. Updating, storing and fetching metadata is handled

through Metadata services. This feature requires *Metadata Repository* feature.

- *Business Rule Engine*

As mentioned before, SaaS applications can be customized and configured by metadata. Workflow may differ for each customer. Business Rules Engine is responsible of metadata execution. It consists of its own rule language, loads the rules and then performs the operations.

- *Integration*

The feature diagram for Integration is shown in Figure 9. In the context of SaaS, all the control, upgrade, and maintenance of user applications and data are handled by SaaS provides. An important challenge in SaaS is the data integration. SaaS applications usually need to use client data which resides at the client’s node. On the other hand, each client may use more than one SaaS application or on-premise application using the same data. The data may be shared among several applications and each application may use different part of it or in different formats. Manipulating the data will usually have an impact on the other applications. Data accuracy and consistency should be provided among those applications. Re-entering or duplicating the data for any application is not a feasible manner to provide data.

There are three different approaches for providing consistent data integration including: *common integration*, *specific integration* and *certified partner integration*. In the common integration approach services are provided for all clients. This feature requires “*Integration.Services.Web Services*” feature. In the specific integration, services are customized for each customer. This feature requires “*Integration.Services.Integration Services*” feature. Finally, in the *Certified Partner* approach the SaaS vendor delegates the integration to another vendor which is a specialist for SaaS integration. The SaaS vendor still needs to provide web services, but it leaves the control to other entities and focuses itself on the application. This feature also requires “*Integration.Services.Web Services*” feature.

The Integration feature describes either *Integration Service* or *Web Service*: In *Integration Service* approach, the SaaS vendor provides custom integration services for customers. Although this is the easiest way for customers, it is hard to manage adding integration service for different needs for vendors and increasing number of customers causes scalability problems. In the *Web Service* approach, the SaaS vendor provides a standard approach for customers as web services. The customers themselves take responsibility for SaaS integration. Compared to the Integration Service approach, customers have to do much more and need extensive experience. On the other hand this is a more scalable solution for vendors.

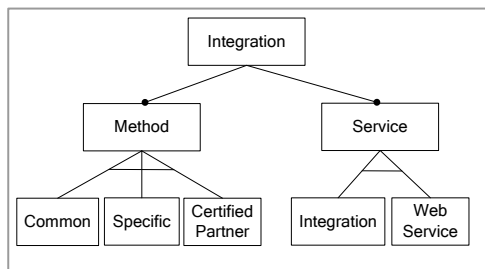


Figure 9. Feature Diagram for Integration

• *Identity Management*

Figure 10 represents the feature model for Identity Management, which deals with identifying individuals in a system and controlling access to the resources in the system by placing restrictions on the established identities of the individuals [7]. The *Directory Management* is responsible for managing the identities.

Identity Management includes two mandatory features *Identity Model* and *Directory Management*. Identity Model can be *Single Sign-On*, *Isolated* or *Federated*. *Isolated Identity Management*: The most common and simplest identity management model is the isolated one. Hereby, each service provider associates an identity for each customer. Despite its simplicity, this model is less manageable in case of the growth of number of users who should remember their login and passwords to their accounts for each service. *Single Sign-On* is a centralized identity management model, which allows users to access different systems using a single user ID and password.

Single Sign-On identity management model [5] can be *PKI-Based*, *SAML-Based*, *Token-Based*, *Credential Synchronization*, or *Secure Credential Caching*. SAML stands for Security Assertion Markup Language and defines the XML based security standard to enable portable identities and the assertion of these identities. The *Token-Based* approach can be either based on *Kerberos* or *Cookie*. The *Secure Credential Caching* can be on the *Server Side* or *Client Side*.

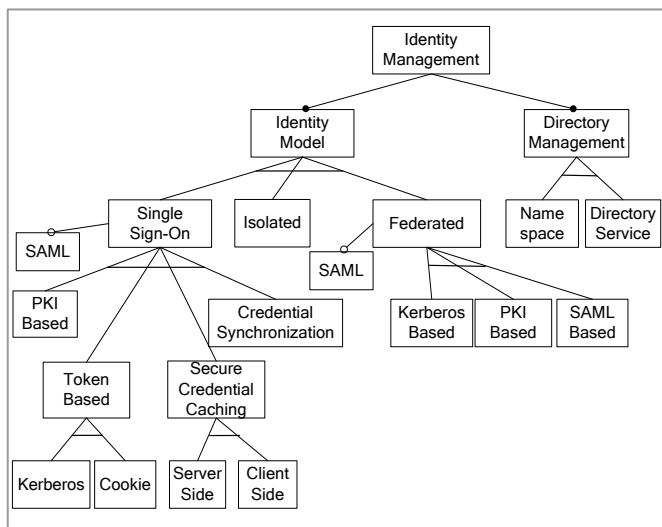


Figure 10. Feature Diagram for Identity Management

The *Federated Identity Model* is very close to *Single Sign-On*, but defined identity management across different organizations [6]. There are three most used approaches, *Kerberos-based Federation*, *PKI-based Federation* or *SAML-based Federation*. *Directory Management* feature includes two mandatory features, *Namespace* and *Directory Service*. *Namespace* maps the names of network resources to their corresponding network addresses. *Directory Service* represents the provided services for storing, organizing and providing access to the information in a directory (e.g. *LDAP*).

• *Communication*

Figure 11 shows the feature model for the *Communication* feature. SaaS vendor needs to provide a communication infrastructure both for inbound and outbound communication. Notification, acknowledging customers, sending feedbacks, demanding approvals are useful for satisfying users. The most common approach for communication is e-mailing. To transfer mails between computers a *Mail Transfer Agent (MTA)* can be used which requires *Simple Mail Transfer Protocol (SMTP)* protocol. Besides of mailing other protocols such as *Short Message Peer-to-Peer Protocol (SMPP)* and *Simple Network Paging Protocol (SNPP)* can be used.

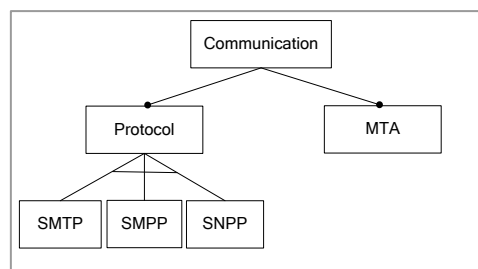


Figure 11. Feature Diagram for Communication

F. *Data Access Layer*

Figure 12 shows the feature diagram for *Data Access Layer*. This layer provides the database management system (DBMS) consisting of software which manages data (database manager or database engine), structured artifact (database) and metadata (schema, tables, constraints etc.).

One of the important, if not the most important, SaaS feature is multi-tenancy [2][12]. Multi tenancy is a design concept where a single instance of software is served to multiple consumers (tenants). This approach is cost saving, scalable, easy to administrate, because the vendor has to handle, update or upgrade and run only single instance. Multi-tenancy is not only about data, this design can be applied in all layers but the most important part of the multi tenancy is multi tenant data architecture. Based on the latter different kind of multi-tenancy can be identified. Multi-tenancy with *Separate Databases* means that each tenant has its own data set which is logically isolated from other tenants. The simplest way to data isolation is storing tenant data in separate database servers. This approach is best for scalability, high performance and security but requires high cost for maintenance and availability. In the *Shared Database, Separate Schemas* approach, a single database server is used for all tenants. This approach is more cost effective but the main disadvantage is

restore is difficult to achieve. Finally, the *Shared Database, Shared Schema* approach involves using one database and one schema for each tenants' data. The tables have additional columns, tenant identifier column, to distinguish the tenants. This approach has the lowest hardware and backup costs.

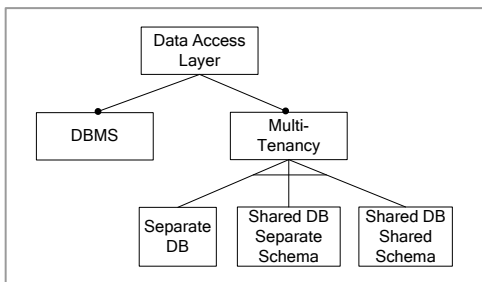


Figure 12. Feature Diagram for Data Access Layer

G. Data Storage Layer

Figure 13 shows the feature diagram for *Data Storage Layer*. The layer includes the feature for Metadata storage, Application Database and Directory Service. Metadata files can be stored either in a database or in a file based repository. Application Database includes the sub-features of Storage Area Network (SAN), Clustering and Caching [2]. SAN is a dedicated storage network that is used to make storage devices accessible to servers so that the devices appear as locally attached to the operating system. SAN is based on fiber channel and moves the data between heterogeneous servers.

Clustering is interconnecting a group of computers to work together acting like a single database to create a fault-tolerant, high-performance, scalable solution that's a low-cost alternative to high-end servers. By caching, disk access and computation are reduced while the response time is decreased.

Directory Service stores data in a directory to let the directory service to lookup for identity management. This data is read more often than it is written and can be redundant if it helps performance. Directory schemas are defined as object classes, attributes, name bindings and namespaces.

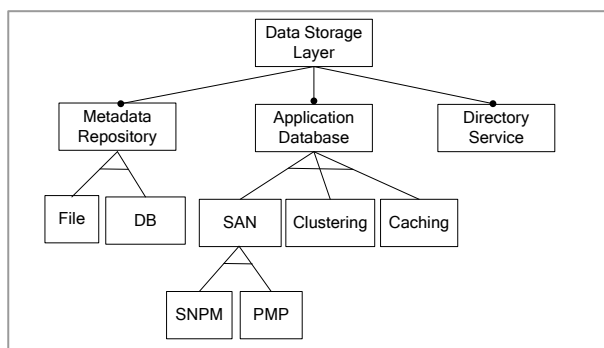


Figure 13. Feature Diagram for Data Storage Layer

H. Supporting Service Layer

Supporting Service Layer is a cross-cutting layer that provides services for all layers. The feature model is shown in Figure 14. As known, SaaS applications have quality attributes such as scalability, performance, availability and security. To keep the applications running efficiently and healthy, the SaaS

system needs to have monitoring system to measure metrics. The monitoring infrastructure can detect failures, bottlenecks, and threats and alert the administrators or trigger automatic operations. Furthermore, SaaS systems may be built on service oriented architecture and may need metering process for service level agreements and billing. A few examples for the metrics are CPU usage, CPU load, network traffic, memory usage, disk usage, attack rate, number of failures, mean time to respond etc.

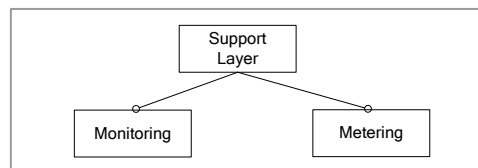


Figure 14. Feature Diagram for Support Layer

IV. EXAMPLE

Figure 15 shows an alternative application architecture design that is derived from the reference architecture shown in Figure 1. To derive this architecture based on the family feature model as discussed in the previous sections, the application feature model is defined. Typically in the application feature model multi-tenancy is selected using a single database management system with a shared database and shared schemas for the tenants.

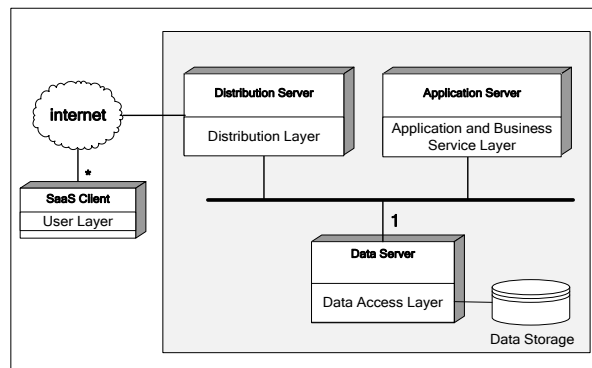


Figure 15. SaaS Application Architecture derived based on corresponding application feature model

V. RELATED WORK

Despite its relatively young history, different surveys have already been provided in the literature on cloud computing and many papers have been published on SaaS. An example survey paper is provided by Goyal and Dadizadeh [8]. However, to the best of our knowledge no systematic domain analysis approach has been carried out to derive a feature model for SaaS.

La and Kim [14] propose a systematic process for developing SaaS systems highlighting the importance of reuse. The authors first define the criteria for designing the process model and then provide the meta-model and commonality and variability model. The metamodel defines the key elements of SaaS. The variability model is primarily represented as a table. The work focuses more on the general approach. The metamodel could be complementary to the reference

architecture in this paper and as presented by SaaS providers. Although the goal seems similar, our approach appears to be more specific and targeting the definition of a proper modeling of the domain using feature modeling.

Godse and Mulik [9] define an approach for selecting SaaS products from multiple vendors. Since the selection of the feasible SaaS product involves the analysis involves analysis of various decision parameters the problem is stated as a multi-criteria decision-making (MCDM) problem. The authors adopt the Analytic Hierarchy Process (AHP) technique for prioritizing the product features and for scoring of the products. The criteria that are considered in the AHP decision process are *Functionality*, *Architecture*, *Usability*, *Vendor Reputation*, and *Cost*. Our work is also focused on selecting the right SaaS product but it considers the design of the SaaS architecture based on feature modeling. The selection process defines the selection of features and not products. However, in our approach we did not outline the motivation for selecting particular features. For this we might add additional criteria to guide the architect also in selecting the features. We consider this as part of our future work.

Nitu [16] indicates that despite the fact that SaaS application is usually developed with highly standardized software functionalities to serve as many clients as possible, there is still a continuous need of different clients to configure SaaS for their unique business needs. Because of this observation, SaaS vendors need take a well designed strategy to enable self serve configuration and customization by their customers without changing the SaaS application source code for any individual customer. The author explores the configuration and customization issues and challenges to SaaS vendors, and distinguishes between configuration and customization. Further a competency model and a methodology framework is proposed to help SaaS vendors to plan and evaluate their capabilities and strategies for service configuration and customization. The work of Nitu considers the configuration of the system after the system architecture has been developed. We consider our work complementary to this work. The approach that we have presented focuses on early customization of the architecture to meet the individual client requirements. The approach as presented by Nitu could be used in collaboration with our approach, i.e. by first customizing the architecture based on the potential clients and then providing configurability and customization support for the very unique business needs.

VI. CONCLUSION

Cloud computing and SaaS is a broad domain that is not easy to understand for novice designers. In this paper we have applied domain analysis techniques to derive a family feature model that represents both the common and variant features of SaaS architecture. Based on the family feature model a particular application feature model can be derived and the SaaS application architecture can be designed accordingly. As such, the family feature model helps both to enhance the understandability of SaaS and the generation of particular applications.

The feature model that we have derived is based on our selection of papers. We do not claim that this is the only

correct or eventual feature model. Enhancing the domain analysis study might refine the feature model that we have presented. Yet, the work should also be considered from an architecture design perspective. An important lesson from this paper is that feature modeling helps to support the architectural design of SaaS systems. In our future work we will develop the required tool support to represent the family feature model, define the link with architecture design decisions and generate application architecture.

VII. REFERENCES

- [1] S. A. Brandt, E. L. Miller, D. D. E. Long, L. Xue. Efficient Metadata Management in Large Distributed Storage Systems, 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies(MSST'03), pp. 290–298, 2003.
- [2] F. Chong and G. Carraro. Building Distributed applications: Multi-Tenant Data Architecture. MSDN architecture center, 2006.
- [3] F. Chong and G. Carraro. Architecture Strategies for Catching the Long Tail, Microsoft, MSDN architecture center, 2006.
- [4] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. Documenting Software Architectures: Views and Beyond. Second Edition. Addison-Wesley, 2010.
- [5] J. de Clercq, Single Sign-On Architectures, Proceedings of the International Conference on Infrastructure Security, p.40-58, October 01-03, 2002.
- [6] Cloud Computing. Wikipedia - [Online]. http://en.wikipedia.org/wiki/Cloud_computing
- [7] FIDIS, "Structured Overview on Prototypes and Concepts of Identity Management Systems", Future of Identity in the Information Society (No. 507512)
- [8] A. Goyal, S. Dadizadeh. A Survey on Cloud Computing, University of British Columbia, Technical Report, 2009.
- [9] M. Godse, S. Mulik. An Approach for Selecting Software-as-a-Service (SaaS) Product, in Proc.of. 2009 IEEE International Conference on Cloud Computing, 2009.
- [10] S. Joshi. Architecture for SaaS applications - using the Oracle SaaS Platform, Oracle White Paper, 2009.
- [11] C. Koppurapu, "Load Balancing Servers, Firewalls, and Caches", Wiley, 2002.
- [12] T. Kwok, T. Nguyen. A Software as a Service with Multi-tenancy Support for an Electronic Contract Management Application. In IEEE International Conference on Services Computing, 2008.
- [13] P.A. Laplante, Jia Zhang, Jeffrey Voas, "What's in a Name - Distinguishing between SaaS and SOA", IT Professional, Volume 10, Issue 3 (May 2008), Pages: 46-50, Year of Publication: 2008,
- [14] H. Jung La and Soo Dong Kim, A Systematic Process for Developing High Quality SaaS Cloud Services, in Proc. Proc. of the 1st International Conference on Cloud Computing, Springer LNCS, Volume 5931/2009, 278-289, 2009.
- [15] K. Lee , K. Chul Kang , J. Lee, Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools, p.62-77, April 15-19, 2002
- [16] H. Liao. Design of SaaS-Based Software Architecture, International Conference on New Trends in Information and Service Science, 2009.
- [17] Nitu. ISEC '09: Proceeding of the 2nd annual conference on India software engineering conference, , pp. 19-26, February 2009.
- [18] C. Spence, J. Devoys, S.Chahal. Architecting Software as a Service for the Enterprise IT@Intel White Paper, 2009.
- [19] Sun Cloud Computing Primer, <http://www.scribd.com/doc/54858960/Cloud-Computing-Primer>, accessed 2011.

Adding Support for Hardware Devices to Component Models for Embedded Systems

Luka Lednicki, Mario Žagar
 Faculty of Electrical Engineering and Computing
 University of Zagreb
 Croatia
 {luka.lednicki, mario.zagar}@fer.hr

Juraj Feljan, Jan Carlson
 Mälardalen Real-Time Research Centre
 Mälardalen University
 Sweden
 {juraj.feljan, jan.carlson}@mdh.se

Abstract—Component-based development promises many improvements in developing software for embedded systems, e.g., greater reuse of once written software, less error-prone development process, greater analyzability of systems and shorter time needed for overall development. One of the aspects commonly left out of component models is communication of software components with hardware devices such as sensors and actuators. As one of the main characteristics of embedded systems is the interaction with their environment through hardware devices, the effects of this interaction should be fully included in component models for embedded systems. In this paper we present a framework that enables inclusion of hardware devices in different phases of the component-based development process, including system design, deployment, analysis and code synthesis. Our framework provides a way for software components to explicitly state their dependencies on hardware devices, promotes reuse of software components with such dependencies and provides a basis for including hardware devices in analysis of component based embedded systems. We evaluate the feasibility of our approach by applying it to the ProCom component model.

Keywords – *Component-based Development, Embedded Systems, Hardware devices, platform modeling*

I. INTRODUCTION

Embedded systems are getting increasingly important in our daily lives, while at the same time getting more complex. Additionally, larger portions of functionality of embedded systems are being put into software, rather than hardware, which results in increased software complexity. Parallel with this trend there is a growing demand on software to be robust, reliable, flexible, adaptable, etc., while shorter time-to-market is desired. One of the approaches to tackle these issues is component-based software engineering (CBSE). CBSE promotes building systems from prefabricated software components, instead of coding from scratch, promising to lower time-to-market, manage complexity and produce software of higher quality. CBSE has proven to be successful in the domains of desktop- and Web applications and enterprise systems. However, embedded systems introduce some domain-specific issues (e.g., safety-criticality, real-time requirements, interaction with the environment), and to fully take advantage of the CBSE potential these must be addressed [1].

In this paper, we focus on enriching existing component models with support for proper handling of the interaction between a software system and its environment, the physical world that the system is embedded into. This interaction is done using *hardware devices*, such as sensors and actuators. The communication between software and hardware devices can be as simple as writing a value to a hardware pin or port, or as complex as invoking a service on a remote device. In all cases, this interaction with the environment implies that software components are dependent on the hardware or middleware used to communicate with the environment. As this affects reusability and analyzability of software components, failure to adequately express these dependencies can hinder the use of a component-based approach in the embedded system domain.

To address the problem of interaction between software components and hardware devices, we have investigated what is needed to properly integrate such devices into software component models for embedded systems, and devised a framework that allows us to describe hardware devices and hardware platforms that we can deploy software systems on, software components dependent on hardware devices. The framework also allows describing a mapping between hardware devices, hardware platforms and software components. Our approach has been developed in the context of ProCom component model [2], but is also applicable to other component models.

In Section II, we describe different ways in which hardware devices can impact the use of a component-based approach when developing software systems for the embedded domain. Section III provides an overview of how interaction of software components with hardware devices is managed in some of the existing component models. Our approach to inclusion of hardware devices in component models is presented in Section IV. Section V gives an example of how our approach can be used in developing software systems that interact with hardware devices, and Section VI concludes the paper.

II. EFFECTS OF HARDWARE DEVICES ON SOFTWARE COMPONENT MODELS

Dependencies of software components on hardware devices, as well as the communication between hardware and

software impact all phases of a component-based development process. In this section we discuss these impacts, in order to be able to address them accordingly. We consider a component-based development process suitable for developing embedded systems, and comprising the following phases: design, deployment, analysis and synthesis. The phases are not strictly sequential and can be iterative.

In the *design* phase, a developer specifies models of (i) the software layer of the system being developed, as a composition of components, and (ii) the hardware layer, as a composition of the hardware devices the system will be deployed on. The former requires a means to manage interaction with hardware devices in the software layer. The latter requires a means to describe the actual instances of hardware devices and how they are connected to a particular instance of a hardware platform.

In the *deployment* phase, a mapping between the software- and hardware layers is defined. In other words, the software components are allocated to the underlying hardware that will execute them. In this phase we must be able to explicitly identify the dependencies of software components on the hardware devices, in order to ensure that the hardware targeted for deployment satisfies these dependencies.

Embedded systems have particularities such as limited resources and real-time requirements, which increase the relevance of extra-functional properties compared to, for example, desktop- and Web applications. In order to guarantee constraints on extra-functional properties, extensive analysis has to be performed. During the *analysis* phase, effects of the hardware devices on the behavior of the software components must be taken into consideration.

During the *synthesis* phase executable code is generated based on the models specified in the design- and deployment phases. During the synthesis we must ensure that the code generated for software components reflects the specifics of the platform, with respect to communication with hardware devices.

As *reuse* is one of key concepts of CBSE, additionally we consider the effects hardware has on the ability to reuse components developed in different contexts. For successful reuse, we must ensure that components dependent on hardware can be deployed on different platforms.

With regards to the aforementioned concerns, the objectives of our work are to:

- provide means to describe hardware elements in a way that they can be integrated into component models for embedded systems;
- enable specification how software components depend on hardware devices, and description of communication between the two;
- allow inclusion of both functional and extra-functional properties of hardware devices and physical platform in analysis of component-based software systems;
- enable analysis of systems in early stages of development, before they are fully implemented; and
- promote reuse of both software components and hardware device descriptions.

III. BACKGROUND AND RELATED WORK

We have identified four different levels of support for hardware dependencies in a component-based context.

A. Outside of the Component Model

Many component models, especially those developed for research purposes, do not provide any method for including hardware devices in system design. All communication with the environment is performed at input and output at the top level of the system. In this approach, functionality must be modeled separately from hardware interaction. Therefore, functionality specifically developed to fit particular hardware is difficult to represent. Furthermore, propagating all hardware interaction to the top level can be particularly cumbersome in complex systems, where many nesting levels exist.

SaveCCM [3] is an example of such a component model. In SaveCCM software components are not allowed to directly communicate with hardware devices. Instead, communication with them takes place outside of the component model.

B. Code Level

Many component models do not provide ways to explicitly state dependencies on hardware devices. However, they allow to communicate with them in the code of software components through direct method calls to the underlying platform. This approach can severely limit reuse of software components, as components with such hard-coded communication with hardware cannot be used on multiple hardware platforms or when the configuration of the hardware platform is changed.

An example of such a component model is Rubus [4]. Rubus was created by Articus Systems for developing dependable real-time systems. Reuse is not the main focus of Rubus, rather it is to provide a higher abstraction layer and better basis for analysis. Thus platform and device dependent information are part of basic software components.

C. Using Specialized Entities

Some component models introduce new entities, separate from software components, which are used to interact with hardware devices. With a way to explicitly describe dependencies and communication with hardware devices, and a clear separation of hardware and software components we can easily reuse parts of systems or include hardware devices in analysis of systems. A drawback of this approach is that it hinders the possibility of hierarchical component composition. As components cannot specify their interaction with hardware devices through their interface, we cannot reuse composite components that contain hardware entities.

A component model that uses this approach is COMDES-II [5]. COMDES-II provides a two-layered component model. The upper layer is defined by active software components named *actors*. The lower layer is used to define the behavior of actors using *function block instances*. Actors interact with hardware devices using entities called *input* and *output signal drives*. Drives can be used to communicate over a network (*communication drivers*) or to sense or actuate physical signals (*physical drivers*).

AUTOSAR [6], also provides similar level of support. AUTOSAR is a component-based architecture created by a

partnership of a number of automotive manufacturers and suppliers. Dependencies on hardware devices are encapsulated in *sensor* and *actuator* software components. These components provide a special interface for managing their interaction with hardware devices. They are dependent on specific sensor or actuator hardware devices. However, AUTOSAR does not provide means for hierarchical composition of components. As it does not provide support to state hardware dependencies for all component types we still argue that sensor and actuator components act as specialized entities.

D. Explicitly Encapsulated in Software Components

Component models can also encapsulate communication with hardware devices in software components, but expose it through the component's interfaces. Compared to approaches that use specialized entities for interaction with hardware devices, this approach enables us to organize components dependent on hardware devices in multiple levels of hierarchy.

Our approach also falls into this category since it provides an explicit way to define how software components are connected to hardware devices. For this we do not use specialized entities, but instead extend the definition of standard software components. This lets us reuse all parts of component model framework and tools while including hardware devices in software component and system definition.

IV. OVERVIEW OF OUR APPROACH

Led by the objectives described in Section II, we have devised a framework that allows us to include hardware devices in component models, and applied it to the ProCom component model.

The ability to reuse components or complete systems is one of the main goals of CBSE. Having components that are dependent on a particular instance of hardware device, or how this device is connected to the platform, can severely limit possibility of their reuse. For this reason we have separated our framework in three layers: *software layer*, *hardware layer* and *mapping layer*. With this separation we are able to independently describe software system and hardware platform, making them suitable for reuse in different scenarios. We can then connect these two layers through the mapping layer when developing a complete system. An overview of how these three layers are connected is given in Figure 1.

In our approach we have a clear distinction between *types* and *instances* for both hardware and software entities. Types are entity definitions that are context-independent. They can be easily reused in different settings or stored to repositories for future use. Once we want to use an entity in a concrete system, we are in fact creating an instance of that entity type. Instances are not copies of the entity, but a representative of the general entity in a specific context. For example, when we are describing a hardware device, we are actually describing a device type. Once we want to use the device in a system we need to create a new instance of that device type. Instances can also refine properties of an entity depending on the usage context.

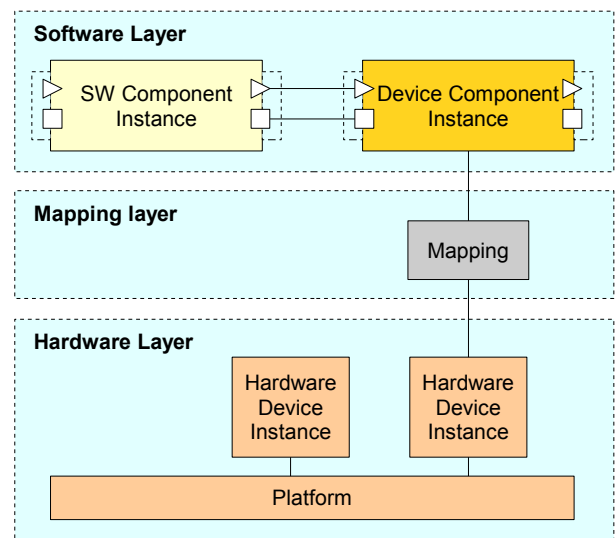


Figure 1: Overview of three layers of our approach relate to each other.

As we handle hardware devices using extended software components, and not specialized entities, we are able to reuse many solutions that already exist in ProCom component model. For the purpose of defining attributes for hardware components we leverage Attribute Framework [7], which allows us to define extra-functional properties for architectural elements of the component model. Also, integration with ProCom allows us to use ProCom Analysis Framework with different types of analysis, such as parametric worst-case execution time analysis [8], model checking of behavioral models [9] and fault-propagation.

A detailed metamodel that describes our approach is given in Figure 2. Next, each of the three layers will be described in more detail.

A. Software Component Layer

To enable interaction of component-based applications with hardware devices we have introduced a new type of component named *device component*. This entity is derived from ordinary software components. Its purpose is to encapsulate dependencies of component-based software system on hardware devices and enable communication with these devices.

When looking at a device component as a black-box, it has the same interface and semantics as all software components. The difference between normal software components and device components is in their internals: device components do not provide the ability for the developer to explicitly specify their realization. This is because they inherit their realization from hardware devices (described in Section IV.B.2)) once the two are mapped together.

Device components are only used to express the existence of dependencies on hardware devices, but not the specifics of a device, i.e., how it is connected to the platform or the code for actual communication with hardware. A device component has exactly one hardware dependency. In case of composite components, its device dependencies must match the combined dependencies of its subcomponents. This way the software

layer stays hardware- and platform-independent. Any system or composite component that contains device components can still easily be reused in a new system or on different platforms.

B. Hardware Layer

The hardware layer allows us to describe *physical nodes* (i.e., processing unit such as microcontrollers or ECUs that runnable code can be deployed to), *hardware devices* such as sensors and actuators and *platforms* which consist of instances of physical nodes and hardware devices and to which we can deploy software systems.

We have designed the hardware layer based on research of what is needed to promote the ability of reuse of software components. However, we also wanted to provide the ability to reuse structures defined in hardware platform. For this purpose we have divided hardware into three separate parts which can be developed independently to each other: *physical node specification*, *hardware device specification* and *platform instantiation*.

1) Physical Node Specification

In our model, physical nodes describe different processing units such as microcontrollers or ECUs. They are reusable as they only describe a type of unit and do not contain any information about how they are used or configured in a particular system.

Physical nodes define a list of inputs and outputs they provide. Inputs and outputs are defined by their type, e.g., one-bit digital I/O, serial communication port, analogue input, etc. Also, for each input or output we define actual program code that will be used for its initialization and data transfer.

Physical nodes can also be characterized by extra-functional properties such as their processing power, available memory, behavioral models, execution times for input or output functions and other similar attributes.

2) Hardware Device Specification

Hardware devices are peripherals such as sensors and actuators that are connected to physical nodes in order to interact with the environment. Each hardware device represents a specific, real-world sensor or actuator.

Each hardware device references a device component for which the device can be used as realization. It should be noted that one device component can be referenced by many different hardware devices. For example, a temperature sensor device component can be referenced by two different implementations of (i.e., hardware devices) temperature sensor. However, a device component (in the software layer) is not dependent on any of these implementations.

Similarly to a list of inputs and outputs provided by physical nodes, hardware devices define a list of inputs and outputs that they require for communicating with them.

A part of hardware device specification is the code for communication with the device. This code is merged with software component code during the synthesis phase of the development process, leaving software components free of hardware-specific code. In that way software components can be reused on different hardware configurations. However, this code leaves out actual function calls needed for communication, which is defined in the physical node specification. This allows us to reuse the same code regardless of which input or output of a physical node the device is connected to, or use it on different physical nodes.

Similar to physical nodes, we can also define attributes that describe extra-functional properties of hardware devices.

3) Platform Instantiation

We have defined platform as a collection of physical node instances on which we can deploy software systems. Except creation of physical node instances, platform instantiation also encompasses creation of hardware device instances and connections of these instances to instances of physical nodes. It should be noted that we do not use type-instance paradigm

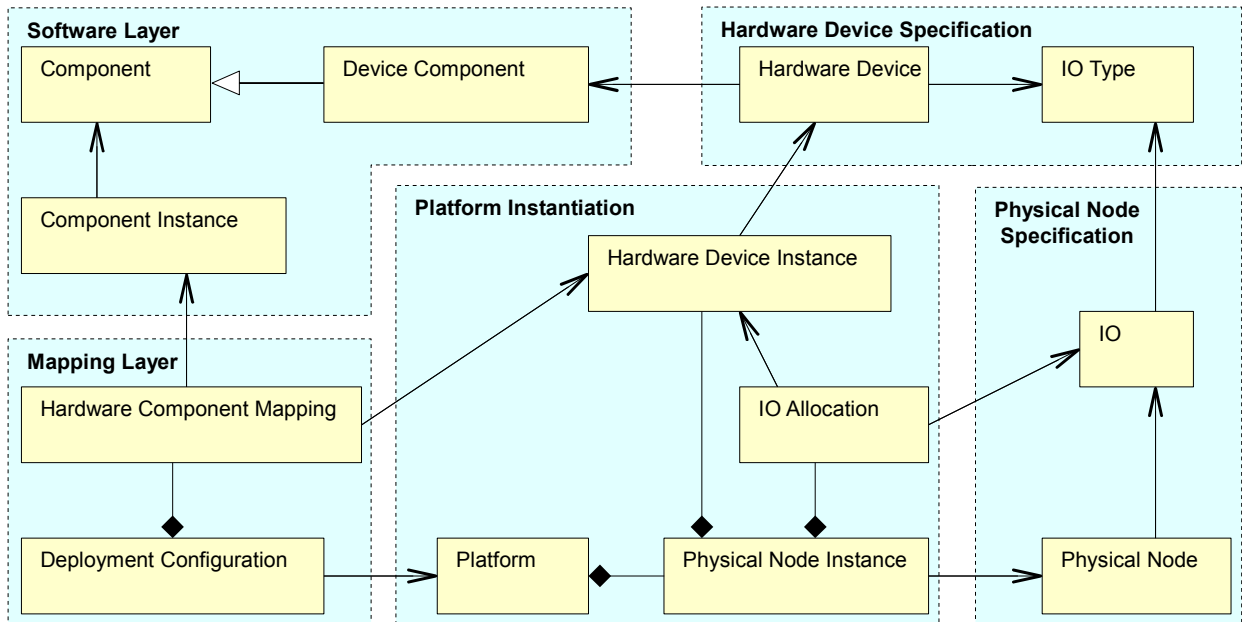


Figure 2: Metamodel that contains all entities we use to add support for hardware devices in software component models.

for platform. We assume that platforms will be collections of reusable physical nodes, and will be specific for every system, there will be no need for their reuse.

Connections between hardware devices instances and physical node instances are implicit: device instances are contained by physical node instances. Allocation of hardware device instances to inputs or outputs of physical node instances is done through *IO Allocation*. Once the allocation of inputs and outputs is defined, we can also validate a platform by checking if requirements of all hardware device instances are fulfilled by inputs and outputs of physical node instances they are connected to.

C. Mapping Layer

As already stated, we have defined software and hardware layers to be as distinct as possible in order to promote reuse of structures defined in them. In order to create systems consisting of both, we had to introduce the mapping layer. The mapping layer allows us to define connections between device component instances in software layer and hardware device instances in hardware layer. By this we put our reusable units in the context of a system and are able to provide platform-specific code for platform independent, reusable software components.

Mapping between the two can be created only if type of hardware device instance references type of device component instance. By having this constraint we can easily assure that a system is deployed (i.e., component instances are allocated to physical node instances) in a valid way.

Besides the platform-specific code, the mapping also allows us to propagate platform- or device-specific values for extra-functional properties.

Our approach supports mapping of component instances to hardware device instances even in early stages of system development. By having reusable descriptions, models and extra-functional properties defined for hardware devices and physical nodes we are able to test and analyze behavior of a system before it is fully implemented. This allows us to detect potential problems and avoid changes in late stages of system development.

Another benefit of separate mapping model is that it allows a more flexible process, where software and hardware can be addressed separately in any order, and interleaved. Also, it enables us to provide partial mappings in early stages of development.

V. EXAMPLE

To illustrate use of our approach, we will demonstrate it on an example. The example will model a simple temperature control system using ProCom component model.

A. The ProCom component model

ProCom is a component model for distributed embedded systems in the vehicular and automation domains. These systems often have a safety-critical role and have to perform in real-time. Therefore, ProCom explicitly addresses extra-functional properties such as timing (e.g., worst case execution time) and resource usage (e.g., static memory, CPU). ProCom follows a model-based methodology centered around a rich

notion of reusable architectural design-time components. A ProCom component can consist of source code, models of timing and resource usage, analysis results and documentation.

The external view of a component consists of ports and attributes. Through the ports the functionality provided by a component can be accessed, while the attributes represent additional information about a component, such as extra-functional properties.

In order to be able to design both the complete system and the low level control functionality, ProCom has been divided into two layers. The upper layer, called ProSys, models a system as a collection of complex, active, concurrent, and typically distributed subsystems that communicate via asynchronous message passing. The lower layer, ProSave, on the other hand models smaller parts of control functionality. ProSave components communicate through trigger (control flow) and data ports (data flow).

B. Temperature Control System

Our example temperature control system consists of two temperature sensors that monitor temperature in a water tank and a heater that will engage if the temperature drops below a defined temperature. A graphical representation of all software and hardware layers of the system, and the mapping between the two layers, is given in Figure 3.

Our software layer consists of a clock (an element that creates periodical triggering signals), two instances of TemperatureSensor device component (TS1 and TS2), one instance of ControlUnit software component (CU1) and one instance of HeaterActuator device component (HA1). The component instances are connected in such a way that the clock triggers both TS1 and TS2. When both of them have finished their execution they forward temperature values to CU1 and generate signals that trigger its execution. Depending on given temperature values, CU1 performs calculations and provides signals to HA1 to be turned on or off.

It should be noted that TS1, TS2 and HA1 just serve just for describing interaction of software components with hardware devices, but are not device-specific. In that way whole software layer is reusable on different hardware platform configurations.

In the hardware layer we need to include specifications of physical nodes and hardware devices, and instantiate our platform. For the purpose of this example we will not fully specify the hardware but will only use parts that satisfy the needs of our system. Physical node specification will consist only of one physical node which we will call MicroCtrl. MicroCtrl will provide three IOs: two analog and one digital. For temperature sensors we use hardware devices that require analog input. We also specify heater hardware device which requires digital output. To instantiate our platform, we will create an instance of MicroCtrl with name Micro1. Micro1 will have two instances of the analog temperature sensor device (AT1 and AT2) and one instance of the heater device (H1). We will allocate the instances of temperature sensor to the analog inputs and the instance of heater device to the digital output of Micro1.

To complete our system, we need to define mappings between device components in software layer and hardware

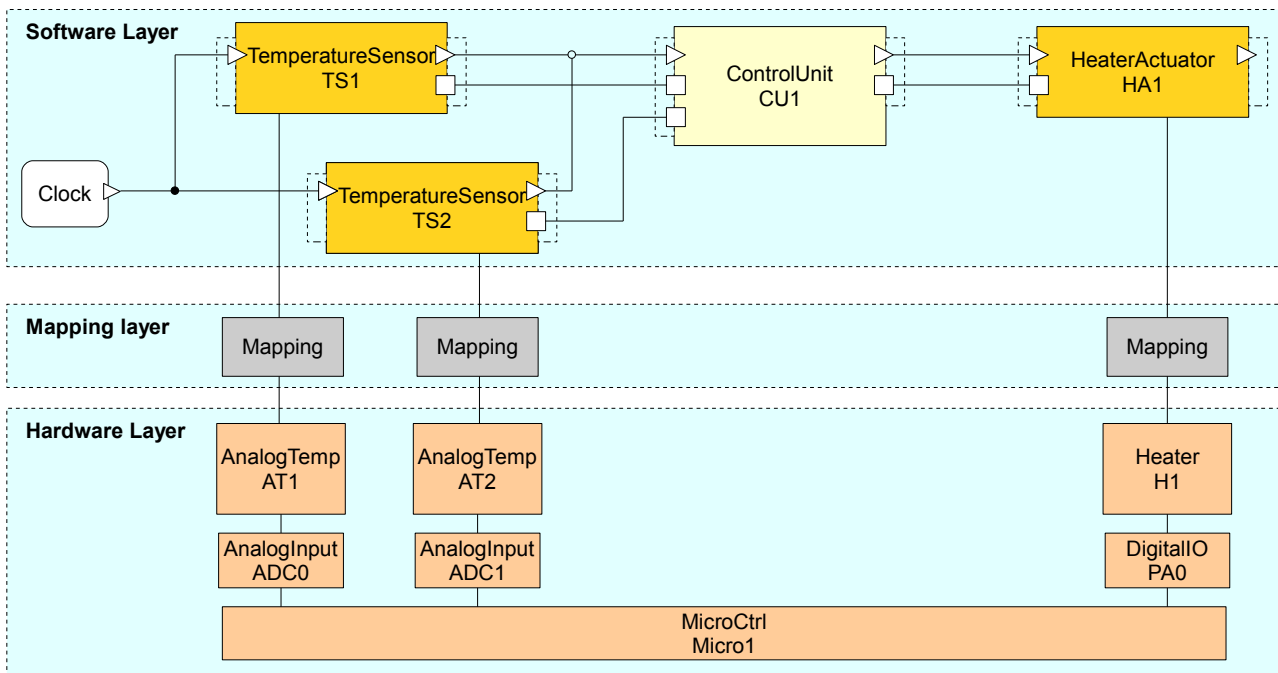


Figure 3: Example of a temperature control system created using ProCom extended with our approach.

devices in hardware layer. For this, we will define mappings between AT1 and TS1, AT2 and TS2, and HA1 and H1.

VI. CONCLUSION

In this paper, we have presented our approach for managing hardware devices such as sensors and actuators in component models for embedded systems. Our framework consists of three layers: software layer, hardware layer and mapping layer. These three layers enable separation of device dependencies in software and models of the actual hardware and allows us to reuse software components and hardware models. The hardware layer enables us to specify all aspects of hardware devices and platforms needed for their integration into component models. In the software layer we enable explicit definition of dependencies of software components on hardware devices. The mapping layer enables us to connect instances of software components to hardware device instances and in that way to design complete systems including software and hardware. The mapping also allows propagation of extra-functional properties of hardware devices to component model. In early stages of system development we can also define just partial mappings. Our approach promotes reuse of software components, hardware device specifications and platform node specification by creating clear distinction between types and instances of these entities, and by removing platform- and device-specific code out of software components.

ACKNOWLEDGMENT

This work was supported by the Unity Through Knowledge Fund via the DICES project, the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS, and the Swedish Research Council project CONTESSE (2010-4276).

REFERENCES

- [1] I. Crnković and M. Larsson, *Building Reliable Component-Based Software Systems*, Artech House Publishers, 2002.
- [2] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic. A component model for control-intensive distributed embedded systems. In *11th International Symposium on Component Based Software Engineering*. Springer Berlin, October 2008., pp. 310-317
- [3] M. Åkerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, and M. Tivoli. The SAVE approach to component-based development of vehicular systems. *Journal of Systems and Software*, May 2007. pp. 655-667
- [4] K. Hänninen, J. Mäki-Turja, M. Nolin, M. Lindberg, J. Lundbäck, and K-Lennart Lundbäck, *The Rubus Component Model for Resource Constrained Real-Time Systems*, 3rd IEEE International Symposium on Industrial Embedded Systems, 2008, pp. 177-183
- [5] K. Xu, S. Krzysztof, and A. Christo, *COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems*, RTCSA '07: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2007, pp. 199-208
- [6] H. Heinecke, W. Damm, B. Josko., A. Metzner, H. Kopetz, A. Sangiovanni-Vincentelli., and M. Di Natale, *Software Components for Reliable Automotive Systems*, Design, Automation and Test in Europe, 2008, pp. 549-554
- [7] S. Sentilles, P. Stepan, J. Carlson, and I. Crnković, *Integration of Extra-Functional Properties in Component Models*, 12th International Symposium on Component Based Software Engineering (CBSE 2009), LNCS 5582, Springer Berlin, East Stroudsburg University, Pennsylvania, USE, June, 2009, pp. 173-190
- [8] T. Leveque, E. Borde, A. Marref, and J. Carlson, *Hierarchical Composition of Parametric WCET in a Component Based Approach*, In *14th IEEE Int. Symposium on Object/Component/Service-oriented Real-time Distributed Computing*, 2011, pp.261-268
- [9] D. Ivanov, M. Orlic, C. Seceleanu, and A. Vulgarakis, *REMES tool-chain – A set of integrated tools for behavioral modeling and analysis of embedded systems*. In *25th IEEE/ACM International Conference on Automated Software Engineering*, 2010, pp. 361-362

A Service Component Framework for Multi-User Scenario Management in Ubiquitous Environments

Matthieu Faure^{*,†}, Luc Fabresse[†], Marianne Huchard[‡], Christelle Urtado^{*}, and Sylvain Vauttier^{*}

^{*}LGI2P / Ecole des Mines d'Alès, Nîmes, France

{Matthieu.Faure, Christelle.Urtado, Sylvain.Vauttier}@mines-ales.fr

[†]Ecole des Mines de Douai, Douai, France

Luc.Fabresse@mines-douai.fr

[‡]LIRMM - UMR 5506, CNRS and Univ. Montpellier 2, Montpellier, France

huchard@lirmm.fr

Abstract—Software dedicated to ubiquitous environments has to deal with the multiplicity of devices and users. It also has to adapt to frequent changes in its environment. Users can easily access and trigger services provided by different devices but also need to implement complex scenarios, *i.e.*, structured compositions of multiple service. State-of-the-art frameworks do not fully meet the expectation we identified. This is why, we designed the SaS (Scenarios As Services) ubiquitous software: a platform for ubiquitous systems that provides a SDL (Scenario Description Language) to support the creation of tailored user-centric scenarios. Our previous work on the subject did not tackle all distribution and concurrency concerns. In this paper, we present SaS's new features. Using the improved SDL, a user can now describe scenarios that combine services even if all of them are not currently available and will never be at the same time. Moreover, different scenario sharing mechanisms coupled with an access right policy are now included in SaS. SaS is currently implemented in a prototype on top of OSGi.

Keywords-Ubiquitous environment; service-oriented computing; user-centric; service composition; scenario creation.

I. INTRODUCTION

More and more electronic devices (such as smartphones, tablet PCs, etc.) assist us in our daily life. They can interact with their environment and propose various functionalities to users. This is the rise of ubiquitous computing [1][2]. These functionalities can be handled as services, and thus, Service-Oriented Computing (SOC) [3] is a suitable paradigm to design software for ubiquitous environments. Service access and system adaptability to environmental changes are already well handled by execution frameworks. However, to our knowledge, these systems fail to meet user expectations to express their needs as complex scenarios involving multiple services. Based on this observation, we designed the SaS (Scenarios as Services) ubiquitous software [4]. SaS features a service component framework that enables end-users to easily define, control and share scenarios. SaS also proposes an SDL to create scenarios as service compositions.

Besides, ubiquitous environments involve multiple users and devices. Consequently, handling previously unknown

device types, sharing information among users and handling control device mobility are challenging issues. First, device types must not be hardwired in the system. It has to be possible to create scenarios with services from specific devices but also from any device of a given type. This capability makes the system more flexible to device change. Second, an access right policy and a process dedicated to sharing scenarios must be specified. Thirdly, handling control device mobility can be seen both as a constraint on the system (that must dynamically adapt to its changing environment) but also as a chance (as the system can benefit from mobility, while executing scenarios that involve services that never coexist in a same environment).

The SaS system is twofold. It divides into a scenario description language called SaS-SDL that provides simple means to describe services, scenarios, environments and an execution framework called SaS platform that provides the processes to support the behavior of the ubiquitous software. In this paper, we focus on SaS's new features. The improved SaS-SDL now manages the environment. In addition, SaS handles scenario sharing among selected users, service memorization for future scenario creation and scenario mobility (execution distributed in multiple places and times).

This paper is further organized as follows. Section II introduces service and scenario declaration in SaS-SDL. Section III presents the new feature of SaS-SDL: context management. Then, Section IV describes how the SaS system executes distributed scenarios. Section V is dedicated to the design of our prototype implementation. Related works are discussed in Section VI. Finally, Section VII concludes this paper and draws perspectives.

II. SERVICE AND SCENARIO DECLARATION WITH SAS-SDL

In this section, we give an overview of service and scenario declaration (a previous version was presented in [4]) using SaS-SDL, the proposed scenario description language. SaS-SDL enables end-users to create scenarios

that correspond to their needs. Improvements are specifically introduced here, such as: multiple operation selection schemes (from a specific device/service or not), the ability to define and set scenario parameters, the ability to specify the execution type (either in sequence or in parallel) of an action list. Compared to other programming languages for service composition (like BPEL [5]), which are imperative and designed for executable processes, our SDL is a high level language, which is declarative and dedicated to end-users. With this SDL, SaS automatically declares services after they are discovered and then, users can declare scenarios.

A. Service declaration

To be interoperable, SaS does not restrict to a protocol but uses a generic pivot mode to declare services. SaS can be specialized by adding bridges to different protocols (as Frascati [6] and EnTiMid [7] already does). SaS-SDL defines a service by a device (its provider), a name and an operation list. Operations have a return type and can have typed parameters. Users can choose if a service operation to compose comes from a specific device and a particular service or not. To do so, the new version of SaS-SDL features the special word *any*, which enables to elude the provider device or the service name. Only the main elements of the grammar are presented in Listing 1.

```
<service> ::= service <device> <service_name> <op_list>
<op_list> ::= ( <operation> ; )+
<operation> ::= operation <operation_name>([<param_list>])
                : <return_type>
<param_list> ::= <parameter_type> (, <parameter_type>)*
<return_type> ::= <type>
<parameter_type> ::= <type>
<device> ::= identifier | any
<service_name> ::= identifier | any
```

Listing 1. Service declaration with the Backus-Naur Form (BNF)

Listing 2 is a *Clock* service declaration example.

```
service clock_Bedroom Clock
operation getTime() : Time;
operation setTime(Time) : void;
```

Listing 2. Service declaration example

B. Scenario declaration

A scenario has a name, some actions and properties. An action can be: (i) an operation invocation, (ii) an alternative (*if - else*), or (iii) a repetition loop.

Listing 3 describes the main elements of a scenario declaration using the BNF notation. With this improved version of SaS-SDL, scenarios have properties, which enable to specify if the scenario is exportable, editable, etc. Moreover, action lists are now executed in sequence by default, however, SaS-SDL enables users to specify some actions to execute in parallel. In addition, users can now leave some parameter values blank at scenario creation. This is represented by the ? value in SaS-SDL. Such eluded parameters become

scenario parameters and must be valued by users every time the scenario is invoked.

```
<scenario> ::= scenario <scenario_name> <action_block>
                [<scenario_properties>]

<action_block> ::= { ( <action> )+ } |
                { ( [ <parallel_exec> ] <action_list> <action_list> ) }
<action_list> ::= ( <action> | <action_block> )+

<action> ::= <op_invocation> ; | <alternative> | <repeat>

<op_invocation> ::= ( <device> ) <service_name> .
                    <operation_name>([<parameter_list>])
<parameter_list> ::= ( <op_invocation> | <parameter_value> )
                    (, ( <op_invocation> | <parameter_value> ) )*
<alternative> ::= if <cplx_condition> <action_block>
                    [<else_clause>]
<else_clause> ::= else <action_block>
<cplx_condition> ::= ( <condition>
                    ( <log_operator> <condition> ) * )
<condition> ::= <op_invocation> <comp_operator>
                ( <op_invocation> | <value> )
<repeat> ::= (while<cplx_condition> | <repeat_value> times)
            <action_block>

<parameter_value> ::= <value> | ?
<parallel_exec> ::= parallel:
<log_operator> ::= and|or|not
<comp_operator> ::= < | <= | > | >= | ==
```

Listing 3. Grammar of the scenario declaration using the BNF notation

Listing 4 illustrates SaS-SDL with a scenario example.

```
scenario night
if ( (any) Clock.getTime() == 6pm and
      (BedroomThermomether) Thermometer.getTemperature() <= 17)
{
  (BedroomRadiator) Heater.setValue(7);
}
```

Listing 4. Scenario declaration example

C. Users point of view

SaS integrates a GUI based on our SaS-SDL to facilitate scenario creation for end-users.

1) *Service selection*: Our GUI presents ordered services in three columns: by device, service and operation. To avoid duplicates, SaS groups services and operations with same name. When users select a device (*resp.* a service), services (*resp.* operations) attached are filtered. It enables users to select a service (*resp.* operation) from a specific device (*resp.* service). In addition, SaS indicates if a service is a scenario.

For users to create conditions on service availability and define alternatives, SaS adds the operation *isPresent* to each service.

2) *Scenario creation*: When users select a service operation to compose, SaS displays corresponding informations (provider device, service name, operation name and result type) and enables users to enter operation parameters. Users can either provide a fixed value or select another operation result (on which they can apply a basic operation such as +, -, *, /). In case the parameter type is complex, SaS only allows users to select an operation result. Figure 1 represents the GUI *sendMail* service operation, with two parameters (second one is complex).

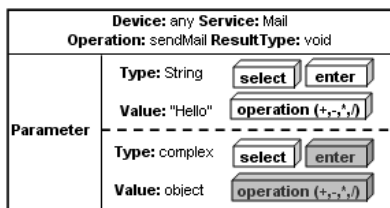


Figure 1. GUI: service operation

SaS provides users with several templates (i.e. *alternative*, *while*, *repeat*, etc.) to create scenarios. Users can combine templates to create the scenario skeleton. Then, users just need to put service operations inside the templates and complete with basic instructions (*and*, *or*, *not*, *<*, *>*, *≤*, *≥*, *==*). Figure 2 illustrates a scenario template and Figure 3 shows an example of scenario creation (scenario operations are represented by pictograms to simplify but they actually are similar to those in Figure 1).

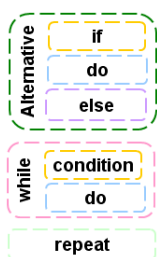


Figure 2. GUI: scenario templates

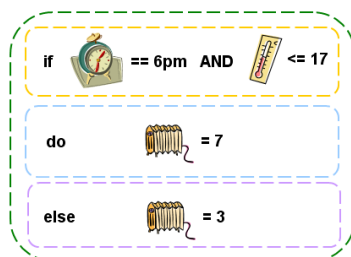


Figure 3. GUI: example of scenario creation

III. CONTEXT MANAGEMENT IN SAS

As seen, our previous version of SaS enables users to create scenarios. To do so, they dispose of SaS-SDL and its graphical representation. We presented in Section II some improvements for service and scenario declaration. Nevertheless, ubiquitous computing implies users mobility and multiplicity. As defined in [8], two characteristics of ubiquitous system are the *social environment* and the *evolving environment*. A ubiquitous system should therefore provide an access right policy and advanced sharing mechanisms. Moreover, this system has to be adaptive but could also benefit from this changing environment.

A. SaS in ubiquitous environment

Ubiquitous environments involve electronic devices. We define two types of devices: simple devices (such as radiator, light) and control devices (such as laptop, pda) which have an advanced user interface (i.e., touch screen), and can be considered as personal and mobile. A SaS container (which contains all SaS mechanisms handled by SaS ubiquitous software) can therefore only be deployed on a control device to constitute a SaS *system*.

B. Service and System Directories

Every SaS system has a unique identifier. As a SaS system is associated to a unique user, sharing scenarios with select SaS systems is equivalent to define access rights. SaS systems (which might not be always available locally) are permanently indexed into a *system directory*. It makes possible to share scenarios with a system even if it is temporary unavailable (failure, mobility). Such a permanent index is also provided for services by the *service directory*. Users can registers services that they discovered or obtain service declarations from a scenario created by someone else. By this means, scenarios can be defined that include temporarily missing services.

To ease directory browsing, services and systems can be grouped into named categories. These categories are like keywords as a service (resp. a system) can be included into several distinct categories. Browsing by categories diminishes the amount of information to be presented to users. They can also be used to collectively export services (which can be equivalent to providing grouped access rights), see Section IV-B1 for details. Examples of categories might be locations (all services available at *home*) or users (all systems owned by *kids*).

Listing 5 represents the main elements of the grammar for context management and Listing 6 illustrates how this part of SaS-SDL can be used. Scenarios in the service directory are highlighted to be differentiate from basic services.

```

<sas_system> ::= system<system_id><system_dir><service_dir>

<system_dir> ::= system_directory { (<system_cat>)* }
<system_cat> ::= category <cat_name> [<system_list>]
<system_list> ::= ( system <system_id> )*

<service_dir> ::= service_directory { (<service_cat>)* }
<service_cat> ::= category <cat_name> [<service_list>]
<service_list> ::= [ services <service_name>
                    (, <service_name>)* ]
    
```

Listing 5. Context Management with SaS-SDL

```

system pda12
  system_directory {
    category mySystems
      platform Nokia3310
      platform Acer TimelineX
    category family
      platform macintosh
  }

  service_directory {
    category home
      [services TV, wakeUp]
    category office
      [services fax, print]
  }
    
```

Listing 6. Service and system directories

The class diagram of Figure 4 provides an alternative compact view of SaS-SDL.

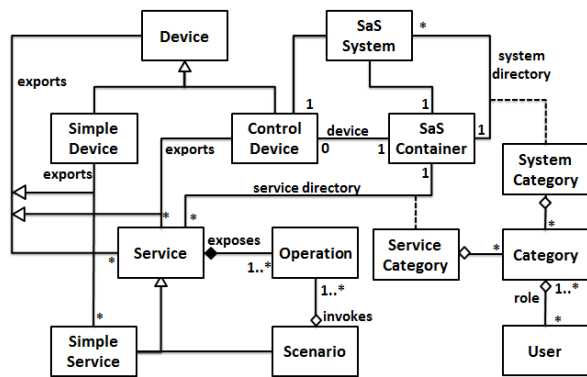


Figure 4. SaS-SDL class diagram

IV. EXECUTION OF DISTRIBUTED SCENARIOS IN SAS

This section presents how the SaS platform supports the execution of distributed scenarios.

A. Scenario execution control

To control scenario execution, SaS handles the *scenarios' life-cycle*. The objective here is threefold:

- provide basic *start, pause, abort and resume* operations for the user to manually control scenario execution,
- provide mechanisms on top of the middleware's detection capability to *dynamically react to detected changes in the environment* (e.g., unpredictable service unavailability consequent to its failure or mobility),
- provide mechanisms that *take advantage of service and scenario mobility to enrich scenario functionalities* (e.g., enabling to combine in a same scenario services that will never coexist on a single SaS platform).

Scenario life-cycle. Scenario execution is externally controlled: users can interfere during execution and changes in the environment can trigger compulsory reactions from the platform (e.g., it is impossible to ignore that a service disappeared while being executed). Therefore, scenario life-cycle needs to be rich enough to encompass specific behavior to dynamically react to many different situations. Scenario life-cycle management is enforced by SaS platform. The state diagram of Figure 5 illustrates the proposed life-cycle. Here, most transitions are initiated by users (except when *finished*, which is automatic) which use the basic *start, pause, resume and abort* service operations for the scenario.

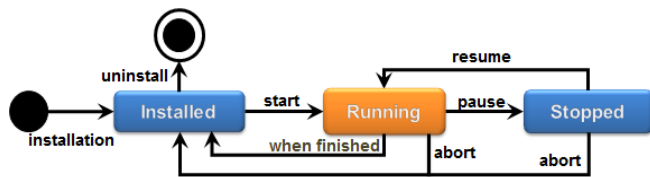


Figure 5. Scenario life-cycle in SaS

Fine, step by step, scenario running. Scenario executions cannot be considered atomic as they involve multiple and distributed service invocations. Moreover, scenario execution can be paused at any time by users or be interrupted at any unpredictable step in case a service disappears.

The *Running* state itself decomposes into a more precise state machine (see Figure 6). SaS considers scenario execution as a succession of steps, and define pre-conditions and post-conditions for each. For example, a pre-condition can be the presence of appropriate services or the execution of a previous step. Post-conditions are threefold: (1) successful execution of the step, (2) a problem occurs (service disappearance or timeout), or (3) interruption by the user. Such capabilities are completed with a logging system that reports scenario step by step execution status. Users can therefore check scenario advancement through the `getScenarioState` operation. Moreover, this enables SaS to retrieve scenario status after an interruption. Transitions are all handled by SaS container.

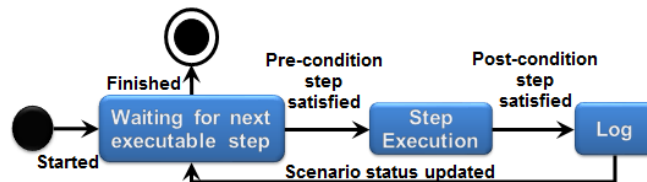


Figure 6. Internal running state diagram

Scenario delayed execution. The step by step running of scenarios has a positive counterpart when considering service mobility. If the user wishes to do so (this option is set at scenario creation), a scenario can be created that comprises operations that are never encountered in a same place at a same time. The user can choose from simultaneous (all services must simultaneously be present) or not. In the latter case, the user has to set a scenario maximum waiting period such as an hour or a day, that limits the duration the scenario might spend waiting for some services to appear.

When the scenario is to be executed, the steps that can be are and the system pauses the scenario until the next step is doable. The satisfaction of the next step precondition will automatically be detected and cause the execution to be resumed. If the device on which the scenario executes has not changed place, this step by step execution might have executed services that are supposed to be present at the same place but not at the same time (e.g., a service offered by a device that moves with its user such as a mobile phone). If the device on which the scenario executes has changed place (e.g., the scenario is executed on a device that moves with the user), this step by step execution might have executed services that are supposed to be present at two distinct places (e.g., a service offered by a device at home and a service offered by another device in a hotel room).

B. Scenario distribution

Registering SaS systems enables users to select who to share scenarios with. Having multiple users generates the need for a scenario access policy. In addition, as devices might fail and scenarios not be shared elsewhere, maintaining scenario availability also lies on SaS's scenario redeployment capability.

1) *Scenario access right policies*: By exporting their scenarios as services, users can share them. However, users might not want everyone to have the same access to created scenarios. SaS provides two modes for sharing scenarios: *individual*, the scenario is shared with a specific system (which provides access to the system's owner) or *grouped*, the scenario is shared with a whole category of systems (which provides access to the set of these systems' owners). Grouped access mode can be used to designate all systems a given user has access to (e.g., dad's) or all systems that pertain to another category (e.g., local network). Three access levels are possible: *private*, the scenario is not shared, *delegated*, the scenario is known and remotely accessible to the systems it is shared with but the owner system possesses the only copy and still executes the scenario, and *copied*, the scenario is copied locally into the system it is shared with and can be executed on the new system autonomously.

2) *Scenario redeployment*: When a user shuts down his/her platform, the solution to maintaining scenario availability is redeployment. Before doing so, SaS first warns the user if a scenario provided by this platform still is running. User can wait for the end of scenario execution. Otherwise, SaS tries to redeploy the scenario on another platform and transfer its current status and execution advancement. The destination platform is chosen from other available SaS systems registered in the system directory. If none of these systems accept, SaS asks other SaS systems present in the environment. If the scenario has not been redeployed on platform restart, SaS asks the user if scenario execution should be resumed.

V. SYSTEM DESIGN AND IMPLEMENTATION

This section describes the design and implementation of the SaS prototype. It is an ongoing work implemented in Java over OSGi [9][10] with iPOJO [11]. OSGi is a popular framework that enables to dynamically manage softwares as sets of decoupled modules called *bundles*. iPOJO is a full fledge Service-Oriented Component Model [12] based on OSGi. The main idea is that a component should only contain business logic as in EJB 3.0 [13] (*EJB entities*); SOC mechanisms should seamlessly be handled by the component container as container-managed cross-cutting services. The already implemented parts of SaS are presented in the previous paper [4].

Scenario delayed execution. Depending on execution rules (parallel or sequence), SaS invokes services present as

defined in IV-A and register the result necessary for some services (as operation parameter).

As defined in [4], SaS translates a scenario in a succession of Java instructions thanks to Javassist [14]. Instead of implementing the whole scenario as the start operation, this version of SaS implements each action block of the scenario in different methods to enable a stepped and delayed execution. A scenario can now be launched even if all services are not present, and it keeps running until it ends, it is stopped, period of validity finishes or, the platform is closed. Leveraging iPOJO the presence of each service independently. So, when all the services involved in an action block become available, the appropriate method is automatically called.

Sharing scenarios. When users share a scenario with all the available platforms, SaS exports the corresponding service as a remote service.. This way, discovery and distribution can be handled automatically by the last version of OSGi. Instead, if users select some other systems to share a scenario with, SaS uses the *UpdateServiceDirectory* service exported by each SaS platform. It enables to send events (service appearance or disappearance) to selected systems.

VI. STATE OF THE ART

This section analyses a representative set of systems that provide a solution for ubiquitous environments and enable scenario creation.

SLCA [15] provides developers with means to compose web services. A composite service contains proxy components bound to involved web services. With **SODAPOP** [16], users specify a goal that the system tries to reach with the available services. The main hypothesis is that each service contains informations about its initial conditions and its effects. **MASML** [17] is a multi-agent system for home automation. Scenarios are defined with an XML syntax and consist of sequences of service operation invocations. Mobile agents are in charge of scenario execution. **SASHAA** [18] is one of our previous work, focused on ubiquitous systems for home automation. It enables end-users to create scenarios with Event - Conditions - Action rules through an appropriate GUI.

The SaS ubiquitous software manages scenario life cycle and provide users with basic *start*, *pause*, *resume* and *abort* operations to fully control scenarios, whereas MASML and SASHAA only enables to start and stop scenarios. The SaS system is the only one to to share scenarios with other users. SASHAA, SLCA and MASML handle adaptation to environmental changes, however, scenarios cannot be executed in different times on multiple places. SODAPOP manages the environment by automatically classifying new services according to pieces of information. However, users have no control on this organization. Moreover, SASHAA enables to specify locations for systems but not register services. Table VI summarizes this study. Symbol ✓ means

that the requirements is fulfilled, - signifies that it is partially accomplished and × represents an absence of solution.

TABLE VI - SYSTEMS COMPARISON

Systems	Scenario Execution Control	Multi User	Adaptability	Context Management
SLCA	×	×	-	×
MASML	-	×	-	×
SODAPOP	×	×	×	-
SASHAA	-	×	-	-
SaS	✓	✓	✓	✓

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented the new mechanisms of our SaS system to manage ubiquitous environments. In addition to enable scenario creation by service composition, the SaS-SDL provides means to organize users’ contexts. Users can register services for a future use. SaS can execute scenarios step by step, at different times, on different platforms. Users can also classify surrounding SaS systems and share scenarios according to different access rights. A graphical representation of SaS-SDL enables end-users to benefit from SaS mechanisms.

For future work we want to add semi-automatic service composition to SaS. Learning from existing scenarios, SaS will propose some possible service compositions to the user. SaS will analyze which scenarios are created and used by users and will extract the more frequent services compositions.

ACKNOWLEDGEMENTS

This work is partially supported by the CARNOT M.I.N.E.S Institute (<http://www.carnot-mines.eu/>).

REFERENCES

[1] M. Weiser, “The computer for the 21st century,” *Scientific American*, pp. 78–89, 1995.

[2] H. Schulzrinne, X. Wu, S. Sidiroglou, and S. Berger, “Ubiquitous computing in home networks,” *IEEE Communications*, pp. 128–135, Nov 2003.

[3] M. P. Papazoglou, “Service-Oriented Computing : Concepts, Characteristics and Directions,” in *Proc. of the 4th Int. Conf. on Web Information Systems Engineering*. IEEE, Dec 2003, pp. 3–12.

[4] M. Faure, L. Fabresse, M. Huchard, C. Urtado, and S. Vauttier, “The SaS Platform for Ubiquitous Environments,” in *Proc. of the 23rd Int. Conf. on Software Engineering and Knowledge Engineering*, July 2011, pp. 302 – 307.

[5] OASIS, “Web services business process execution language version 2.0,” april 2007, [Last consulting: July 2011]. [Online]. Available: <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>

[6] D. Romero, R. Rouvoy, L. Seinturier, and P. Carton, “Service Discovery in Ubiquitous Feedback Control Loops,” in *Proc of the 10th IFIP Int. Conf. on Distributed Applications and Interoperable Systems*, ser. LNCS, F. Eliassen and R. Kapitza, Eds., vol. 6115. Springer, Jun 2010, pp. 113–126.

[7] G. Nain, E. Daubert, O. Barais, and J.-M. Jézéquel, “Using mde to build a schizophrenic middleware for home/building automation,” in *ServiceWave’08: Networked European Software & Services Initiative (NESSI)*, Madrid, dec 2008, p. 49–61.

[8] G. Banavar and A. Bernstein, “Software infrastructure and design challenges for ubiquitous computing applications,” *Communi. of the ACM*, vol. 45, no. 12, pp. 92–96, 2002.

[9] OSGi Alliance, “OSGi Service Platform Core Specification Release 4,” 2005, [Last access: July 2011]. [Online]. Available: <http://www.osgi.org/download/r4v40/r4.core.pdf>

[10] —, “OSGi Service Platform Enterprise Specification,” pp. 15–27, 2010, [Last access: July 2011]. [Online]. Available: <http://www.osgi.org/download/r4v42/r4.enterprise.pdf>

[11] C. Escoffier and R. Hall, “Dynamically adaptable applications with iPOJO service components,” in *Proc. of the 6th int. Conf. on Software composition*, ser. LNCS, vol. 4829. Springer, Mar 2007, pp. 113–128.

[12] H. Cervantes and R. Hall, “Autonomous adaptation to dynamic availability using a service-oriented component model,” in *International Conference on Software Engineering (ICSE)*. IEEE, May 2004, pp. 614–623.

[13] Sun Microsystems, “Enterprise javabeans specifications,” may 2006, [Last consulting: July 2011]. [Online]. Available: <http://java.sun.com/products/ejb/docs.html>

[14] S. Chiba and M. Nishizawa, “An Easy-to-Use Toolkit for Efficient Java Bytecode Translators,” *Proc. of the 2nd int. conf. on Generative programming and component engineering*, pp. 364–376, Sept 2003.

[15] V. Hourdin, J. Tigli, S. Lavirotte, G. Rey, and M. Riveill, “SLCA, composite services for ubiquitous computing,” in *Proc. of the Int. Conf. on Mobile Technology, Applications, and Systems*. New York, USA: ACM Press, 2008, pp. 1–8.

[16] J. Encarnação and T. Kirste, “Ambient intelligence: Towards smart appliance ensembles,” in *From Integrated Publication and Information Systems to Information and Knowledge Environments*. Springer, Dec 2005, pp. 261–270.

[17] C.-L. Wu, C.-F. Liao, and L.-C. Fu, “Service-Oriented Smart-Home Architecture Based on OSGi and Mobile-Agent Technology,” *IEEE Trans. on SMC, Part C*, vol. 37, no. 2, pp. 193–205, Mars 2007.

[18] F. Hamoui, M. Huchard, C. Urtado, and S. Vauttier, “Specification of a component-based domotic system to support user-defined scenarios,” in *Proc. of 21st Int. Conf. on Software Engineering and Knowledge Engineering*, July 2009, pp. 597–602.

A Graph-Based Requirement Traceability Maintenance Model

Facilitating Chronological Evolution

Vikas Shukla^{1,2}, Guillaume Auriol^{1,2}, Claude Baron^{1,2}

¹LAAS-CNRS,

7 avenue du Colonel Roche, F-31077 Toulouse, France

² Université de Toulouse; UPS, INSA, INP, ISAE; UT1, UTM, LAAS ;

F-31077 Toulouse, France

{vshukla, gauriol, cbaron}@laas.fr

Abstract—Requirement traceability remains a challenging task for the software developers. It helps stakeholders to understand the various relationships between the artifacts produced during the development process. During this requirement evolution process, information is produced and is stocked as trace. Some part of this information is lost owing to traceability maintenance process as links are deleted and removed from the system. This lost information is very useful while making decisions during the development process. In this paper we discuss a graph-based traceability model, which allows easy maintenance without any significant information loss. We show that both nonfunctional and functional requirements can be traced forward and backward using our proposed graph-based traceability model.

Keywords-Requirement Traceability; Graph; Maintenance; Decision making.

I. INTRODUCTION

Requirement traceability is the ability to describe and follow a requirement in both forward and backward direction in a software development life cycle [1]. Requirement traceability is seen as an index of software quality, it is one of the recommended activities for the system requirement specifications [2], CMMI and ISO 15504 consider it as ‘best practice’ and strongly suggest its usage. Requirement traceability allows various stakeholders to understand the various existing relationships among the produced artifacts during the product development process.

A requirement is traceable if you can discover who suggested the requirement, why the requirement exists, which requirements are related to it and how that requirement relates to other information such as systems design, implementation and user documentation. Traceability information helps you discover which other requirements might be affected by requirement changes.

Requirement traceability is always associated with artifacts, we define artifact as any product which may have originated during the course of development process or is utilized during the development process or later and is important for the success of project.

Every organization implements its own suitable guiding principles for requirement traceability which are known as ‘traceability policies’. Traceability policies define which information dependencies between requirements should be maintained and how this information should be used and managed.

Traceability means different things for different types of users depending on the types of users high-end or low-end [3, 4]. Usually, quality requirement of a system, which are mostly nonfunctional requirements, are high-end users requirements associated with management people. Low-end users are usually developers, programmers or people involved with testing, verification or validation.

For high-end users it implies how the client needs have been fulfilled but usually the low-end users find it unnecessary work overload [3], Tracing of nonfunctional requirements satisfies their needs. Similarly the traceability need of low-end users is satisfied with functional tracing.

We have contributed to the existing state of art by proposing a valid solution to the maintenance problems, i.e., the information loss, and dangling traces. Our paper addresses solution for the existing requirement traceability maintenance problems using graph-based methodologies, based on event-based traceability [5]. We show how we can increase the value of trace for the low-end users and hence involve them rigorously in traceability process. Our approach shows the interesting solution for the dangling-trace and information-loss problem and shows how our technique can be suitably used for minimizing cost of maintenance.

The paper is organized as follows. Section 2 of this paper highlights the current traceability maintenance problems. Section 3 presents the existing related works. Section 4 presents our graph-based traceability maintenance model. Section 5 discusses various aspects of our maintenance scheme and discusses feasibility and scalability issues linked, and equally the various combinations possible with recovery schemes. Section 6 concludes the paper and brings the possible problems and solutions linked to our approach. Finally, Section 7 presents the future perspective works envisaged.

II. TRACEABILITY MAINTENANCE PROBLEM

The requirement traceability is a continuous activity, involving peoples of various levels to participate continuously and maintaining a perfect communication channel among them for avoiding any information lapse. A good communication channel can help to figure out inconsistencies in the interpretation of requirements among various stakeholders which is very necessary for requirement engineering activities. Besides the communication there are various issues in traceability maintenance. Maintenance is the activity of updating and modifying already existing traceability relationships [6]. We discuss a few of the

existing important maintenance problems, which we address in this paper.

A. Cost of Maintenance

As the requirements are continuously evolving through the life of a project, requirements are added, removed or modified. The links between these evolving requirements need to be maintained. In a sufficiently complex system, the number of requirements can vary up-to few thousand requirements depending upon the granularity. Maintaining these requirements can be tedious task involving lot of computational and human resources.

B. Dangling Trace

A dangling trace is one which points nowhere or it lacks either a source or a target [7]. Such situation may arise due to human or system error during the course of a continuous evolution of a fairly complex system. They may also arise due to changes in the system model rendering some part of old system out of the boundaries of new system and hence it becomes difficult to trace them with respect to new requirements.

C. Information Loss

Whenever a new requirement is added to the system it needs to be linked to other requirements and available artifacts. The corresponding owners of the linked artifacts should be informed and advised to bring up the necessary changes. Similarly whenever an artifact is removed or altered or its dependency changes all the information should be communicated to the various stakeholders. This task usually involves maintaining these fine grained relationships and continuous update of such information usually leads to loss of data and hence information. We claim this information to be important as they are result of earlier high level discussions and decisions which involved certain cost.

If any such information is deleted permanently then in case of a future discussion there is chance that development team may reach a similar decision which was earlier found to be inutile. This may happen due to a probable change in the team or may be just of a simple absence of a member, which is quite possible as project development may take sufficiently long time.

D. Increasing Value of Trace for low end users

As mentioned earlier, for the low end users traceability seems to be a monotonous task and they are reluctant to involve themselves in traceability process. They do not find it very useful for their objectives and hence traceability does not offer them sufficient valorization for their work.

Whereas with every change brought to an artifact during the course of development there is an inherent risk attached to every dependent artifact involved which may jeopardize the success of project. We show in the following section that this risk evaluation factor can be used as a tool to valorize the work of low-end users and hence to continuously involve them in traceability mechanism. This associated risk can then be utilized in change impact management.

III. OTHER REALATED WORKS

Current literature on traceability contains ample work on need, and generation of traceability [1]; however, fewer work has been produced regarding the maintenance of traceability [5, 7, 9, 10, 14] the existing ones do not address properly the information loss problem. Cleland-Huang *et al.* [5] proposes publish-subscribe mechanism, a relationship between artifacts is registered to a central server. The evolution is represented by the series of change event. When a requirement is changed, the subscribers are notified about the change and they may bring the potential changes to their artifacts. It allows complete removal of requirements.

Another event-based scheme [14] uses a tool called Tracemaintainer but it uses only UML structural models. Another similar tool to Tracemaintainer is ArchTrace [13], it addresses the consistency and evolution of trace links between software architecture models and their associated code. Another approach for evolving traceability for heterogeneous artifacts [11] gives interesting insights about which information should be traced for corresponding artifacts so that fine-grained differencing can be used to identify evolution. The graph-based traceability schemes exist in literature like [6, 15, 16]. Schwarz *et al.* [6] recommends the complete deletion of traceability links hence in this respect it is like our maintenance model, but it insists the trace maintenance using the technique based on [5], but essentially they are based on transformation models, while this paper is based on classical techniques. Some earlier works have recommend versioning schemes for traceability maintenance of artifacts [9], but with the versioning schemes it becomes hard to see the evolution at an instant. The other approaches are state-based [7], and scenario-based traceability. The state-based techniques employ syntactic differences between different versions of model. Some use text differencing to identify change. The other techniques for managing traceability, based on evolution, use policy-based support [10].

An important aspect of various traceability models is of the traceability recovery scheme. To reduce the cost of traceability, use of semi-automatic and automatic mechanism for traceability recovery is advocated. This is an important aspect, as for a fairly large sized project creating traces manually can be tardy.

ADAMS [16, 17] uses a latent semantic indexing scheme for traceability recovery from the checked in artifacts. There are many schemes based on IR (information retrieval) and vector space model techniques. The majority of traceability tools equipped with semi-automatic or automatic recovery techniques are plagued with ‘false positive’ problem [16]. The tool ADAMS uses an event notification scheme and claims automatic traceability recovery scheme and other modules for project management. It also uses a versioning scheme for traces, but still some information loss is still possible owing to complete removal of artifacts before the version release.

There are many traceability models, but most of the systems are overly complex and do not address the chronological evolution and information loss problem in particular. Valorizing traceability can be used as a tool in software configuration management [8].

IV. GRAPH-BASED TRACEABILITY MAINTENANCE MODEL

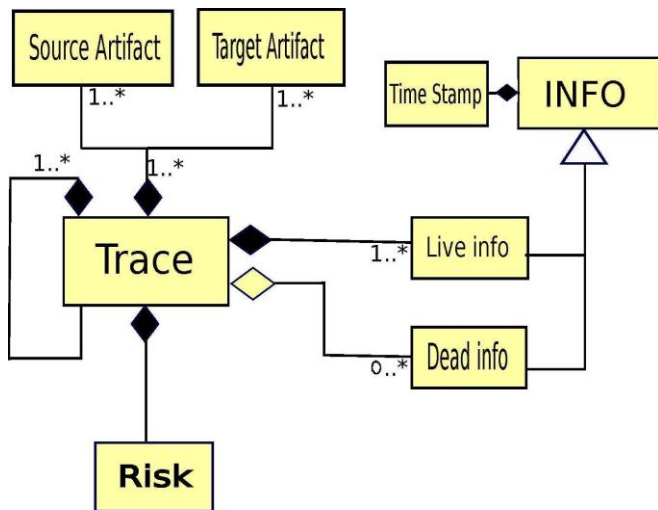


Figure 1 .Trace meta-model

Our graph-based traceability maintenance model is comprised of two entities: trace meta-model and traceability mechanism.

A. Meta-model

We propose our solution to the aforesaid problems; we assume that the information that a trace should contain are decided by traceability policies of the enterprise. We define our traceability meta-model, as shown in Figure 1. We have introduced the concept of live and dead information in our meta-model. Live information is one which is coherent till date and is represents the current state of artifact, whereas dead information is one which is obsolete with respect to current state of artifact but still holds information which shows the chronological evolution of system.

The trace meta-model defines trace as composition of other traces; a trace always contains at least one source and at least one target artifact. A trace contains two types of information live information and dead information.

Information is always associated with a time stamp indicating the period during which it was conceived or created. A trace should contain at least single live information and may not contain dead information. A trace always contains a risk associated apart from information. We recommend link model of [11] data to be taken in consideration for representing a trace information.

B. Traceability mechanism

Traceability mechanism is based on the graphical traceability techniques in which artifacts are represented as nodes and traces are links between the two or more artifact. The need of a product or product is considered as the root of the tree, non-functional requirements (NFRs) and functional requirements (FRs) are the immediate nodes to the root. As most of the NFRs are implemented as FRs, the NFRs are later linked to FRs and artifacts in next level at finer granularity.

In our traceability mechanism, we define three actions addition, modification, and rejuvenation; they can be applied both on traces and artifacts; there is no deletion operation but instead another sub-operation of modification called suspension. Suspension is envisaged to provide similar functionality like deletion, which permits to keep the track of trace evolution.

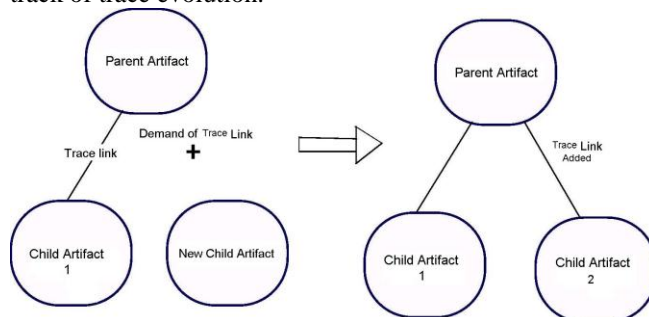


Figure 2. Addition operation

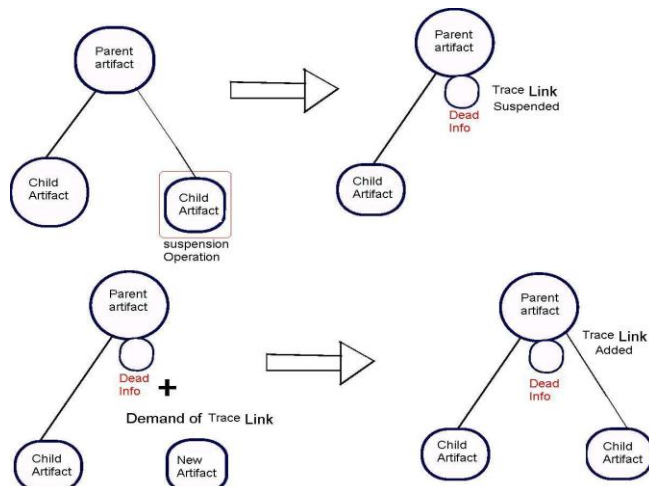


Figure 3. Modification-suspension operation

Each node/artifact maintains two additional lists, one for the dependencies or links, which are pointing to a dead artifact, and one which maintains the names of dead child artifacts.

1) Addition operation

Figure 2 shows the addition operation, when an artifact is created a trace is created pointing from the parent node to the recently created node. All the necessary data are filled and the node is initialized.

2) *Modification operation*

Modification operations are of two types change and suspension.

a) *Change*

In case of modification change operation whenever data are updated the earlier existing data are marked dead and the newer ones take their place and are marked alive.

b) *Suspension operation*

Modification-suspension operation is one when an artifact is no longer coherent with the current system state, and user actually wants to remove it, in this case the artifact is marked dead and is suspended and instead of complete deletion from tree it is moved one level up and is added to the list of dead artifacts of corresponding node. Figure 3 shows the modification–suspension operation. The other consequence to modification–suspension operation is that all the links from the various other artifacts which were pointing to dead artifact are added to the list of dead pointers.

3) *Rejuvenation operation*

A rejuvenation operation permits to change the status of a trace from dead to alive. This operation can only be applied when all the pre-artifacts to current artifact are alive or controlled, i.e., all the earlier artifacts which were the existential reason for the current artifacts should have been taken in account suitably.

V. DISCUSSION

In principle, the majority of graph-based traceability tools are more or less similar, plagued with similar deficiencies. We would recommend a semi-automatic traceability recovery technique. As, in a fairly large system a fully automatic mechanism can lead to false-positive notifications, which can be errant for requirement engineers. The current traceability mechanisms based on information retrieval (vector space models, latent semantic indexing, and probabilistic model etc.), structural rule-based, linguistically rule-based, transformation rule-based or other hybrid techniques are still error prone and needs to be improved.

Our traceability maintenance technique can be coupled with any traceability recovery technique, and used efficiently. Our paper addresses vital issue of information loss; for example, in a fairly large project which has duration of several years, it is possible that one artifact which was previously decided not to be included in the product owing to a certain constraint, is reintroduced. If the analyst had removed this artifact from system, the information regarding its exclusion was lost which was valuable to the project, and hence it costs again time and money, only to be discovered later regarding its deficiency. We claim that this ‘artifact evolution information’ is useful and should not be lost whether the decision regarding the artifact is finally affirmative or negative.

The major limitation of event-based traceability approach is of scalability; as the number of messages

generated passes a certain limit, it becomes difficult to handle so many notifications manually [17]; even reduced subscription cannot answer this problem. This maintenance problem is addressed by our technique. The cost of maintenance using our technique is fairly less, as compared to other techniques. For every artifact updated, the information which is obsolete becomes part of the parent node in the form of dead information, and the pointing trace is also removed and stocked as dead information with parent node, this eases the work of requirement engineer. In a large project with an event-based notification procedure, using our proposed technique, the deletion operation on any artifact could be executed without the overhead of notifications, and overhead of follow-up trace deletion requests from lower level artifact owners to higher level artifact owners.

Our traceability model includes risk evaluation of every trace created, this helps to valorize the traceability task of requirement engineer. The risk involved can be the information vital information regarding the dependencies or the rationale behind the existence of the artifact. We claim that, this can help requirement engineer to valorize his work and renders the tracing activity interesting by coupling analysis together, which can be used later, for calculating ripple effect.

VI. CONCLUSION

This paper has presented a new approach for traceability maintenance scheme, trying to address chief problems of current trace processes. The proposed traceability model emphasizes on maintenance with efficient maintenance schemes, we are developing a tool which comprises our technique, and we are yet to obtain results and observations which support our claims. Our technique provides interesting solution to the dangling trace problem, which can immensely help to reduce the tediousness of tracing process. Our solution offers a plausible solution to the information-loss problem as the information ever generated in the development process remains in system to provide the exact trace of evolution of the system.

With the ease in trace maintenance process the cost of maintenance can be reduced noticeably as the dangling pointer problem is solved the effort in maintenance is reduced and hence less time and less human resources are engaged to do the same task.

We claim that our technique can bind tightly the low-end users to the traceability process and can help them to valorize their work by involving them in risk assessment process of every artifact they own. Usually in the system development process there are numbers of iterations before an artifact is finally accepted as the part of system, our technique allows retaining the information regarding iterations and chronological evolution and hence helps in better decision making.

We can still not trace 100% of information as it is always difficult to trace the informal aspects of many artifacts. We advocate the usage of semi- automatic trace mechanism with

event specific human intervention for the optimal benefits of traceability.

VII. FUTURE WORK AND PERSPECTIVES

We are currently working to fully implement our technique, which addresses maintenance issues which we discussed in this paper. In spite of these facts there are other issues which need to be addressed like heterogeneous traceability schemes for capturing informal aspects.

Usually graph becomes large and hard to understand [12], our technique can be constrained to map intra-level traceability, reducing size and increasing the understandability of graph. Our technique can be evolved further to enable global distributed traceability.

There are still issues like increasing the value of trace and methods to augment the usability of trace in organization and how to holistically link the various aspects of system development with the traces. Can we utilize traces for rapid development process? Can traceability patterns be used for product development? How to evolve traceability techniques as a tool for change impact analysis? These are the numerous issues which need to be addressed by research communities.

ACKNOWLEDGEMENTS

The research leading to above results has received funding from the European Community' Seventh Framework Program (FP7/2007-2013) under grant agreement n° 234344.

REFERENCES

- [1] Gotel, O.C.Z., and Finkelstein, C.W., "An analysis of the requirements traceability problem," Proceedings of the First International Conference on Requirement Engineering (ICRE 1994), pp. 94-101, 18-22 Apr-1994, doi: 10.1109/ICRE.1994.292398.
- [2] "IEEE Recommended Practice for Software Requirements Specifications," IEEE Std 830-1998, 1998 doi:10.1109/IEEESTD.1998.88286.
- [3] Ramesh,B., "Factors influencing requirements traceability practice," Commun. ACM 41, 12 (December 1998), pp. 37-44. doi=10.1145/290133.290147.
- [4] Ramesh, B., and Jarke, M., "Toward reference models for requirements traceability," IEEE Transactions on Software Engineering, vol.27, no.1, pp. 58-93, Jan 2001 doi: 10.1109/32.895989.
- [5] Cleland-Huang, J., Chang, C.K., Christensen, M., "Event-based traceability for managing evolutionary change," IEEE Transactions on software engineering, vol.29, no.9, pp. 796- 810, Sept. 2003, doi: 10.1109/TSE.2003.1232285.
- [6] H. Schwarz, J. Ebert, and A.Winter., "Graph-based traceability: a comprehensive approach," *Softw. Syst. Model.* 9, 4 (September 2010), pp. 473-492. doi=10.1007/s10270-009-0141-4.
- [7] N.,Drivalos-Matragkas; D.S. Kolovos. R. F. Paige; and K.J. Fernandes., "A state-based approach to traceability maintenance," Proceedings of the 6th ECMFA Traceability Workshop (ECMFA-TW '2010). ACM, New York, NY, USA, pp. 23-30. doi=10.1145/1814392.1814396.
- [8] K.Mohan, P.Xu, LCao, B.Ramesh., "Improving change management in software development: Integrating traceability and software configuration management," *Decision Support Systems*, Volume 45, Issue 4, Information Technology and Systems in the Internet-Era, November 2008, pp. 922-936, ISSN 0167-9236, doi: 10.1016/j.dss.2008.03.003.
- [9] T. N. Nguyen, C. Thao, and E. V. Munson., "On product versioning for hypertexts," Proceedings of the 12th international workshop on Software configuration management (SCM '2005), ACM, New York, NY, USA, pp. 113-132. doi=10.1145/1109128.1109137.
- [10] A.Seibel, S. Neumann, and H.geise., "Dynamic hierarchical mega models:comprehensive traceability and its efficient maintenance," *Softw. Syst. Model.* 9, 4 (September 2010), pp. 493-528. doi=10.1007/s10270-009-0146-Z.
- [11] Hong, Y; Kim, M; Lee, S-W., "Requirements Management Tool with Evolving Traceability for Heterogeneous Artifacts in the Entire Life Cycle," Proceedings of the Eighth ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2010), pp. 248-255, 24-26 May 2010 doi: 10.1109/SERA.2010.39.
- [12] Winkler, S., and Pilgrim, J.V., "A survey of traceability in requirements engineering and model-driven development," *Softw. Syst. Model.* 9, 4 (September 2010), pp. 529-565. doi=10.1007/s10270-009-0145-0.
- [13] Murta, L.G.P.,van der Hoek, A., Werner, C.M.L., "ArchTrace: Policy-Based Support for Managing Evolving Architecture-to-Implementation Traceability Links," Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE '06), pp. 135-144, 18-22 Sept. 2006 doi: 10.1109/ASE.2006.16.
- [14] P.Mäder, O.Gotel, and I.Philippow., "Enabling Automated Traceability Maintenance through the Upkeep of Traceability Relations," Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA '09), LNCS 5562, pp. 174-189, doi: 10.1007/978-3-642-02674-4_13.
- [15] Pinheiro, F.A.C., Goguen, J.A., "An object-oriented tool for tracing requirements," Proceedings of the Second International Conference on Requirements Engineering (ICRE 1996), pp. 219, 15-18 Apr 1996, doi: 10.1109/ICRE.1996.491449.
- [16] De.Lucia, A., Fausto, F., Rocco, O., and Genoveffa, T., "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Trans. Softw. Eng. Methodol.* 16, 4, Article 13 (September 2007). doi=10.1145/1276933.1276934.
- [17] De.Lucia, A., Fausto, F., Rocco, O., and Genoveffa, T., "Fine-grained management of software artefacts: the ADAMS system.," *Softw. Pract. Exper.* 40, 11 (October 2010), pp. 1007-1034, doi=10.1002/spe.v40:11.

A Systematic Mapping Study on Patient Data Privacy and Security for Software System Development

Isma Masood

Department of software engineering
International Islamic University
Islamabad, Pakistan
ismamasood786@gmail.com

Saad Zafar

Faculty of Computing
Riphah International University
Islamabad, Pakistan
saadzafar@riu.edu.pk

Abstract-The exchange of Electronic Health Records (EHR) has increased threats to patient data privacy and security. The software systems developed for healthcare sector are required to explicitly address patient data privacy and security. A number of solutions have been proposed to incorporate these requirements into the software systems. However, there is no comprehensive study that synthesizes the different research initiatives according to any predetermined criteria. The main focus of this paper is to survey the various proposed solutions in the literature to incorporate patient data privacy and security into software systems. The proposed solutions are mapped against: (1) the software development stage for which the solution has been proposed, and (2) the established patient privacy and security principles. The existing literature has been surveyed using a systematic mapping study by phrasing two questions. In the mapping study, a total of 58 studies, dating from 2000 to 2011, were evaluated and mapped against the aforementioned categories.

Keywords-Systematic mapping study; Electronic Health Records (EHR); Patient data privacy and security; Software system development.

I. INTRODUCTION

Health information and medical records contain sensitive personal information including diagnosis and testing information along with person's family history, genetic testing, history of diseases and treatments, history of drugs used, sexual orientation and practices, and testing for sexually transmitted diseases [1]. Nowadays, digitized health records are not only used for diagnosis and treatment but they are also used for other purposes like improving efficiency of the healthcare system, drive public policy development administration, conduct medical research, and to provide effective health services that can be tracked and evaluated [2,3].

Increasingly, the electronically shared information within healthcare sector is receiving new threats to patient data privacy and security. Threats to patient data privacy and security become a major cause of inaccuracies and improper disclosure of information, which threaten individual's personal life and financial well being [3, 4]. Therefore, many laws and policies in different countries have been

implemented to protect patient data privacy and security especially for EHR [5].

To bridge the gap between different patient privacy rules, regulations and policies, Markle Foundation has proposed a set of principles under a *Common Framework* for uniform implementation of health information exchange across the health sector [9]. Markle Foundation works for advancement of health and national security through information and information technology in the United States of America. One of the major objectives of the Common Framework is to ensure patient privacy and seamless connectivity among various organizations related to the health sector. The privacy principles defined under the framework are described later in the paper.

A number of initiatives have been taken to propose effective integration these policies into software systems. However, effective implementation of *all* the policies and principles related to patient privacy and security into software systems remains a challenge.

Therefore, there is a room for new and improved solutions in this field. But before performing any new research, there is a need to synthesize the existing work in the area and to understand the need for improvement or to identify any new solution to an unresolved matter. Typically, a systematic literature review [SLR] is performed for this purpose. The idea of conducting SLR in the field of software engineering has been proposed by Kitchenham [6]. Often, a pre-requisite for conducting SLR is a mapping study, which is performed as an initial step to assess the feasibility of a complete SLR. In this paper we have conducted a mapping study as we could not find any SLR on the proposed solutions related to the Patient Data Privacy and Security in the field of software engineering. For this mapping study, we have followed the guidelines published in [7, 8].

We have presented the results of mapping study to identify available solutions on patient data privacy and security for software system development and have categorized these solutions against: (1) software development stages in software development cycle, and, (2) the well established policy principles for patient data privacy and security presented in [9]. Specifically, our mapping study addressed the following research questions (1) which solutions of patient data privacy and security have been

proposed for software system development? (2) Can we categorize these solutions using the Markle Foundation’s Common Framework?

In Section II, we have described our systematic mapping process; in Section III, we provide explicit answers to our research questions; the discussion of the results is provided in Section IV; conclusion and the future work are given in the last section.

II. THE SYSTEMATIC MAPPING PROCESS

For our mapping study, we following the guidelines provided in [7, 8]. Accordingly, our mapping study was conducted in three stages. In Stage 1, we define the scope, the search strategy and the selection criteria. In the second stage primary studies were selected applying the search strategy and the selection criteria. Lastly, in Stage 3, the selected studies are classified into the different categories.

A. Stage 1: Defining Scope, search strategy and selection criteria

We define the scope of the study as follows. The *population* of the study is selected as the set of articles addressing patient data privacy and security. As *intervention*, we selected any patient data privacy and security solution proposed for any of the software development cycle (e.g., requirements engineering, design, testing, etc.). The *outcome* of our study is a mapping of selected solutions to the patient data privacy principles found in [9]. Our search string for conducting the research was:

Patient AND Data AND (Privacy OR Security)

The research sources selected for our study were IEEE Digital Library, ACM Digital Library, Science Direct and Springerlink. To select relevant studies, we used the following inclusion and exclusion criteria.

Inclusion Criteria: A study contribution related to any stage of the software system development lifecycle. The study should also discuss *at least* one or more than one principles of patient data policy. For this purpose we read abstract, conclusion, introduction, or the full paper (if required).

Exclusion Criteria: Any study not related to the domain of software engineering, patient data privacy or security is not selected. The studies related to patient data privacy and security for images, sensor network and wireless transmission are also not included.

B. Stage 2. Selecting primary studies

In the first iteration, the search string was used at each resource. All references along with their abstracts were downloaded in Endnote [11] reference library. At this stage, we downloaded 4,670 references. In the second iteration, abstract of all reference were read and relevant studies which explicitly addressed the patient data privacy or security with contribution towards software system development were selected and placed in another library of selected papers. In this iteration, 120 studies were selected. We selected 93 papers from IEEE, 6 papers from ACM, 17 papers from

Science Direct and 4 papers from Springerlink. In the third iteration, full texts of these 120 studies were downloaded. We read all the articles one by one and applied the inclusion and exclusion criteria and finally selected 51 studies in our third iterative phase. We placed our 12 doubtful studies in the pending folder. In the fourth iteration, we discussed these doubtful studies and decided to accept 7 studies and to reject 5 studies. The breakdown of the results from each of the source is presented in Table 1, whereas Table 2 shows the distribution of our four iterative phases and the number of studies which were retained in each phase. In Table 3, we summarize the most relevant publication channels.

TABLE 1. NO. OF STUDIES AT EACH RESOURCE

Resource	No. of studies	No. of selected studies	Percentage
IEEE	4,540	44	0.96%
ACM	74	6	8.1%
Science Direct	40	8	20%
Springerlink	16	0	0%
Total	4,670	58	1.2%

TABLE 2. NO. OF STUDIES AT ITERATIONS

1st iteration	2nd iteration	3rd iteration	4th iteration
4,670	120	51	58

TABLE 3. MOST RELEVANT PUBLICATION CHANNELS

Acronym	Type of publication	Percent
International Journal of medical informatics	Journal	13.7%
Information Technology in Biomedicine	Journal	6.8%
CCSW	Workshop	5%
ICBECS	Conference	3.4%

The IEEE Digital Library had yielded the most number of papers (4,670), followed by ACM (74), Science Direct (40), and Springerlink (16). It is noteworthy that the most relevant studies were found in Science Direct (20%) and the least were found in Springerlink (0%). ACM had 8.1% and IEEE Digital Library had 0.96% relevant studies, respectively. Most of the relevant studies were found in International Journal of Medical Informatics (13.7%). This was followed by Information Technology in Biomedicine (6.8%). The rest of the relevant studies were found in two conferences: Workshop on cloud computing security (CCSW) (5%) and International Conference on Biomedical Engineering and Computer Science (ICBECS) (3.4%).

As part of our inclusion criteria, we included studies from the year 2000 to 2011. For the year 2000 we did not find any relevant study. However, from the years 2001 to 2008 the number of relevant studies increased steadily with a sharp increase in the year 2008 (frequency=17). The only exception to the trend is the year 2009 where the total number was reduced to only 4. In 2010 the number was again increased to 10 studies showing a positive trend. Only one study was found to be relevant in the first quarter of

2011. This trend of number relevant studies per year is given in Table 4.

TABLE 4. PERCENTAGE OF STUDIES AT EACH YEAR

Years	Relevant Studies	Selected Studies	Percentage
2000	2	0	0%
2001	5	2	3.4%
2002	6	1	1.7%
2003	8	3	5.1%
2004	8	3	5.1%
2005	10	3	5.1%
2006	10	4	6.8%
2007	23	8	13.7%
2008	25	17	29.3%
2009	30	4	6.8%
2010	23	10	17.2%
2011	22	1	1.7%
Total	172	58	

C. Stage 3. Classifying selected Studies

In the next stage, we divided our studies according into three categories. In the first category, we classified the studies according to the research approach used in the selected primary studies. We divide the research approaches according to the classification proposed by Weiringa et al. [10]. The *validation research* is used for those novel techniques that have not been implemented and are validated through experiments in a lab-like environment. The *evaluation research* is used to evaluate the techniques that have been implemented in practice. This research type explores how well the technique has been implemented. In the *solution proposal* either a novel solution is proposed or an existing solution is extended significantly. The *philosophical papers* propose either a conceptual framework to structure concepts into a new taxonomy. On the other hand *opinion papers* express personal opinion of the authors about a technique and the *experience papers* explain the experience of the authors of how a technique has been implemented in practice.

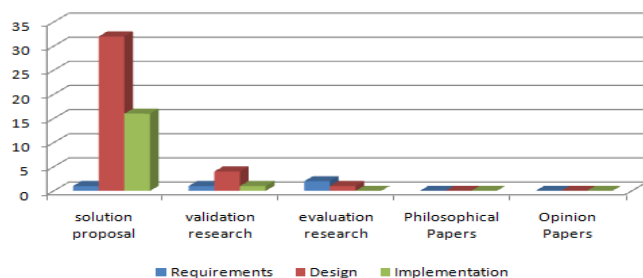


Figure 1: Mapping of studies according to research types

TABLE 5. RESEARCH TYPE AND SOFTWARE DEV.PHASE FACETS

Context	Solution	Validation	Evaluation	Total
Req.	1	1	2	4
Design	32	4	1	37
Imp.	16	1	0	17
Ver.	0	0	0	0
Maint.	0	0	0	0
Total	49	6	3	58

Table 5 shows the distribution of research type facet of the selected studies. An overwhelming majority of research approaches in the selected primary studies proposed a new solution ($f=49$). The next approach used the most was validation research ($f=6$) followed by evaluation research ($f=3$). However, we did not find any study that could be classified into any of the other research type categories. The results of this classification are summarized in Figure 1.

We also classified the studies on the basis of different stages of software development. Specifically, we grouped the software development stages into: *requirements*, *design*, *implementation*, *verification*, and *maintenance*. The breakdown of the classification of the selected studies is given in Table 5. The majority of selected primary studies addressed the Design phase of the software development ($f=37$), followed by the Implementation phase ($f=17$), while some of the studies were classified under the Requirements phase ($f=4$). We did not find any study related to software Verification and Maintenance phases.

Our next categorization was based on the Markle Foundation’s privacy principles [9]. The first principle of (1) *Openness and Transparency* mandates that there should be an overall policy of openness regarding personal data. The individuals should be aware of the nature stored data, its location and its access control policy. The (2) *Purpose Specification and Minimization* principle requires that the data collection purpose should be defined at the time of collection and its use should be limited to the intended purpose. Under the (3) *Collection Limitation* principle the personal health information must only be collected lawfully and with the knowledge and consent of the concerned individual. The (4) *Use Limitation* principle states that personal data must not be disclosed, made available or used in any manner other than the specified purposes. The (5) *Individual Participation and Control* principle requires that individuals have the right of access and control over their stored personal information. The (6) *Data Integrity and Quality* states that only the relevant data is stored and that the data is always accurate, complete, and current. The (7) *Security Safeguards and Controls* requires there should be reasonable security safeguards against the risks of loss of data or unauthorized access. The accountability of entities responsible for keeping and maintaining the personal data according to stated principles is covered under the (8) *Accountability and Oversight* principle. Lastly, the (9) *Remedies* principle states that there are adequate legal and financial remedies to address any security breaches or privacy violations.

Table 6 shows the distribution of studies according to the aforementioned privacy principles. As reflected in the data

shown in the table, we found many single studies that address multiple privacy principles. The most coverage was given to the Use Limitation principle ($f=38$). This was followed equally by the Individual Participation and Control, and Security Safeguard and Control principles ($f=24$). After them the most covered principle was Data Integrity and Quality principle ($f=16$), followed by the Purpose Specification Principle ($f=14$). The next principle covered the most was the Accountability and Oversight principle ($f=13$), whereas, the Remedies and Collection Limitation had the least coverage with a frequency of 3 and 1, respectively.

TABLE 6. CLASSIFICATION OF STUDIES ACCORDING TO PRIVACY PRINCIPLES

Principle	Req.	Design	Impl.	Total
Openness	2	5	1	8
Purpose Specification	1	9	4	14
Collection Limitation	1	0	0	1
Use Limitation	1	23	14	38
Individual Participation and Control	1	17	6	24
Data Integrity and Quality	1	13	2	16
Security Safeguards and Control	2	17	5	24
Accountability and Oversight	2	9	2	13
Remedies	0	2	1	3

III. RESEARCH QUESTIONS

Based on the above data, we now answer our two research questions.

RQ1: Which solutions of patient data privacy and security have been proposed for software system development?

In our mapping study we found 58 relevant primary studies. Out of these studies 63% of the studies were related to the Software Design. While 27% of the studies contributed towards Software Implementation and only 6% aimed at Software Requirements. Therefore, we can conclude that the most research is being conducted on how to effectively design software systems related to the requirements of patient data privacy and security. Similarly, there is also significant focus in the research community on how to effectively implement the patient data privacy and security requirements. Surprisingly, much less studies are focused on requirements analysis and specification phase of software development (see Figure 2).

RQ2: Can we categorize these solutions using the Markle Foundation's Privacy Principles [9]?

The mapping of selected studies against the Markle Foundation's Privacy Principles is given in Figure 3. As discussed earlier, a single study was often mapped against

multiple principles. But we found the solutions in the studies mapped reasonably well against the privacy principle. It is important to note that the Use Limitation was covered in 41.4% of the studies, followed by Individual Participation and Security Control principles with 41.4% studies. The other two principles covered in the selected studies were Data Integrity and Quality, and Purpose Specification with 27.6% and Purpose Specification 24.1%, respectively. The coverage of rest of the principles was not very significant.

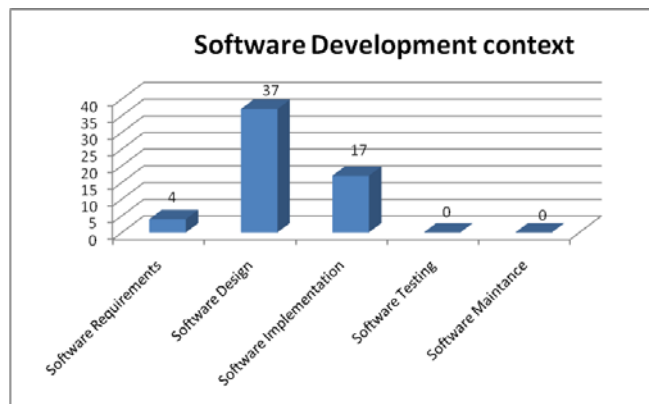


Figure 2: Mapping of studies according to software development context

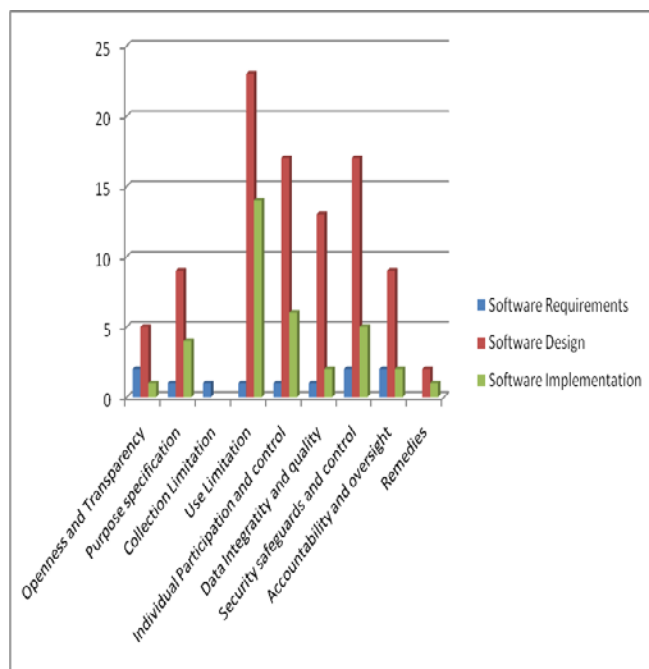


Figure 3: Mapping of studies against privacy principles

IV. DISCUSSION

The amount of personal information stored and exchanged by the health information systems is increasing by the day. With the increase in the volume of data the concern about the patient data privacy and security is also

increasing. The data stored about the patient include sensitive information like history of diseases and treatments, history of drugs used, sexual orientation and practices, results of sexually transmitted diseases, etc. As a result, a number of rules, regulations and best practices have been proposed to ensure that the stored data does not violate individual's privacy and that the data is never use inappropriately. Consequently, there has been a steady increase in research community to ensure that the software systems deployed must effectively integrate all the requirements related to patient data privacy and security.

The motivation behind our study was to investigate the feasibility for conducting a complete Systematic Literature Review. Here we cover the breadth of patient data privacy and security presented in the literature. The subsequent SLR studies can investigate the depth based on the results presented in our work.

The steady increase in the related primary studies from the year 2001 to 2010, with a few possible exceptions, indicates a growing interest in this significantly important research area (see Table 4). Similarly, the need of implementation of patient data and security requirements is reflected from the fact that most of the selected studies are concerned about the Design and Implementation of the privacy related requirements and less attention is paid to critically important phases of Requirements Analysis and Specification, Verification and Maintenance. This notion is further reinforced by the fact that the most common research approach used in the primary studies is Solution Proposal, with much less studies on validation and evaluation research. Likewise, we did not find any study based on experience reports, philosophical papers, or opinion papers.

Perhaps, not surprisingly the most importance is given to the Use Limitation, Individual Participation and Security Control principles. However, less coverage is given to the rest of the privacy principles, without which any software system cannot effectively implement a complete set of patient data privacy and security requirements.

We identify the following two limitations of our study: (1) some studies may have been missed due to the diverse use of the terms used in the search string; and (2) studies published in English language were selected in the search.

V. CONCLUSION AND FUTURE WORK

In this study, we have presented initial findings on solutions available for patient data privacy and security to develop software system. On this topic, we found 58 papers published in the years from 2000 to the first quarter of 2011. We have mapped these solutions against principles of

privacy policy to cover all aspects of patient data privacy and security. A large number of studies focused on Software Design as compared to Software Implementation and Software Requirements while, no study found on testing and maintenance. The Use Limitation principle along with Individual Participation and Control, and Security Safeguard and Control had most coverage in the selected studies. Our future work includes performing in-depth Systematic Literature Review on various aspects of Patient Data Privacy and Security identified in this study.

REFERENCES

- [1] U.S. Congress, Office of Technology Assessment, "Protecting Privacy in Computerized Medical Information" OTA-TCT 576. Washington, DC, US Government Printing Office, Sept 1993.
- [2] A. Appari and M.E. Johnson., "Information Security and Privacy in Healthcare: Current State of Research." *International Journal Internet and Enterprise Management*, vol. 6, pp. 279-314, Oct. 2010.
- [3] L.Gostin., "Health Care Information and Protection Privacy : Ethical and Legal Considerations" in ETATS-UNIS, 1997, pp. 683-690.
- [4] C. H. Liu, Y. F. Chung, T. S Chen, and S. D Wang, "The Enhancement of Security in Healthcare Information Systems." *International Journal of Medical System*, pp. 1-16, Nov. 2010.
- [5] M.Eichelberg, T. Aden, J. Riesmeier, A. Dogac, and G. B. Lalecil., "A Survey and Analysis of Electronic Healthcare Records Standards." *Journal ACM Computing Surveys*, vol.37, pp. 277-315, Dec. 2005.
- [6] B. Kitchenham and S.Charters., "Guidelines for performing systematic literature reviews in software engineering", Technical Report, EBSE-2007-01, Keele University, 2007.
- [7] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson., "Systematic mapping studies in software engineering.", in 12th International Conference on Evaluation and Assessment in Software Engineering (EASE), pp. 71-80 , June. 2008.
- [8] W. Afzal, R. Torkar, and R. Feldt., "A systematic mapping study on non-functional search-based software testing", in 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2008.
- [9] Markle Foundation, Connecting for Health Common Framework. January 10, 2011. <www.connectingforhealth.org>
- [10] R.Wieringa, N.Maiden, N.Mead, and C.Rolland, "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion", *Journal Requirements Engineering* . vol. 11, pp. 102-107, Dec. 2005.
- [11] T.Reuters,"EndNote-Your smater refrence assistant"Internet. June 5, 2010. <<http://www.endnote.com/>>

Impact on the inclusion of security in the UPnP protocol within the Smart Home

Alberto Alonso Fernández, Alejandro Álvarez Vázquez, M.P. Almudena García Fuente, Ignacio González Alonso
 Computer Science Department University of Oviedo
 Oviedo, Spain

alonsoalberto@uniovi.es, alvarezvalejandro@uniovi.es, agarciaf@uniovi.es, gonzalezaloinacio@uniovi.es

Abstract — This paper describes the impact caused by an encryption security system on a protocol for interoperability between robots and home automation. DHCompliant is an open source interoperability protocol supported by the UPnP standard. Until today, UPnP does not provide mechanisms for secure communications, since messages are transmitted over the network unencrypted and anyone can intercept and read its contents. The proposed security system is intended to provide DHCompliant with a dual security mechanism based on RSA and AES algorithms. The use of these algorithms can influence the performance of the protocol and the present work is focused on describing the real impact of the inclusion of such security mechanisms. Our results show that hiding information in a Smart Home interoperability protocol by the inclusion of a security system is viable and does not imply great consequences in CPU memory consumption.

Keywords – DHCompliant; Security; Data Encryption; UPnP.

I. INTRODUCTION

Security and interoperability are key issues in computer systems. In a system designed for the Digital Home, in which several technologies coexist handling data from devices as well as from the users, it is needed to cover the security of them as well as the interoperability of the whole system.

A. Security in the Digital Home

Having smart devices in the Digital Home is very useful. Once all the devices in a home are automated and connected through a network, it is important to consider security issues, authentication and access control [1]. There is a need for each device and each user to be authenticated in the system at the same time in order to interact. Regarding the interoperability protocols into the Smart Home, information related to its inhabitants and its habits are managed. This information is confidential and mechanisms, which make it inaccessible and/or illegible for entities from outside the Home, must be developed. At the same time, the devices that compose the system must be validated and accomplish a group of requisites in order to be part of the web, avoiding malignant devices to take control of the installation or allow a leak of information. The information managed in these environments includes all the values gathered by all the Smart Home sensors, as well as behavior patterns of the inhabitants (e.g., daily tasks, timetables and other personal information).

Without the existence of security in the Smart Home, its inhabitants' personal life information is exposed. It is necessary that this situation does not occur in order to extend the concept of Smart Digital Home in the society, this way the users will trust a system with a high level of reliability, which does not allow situations in which information and devices can be compromised.

B. Interoperability and security

The development of software systems incorporating heterogeneous components has a great potential, reducing costs and increasing productivity and flexibility to future changes, but on the other hand it is prone to suffer threats in non functional aspects of the system [2]. One of the problems identified is how to build a secure system from components, which may or not be safe by themselves. In this study, an example of components can be robotic adapters developed in different programming languages and executed in different platforms, the OpenID identity supplier or the software component, which administer the control and events of the home automation installation inside the Digital Home. The security of all the system cannot fall on an only component and the interoperability in the security of integrated systems is not a trivial problem [3]. It is possible that each component can implement different policies and security mechanisms, which may not be interoperable among them. This is the reason why it is highlighted the need of providing these systems with security mechanisms common to all components in order to preserve interoperability among them with a security guarantee.

Another aspect to be considered is the quality of the service provided (QoS) [4]. The main concern is the delay that may occur to access, transmit and display the information, which is exchanged in the Digital Home environment. In order to guarantee all the aspects previously stated, in the present study different options regarding security issues were evaluated. The principal aim was to choose a group of security mechanisms and algorithms already proven that endow a domo-robotic interoperability protocol with the security needed for preserving communications and confidential information that can circulate through the network.

C. Digital Home Compliant (DHCompliant)

DHCompliant [5] project aims to integrate home automation and robotics in the digital home and media communications network based on the Universal Plug and Play (UPnP) technology [6]. DHCompliant proposes a solution to develop collaborative tasks between robots

taking into account the information that home automation devices can provide, such as lighting conditions, humidity parameters or presence detection. All the information is handled to perform tasks managed by UPnP. From the automatic discovery of devices to remote invocations of robot actions are controlled by the UPnP protocol.

The paper is structured as follows: Section number two breaks down the current state of the art in the field of security and it is exposed the main motivation for this work. Section three describes the methodology used and Section four describes the experiments that have been included. Finally, Section five presents the results obtained and Section six assess all these results to draw conclusions and propose several future works.

II. MOTIVATION AND STATE OF THE ART

The main motivation of this study is to assess the impact of the proposed security system for protection of communications in the digital home. Due to the lack of security in the UPnP protocol, it has been studied the mechanisms and security encryption algorithms to choose an optimal solution to provide the required security system to safeguard the privacy of users. Today the latest specifications of the UPnP protocol does not provide any security mechanism for messages transmitted over the network or to authenticate users on the network as well as concepts of privacy.

One of the goals of this study is to provide a safety mechanism for interoperability protocol DHCompliant based on UPnP. Another goal is to evaluate how it affects the security system on the overall performance of the protocol.

In the present section, the main data encryption systems will be presented, as well as the DHCompliant protocol.

A. Data encryption

1) RSA (Rivest, Shamir y Adleman)

It is a public key cryptographic system developed in 1977. The safety of this algorithm lies in the problem of factoring integers. Sent messages are represented by numbers, and the operation is based on the product of two random large prime numbers in a secret way.

When you want to send a message, the speaker looks for the recipient's public key, encrypts the message with that key, and once the encrypted message reaches the receiver, it's decrypted using its private key.

RSA was believed to be safe until it was not known the quick ways to decompose a large number of prime products. Quantum computing could provide a solution to this problem of factoring.

RSA is used in multiple applications including electronic cash, secret broadcasting, secret balloting systems, various banking and payment protocols, smart cards, and biometrics [7].

2) AES

Advanced Encryption Standard (AES), also known as Rijndael is a schematic block cipher adopted as an encryption standard by the U.S. government.

AES has a fixed block size of 128 bits and key sizes 128, 192 or 256 bits. Rijndael is a block cipher with both a

variable block length and a variable key length. It would be possible to define versions of Rijndael with a higher block length or key length, but currently there is no need for it [8]. By design, the DES and TDES are slow algorithms. AES can be up to 6 times faster and, besides, not vulnerable [9].

AES has multiple libraries for the development of secure applications in several programming environments as C, C++, Java, C# o Python. Among all its uses, file compression, disk encryption, security in local networks (LAN) or as part of other applications as GPL [10] o Pidgin [11] are highlighted.

3) DES and 3DES

Data Encryption Standard (DES) is a method for encrypting information, chosen as FIPS in the United States in 1976, its use has spread widely throughout the world, [12].

Today, DES is considered insecure for many applications. This is mainly because the key size of 56 bits is short. DES keys have been broken in less than 24 hours. There are also analytical results, which demonstrate theoretical weaknesses in the cipher, although they are unworkable in practice. It is believed that the algorithm is safe in practice as a variant of Triple DES, although there are theoretical attacks.

Triple DES is also known as TDES or 3DES, was developed by IBM in 1998 [13]. The Triple DES is slowly disappearing, being replaced by the AES algorithm. However, most credit cards and other electronic payments have as standard Triple DES algorithm (previously used the DES) [14]. By design, the DES and TDES algorithms are slow.

4) BLOWFISH

Is a public domain symmetric block encoder, designed by Bruce Schneier [15] in 1993 and included a large number of sets of encoders and encryption products. While no analyzed Blowfish cipher has been found effective today, it has been given more attention than decoding blocks with larger blocks, like AES.

Blowfish was designed as a general purpose algorithm, which attempted to replace DES and avoid the problems associated with other algorithms for use in performance-constrained environments such as embedded systems [16].

5) IDEA

Is a block cipher designed by Lai and James L. Xuejia Massey of the Federal Polytechnic School in Zurich and was first described in 1991 [17]. An algorithm was proposed as a replacement for DES.

The designers analyze IDEA to measure its strength against differential cryptanalysis and concluded that it is immune under certain assumptions. Non successful linear or algebraic weaknesses have been reported. One of the most popular uses is within the framework of PGP [18].

B. DHCompliant architecture

DHCompliant protocol is divided into a number of subsystems that can meet existing needs in a home automation environment. It is a protocol set up over UPnP and it includes the following subsystems: Groups, Localization, Intelligence, Energy, and Security & Privacy.

- **DHC-Groups:** Is the service that manages the collaborative tasks. It transmits to the connected robots the task information to be executed and responds to requests that they are later made to form a hive of robots capable of performing a particular task.
- **DHC-Localization:** Allows obtaining the position of the robots in the house. The robots take the coordinates of the location system to navigate to the point where the task is performed.
- **DHC-Energy:** Enables power profiles management to perform collaborative tasks and calculations of costs and fees for expenditure control.
- **DHC-Intelligence:** Here are included semantic tagging capabilities, building and testing user-defined rules and machine learning. In this module is the Machine Learning [19] technology that provides the system with learning capabilities for making decisions in a more autonomous way.
- **DHC-Security&Privacy:** Allows the encryption of communications in the DHCompliant UPnP network protocol established [20]. Through the RSA asymmetric encryption algorithm is sent to all devices on the network a system password to be used by the AES encryption as its symmetric key. In the next section you can see the process in a more detailed reflection in a SysML diagram of sequence (Figure 2).

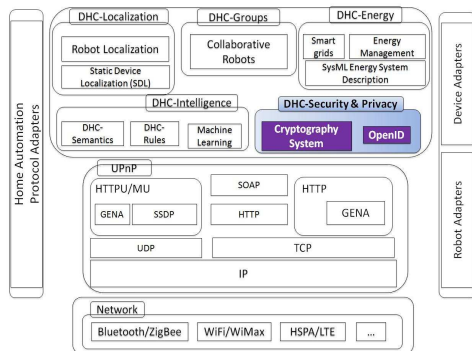


Figure 1. DHCompliant Architecture.

III. METHODOLOGY

This section describes the tools and elements required for including a security system into the DHCompliant protocol, as well as for performing the experiments.

A. Tools

To carry out the tests several tools have been used. A simple and effective technique has been used for measuring execution times for the .NET platform. It consists of the use of the basic classes and methods to measure time like TimeSpan and the attribute Ticks.

The method consists of the introduction of a Date .Now instruction in the source code at the beginning of what it is wanted to measure and a statement at the end of the method or code section. The two times are subtracted to get how many milliseconds.

To analyze the performance of the system it has been chosen a profiler for the NET platform, the YourKit Profiler [21]. It provides zero-overhead profiling for your .NET applications and makes code profiling and memory usage optimization simple and fast. The remote option has been used in all the experiments because it does not interfere with measurements. Measuring time and resources usage remotely is needed to obtain the better results.

B. Items

It can be distinguished two types of elements in consideration in conducting the experiments, hardware items and software items.

1) Hardware items

The following table (Table 1) describes the characteristics of the equipment used to perform all experiments.

TABLE I. LIST OF HARDWARE COMPONENTS USED FOR EXPERIMENTS

Computer	1	2
OS	Windows XP Professional 32 bits	Windows Vista Business 64 bits
CPU	AMD Athlon X2 4000+ 2.11GHz	AMD Athlon X2 4000+ 2.11GHz
RAM	3GB DDR2	2GB DDR2
HD	Western Digital 7.200 rpm	Western Digital 7.200 rpm

The computer number one is the machine that contains all the DHCompliant system. The computer number two is responsible for running an instance of the YourKit profiler to run tests remotely.

2) Software items. DHCompliant protocol.

DHCompliant protocol modules involved throughout all the tests are the following:

- **GUI:** It is the user interface from where the tasks are created and launched to be performed by the robots. The interface consists primarily of a form in which the user enters data for the task as the task name, the number of robots that are to be used, the target room and other necessary parameters for the job. It also allows the creation of user rules, selection of the energy profile for the task and the cancellation of tasks.
- **DHC:** Is the main part of DHCompliant. It contains all the protocol services: DHC-Groups, DHC-Localization, DHC-Energy, DHC-Intelligence and DHC-Security.
- **Adapters:** It is the software component that acts as a link between the physical robot and the DHCompliant protocol. It implements all the protocol functionality to perform the tasks sent by the user. It communicates with the API of each robot to use its features [22]

The experiments described in the following sections have been carried out to demonstrate what is the real impact of the inclusion of security and privacy in an interoperability protocol. The goal is to demonstrate what is the time penalty and performance when compared with the same protocol without restraint.

IV. DESIGN OF THE EXPERIMENT

The experiment was performed to study the impact of the security system consisted in executing a video surveillance task within the protocol DHCompliant. The objective is to perform a task from the user interface to be carried out by a

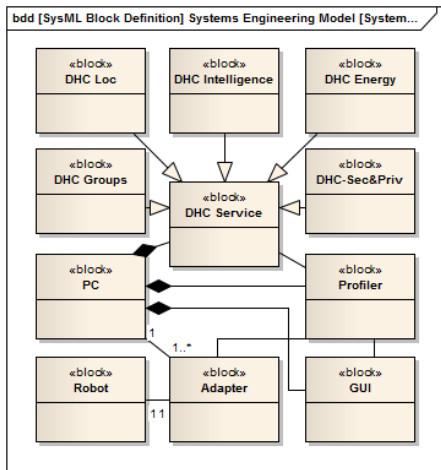


Figure 2. SysML package diagram of the experiment components.

robot. The DHC module is located between the robot and the user interface, and is responsible for the tasks management (choose appropriate robot, location service, energy service ...).

Because the encryption is included in each of the entities involved in the flow of execution of a task, it was decided to divide the experiment into three stages to obtain more accurate results and more data for analysis. One stage was chosen for the flow of execution in the graphic user interface, another for the DHC device, and the last one for the robot adapter.

First, the GUI generates the internal system, which will be the future symmetric key for the AES encryption algorithm to encrypt all protocol communications. This key must be shared with other devices in a safe way, so it is sent encrypted using the RSA algorithm.

Once the devices (DHC and adapters) are subscribed to the UPnP security service of the GUI, they perform an invocation to obtain the system key. The devices also implement the RSA algorithm so in the previous involution the GUI sends your public key. Next, the GUI key system encrypts the public key of each device. The value returned by the invocation is the key encrypted with the public key of each device that relies on the security action interface. After receiving the key, the device decrypts with its private key and initializes the AES symmetric cipher with the key obtained.

Once the processing is completed, the devices can subscribe to other services of the encrypted communications system.

V. RESULTS

This section describes the results obtained with and without the inclusion of a security system in DHCompliant.

A. Time measurements

Measure ranges were the following:

- In the interface, time was measured since the launching of a task until the last change of state variable (including the specification of the energy profile).
- In the DHCompliant central system, time was measured since the detection of the first state variable change until the last change of variable.

- In the robot adapter, time was measured since the change in the TaskID variable until it receives the response from the first request for coordinates to the location system.

The tables (Tables II and III) show the measurements obtained, with a system in which the data is unencrypted and another system in which data is encrypted.

TABLE II. TIME MEASUREMENT WITHOUT DATA ENCRYPTION

Iteration Number	Average Time (ms)
Interface	49,99696
DHC	462,29
Adapter	17946,77
Total	18459,06

TABLE III. TIME MEASUREMENTE WITH DATA ENCRYPTION

Iteration Number	Average Time (ms)
Interface	112,492
DHC	1118,722
Adapter	18444,75
Total	19675,96

B. CPU and memory consumption

With the profiler it has been taken samples from memory and CPU consumption during the course of

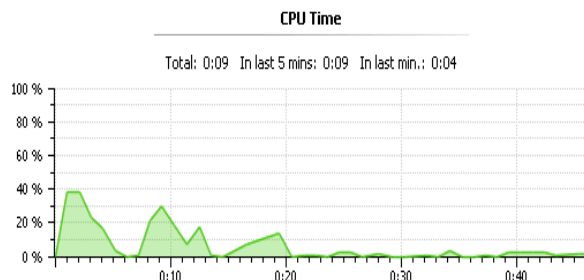


Figure 3. CPU time consumed in the adapter without encryption carrying out the experiment described above. To obtain more reliable results without interferences, tests have been carried out with the remote profiler option.

As it has been described in previous sections, the task was divided in three sections, one for every device involved and tests have been performed on the system with and without the encryption system. Samples from memory and

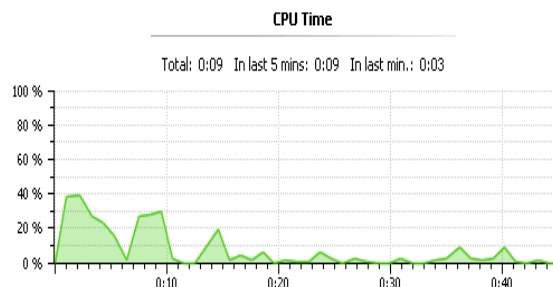


Figure 4. CPU time consumed in the adapter with encryption

CPU consumption have been taken with the profiler during the course of carrying out the experiment described above. To obtain more reliable results without interferences, tests have been carried out with the remote profiler option.

The pictures above show the most significant results obtained. Figure 3 and 4 illustrate the task execution flow in

the adapter in each case. This flow starts when the adapter is started until it receives the last task parameter from the DHC device.

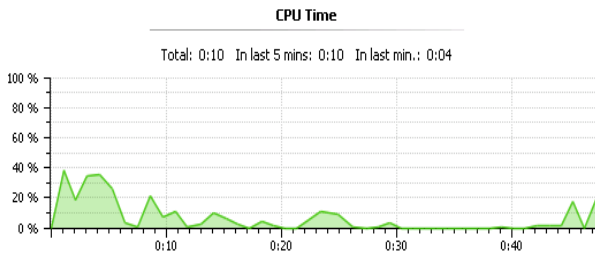


Figure 5. CPU time consumed in DHC without encryption

At the beginning there is no difference between two systems in terms of CPU load, but in the final moments it is noticed a small increase in the adapter with encryption system due to the obtained data from the task. The adapter ask DHC device for the task parameters so DHC answers the adapter with those parameters encrypted and the adapter must decrypt them. This process has a little increase of about 5% in the CPU load.

Figure 5 and Figure 6 show the results in the DHC device with and without the encryption system, respectively. In the first 15 seconds the adapter receives all the task

VI. CONCLUSIONS AND FUTURE WORKS

In this paper it is shown the most widely used encryption

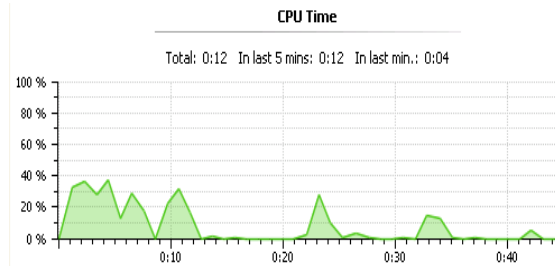


Figure 6. CPU time consumed in DHC with encryption

algorithms applied to a Smart Home protocol. Each system has its advantages and disadvantages but it has been decided to use RSA and AES systems for several reasons.

In the case of RSA, the main advantage of public key cryptography is an increased security and comfort, as the private key is not sent to any network device. In a secret key, however, the secret keys must be transmitted (either manually or through a communication channel), because the same key is used for encryption and decryption. A major problem is that there may be a possibility that an intruder

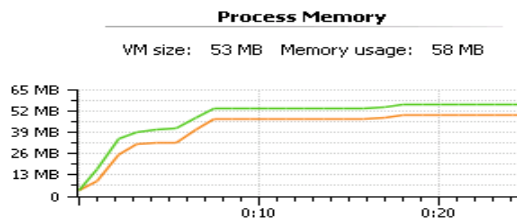
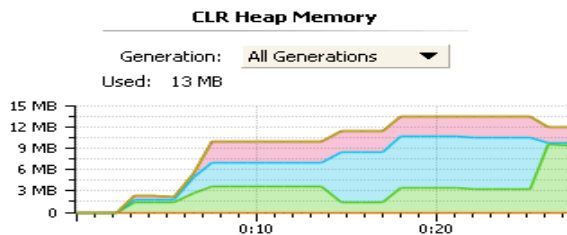


Figure 7. Memory usage in the GUI device with encryption system

parameters from the user interface and this information has to be decrypted. In this case, a peak in the encrypted system can be seen. This is because DHC receives parameters

can discover the secret key during transmission. This is why the private key is transmitted using the RSA system. The function of using a system based on public/private key

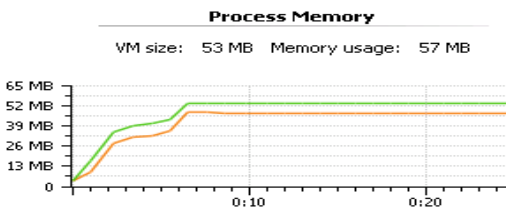
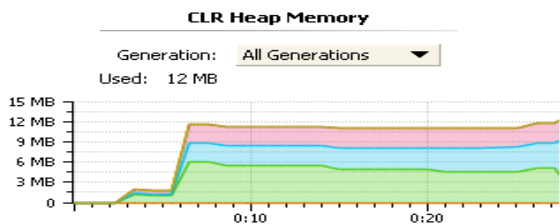


Figure 8. Memory usage in the GUI device without encryption system

encrypted and it has to decrypt and encrypt them again to send them to the adapter.

At about the twentieth third second a small increase in CPU load occurs. This stage corresponds to the stage when the adapter asks DHC for task information.

Figure 7 and Figure 8 shows the memory consumption in the user interface device with and without encryption system. All memory generations are shown. The encryption system consumes 1 MB of memory more than the non-encrypted.

encryption is to guarantee transmission of the AES key used to encrypt communications.

In the case of AES, the National Institute of Standards and Technology (NIST) with the joint work of Belgian researchers Vincent Rijmen and Joan Daemen selected Rijndael in October as a basis for AES. Rijndael was selected from among five finalists in a process that took more than three years [23]. Compared with other AES encryption algorithms, Rijndael had more elegant mathematical formulas behind, and only requires one pass to

encrypt the data. AES has been designed from scratch to be faster, unbreakable and capable of supporting smaller computing devices imaginable. The big differentiators between AES and other systems are safety, superior performance and better use of resources. Another reason to choose AES is that it provides strong encryption and has been selected by NIST as Federal Information Processing Standard in November 2001. In June 2003 the U.S. Government (NSA) announced that AES is secure enough to protect classified information up to TOP SECRET level, which is the highest level of security over the information, and which disclosure to the public would cause exceptionally damage to national security.

The experiments performed in this paper show that the inclusion of an encryption system in a protocol of interoperability provokes only a slight increase in consumption of RAM and CPU. Taking this into account, it can be concluded that the inclusion of a security system in the interoperability protocol in the Smart Home hides information is viable.

As future work, it would be interesting to implement other encryption systems for the DHCompliant protocol and compare them with the proposed solution of RSA + AES in order to get real data on the performance of each of the alternative algorithms. It is advisable to extend the encryption system to not only to encrypt the contents of the variables that contain information of the tasks within the digital home, also to encrypt the names of these variables.

Another aspect is to consider in the future the implementation of policies and recommendations on privacy issues. For products made in the European Union, the system proposed must comply with the Data Protection Directive 95/46/EC (European Union, 1995) and Regulation (EC) 45/2001 (European Union, 2001) and according to the instructions of the European Data Protection Supervisor (European Data Protection Supervisor, 2010). For products made in USA, it must comply with the Guide to Protecting the Confidentiality of Personally Identifiable Information (PII) (NIST (National Institute of Standards & Technology), 2010) [23]. Finally, Adapter, Robot or DHC service manufacturer MUST comply with the ISO / IEC 27002 (ISO / IEC - International Standard Organization, 2009) for information security.

REFERENCES

[1] J. A.-M. Manish Anand and R. C. M. Dennis Mickunas, «Secure Smart Homes using’ Jini and UIUC SESAME», *ACSAC ’00 Proceedings of the 16th Annual Computer Security Applications Conference*, 2000.

[2] E. A. O. Lawrence Chung, «Analyzing Security Interoperability during Component Integration», *IEEE/ACIS International Conference on Computer and Information Science*, 2006.

[3] K. M. K. J. Han,, «Composing security-aware software», *IEEE Software*, vol. 19, pág. 34–41, Feb. 2002.

[4] C. L. Samuel Pierre, «Security, Interoperability, and Quality of ServiceAspects in Designing a

TelecommunicationsPlatform for Virtual Laboratories», *IEEE Electrical and Computer Engineering*, 2000.

[5] Infobotica Research Group, «DHCompliant web site», *dhcompliant.com*, 2010. [Online]. Available: <http://dhcompliant.com/>. [Accessed: 04-Mar-2011].

[6] «UPnP Forum», <http://www.upnp.org/>, Oct-2010. [Online]. Available: <http://www.upnp.org/>. [Accessed: 10-Nov-2010].

[7] Richard A. Mollin, *RSA and Public-Key Cryptography*. Chapman & Hall/CRC, 2002.

[8] Joan Daemen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[9] A. A. H. Abul Ahsan and Md. Mahmudul Haque, «A Comparative Study of the Performance and Security Issues of AES and RSACryptography», presented at the Third 2008 International Conference on Convergence and Hybrid Information Technology, 2008.

[10] gnupg.org, «The GNU Privacy Guard - GnuPG.org», *The GNU Privacy Guard - GnuPG.org*. [Online]. Available: <http://www.gnupg.org/>. [Accessed: 26-May-2011].

[11] pidgin.im, «Pidgin, the universal chat client», *Pidgin, the universal chat client*. [Online]. Available: <http://www.pidgin.im/>. [Accessed: 26-May-2011].

[12] Mikael J. Simovits, *The Des: An Extensive Documentation and Evaluation of the Data Encryption Standard*. Aegean Park Pr, 1996.

[13] «IBM Press room - 1998-02-23 IBM Offers S/390 Customers Wider Safety Net to Conduct e-business - United States», *IBM Offers S/390 Customers Wider Safety Net to Conduct e-business*, 1998. [Online]. Available: <http://www-03.ibm.com/press/us/en/pressrelease/2780.wss>. [Accessed: 26-May-2011].

[14] William C.Barker, «Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher». NIST, 2008.

[15] «Schneier on Security». [Online]. Available: <http://www.schneier.com/>. [Accessed: 09-Mar-2011].

[16] Bill Gatliff, «Encrypting data with the Blowfish algorithm», Ago-2003.

[17] José M. Granado, Miguel A. Vega-Rodríguez, Juan M. Sánchez-Pérez, and Juan A. Gómez-Pulido, «IDEA and AES, two cryptographic algorithms implemented using partial and dynamic reconfiguration», *Microelectronics Journal*, Jul-2009.

[18] «The International PGP Home Page». [Online]. Available: <http://www.pgpi.org/>. [Accessed: 27-May-2011].

[19] Nils J. Nilsson, «Introduction to Machine Larning». Stanford University, Nov-1998.

[20] Infobotica Research Group, «Draft specification for data protection, user data privacy and access restriction». Dic-2010

[21] «.NET Profiler - Java Profiler - The profilers for .NET and Java professionals». [Online]. Available: <http://www.yourkit.com/.net/profiler/index.jsp>. [Accessed: 03-Mar-2011].

[22] Alejandro A. Vázquez, Ignacio G. Alonso, and M.P. Almudena García Fuente, «UPnP adapter for collaborative tasks development over the open protocol DHCompliant», presented at the INTERA 2011, Oviedo, Spain, 2011.

[23] «Goodbye DES, Hello AES», Jul-2001. [Online. Available: <http://www.networkworld.com/research/2001/0730feat2.html>. [Accessed: 07-Mar-2011].

OntoLog: Using Web Semantic and Ontology for Security Log Analysis

Clovis Holanda do Nascimento¹, Felipe Silva Ferraz²,
Rodrigo Elia Assad², Danilo Leite e Silva¹, Victor Hazin da Rocha²

¹CESAR – Recife Center for Advanced Studies and Systems
{clovishn,daniloleite2}@gmail.com

Informatics Center

²Federal University of Pernambuco (UFPE) Recife – PE, Brazil
{fsf3,rea,vhr}@cin.ufpe.br

Abstract—Along with the growth of available information on the internet, grows too the number of attacks to the Web systems. The Web applications became a new target to those invaders, due to the popularization of the Web 2.0 and 3.0, as well as the access to the social networks system's API's, the cloud computing models and SaaS. In this context, the identification of an eventual attack to those applications has become an important challenge to the security specialists. This article presents a proposition of using Semantic Web and Ontology concepts to define an approach to analyze Security logs with the goal to identify possible security issues.

Keywords-Security; Log Analysis; Ontology

I. INTRODUCTION

Log Analysis to search for information that can provide data about the process of identifying evidence, events and user profiles related to the system, consists in an ordinary and necessary activity for the teams that administer and manage systems. With the growth and popularization of Web systems [7] [12], the volume of information generated in logs has grown considerably.

The growth of generated logs made the most common techniques used to analyze them, such as looking for evidence of certain attacks and even compromising those by finding patterns in the logs, not as effective as they were before [4]. This scenario become even more complex when there is the need to identify co-relation between the events that are in the logs, such as identifying which operations a determined user in which systems in the last 3 days?

Alongside the problems described, we are maturing the definition of what can be defined as an attack and how an eventual attacker would use it [17] [24], what allowed to be adopted more sophisticated mechanisms, generating detailed data about the event, but making the log analysis more complicated.

In this context, the use of Semantic Web technologies, specifically, the use of ontologies, in the context of security log analysis, showed itself as a possibility of improving the results of the searches in the log files. Generally, is expected that the ontologies can help in the interpretation process of interpretation of the great diversity of information that are present in this kind of archive [3] [5] [6].

Fundamentally, the role of ontology is to enable the construction of a representation model of a given area through the representation of a terms and relations vocabulary [21]. According to Gruber [9] , Ontology is a formal and explicit specification of a shared conceptualization. Thus, as a formal specification, the ontology can be processed by computer software with precision in the analysis, facilitating the search in the log files and thus improving the efficiency of the results analysis [8].

This article aims to concisely present the proposal for the use of ontologies to analyze and process logs generated by web application firewall [15], identifying the generated types of information, its importance and the problems related to the log files.

The remaining sections of this paper are divided as follows: Section 2 presents the difficulties related to log analysis and security. Section 3 presents the use of ontologies for log analysis. Section 4 presents the results of the experiment. Finally, Section 5 presents the conclusions.

II. LOG ANALYSIS DIFFICULTIES AND SECURITY

The information stored in the logs are invaluable to the security area, for they have the attacks records, input ports, IP numbers, evidence of invasion, typed commands, among others. In addition, logs can be considered as a source in constant growth, due to the use of systems on a daily basis. Kimball and Merz [11] present some problems found in the log files, such as; multiple file formats, dispersed information, incomplete, inconsistent and irrelevant data, which makes the analysis and extraction of information from these files harder to accomplish.

The security auditor or system administrator has as part of their daily routine duties a rather hard activity, the research and analysis of logs. This task is considered difficult and time consuming, because the log files are created without a semantic description of their format, making the extracting of the meaning of the data impracticable, showing only the words typed in the search, resulting in poor quality results. According to Guarino [10], this limitation occurs because when the data is generated without the use of ontology, it can present ambiguities and vagueness of

elements. This situation becomes more serious when we face major files with gigabytes of information.

III. THE USE OF THE ONTOLOGY FOR LOG ANALYSIS

There are various definitions found in literature about what is ontology. Originally the term was born in the field of philosophy, being a word of Greek origin, which deals with the nature of being and its existence. In the field of Computer Science, it was first applied in the artificial intelligence field to computational representation of knowledge, being considered an abstract model of a domain. Below are some of the most used definitions for the term ontology:

- According to Gruber [9], "ontology is a formal and explicit specification of a shared conceptualization."
- The W3C consortium [25] defines ontology as: "the definition of terms used to describe and represent an area of knowledge."
- According to Noy and McGuinness [16], there is no one correct way to model a domain, meaning that there is more than one way to develop an ontology.

The basic components of ontology are classes (organized in taxonomy), relations (used to connect the domain concepts), axioms (used to model sentences that are always true), properties (describe characteristics common to the instances of a class or relationships between classes) and instances (used to represent specific data).

Ontologies are used for modeling data from specific domains and also allow inferences to discover implicit knowledge in these. Despite the considerable increase in the use of ontologies, build a complete ontology covering all the expressiveness of the domain continues to be a hard work, making the work of a multidisciplinary team a necessity, in which case it would be an ontology engineer, a security expert, among others, and acting in a participatory and integrated [22] [24].

More specifically, in this work, we are interested in building ontology for the representation of data available in security logs of web applications. In this context, ontologies can be useful for improving the classification of the attacks occurred and the identification of related events.

In the next session, we present an overview of ontology, and describe the methodology used for its creation.

A. General description of the proposed ontology

The proposed ontology, OntoSeg, has as main objective the representation of data generated by the application firewall log ModSecurity on this work. From a detailed analysis of several samples of the log we identified various classes and their relations. Table 1 presents a brief description of the main classes that compose the ontology for the representation of the security log.

TABLE 1. MAIN CLASSES OF PROPOSED ONTOLOGY

Class	Definition
Audit log header	Represents the log header, and contains the following information: date and time, ID (transaction ID, with a unique value to each transaction log), source IP, source port, destination IP and port of destination.
Request Headers	Represents the request Header, contain the information of the request and the header of the solicitation that was sent by the client.
Request Body	Represents the body of the transaction, contains the contents of the client request.
Intended Response Headers	Represents the status line and headers, this part is reserved for future use, so it is not implemented
Intended Response Body	Represents the body of the response of the transaction, the response body contains the actual case the transaction is not intercepted.
Response Headers	Represents the header of the actual response sent to the the client, contains data and headers
Response Body	Represents the body's effective response to the request, but this part is reserved for future use, so it is not implemented
Audit Log Trailer	Represents additional data, contains additional meta data of the transaction obtained from the web server or ModSecurity
Reduced Multipart Request Body	Represents the body of the request reduced, Part I is designed to reduce the size of the request body with irrelevant content from the standpoint of security, transactions that deal with uploading large files tend to be big
Multipart Files Information	Represents information about the files, the objective of this part is to write the metadata information about the files contained in the request body, this part is reserved for future use, so it is not implemented
Matched Rules Information	Figure 1. Represents the rules, contains a record with all the rules that occurred during the processing of transactions ModSecurity
Audit Log Footer	Represents the end of the log, your goal is just to indicate the end of a log file

Figure 1 represents the relationships proposed in OntoSeg. As can be noted several branches show the complexity of the domain.

The basic relationships between the classes are performed using the property ID (transaction ID, with unique value to each transaction), derived from the log ModSecurity [15] that defines the ModSecurity_log main class, and from this we have the following derived classes:

- Response_header: contains all the information related to the HTTP header response
- Request_headers: contains all the information related to the HTTP request header
- Audit_log_header: contains all the information related to the header of IP and TCP
- There still is a set of information that derive from ModSecurity_log class that contains information about the HTTP message body from these subclasses we have the derivation of other subclasses that contains each of the basic elements of ontology OntoSeg.

Figure 3 shows an excerpt of the ModSecurity log, where you can see that there is no semantic description associated with its data, thus limiting the expressiveness of the concepts.

```

--a2ab5e2b-A--
[13/Apr/2010:13:31:01--0300]S8SAOMja0G0AAHXEGgAABEM2.2.2.34571
192.168.1.180
--a2ab5e2b-B--
GET /teste.asp?VARIABLE=!%20and%201=1%20and%20="" HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Host: www.example.com
Cookie: ASDF=ABC; ASPSESSIONIDQSCRBASD=PADSFAFASDADASdAADA
--a2ab5e2b-F--
HTTP/1.1 200 OK
X-Powered-By: ASP.NET
Content-Length: 88045
Content-Type: text/html; charset=ISO-8859-1
Expires: Tue, 10 Apr 2000 13:31:01 GMT
Vary: Accept-Encoding,User-Agent
--a2ab5e2b-E--

--a2ab5e2b-H--
Message: Warning. Pattern match
"(?:b(?:s(?:electb(?:{1,100}?b(?:?:length|count|top)b.{1,100}?bfrom|fromb.{1,
100}?bwhere).)*?b(?:d(?:umplb.*bfrom|data_type))|(?:to_(?:numbe|cha)|inst(r))|p_(?:
?:addextendedpro|sql|xex)c(?:oacreat|prepar)e|execute(?::sql)?|makewebtask)|qL(?:
..." at ARGS:VARIABLE. [file
"/etc/httpd/conf/extra/mod_security/modsecurity_crs_40_generic_attacks.conf"] [line
"66"] [id "950001"] [msg "SQL Injection Attack"] [data "and 1="] [severity "CRITICAL"]
[tag "WEB_ATTACK/SQL_INJECTION"]
Message: Warning. Pattern match "\b(id+) ?=\1\b|["'](\w+)|["'] ?= ?["']2\b" at
ARGS:VARIABLE. [file
"/etc/httpd/conf/extra/mod_security/modsecurity_crs_40_generic_attacks.conf"] [line
"70"] [id "950901"] [msg "SQL Injection Attack"] [data "1="] [severity "CRITICAL"]
[tag "WEB_ATTACK/SQL_INJECTION"]
Apache-Handler: proxy-server
Stopwatch: 1271169080460234 798372 (133 856 -)
Response-Body-Transformed: Dechunked
Producer: ModSecurity for Apache/2.5.9 (http://www.modsecurity.org/); core
ruleset/1.6.1.
Server: Apache
WebApp-Info: "www"."."."
--a2ab5e2b-Z--
    
```

Figure 3. Example of an excerpt from the log of ModSecurity, which represents a real SQL injection attack (some data has been changed to preserve confidentiality of the company)

For the creation of the ontology were chosen: a language for representing ontologies, a specific tool for working with ontologies, and some methodologies for the constructions of ontologies with defined roles. The below summarizes the choices that were made during the creation of ontology:

- Language for definition of the ontology: In order to create an ontology that can be semantically processed by computers, the OWL language was adopted, that has these characteristics, and currently is the language recommended by the W3C consortium [1].
- Methodologies for building ontologies: In the proposal, the following methodologies were used to develop ontologies [2]:
 - 101: Methodology of the simplified and interactive process, which has the following steps: Determine the domain and scope, consider reuse, list relevant terms, define classes and their hierarchy, define classes properties, set property values and creating instances.
 - Uschould and King: Methodology consists in four distinct stages: Identifying

the purpose of the ontology, construction, evaluation and documentation.

- Methontology: Methodology that suggests a life cycle of evolutionary model, composed by the following phases: planning, specification, knowledge acquisition, conceptualization, formalization, integration, implementation, evaluation, documentation and maintenance.
- Tool for the creation of the ontology: The Protégé [20] tool was chosen that allows the construction and edition of ontologies through a graphical interface of easy interaction. It also allows the insertion of new capabilities by installing plug-ins. In our case the OWL and Jambalaya plug-ins were installed. Other features of this tool are the importing and exporting of ontologies, open source and be developed in Java. Based on the comparative study of SILVA, Daniel Lucas; ROCHA SOUZA, Renato; ALMEIDA, Mauricio Barcelos [2], which presents the methodologies for building ontologies, three methods were selected according to the activities involved in the proposal, as described in Table 2.

TABLE II. METHODOLOGIES USED IN EACH STEP OF THE LIFE CYCLE OF THE ONTOLOGY

ACTIVITY	METHODOLOGY		
	Uschould e King	101	Methontology
Determination of the propose of the ontology.			
Definition of classes, properties and instances.			
Construction of the conceptual model.			
Implementation.			
Verification and validation.			
Maintenance.			

IV. RESULTS AND TESTS

Among the benefits of using ontologies are: the classification of the terms of the logs, relationships, inferences, formalization, reuse, sharing, among others, we will show some gains in the context of research.

To prove the improvements in research in the generated logs, was used the ontology described in the previous sections to analyze the logs. In addition, the ontology is necessary to use a query language, called SPARQL [18], which since January 2008 is recommended as the standard by the W3C Consortium [14] [19].

This language allows querying ontologies through clauses, which can combine several classes at the same time

and also make filters in the searches. A compiler for SPARQL [18] is already integrated in the Protégé tool [20].

To strengthen the evidence of the improvements in research involving the use of ontology in comparison with the traditional keyword searches, see below some situations:

1. A security auditor must analyze the log events that occurred in the 10th day of a month until the 10th day of the following month, that has the ip address range of 192.168.0.13 to 192.168.0.97 with destination ports 3306 and 443, and is on schedule from 21:30 to 24:00 h.
2. A security auditor wishes to know what IP address or group of addresses generated more attacks and in what times.

Considering the situations above, we have one, between two ways to proceed:

1. Search with the use of Ontology: It would be enough to use the select command with the filter clause containing the classes mentioned above, and be answered with only the desired data to analyze.

It could also create a search interface using the language SPARQL query language or another, to prevent the auditor to need to know and type the commands queries.

2. Searches without the use of ontologies :

Quite difficult to be generated because that for the auditor to make the analysis he wants, he would first have to read many logs manually separating by keyword and then make de co-relation, with the risk of losing data, since only the search for information, he will be ignoring the other data you want.

Below are the consult solutions in SPARQL to the situations described above:

```
SELECT ?ID ?DestinationIP ?DestinationPort ?SourceIP
?SourcePort ?Timestamp ?Cookie ?mod_security-message
WHERE { ?ID rdf:type :ID . ?DestinationIP rdf:type
:DestinationIP .
?DestinationPort rdf:type :DestinationPort . ?SourceIP
rdf:type :SourceIP . ?SourcePort rdf:type :SourcePort .
?Timestamp rdf:type :Timestamp . ?Cookie rdf:type
:Cookie . ?mod_security-message rdf:type :mod_security-
message
FILTER((?TimeStam > xsd:date("2010-07-09") &&
?TimeStam < xsd:date("2010-08-11") &&
(?DestinationIP > 192168012 && ?DestinationIP <
192168098) && (?DestinationPort = 3306 ||
?DestinationPort = 443) && (?TimeStam >
xsd:time("21:29:00") && ?TimeStam <
xsd:time("24:00:01")) ) }
```

Figure 4. SPARQL [18] consult in the Ontology, Situation 1

```
SELECT ?ID ?SourceIP ?DestinationIP ?Timestamp
WHERE { ?ID rdf:type :ID . ?SourceIP rdf:type
:SourceIP . ?DestinationIP rdf:type :DestinationIP .
?Timestamp rdf:type :Timestamp
GROUP BY
?SourceIP } ORDER BY ASC(?SourceIP)
```

Figure 5. SPARQL Consult in the Ontology, situation 2

In this sense, the implemented ontology fulfilled its role very well, according to what was previously planned, the searches were carried out in a simple way in the ontology producing the most interesting and visible results in comparison with the traditional consultations using only key words, obtaining better results for event logs identification.

It is seen that with the approach proposed in this paper, the activity log analysis was made simple and independent of the complexity of the log and the generated data volumes allowing the realization of co-relations between events more efficiently

V. CONCLUSION AND FUTURE WORKS

This study aimed to take the first steps in using ontologies for analysis of security logs. For that purpose the logs generated by the program ModSecurity were initially used. As a starting point the log that was generated by this tool on a web server of CESAR (Center for Advanced Studies and Systems of Recife) was used. The ontology modeling was accomplished from the understanding of the logs the model the ontology.

The performed tests proved that there was an easier log interpretation and analysis, allowing the performing of more complex consultations and the implementation of co-relation of events very effectively.

Finally, we proved that the demand for log audits that use ontologies is very large, for the tools and current research procedure is very limited, constituting a critical point in analyzing logs. In this context, this work was made to contribute to the attending of this demand.

ACKNOWLEDGMENT

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES <http://www.ines.org.br>), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.

REFERENCES

- [1] D. Allemang and J. Hendler, Semantic Web for the Working Ontologist, Effective Modeling. in: RDFS and OWL, Cambridge, Morgan Kaufmann, 2008.
- [2] M. B. Almeida and M. P. Bax, Uma Visão Geral sobre Ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção in Cin:Inf., Brasília, 2003.
- [3] A. Grigoris and V. H. Frank, A Semantic Web Primer, Second Edition. London, The Mit Press, 2008.
- [4] T. Berners-lee, J. Hendler, and O. Lassila, The Semantic Web, Scientific American Publishing, New York, 2001.

- [5] A. Brandão, A. R. A. Franco, F. Lucena and C. J. Pereira, Uma Introdução à Engenharia de Ontologias no contexto da Web Semântica, Rio de Janeiro, 2002.
- [6] K. Breitnam., Web Semântica, a Internet do Futuro. Rio de Janeiro, LTC Publishing, 2005.
- [7] R Cannings, D. H. Lackey, and H. Zane. Hacking Exposed™ WEB 2.0: Web 2.0 Security Secrets And Solutions. New York: Mc Graw Hill, 2008. - DOI: 10.1036/0071494618
- [8] M. C. Silveira. Um estudo sobre XML, Ontologias e RDF(S), http://www.inf.pucrs.br/~mchaves/pg_portugues/tc/paperxml.pdf, Accessed on 01/10/2010.
- [9] T. Gruber and R. Toward, principles for the design of ontologies used for knowledge sharing. In Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publishers, 1996.
- [10] P. N. Guarino and A. Roberto, Formal ontology, conceptual analysis and knowledge representation. International Journal of Human-Computer Studies, Volume 43 Issue 5-6, Nov./Dec. 1995
- [11] R. Kimball and R. Merz, The Data Webhouse Toolkit, New York, John Wiley and Sons, Inc, Wiley Computer Publishing 2000.
- [12] M. Lytras, D. Damiani, P. Ernesto and Patricia O. WEB 2.0 The Business Model. New York: Springer, 2009. ISBN-13: 978-0-387-85894-4
- [13] D. L. Mcguinness and F. V. Harmelen. OWL Web Ontology Language , <http://www.w3.org/TR/owl-guide/>, Accessed on 10/10/2011
- [14] Overview. W3C World Wide Web Consortium (<http://www.w3.org/TR/owl-features/>), 2004. Accessed on 08/16/2011
- [15] Modsecurity. Breach, ModSecurity 2 Data Formats, 2009, Copyright © 2004-2009 Breach Security, Inc. (<http://www.breach.com>). Accessed on 08/16/2011
- [16] N.F. Noy and D. L. McGuinness, "Ontology Development 101: A Guide to Create Your First Ontology", <http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf>, Accessed on 10/10/2011
- [17] Owasp, 2008, owasp testing guide 2008 v3.0.: http://www.owasp.org/index.php/category:owasp_testing_project Accessed on 08/16/2011
- [18] E. Prud'hommeaux and A. Seaborne, SPARQL Query Language for RDF, <http://www.w3.org/TR/rdf-sparql-query/>, Accessed on 10/10/2011
- [19] W3C- World Wide Web Consortium (<http://www.w3.org/TR/rdf-sparql-query/>). Accessed on 08/16/2011.
- [20] Protégé; Ontology Editor and Knowledge Acquisition System (<http://protege.stanford.edu/>>). Accessed on 08/16/2011
- [21] R. Studer, et al., Situation and Perspective of Knowledge Engineering, Stanford University, http://infolab.stanford.edu/~stefan/paper/2000/ios_2000.pdf, Accessed 10/10/2011 .
- [22] D. Stuttard and M. Pinto, The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws, 2008.
- [23] Us-cert - technical cyber security alerts, 2009. <http://www.us-cert.gov/cas/techalerts/> . Accessed on 08/16/2011
- [24] W3C, 2010, <http://www.w3.org/>. Accessed on 08/16/2011.
- [25] Stanford Jambalaya plug-in, <http://protege.stanford.edu/plugins/jambalaya/jambalaya-simple-backup.htm> , Accessed on 10/10/2011

Intrusion Detection with Symbolic Model Verifier

Ines Ben Tekaya

PRINCE Laboratory
4011 Hammam Sousse, Tunisia
bentekaya.ines@voila.fr

Mohamed Graiet

MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
mohamed.graiet@imag.fr

Bechir Ayeb

PRINCE Laboratory
4011 Hammam Sousse, Tunisia
ayeb_b@yahoo.com

Abstract— Many intrusions came from internal users. This behavior can cause damage without human intervention: viruses, worms, trojan horses, etc. This paper describes our intrusion detection method in Linux/Unix commands using formal verification. The main features of this work are twofold. It exploits formal method in the intrusion detection field. It presents our tool TLID which can transform Linux code to Symbolic Model Verifier.

Keywords— attacks; intrusion; security; scenarios; Linux commands; model verifier.

I. INTRODUCTION

The intrusion field was introduced by Anderson. It was defined as an attempt or a threat to be the potential possibility of a deliberate unauthorized attempt to access information, manipulate information, or render a system unreliable or unusable [1]. The difference between intrusion and attack consists of the fact that intrusion is a malicious, externally or internally induced fault resulting from an attack that has succeeded in exploiting vulnerability, while a fault is the adjudged or hypothesized cause of an error, the cause of which is intended to be avoided or tolerated. An attack is a malicious technical interaction fault aiming to exploit vulnerability as a step towards achieving the final aim of the attacker [2].

A statistical study shows that 98% of enterprises have a firewall to be protected from external attacks; however, 80% of attacks came from internal users [3]. Detecting internal normal user behavior is a difficult problem because a user can have much dynamic behavior and it will be almost impossible to create user profiles that determines the normal behavior. Using a system to distinct normal user from intruders is necessary. This system is called Intrusion Detection System (IDS). It is defined as a security technology attempting to identify and isolate computer systems intrusions [4].

We choose to work with Unix/Linux operating system because in people's minds, if it is non-Windows, it is secure [5]. This hypothesis will be countered here. More details for Unix/Linux system can be found in [6].

The literature on detection using Linux/Unix commands offers a variety of methods. Despite their diversity, their common objective is: to distinguish between a normal behavior and an intrusive behavior. From an abstract view point, we organize these work into one main approach:

empirical approach. This classification included methods based on aggregative, training or experimental past data. The present work falls mainly within the model approach. The data are not based in the past event but they compose a model. It is a theoretical representation of a system which is composed of elements and relation.

The reminder of the paper is organized as follow. Section 2 deals with intrusion background. Section 3 describes our method. Section 4 proposes practical tool and experimental results for intrusion scenarios. Section 5 summarizes the paper, with concluding remarks.

II. INTRUSION BACKGROUND

The next subsections summarize attacks topology, some dataset used in the literature for intrusion detection and show detection methods using Unix commands

A. Attacks topology

Attacks take several forms to break one or more of the security properties. They can be grouped according to their functionality as described in the following subsections [7]:

- **Gathering Security-relevant Information:** Before experiencing an attack, a hacker tries to obtain necessary information that is probably sensible about the targeted system, which can be employed later to obtain access to this system. Useful information can be obtained by different ways such as network scanning and vulnerability scanning or even by using public search engines such as Google or social engineering methods.
- **Access Gain Attacks:** With information gathered by the above methods, attackers try to obtain a privileged access on a system by exploiting vulnerabilities in the services or the applications installed on this system or a bad configuration of the network. This kind of attacks primarily grants unauthorized access to the targeted system. For example, one of the configuration problems is the use of weak passwords in systems where a bad policy of password definition allows users to choose simple and easy guessable passwords. Otherwise, an attacker can use cracking tools such as “john the ripper” [8] to obtain passwords by brute-force. Buffer-overflow attacks are another example that allows attackers to execute arbitrary code on the targeted hosts.

- Denial of service (DoS): DOS attacks are designed to overload or disable the capabilities of a machine or a network, and thereby render it unusable or inaccessible. An example of denial of service is a fork bomb. It works by creating a large number of processes very quickly in order to saturate the available space in the list of processes kept by the computer's operating system. If the process table becomes saturated, no new programs may start until another process terminates.
- Malware Attacks: This category of attacks can result in damages as simple as displaying a simple flicker to catastrophic damages such as completely formatting hard disks. It groups virus, worm, Trojan horse, spyware, rootkit [9] and spam.

B. Detection Using UNIX Commands

The object of intrusion can be files, data bases, network connection, Input/output systems or commands Linux/Unix. In this paper we are interested about intrusion using Linux/Unix commands because it can characterize user behaviour more efficiently than other object. The followings paragraphs present some works about methods using Unix commands. These works are classified into two classes: the class of intrusion detection and the class of masquerade detection.

Ilgun, et al. present the state transition analysis method [10][11]. They used the known Unix intrusion to create a penetration scenario. A penetration is viewed as a sequence of actions performed by an attacker that leads from some initial state on a system to a target compromised state, where a state is a snapshot of the system representing the values of all volatile, semi-permanent and permanent memory locations on the system. The initial state corresponds to the state of the system just prior to the execution of the penetration. The compromised state corresponds to the state resulting from the completion of the penetration. Between the initial and compromised states are one or more intermediate state transitions that an attacker performs to achieve the compromise.

This method is based on sequence matching. The incoming stream event is segmented into overlapping fixed-length sequences. The choice of the sequence length, l , depends on the profiled user. In practical, it's fixed to the value $l = 10$ in the SEA dataset [12]. Each sequence is then treated as an instance in an l -dimensional space and is compared to the known profile. The profile is a set, $\{T\}$, of previously stored instances and comparison is performed between all $y \in \{T\}$ and the test sequence via a similarity measure. Similarity is defined by a measure, $\text{Sim}(x, y)$, which makes a point-by-point comparison of two sequences, x and y , counting matches and assigning greater weight to adjacent matches.

The maximum of all similarity values computed forms the score for the test command sequence. Since these scores are very noisy, the most recent 100 scores are averaged. If the average score is below a threshold an alarm is raised. The threshold is determined based on the quantiles of the empirical distribution of average scores [13].

Another method, used statistical method, is called uniqueness. It is based on the idea that commands not previously seen in the training data may indicate an attempted masquerade. Uniquely used commands account for 3% of the data. A command has popularity i if exactly i users use that command. They group the commands such that each group contains only commands with the same popularity. They define a test statistic that builds on the notion of unpopular and uniquely used commands. They assign the same threshold to all users. This threshold is estimated via cross validation: They split the original training data in the SEA dataset into two data sets of 4000 and 1000 commands. Using the larger data set as training data, they assign scores for the smaller one. This is repeated five times, each time assigning scores to a distinct set of 1000 commands. They set the threshold to the 99th percentile of the combined scores across all users and all five cross validations. For their data, the resulting threshold is 0.2319 [12][14].

Another method is called Bayes 1-Step Markov Model. It is proposed by Schonlau, et al. The authors use the information of 1-step command transition probabilities. They build transition matrices for each user's training and testing data. The detector triggers the alarm when there is a considerable difference between the training data transition matrix and the testing data matrix. This technique was the best performer in terms of correct detections, but failed to get close to the desired false alarm rate [12].

Maxion use Naive Bayes classifiers and detect masqueraders by looking at the classifiers misclassification behavior [15]. This method use command occurrence probability distribution modeling the UNIX sequence. The goal of the training procedure is to establish profiles of self and nonself, and to determine a decision threshold for discriminating between examples of self and nonself. For each User X in the SEA dataset, a model of Not X can also be built using training data from all other victims. The probability of the test sequence having been generated by Not X can then be assessed in the same way as the probability of its having been generated by User X . The larger the ratio of the probability of originating with X to the probability of originating with Not X , the greater the evidence in favor of assigning the test sequence to X . The exact cut-off for classification as X , that is the ratio of probabilities below which the likelihood that the sequence was generated by X is deemed too low, can be determined by a cross-validation experiment during which probability ratios for sequences which are known to have been generated by self are calculated, and the range of values these legitimate sequences cover is examined.

C. Limitations in existing methods

The intrusion detection method in Linux/Unix commands using formal verification seeks to improve on some of limitations that the authors observed in the existing methods. This section briefly identifies some of their characteristics.

The major weakness of these methods is that they depend on aggregative, training or experimental past data. The results of static methods are closed to the training data

while the result of state transition analysis method is depend with the defined penetrations attacks which are non valuable now.

Another limitation is they are based on analysing command by command (line per line). This local analysis can not be equivalent to a global analysis (all of lines).

Lastly, they cannot make difference between the orders of commands in the sequence used. The statical methods are based on the command frequency while a state transition analysis method can't detect the attacks based in frequency such as deny of service.

In the following, we focus in these limitations to present our method based on model using formal verification with Symbolic Model Verifier (SMV).

III. INTRUSION DETECTION IN LINUX/UNIX COMMANDS WITH SMV

This section presents our method. It combines tests on the direct and indirect ways to detect the intrusions. It focuses on global analysis. The following proposition plays a central role here.

Proposition 1. *A global analysis can not be realized in k local analysis.*

Example 1. Let GA is a global analysis and $LA = \{u_1, u_2, \dots, u_k\}$ a k local analysis. Suppose that GA can be realized in k local analysis. In this case, if GA is false, we must have one or more u_i is false.

This supposition is false because we can find GA is false while LA is true. The example is here: We have two users X and Y . User X can execute the following actions : modify all executable files, named F and that he have write permission, owned by user Y . X append some code to files F . When any users, that have write permission in these file, execute F , all F files will be infected. These actions can be:

1. X search all Y executables files, that X have write permission,
2. X append some legal code to infect files F
3. Any authorised users execute one of F files
4. All F files will be infected

The local analysis for actions 1, 2 and 3 are legal. They have a true value, but the global analysis gives a false value: all F files will be infected.

To perform a global analysis we should specify what are the anti-properties that characterize an attack script.

The anti-properties (AP) are unwanted properties that can cause damage in our system. They can be:

- AP1: Execute some illegal commands,
- AP2: Change source or command destination,
- AP3: Execute illegal actions (parameters, etc.),
- AP4: Having infinite loop,
- AP5: Having auto-replication,
- AP6: Detain a resource infinitely
- ...

The system specification are formalizes using the AP. They can be expressed in proportional logic or temporal logic.

Propositional logic is the branch of logic that studies ways of joining and/or modifying entire propositions,

statements or sentences to form more complicated propositions, statements or sentences, as well as the logical relationships and properties that are derived from these methods of combining or altering statements.

The temporal logic is used within the framework of the reagent systems, which where the software is supposed to maintain a relation of coherence between the input flows and the output flows. The temporal logic allows expressing the state evolution of a system.

We choose the temporal logic because temporal logic is an extension of propositional logic. Either in temporal logic, propositions are qualified in terms of time.

The following paragraph explains how to write the anti-properties AP to properties (P) using temporal logic.

AP1: Execute some illegal commands

The AP1 consider that user can execute some commands. For example, if the user is an administrator, he can execute commands like adduser, userdel, etc.

P1: Do not execute some illegal commands

$$P1 = \{(U_i, C_j) / U_i \in U \text{ et } C_j \in C\}$$

where: U : set of users

C : set of illegal commands

(U_i, C_j) : U_i can use C_j

$Use(U_i, C_j) \rightarrow (U_i, C_j) \notin P1$

AP2: Change source or command destination

The AP2 consider that the command path was modified.

P2: Do not change source or command destination

$$P2 = \{(U_i, F_j) / U_i \in U \text{ et } F_j \in F\}$$

where: U : set of users

F : set of illegal folder

(U_i, C_j) : U_i can't write on F_j

$Write(U_i, F_j) \rightarrow (U_i, F_j) \notin P2$

An example is: `write(user1, /bin/cp)`

AP3: Execute illegal actions (parameters, etc.),

The AP3 consider that some user can use or modify objects of other users that he don't have a permission.

P3: Do not execute illegal actions (parameters, etc.)

$$P3 = \{(U_i, O_j) / U_i \in U \text{ et } O_j \in O\}$$

where: U : set of users

O : set of illegal objects.

(U_i, O_j) : U_i can read O_j

$Read(U_i, O_j) \rightarrow (U_i, O_j) \notin P3$

AP4: Having infinite loop

The AP4 consider that user can modify the system performance. So they consume memory to overload the system.

P4: Do not have infinite loop

$$AP4 = G \wedge \neg (a_i \wedge a_j)$$

let: G : always

\wedge : and operator

\neg : not operator

a_i : loop and a_j : loop condition

An example is: `while(true)`, `while(i := i+1)`, etc.

Some others anti-properties can be formalized such as having auto-replication detain a resource infinitely, etc.

The user observed behavior is the possible behavior. It is deduced from Linux/Unix terminal. We are interested about a script not about a line of commands.

In this paper, we concentrate on formal verification technique that is based on temporal logic, because that allows in general less involvement of the user in the verification process: model checking.

Our basic idea is to exploit model checking. This model use algorithms, executed by computer tools, to verify the correctness of our system. The user inputs a description of a model of the system (the possible behavior) and a description of the requirements specification (the desirable behavior) and leaves the verification up to the machine. If an error is recognized the tool provides a counter-example showing under which circumstances the error can be generated. The counterexample consists of a scenario in which the model behaves in an undesired way.

In the rest of this paper, we use the term Linux, which can be interchanged with Unix. Our method is based in the user's observed behavior and in the system specification. The user's observed behavior is modeled by a Linux script. It will be transformed into SMV code. However Linux script differs from SMV code. We propose LSc2SMV (Linux Script to Symbolic Model Verifier) tool to do the transformation.

--The user observed behavior is transformed by our proposed tool, named LSc2SMV (Linux Script to Symbolic Model Verifier), to SMV code.

We obtain a SMV program containing logical properties which we verify by SMV tool. The result will be verified properties if the behavior is normal or violated properties if the behavior is intrusive. Figure 1 illustrates this schema.

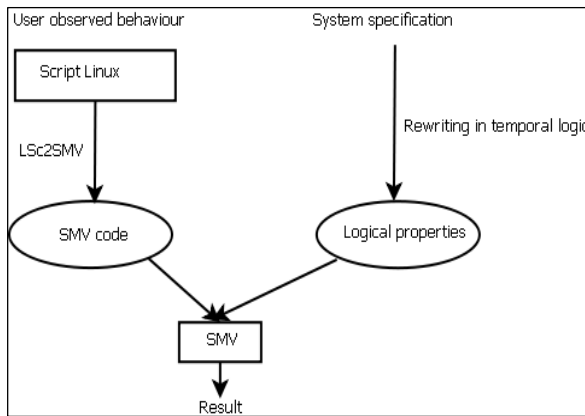


Figure 1. A diagram tracing our method.

The LSc2SMV tool will convert Linux script to an SMV code. It will be in the form of main module (). We show the transformation in constant, in variables, in arrays, in expressions, in functions, and in loops and conditions. Tables I, II, III, IV and V give this direct transformation.

Table I shows the transformation in constants and variables.

TABLE I. VARIABLES AND CONSTANTS CASES

Type	LSc	SMV
Integer variable	varname = valeur	VAR <signal> : number ;
Variable of an interval	for i in 0 1 2 3 4	VAR <signal> : 0..4 ;
Constant	SIZE=32	#define SIZE 32
Initialisation	signal = ready	init(signal) := ready ;
Modification	signal = busy	next(signal) := busy ;

Table II shows the transformation in arrays cases.

TABLE II. ARRAYS CASES

Type	LSc	SMV
Array	declare -a nametab	<nametab> : array <x>..<>y> of <type> ;
Matrix	char mat[2][2]	mat : array 0..1 of array 2..0 of boolean ;

Table III shows the transformation in expressions cases.

TABLE III. EXPRESSIONS CASES

Type	LSc	SMV
Boolean operators	-a (and) -o (or) !(not)	("and","or","not")
Condition operators	if-then-else case switch	if-then-else case switch
Arithmetical operators	+, -, *, /, %	+, -, *, /, mod
Comparison operators	-eq, -ne, -lt, -gt, -le, -ge	"=", "<", ">", ">=", "<=")

Table IV shows the transformation in the function case form.

TABLE IV. FUNCTION CASE

Type	LSc	SMV
function	function name() {...} ;	MODULE name(input, output) {...}

Table V shows the transformation in the condition and loop cases form.

TABLE V. CONDITIONS AND LOOP CASES

Type	LSc	SMV
Condition	if[<condition>] <stmt1> else <stmt2> fi	if(<condition>) <stmt1> else <stmt2>
Case	case \$variable in val1) stmt1) ; ; *) <stmtn> ; ; esac	case { <cond1> : <stmt1> ... <condn> : <stmtn> [default : <dftlstmt>]}
Switch	switch(<expr>) <case1> : <stmt1> breaksw <casen> : <stmtn> breaksw default : <dftlstmt> breaksw	switch(<expr>){ <case1> : <stmt1> ... <casen> : <stmtn> [default : <dftlstmt>]}

	endsw	
for	for var in \$files ; do	for(var = init ; cond ; var = next) <stmt>
while	while condition ; do <stmt> done	-

The indirect transformation is based on properties to verify and in Linux script.

Some other conversion in the file name or in the folder name can be made. This is because SMV cannot support some character like . or / in the variable name. The Table VI gives some conversion.

TABLE VI. NAME TRANSFORMATION

Type	LSc	SMV
File name	/etc/passwd,/etc/inittab, /etc/ld.so.conf, etc/lilo.conf,etc/group	etcpasswd,etcinittab, etcldsoconf,etcililoconf, etcgroups
Folder name	/var,/usr/bin,/dev, /etc/security, /var/spool,/etc, /usr/etc,/usr,/usr/lib/	var,usrbin,dev,etcsecurity, varspool,etc,usretc,usr, usrlib,slash

IV. TLID: TOOL FOR LINUX INTRUSION DETECTION

There are two solutions to survey a user:

- The first solution consists in using the file .bash_history. But this file cannot give a strengthened and real-time history because when you use other shell, like csh., this method cannot save the history. Either when you type kill -9.
- The second solution is to develop a patch. It consists to modify file system which are bashhist.c, histexpand.c, histfile.c, history.h and history.c (to obtain the patch e-mail : bentekaya.ines@voila.fr). When a user writes anything in the console, it will be saved in a file using his name. This patch can be used in every system to survey a command user.

Figure 2 gives some functionality of TLID. You can choose a user, a day and we obtain the behavior. It is composed by time, PID and commands.

After that you can choose a property to verify. In this example, we choose to verify the use of illegal parameters. The button LSc2SMV became enabling. When we click below, we obtain the SMV file. This file contains the verification of action 1: cd /tmp and action 2: cp /etc/ld.so.conf /tmp. It consists to verify the permission of using folder /tmp and /etc/ld.so.conf file. This is given by SMV file in Figure 4. The two properties we specified are file confidentiality (conf) and folder confidentiality (confo). We choose "Prop|Verify all" to verify if the properties we specified in fact hold true or false for all time. The result is given by Figure 5. The conf property should be false, and a counterexample appears in the trace page. This because ines user use a file that he don't have a permission.

TLID can do a local analysis a global analysis between users.

Intrusion scenario Sc between users can be defined as:

Sc = {A, V, S} with:

A: an attacker

V: a victim

S = {s1, s2... sn}: a set of steps

Every step is a sequence of commands with their parameters. The next paragraph shows an example of scenario. It have been developed and tested in Linux Red Hat Enterprise version 5 and we use TLID and SMV for verification.

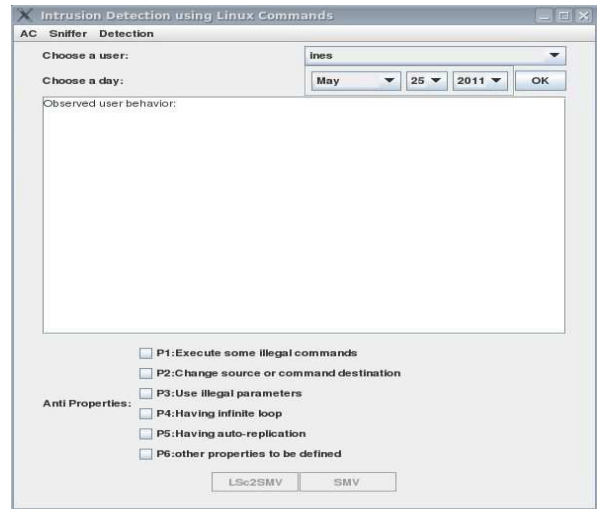


Figure 2. TLID

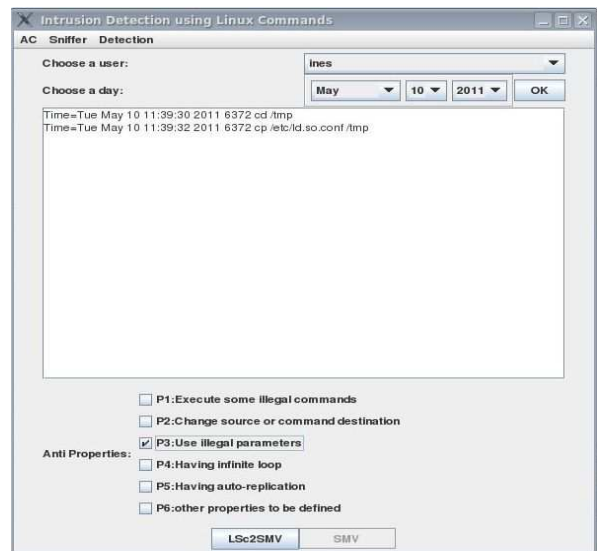


Figure 3. Observed ines behavior in May-10-2011

```

View SMV file
module main(lectfo,paramfo)
{
  lectfo : boolean;
  paramfo : {tmp};
  folder_nwrite_user1 : {var,usrbin,dev,etc,security,varspool,etc,usr,usrlib,slash};
  init(lectfo) := 0;
  init(paramfo) := {tmp};
  if (paramfo in folder_nwrite_user1)
  {
    next(lectfo) := 1 ;
  }
  else
  next(lectfo) := 0 ;
  lect : boolean;
  param1 : {etc,liloconf};
  file_nread_user1 : {etc,passwd,etc,inittab,etc,ldsoconf,etc,liloconf,etc,groups};
  init(lect) := 0;
  init(param1) := {etc,liloconf};
  if (param1 in file_nread_user1)
  {
    next(lect) := 1 ;
  }
  else
  next(lect) := 0 ;
  conf : assert G (param1 -> lect = 0);
  conffo : assert G (paramfo -> lectfo = 0);
}
    
```

Figure 4. SMV file

Using TLID, we choose to the anti property: Having infinite loop. If we don't know how a property to choose, we can mark all checkbox. The result is given by Figure 7.

Property	Result
loop	false

Figure 7. The result

Property	Result
conf	false

Figure 5. Verification with SMV

V. CONCLUSION

In this paper, we are interested by attacks using Linux commands. We have presented their topology. We have shown that their impact can be inoffensive or can destroy information system.

We have proposed a method that exploits model checking. This model use algorithms, executed by computer tools, to verify the correctness of our system. It combines security field with formal verification. The user inputs a description of a model of the system (the possible behavior) and a description of the requirements specification (the desirable behavior) and leaves the verification up to the machine. If an error is recognized the tool provides a counter-example showing under which circumstances the error can be generated. The counterexample consists of a scenario in which the model behaves in an undesired way.

This method is applied to distinct normal user behavior from intruders' behavior. It has lead to the TLID tool development. We give some experimental results to show how the TLID works under some attacks.

There is another attacks group which can be named unknown attacks. In this new group, attacks could cause the intrusion detection systems crash and thus incomplete testing. It becomes clear that present approaches to evaluate intrusion detection system are limited to some known attacks.

We divide our future work into two main parts: refine and improve attacker competence and extend scenario to include multi-attacks and equivalent attacks.

REFERENCES

- [1] J. P. Anderson, "Computer Security Threat Monitoring and Surveillance," Technical report, Washing, PA, James P. Anderson Co., 1980.
- [2] D. Powell and R. Stroud, "Conceptual Model and Architecture of MAFTIA", Eds., MAFTIA (Malicious and Accidental Fault Tolerance for Internet Applications) project deliverable D21, LAAS-CNRS Report 03011, 2003.

We have two users. The victim is named 'troismille' (user-id: 3000) and the attacker is named 'ines' (user-id: 5502).

```

[root@localhost ~]# cat /etc/passwd
Result:ines:x:5502:5502::/home/ ines:/bin/bash
troismille:x: 3000: 3000::/home/ troismille:/bin/bash
    
```

This scenario consists of sending many mail from user ines to user troismille to saturate his mail. In this case, the user troismille cannot access to his e-mail. The scenario is given by Figure 6.

```

ines@localhost:~$ while true;
do
mutt -s "subject" -a fiche.txt troismille@localhost.localdomain <corps.txt;
done &
    
```

Figure 6. An example of scenario

- [3] C. Matheï., (2004) "Ouverture des réseaux IP d'entreprise : risques ou opportunité ?" [Online]. Available: [http://www.awt.be/contenu/tel/res/IPforum23-04_Réseau unifié et sécurisé.pdf](http://www.awt.be/contenu/tel/res/IPforum23-04_Réseau_unifié_et_sécurisé.pdf).
- [4] B. E. Cloete and L. M. Venter, "A comparison of Intrusion Detection systems" *Computers & Security*, vol 20, Issue 8, pp. 676-683, Dec. 2001.
- [5] A. Patrizio. (2006) "Linux Malware On The Rise. " [Online]. Available: <http://www.internetnews.com/dev-news/article.php/3601946>.
- [6] M. Santana, "Chapter 6 - Linux and Unix Security, Computer and Information Security" Handbook 2009, pp. 79-92.
- [7] M. E. S. Gadelrab, "Évaluation des Systèmes de Détection d'Intrusion," thèse, Université de Toulouse - Paul Sabatier, France, Dec. 2008.
- [8] M. F. Krafft (2007) "John the Ripper password cracker:" [Online]. Available: <http://www.openwall.com/john/>.
- [9] G. Hoglund, and J. Butler, "Rootkits: Subverting the Windows" Kernel, Addison-Wesley Professional, 2005.
- [10] Koral Ilgun , Richard A. Kemmerer , Phillip A. Porras. "State Transition Analysis: A Rule-Based Intrusion Detection Approach. " *Journal IEEE TRANSACTIONS on Software Engineering*, Vol. 21, No. 3, pp. 181-199, 1995.
- [11] K. Ilgun. "USTAT - A Real-time Intrusion Detection System for UNIX," Master's Thesis, University of California at Santa Barbara, Nov. 1992.
- [12] M. Schonlau, W. DuMouchel, W. H. Ju, A. F. Karr, M. Theus and Y. Vardi. "Computer Intrusion: DetectingMasquerades" *Statistical Science*, Vol. 16, No. 1,pp 1-17, 2001.
- [13] T. Lane and C E. Brodley. "Sequence matching and learning in anomaly detection for computer security." In *AAAI Workshop : AI Approaches to Fraud Detection and Risk Management*, pp. 43-49. AAAI Press (1997).
- [14] M. Theus and M. Schonlau. "Intrusion detection based on structural zeroes." *Statistical Computing and Graphics Newsletter* 9, pp. 12-17, 1998.
- [15] M. Roy. "Masquerade detection using enriched command lines." In: *Proceedings of international conference on Dependable Systems and Networks (DSN-03)*, pp. 5-14, June 2003.

Security Quality Assurance on Web Applications

Rodrigo Elia Assad^{1,2}, Felipe Ferraz^{1,2}, Henrique Arcoverde³, Silvio Romero Lemos Meira^{1,2}

¹Centro de Estudos e Sistemas Avançados do Recife(CESAR) - Recife – PE – Brazil

²Centro de Informática
Universidade Federal de Pernambuco (UFPE) – Recife, PE – Brazil

³Tempest Security Intelligence

assad@cesar.org.br, fsf3@cin.ufpe.br, henrique@tempest.com.br, srlm@cesar.org.br

Abstract: Historically, it is well known that issues related to security of software applications are normally omitted by the development teams owing to a lack of expertise or knowledge in security policies. With the emergence of WEB technologies, this situation became more serious. Entire systems, complex or not, have outstanding access availability and therefore are highly vulnerable to threats. This work aims to discuss how the security requirement, design patterns and tests should be elaborated in order to making easier the execution of its tests and consequently improving the quality of the solution developed.

Keywords-security requirements; design patterns; security tests validation, quality assurance

I. INTRODUCTION

Since the popularization of the Internet through its commercial use occurred during the decade of 1990, attacks on computer systems have become more frequent. Initially the attacks was more focused on operating systems and network services, as can be seen in the attacks reports generated from various institutes such as CERT [47].

With the rapid growth of attacks, companies, governments, universities invested heavily in security solutions, such as firewall, intrusion detection systems, anti-virus, patch management and so on. Also there was investment on development of security procedures and processes for managing information, such as ITIL, COBIT and SOX [46]. And also a definition of specific legislation to support the security analysts.

All these initiatives, associated with maturation time, related to security issues comprehension and security standards adoption, occurred from 1997 to 2007; the rate of attacks reported to security holes in operating systems and computer networks have decrease significantly, as shown in Figure 1.

Analyzing these numbers, we see that the definition of procedures, comprehension of security flows produces an improvement on a perceived security quality - QA - Quality Assurance - in relation to services provided by system administrators, security consultants and security engineers that support computer networks and operation system.

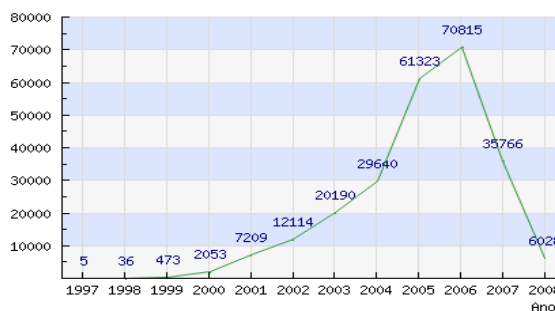


Figure 1: Attacks on network operating systems and reported by CERT.br [47]

The improvement and maturation of Security Quality Assurance procedures do network and operation systems resulted in change of security focus, now applications have become the primary target.

It is undeniable that the security problems still persist, however, are not only related to flaws in operating systems or network services, but the major focus has changed and is currently in web applications, as seen in Figure 2.

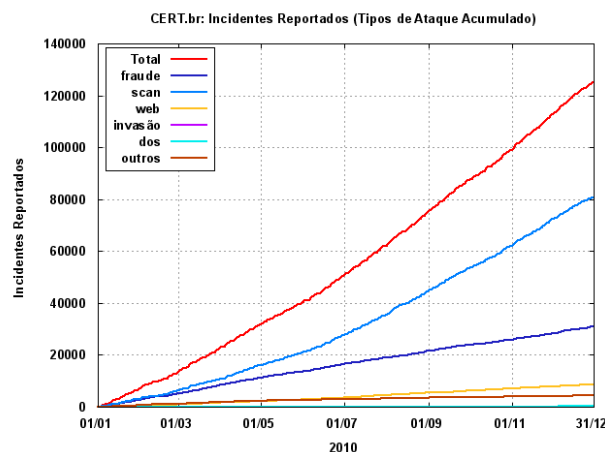


Figure 2: Attacks reported on 2010 to Cert.br [47]

The attacks on web applications and began more popular on 2007. It's can be evidenced in several ways, among them, through consultations on Google cache

showed on Figure 4. As observed before 2007 there are few records of consultations about web application security. Using the same methodology shown in Figure 3 queries related to network security - a subject of greater scope related to operating system security and network services - has been decreasing year after year as a result of the quality assurance process described before.

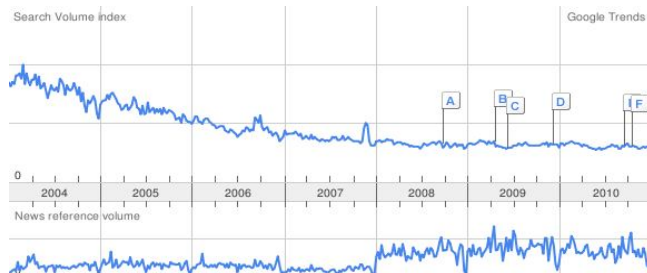


Figure 3: Query to Google on “network security”[50]

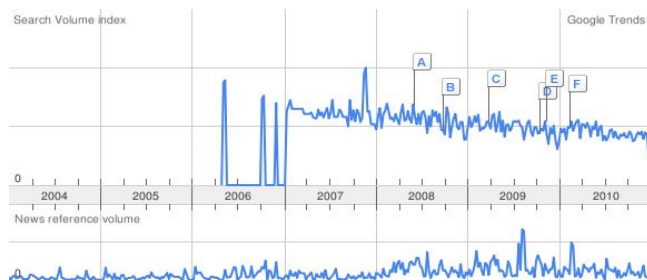


Figure 4: Query to Google on “web application security ”[50]

Another fact should be observed that’s collaborate to attacks migration to web applications, we have in this period the emergence of applications WEB2.0, Web3.0 [37], consolidation of the browser as a gateway to all applications, the emergence of API’s development as the proposed Google apps and Microsoft Live and more recently the cloud computing [38][44][48].

We can believe that, as happened as with the attacks on operating systems and network services, to protect web application we will require efforts on research development, new product development and procedures specifications, and consequently the maturation of software developers in order to improve the code produced for web applications, it can be called SSQA: Security Software Quality Assurance:.

To define the security Quality Assurance demanded by an application, it is required experienced and trained stakeholders with abilities in all disciplines with a focus on security. Throughout the development of a web application, it is important that the activities of elicitation and specification of requirements to be followed by architecture definition and a process of validation. It is essential to track and approve if the security requirements are being satisfied on applications. The traceability of functional and non-functional security requirements is naturally a complex task,

because in general, they affect the entire system. To carry out successful safety tests on a application, is necessary to identify what types of vulnerabilities could enable attacks on the system. In other words, we must understand where the flaws may be present to learn how to avoid them.

From this point of view, we identified that one of the possible causes to security flaws relies on the low quality of software security requirements and consequently in its implementation, validation and tests phases.

It should consider the present scenario of IT companies in relation to technologies used in the development of web applications, we have the main highlights:

- a) Use agile methodologies
- b) Software reuse
- c) Development framework

This section presents a proposal for the integration of the above themes, throws specifying a security quality assurance process that can be used by companies to promote the development of secure applications on certain assumptions, keeping the agreed deadlines and focusing on quality assurance of the safety of software. It’s examines the possibility of adopting the same methodology used successfully between 1997 and 2007 that brought a significant drop in network security problems, they are:

- a) Understanding of attacks and its operating mechanism
- b) Development of defense models in relation to existing technology
- c) Adopting an agile and reusable
- d) Establishment of a pricing mechanism for the easy development of secure solutions.

II. SECURITY QUALITY ASSURANCE WEB

The proposal of this paper is to guarantee the security quality assurance of web applications, by defining a methodology that could be reused and agile. So the first objective is identifying the main problems of web application. To do it, we used a real case scenario of a security company of Brazil called Tempest Security Intelligence [49] that sales web penetration test service.

The whole universe of the research described here corresponds to 467 reported vulnerabilities in the Tempest Security Intelligence analysis projects and web application ethical hacking of web applications, not considering the analysis projects of infrastructure. An importantly point is the vulnerabilities are spread across various customers and

do not correspond to points of vulnerability to be explored but real flows on web application. For example, given an analysis in the foo app, there are 15 points where you can perform SQL attacks, however, the vulnerability is reported only once. The numbers represent only the vulnerability in the application and not the amount of exploitable points in each application.

The research uses as base the perspective of the OWASP Top 10 2010 [45] version vulnerabilities, successor version of OWASP Top 10 2007 version, so the data used are restricted to projects reported between the years 2008 and 2010. On Figure 5 we present the workflow used.

Data collection corresponds to real cases but to preserve the client we uses a fictitious names.

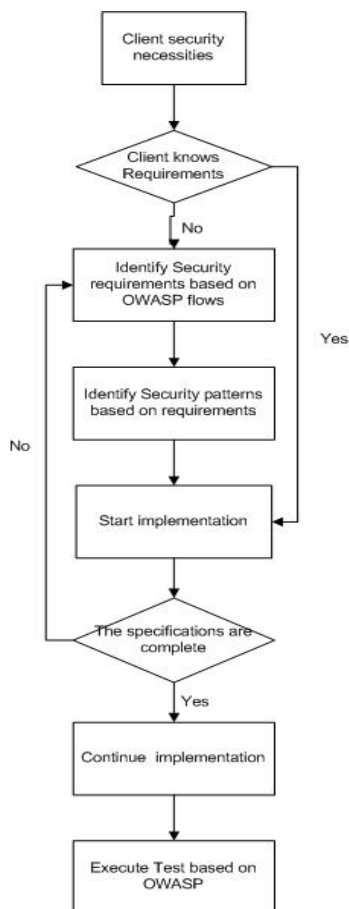


Figure 5: Software Security Quality Assurance workflow

The sample profile collected is determined in two characteristics: year of publication and type of vulnerability. First characteristic determines the year in which the vulnerability was discovered and published to the client. As previously described the data for the years 2008, 2009 and 2010. It was observed that 17% of vulnerabilities were reported in 2008 (Figure 6), 28% were reported in 2009 and 55% were reported in 2010. (Figure 6)



Figure 6: Vulnerabilities per year [49]

Another information collected was the type of collected vulnerable applications, showed on Figure 6.

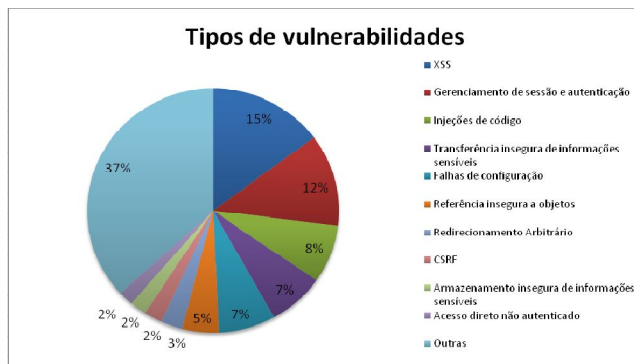


Figure 7 Types of vulnerabilities [49]

Resuming, it is possible to observe that 10 of vulnerabilities reported by Tempest Security Intelligence in the years 2008, 2009 and 2010 are: 15% of XSS vulnerabilities are observed, session management and access control are 12% of the vulnerabilities, 8% of the vulnerabilities are code injection, 7% of the vulnerabilities are flaws in the configuration, with 7% of the vulnerabilities are flaws transfer insecure credentials, the reference objects unsafe to correspond to 5% of the vulnerabilities, 3% of vulnerabilities are related to arbitrary redirection, 2% of vulnerabilities are related to direct access to unauthenticated, 2% for safe storage of sensitive and 2% of CSRF. The table bellow makes a comparison between Owasp reports and Tempest results.

TABLE 1: COMMON VULNERABILITY ACCORDING TEMPEST.

Tempest Top 10			
2008	2009	2010	General
XSS	XSS	Session authentication management	XSS
Code injection	Session authentication management	XSS	Session authentication management
Unsecure	Code injection	Configuration	Code injection

transmission of sensitive data		flows	
Session authentication management	Unsecure transmission of sensitive data	Unsecure reference to objects	Configuration flows
Configuration flows	Configuration flows	Code injection	Unsecure transmission of sensitive data
Direct access with no authentication	Unsecure reference to objects	Unsecure transmission of sensitive data	Unsecure reference to objects
Unsecure reference to objects	Direct access with no authentication	Unsecure sensitive and storage information	Arbitrary redirect
Unsecure sensitive and storage information	Arbitrary redirect	CSRF	CSRF
CSRF	CSRF	Arbitrary redirect	Unsecure sensitive and storage information
Arbitrary redirect	Unsecure sensitive and storage information	Direct access with no authentication	Direct access with no authentication

III. REQUIREMENTS

The requirements engineering on a business, systems, applications and components is more than just document that describes functional requirements of the application. Even though, most system analysts dedicate the bigger art of their time to elicit some quality requirements such as interoperability, availability, performance, portability and usability, many of them still sin with regard to addressing issues related to security.

Unfortunately, documenting specific security requirements is difficult. These tend to cause a high impact for many functional requirements. Furthermore, security requirements are usually expressed in a document the terms of how to achieve security not as the problem that needs to be resolved [27].

Most system requirements analysts have no knowledge in Security, the few who received some training had only a general overview of some security mechanisms such as passwords and encryption rather than meet real requirements in this area [5][28][29].

Security requirements deal with how the assets of a system must be protected against any kind of evil [27][30]. An asset is something within the system, tangible or not, that must be protected [31]. A threat or harm from which a system must be protected, is a potential vulnerability that can reach a well. A vulnerability is a weakness of a system

that tends to exploit an attack. Security requirements are constraints on the functional requirements in order to reduce the scope of vulnerabilities [27].

On the Bellow table we make a comparison between Donald Firesmith [28][29] security requirements proposal: Identification, Authentication, Authorization, Non-Repudiation, Privacy, Immunity, Integrity, Intrusion Detection, Security Audit, Maintenance Systems Security and Physical Protection; and OWASP Web vulnerability list, it relates each vulnerability and the correspondent requirements

TABLE 2: REQUIREMENTS X VULNERABILITIES

Requirement	Owasp Test
Identification	OWASP-IG-003, OWASP-IG-004
Authentication	OWASP-AT-001, OWASP-AT-002, OWASP-AT-003, OWASP-AT-004, OWASP-AT-005, OWASP-AT-006, OWASP-AT-007, OWASP-AT-008, OWASP-AT-009, OWASP-AT-0010
Authorization	OWASP-AZ-001, OWASP-AZ-002, OWASP-AZ-003
Immunity	OWASP-IG-005, OWASP-IG-006, OWASP-CM-002, OWASP-CM-003, OWASP-CM-006, OWASP-CM-008, OWASP-DV-001, OWASP-DV-002, OWASP-DV-003, OWASP-DV-004, OWASP-DV-005, OWASP-DV-006, OWASP-DV-007, OWASP-DV-008, OWASP-DV-009, OWASP-DV-0010, OWASP-DV-0011,OWASP-DV-0012,OWASP-DV-0013,OWASP-DV-0014, OWASP-DV-0015,OWASP-WS-002, OWASP-WS-003, OWASP-WS-004, OWASP-WS-005, OWASP-WS-006, OWASP-WS-007, OWASP-AJ-002
Integrity	OWASP-SM-001, OWASP-SM-002, OWASP-SM-003, OWASP-SM-004, OWASP-SM-005
Intrusion Detection	OWASP-CM-005
Non – Repudiation	
Privacy	OWASP-IG-001, OWASP-CM-001
Security Audit	
Fault Tolerance	OWASP-DS-001, OWASP-DS-002, OWASP-DS-003, OWASP-DS-004, OWASP-DS-005, OWASP-DS-006, OWASP-DS-007, OWASP-DS-008
Physical protection	
Maintenance of Security Systems	OWASP-CM-004, OWASP-CM-007

Since we have a relation that puts security vulnerability and system requirements we can elaborate reusable system

requirements based on security flows that could be addressed by system analysts during the project conception.

IV. DESIGN PATTERNS

Now, that we have a relation between security vulnerabilities and system requirements the next step is try to use a design patterns concept to propose a methodology to use it, giving to the user a choice to make a latter implementation of security code. To do it we use a classification given by the GoF Design Patterns.

Initially, we will consider the following requirements: Identification, Authentication, Authorization, Non Repudiation and Privacy. Reviewing these requirements, we can observe that all of this belongs to the same subject, identification of an actor, be a user, system or other entity that interacts with the system in question. All of them deal with the identification of an actor. Respectively have the actor ID, proof of ID, permissions of identity, confirmation of the shares of the entity and finally the secrets or secret identity. All of these requirements revolve around the creation of an identity.

Going forward on requirements analysis, we can separate the requirements for immunity and integrity, as two conditions that directly affect the structure of the system. From this viewpoint, the requirements for immunity vision ensure that the system is immune to contamination by parts of the actors and the requirements of Integrity vision ensure that the structure of an integrated system communication between these actors. Both requirements have a direct relation with the structure of the system since it will be necessary to change the structure of the system to adopt solutions to these requirements.

Following, we have the requirements for Intrusion Detection, Security Audit and Fault Tolerance, which deal with issues related to actions taken by the system. In the first case detection, we have a requirement that works as a prevention, which aimed to provide a mechanism for detection and notification in case of unauthorized access, since the audit comes as a mechanism to work issues in a more reactive, or attitudes that can be taken from the evidence and observation of actions, an audit requirement must include the registration of shares as well as mechanisms for future reference [11], different fault tolerance as well as reactive, which defines the behavior of the system will have in case of failure, is also to ensure that preventive flaws in system entities do not jeopardize the rest of the system. Therefore, the requirements of work on the issue of the conduct taken within the system.

Finally analyzing the requirements for Maintenance of System Security and Physical Protection have, this is a requirement that is more than physical matter, as the name refers, where the concern goes beyond the scope of

software, both outside the scope of our analysis. Since the requirement for maintenance has a horizontal behavior in relation to other requirements, since this deals with the maintenance of the system's other needs related to security, he is indirectly responsible for such requirements needs. Appears not a requirement for so considerable in terms of software and one that is the sum of the other requirements.

Organizing them according to their characteristics are:

- 1) Requirements Identification, Authentication, Authorization, Non-Repudiation and Privacy and related creation.
- 2) Integrity and Immunity Requirements related to the structure of the system.
- 3) Requirements for Intrusion Detection, Audit and Fault Tolerance-related behaviors of the actors in the system.
- 4) Requirements for Maintenance of Security Systems related to the other requirements.

Under this approach, using some of the classifications of GoF, we can separate the requirements according to their purposes. From the characteristics presented, we will separate them into three groups according to this criterion purposes, they are, Creation, Structural and Behavioral.

TABLE 3: RELATING PATTERNS AND REQUIREMENTS

Purpose		
Creation	structural	behavioral
Identification requirement	Immunity Requirement	Intrusion detection Requirement
Authentication Requirement	Integrity Requirement	Security audit Requirement
Authorization Requirement		Fault tolerant Requirement
Non-repudiation Requirement		
Security Maintenance Requirement		

Physical protection requirements that deal with physical issues related to the physical system are not addressed within this framework.

V. CASE STUDY

A) Reusable requirements

The tasks described by this article were used on the development of some IT projects on C.E.S.A.R (Center of Studies and Advanced Systems of Recife) and UNIMIX. These IT companies are needing to realize a detailed analysis of security issues in some projects with the purpose of making sure that system that are considered critical be tested and validated. As a consequence, this ensures that everything agreed on the contract is respected

by the service provider and cannot be questioned by the client.

Due to contracts issues, we are not allowed to give any further information about the context in question. However, some points must be cited, such as:

- a) The process of writing requirements has been validated in three projects with companies that act on these sectors: telecommunications and informatics. In these cases, the objective is only write the requirements document and the proposed methodology was employed. As a result, customers noticed an improvement in the problems understanding in early stages of the project.
- b) During the risk analysis, the suggested changes in the templates for requirements elicitation indicated a greater understanding of the problems, possible solutions for them and mitigating strategies.
- c) Preparation of a repository of reusable security requirements and their test cases based on the recommendations of tests developed by OWASP. This was the case with the requirements of P2P FINEP project, which aims to deploy solutions in environments peer-to-peer. Their requirements and test cases were used for decision making in relation to which requirements and test cases as well as risk analysis for the management solution desktop dreams ([HTTP:// www.dreamsweb.com. br](http://www.dreamsweb.com.br))
- d) We have a case study in the development of corporative site of a technology company of Pernambuco. This scenario was run throughout the full proposed cycle in this paper. It was observed that: a) there was significant improvement of the safety requirements of the portal, b) in the testing phase were found about 11 flaws in the site that did not meet the requirements, some of them quite serious, c) Another project in onset may benefit from the basic set of requirements, d) Part of the scripts could also be reused. Unfortunately for security reasons the company was not authorized to divulge more details of the results, as problems are identified security.
- e) Observers that the methodology described here is used with extreme efficiency and trends in the proposals brought to the development of systems that must function in an environment of cloud computing. This is because in this environment issues of SaaS, PaaS and IaaS introduce the characteristics of infrastructure as something bringing programmable horizontal scalability for applications. It is undeniable that as we have the scalability of an application being made across the board problems and new security risks arise. These problems not previously considered relevant. Mainly on issues related to security [44]. However,

the proposed solutions have a way to specify and reuse them efficiently because the strategies do not vary much scalability. The main result of this work, we observed an improved understanding of the technical requirements and their implementation by software security engineers and the ability to produce more accurate tests and that met the needs of customers. Thus reducing the need for correction of deficiencies identified in the test phase, which is one of the main mistakes made in building secure software [43].

Also as a result it will have a better quality software, on the point of view of ensuring the functionality specified by carrying out a process of validation and elaboration.

Another important result presented in this paper is to provide project managers the ability to quantify risk in relation to the implementation or not a particular requirement before starting the coding phase.

As a consequence of this work, we observed a satisfactory improvement on the comprehension of technical needs and its implementation by software engineers besides the ability to produce test more precise that meet clients need. Consequently, we were able to develop software with better quality from the point of view of the functionality assurance through the performance of a validation process more elaborated.

B) Design Patterns and a late implementation

The purpose of the case study was to validate the proposed relationship between the GoF design patterns as a way to represent security requirements. As mentioned in the work we try to have a more practical assessment of our study to evaluate the feasibility of using these standards as a tool in implementing security requirements and to facilitate the understanding of security requirements for developers in genera during the software development process.

Initially our study was conducted in a project expected to last 3 (three) months, we will call this a Test System. This project would serve initially as a proof of concept for a larger project, with issues related to client confidentiality and NDA cannot go into further detail concerning the applicant, project name and other sensitive information to be omitted.

In a second moment relationships proposed in this paper was applied again in the second Test System that time this system was already in a more consolidated stage requiring greater attention as we shall see below.

Finally, a third opportunity was presented to us where we suggest an approach to two related structures created in this work. Unlike the two previous occasions the third opportunity is still being implemented.

For these tree applications at the beginning of the project the security requirements was not so clear, but once completed the implementations we could see a positive result of implementing the proposals made by the job. One of the best insights that should be observed was that the changes on requirements were not very intrusive, low impact and easy modification.

So the opportunity to apply the propositions made in this initial work in a context outside of the web, mainly located on the server side, served as an initial validation of a positive result. Besides these the possibility of making a second application using a macro context of the application, addressing GWT, RPC calls, proved satisfactory.

Furthermore the application of the propositions in a macro context has generated the perception that the adoption of standards, as related to ambiguous or poorly written requirements, may present as a data point requiring a second more detailed approach to understanding how to address . Still, the approach was extremely valid and consistent highlighting the importance of future studies mentioned in the next section.

VI. CONCLUSION

The software quality cannot be measured only by the assurance of the execution of a process, but by the results of its execution and necessary validations. Within this context, this paper aimed to define tasks, recommendations and process that should be introduced on the cycle of software development with the purpose of guiding the test and validation phase to produce more elaborated and precise results from the point of view of security issues.

The process proposed by this paper is being introduced on the software cycle development s at C.E.S.A.R as specific security needs are required.

The adoption of this process allowed making a more critical analysis of the new features introduction on new projects as well as the test team comprehension at executing these tasks thus improving the software quality observed by the clients.

As a final contribution, we were able to validate the proposal software security requirements reuse and its test cases in other projects inside C.E.S.A.R, proving that this process is extensible as proposed.

ACKNOWLEDGMENT

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES <http://www.ines.org.br>), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.

REFERENCES

- [1] R. Lutz, "Software engineering for safety: a roadmap," Proceedings of the Conference on The Future of Software Engineering, ACM, 2000, pp. 213–226.
- [2] B. Matt, "What Is Computer Security?," Computer, 2003, pp. 67-69.
- [3] I. Sommerville, T. Rodden, P. Sawyer, R. Bentley, and M. Twidale, Integrating Ethnography Into the Requirements Engineering Process, 1993.
- [4] D.G. Rosado, C. Gutiérrez, E. Fernández-medina, M. Piattini, P.D. Universidad, C. Real, D. Grosado, E. Fdez-medina, S.T. Calle, and M. Tovar, "A Study of Security Architectural Patterns," Information Systems, vol. 1, 2006, pp. 2-9.
- [5] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security," Urbana, vol. 51, p. 61801.
- [6] P.T. Devanbu and S. Stubblebine, "Software Engineering for Security: a Roadmap," The future of Software Engineering, ACM Press, 2000, pp. 227-239.
- [7] N. Yoshioka, H. Washizaki, and K. Maruyama, "A survey on security patterns," Progress in Informatics, 2008, p. 35.
- [8] G. Sindre and A.L. Opdahl, "Eliciting security requirements with misuse cases," Requirements Engineering, vol. 10, 2004, pp. 34-44.
- [9] W.C. Summers, "Password Policy: The Good, The Bad, and The Ugly.", Proceedings of the Winter International Symposium on Information and Communication Technologies, 2004
- [10] P. Samarati and S.D. di Vimercati, "Access Control: Policies, Models, and Mechanisms," FOSAD, R. Focardi and R. Gorrieri, Springer, 2000, pp. 137-196.
- [11] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, Security Patterns : Integrating Security and Systems Engineering (Wiley Software Patterns Series), John Wiley & Sons, 2006.
- [12] F. Khomh and Y.G. Gueheneuc, "Do Design Patterns Impact Software Quality Positively?," Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on, 2008, pp. 274-278.
- [13] J. Katz and Y. Lindell, Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series), Chapman & Hall/CRC, 2007.
- [14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented, Addison-Wesley Professional.
- [15] I. 7498-2, Information processing systems -- Open Systems Interconnection -- Basic Reference Model -- Part 2: Security Architecture, 1989.
- [16] D. Zerkle and K. Levitt, "NetKuang -- A Multi-Host Configuration Vulnerability Checker," in Proceedings of the 6th USENIX Unix Security Symposium, 1996.
- [17] P. Steiner, "On the internet nobody knows you're a Dog," The New Yorker, vol. 69, 1993, p. 61.

- [18] P. Kuniyu, "An identity authentication system based," Identity, 2009.
- [19] Z. Wang, M. Li, M. Chen, and C. Chang, "A New Intelligent Authorization Agent Model in Grid," 2009 Ninth International Conference on Hybrid Intelligent Systems, 2009, pp. 394-398.
- [20] G. Dhillon, Principles of Information Systems Security: Texts and Cases, Wiley, Ed. Virginia Commonwealth Univ March 2006
- [21] H. Peiris, L. Soysa, and R. Palliyaguru, "Non-Repudiation Framework for E-Government Applications," 2008 4th International Conference on Information and Automation for Sustainability, 2008, pp. 307-313.
- [22] J. Adikari, "Efficient Non-Repudiation for Techno-Information Environment," First International Conference on Industrial and Information Systems, 2006, pp. 454-458.
- [23] L.F. Soares, G. Lemos, and S. Colcher, Redes de Computadores: das LANs, MANs e WANs às redes ATM, Rio de Janeiro: Campus, 1995.
- [24] C. Alexander, S. Ishikawa, and M. Silverstein, A pattern Language, Oxford University Press.
- [25] R.P. Gabriel, Patterns of software: tales from the software community, Oxford University Press, Inc. New York, NY, USA, 1996.
- [26] I. Oliveira, "Uma Análise de Padrões de Projeto para o Desenvolvimento de Software Baseado em Agentes," 2001.
- [27] F. Khomh, Y. Guéhéneuc, and P. Team, "An empirical study of design patterns and software quality," 2008, pp. 1-19.
- [28] P.S. Sandhu, P.P. Singh, and A.K. Verma, "Evaluating Quality of Software Systems by Design Patterns Detection," 2008 International Conference on Advanced Computer Theory and Engineering, 2008, pp. 3-7.
- [29] M. Bernardi and G. Di Lucca, "Improving Design Pattern Quality Using Aspect Orientation," 13th IEEE International Workshop on Software Technology and Engineering Practice, 2005, Ieee, 2005, p. 206-218.
- [30] C.B. Haley, R.C. Laney, and B. Nuseibeh, "Deriving security requirements from crosscutting threat descriptions," AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development, New York, NY, USA: ACM Press, 2004, pp. 112-121.
- [31] D. Firesmith, "Engineering security requirements," Journal of Object Technology, vol. 2, 2003, p. 53-68.
- [32] D. Firesmith, "Analyzing and Specifying Reusable Security Requirements," Eleventh International IEEE Conference on Requirements Engineering (RE'2003) Requirements for High-Availability Systems (RHAS'03) Workshop, Citeseer, .
- [33] C.B. Haley, J.D. Moffett, R. Laney, and B. Nuseibeh, "A framework for security requirements engineering," SESS '06: Proceedings of the 2006 international workshop on Software engineering for secure systems, New York, NY, USA: ACM Press, 2006, pp. 35-42.
- [34] Information Technology - Security Techniques - Evaluation Criteria for IT Security, Geneva Switzerland: ISO/IEC Information Technology Task Force (ITTF) .
- [35] M. Weiss and H. Mouratidis, "Selecting Security Patterns that Fulfill Security Requirements," 2008 16th IEEE International Requirements Engineering Conference, 2008, pp. 169-172.
- [36] Structural Patterns at Source Making.", http://sourcemaking.com/structural_patterns, last accessed 8/10/2011 .
- [37] G. Inc, "GWT, Google Web Toolkit," <http://code.google.com/webtoolkit/>, last accessed 8/10/2011.
- [38] J. Company, "Hibernate," <http://www.hibernate.org/>, last accessed 8/10/2011.
- [39] Gilead, Generic Light Entity Adapter, <http://noon.gilead.free.fr/gilead/>, last accessed 8/10/2011.
- [40] P. Pawlak, B. Sakowicz, P. Mazur, and A. Napieralski, "Social Network Application based on Google Web," Source, 2009, pp. 461-464.
- [41] M. Dhawan and V. Ganapathy, "Analyzing Information Flow in JavaScript-Based Browser Extensions," 2009 Annual Computer Security Applications Conference, 2009, pp. 382-391.
- [42] E. Ofuonye and J. Miller, "Resolving JavaScript Vulnerabilities in the Browser Runtime," 2008 19th International Symposium on Software Reliability Engineering (ISSRE), 2008, pp. 57-66.
- [43] Meier J. Web application security engineering. IEEE Security & Privacy Magazine. 2006;4(4):16-24.Available at: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1667998>, last accessed 8/10/2011.
- [44] SUN, Introduction to Cloud Computing architecture White Paper 1st Edition, June 2009
- [45] Owasp, 2008, owasp testing guide 2008 v3.0. Disponível em: http://www.owasp.org/index.php/category:owasp_testing_project, last access: 06/05/2009
- [46] PMBOK, Project Management Institute (PMI) standards committee: A guide to the Project Management Body of Knowledge (PMBOK) Third edition,2008.
- [47] Us-cert - technical cyber security alerts, 2009. Disponível em: <http://www.us-cert.gov/cas/techalerts/> . Last access: 29/04/2009
- [48] Cloud Computing Use Case Discussion Group Version ; Cloud Computing Use Cases A white paper produced by the 2.0 30 October 2009
- [49] Tempest Security Intelligence, www.tempest.com.br, last accessed 05/05/2011
- [50] Google Trends Service, www.google.com/trends, last accessed 8/10/2011.

On Generating Security Implementations from Models of Embedded Systems

Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin
 Mälardalen Real-Time Research Centre (MRTC)
 Mälardalen University, Västerås, Sweden
 {mehrdad.saadatmand, antonio.cicchetti, mikael.sjodin}@mdh.se

Abstract—Designing secure embedded systems is a challenging task. Many of the challenges unique to embedded systems in this regard are due to the constraints that these systems have and thus impacts that security features will have on other properties of the system. Therefore, security decisions should be considered from early phases of development and together with other requirements. In model-driven methods, this means including security features in the design models. On the other hand, code generation from models is one of the promises of model-driven approaches. In this paper, by discussing the impacts of security design decisions on timing properties, we present the idea of automatic security code generation. We identify what issues a model for an embedded system should be able to answer and cover so that the security implementations that are later generated from it, will be consistent with the timing constraints and specifications of the system.

Index Terms—Embedded security; MDA; Code generation; UML modeling

I. INTRODUCTION

In the design of systems, security should be considered from early phases of development and along with other aspects of the system. While this approach to security is important for consistent and efficient security decisions, it becomes critical in case of embedded systems. Due to resource constraints in embedded systems, it is important to perform careful balance among different properties to satisfy all the requirements. Therefore, security should not be considered just as an addition of features but as a new dimension and metric [1].

Regarding design complexity of embedded systems, model-driven methods are a promising approach in raising abstraction levels and coping with the complexity of embedded systems. However, due to the characteristics of embedded systems, security requirements cannot be considered in separation from other requirements, and the modeling solutions that are used should be able to model security aspects along with other requirements such as timing, performance, and power consumption. This is especially important not to just document security requirements in the model, but also to enable analysis of them and their impacts on other requirements of the system, and generation of code that includes security features and implementations. The possibility to perform such analyses is the key to ensure correct design of an embedded system and that the code to be generated will be consistent with the specification. However, it should also be noted that the actual behavior of the generated code at runtime may deviate from

what is specified in the model and expected. One reason is that some information may only be available at runtime. These deviations can be detected and controlled by using runtime verification and monitoring methods [2].

In this paper, considering challenges of designing secure embedded systems, we discuss what is needed at model level to enable proper security code generation. We do this by identifying necessary analyses that are required to realize implications of security design decisions and therefore predict the side effects of generated security implementations on other aspects, particularly timing properties.

The remainder of the paper is structured as follows. Section II discusses security challenges in embedded systems and implications of security design decisions in general. In Section III, we describe automatic payment system for toll roads, and explain the relation between timing and security requirements in this system. In Section IV, we will have a look at several UML profiles for modeling security and discuss their suitability for generation of security implementations. We propose a solution for modeling security by extending MARTE [3], and present our partial work on that (in the scope of this work, we focus only on UML profiles and not other ways of defining domain specific languages). Finally, Section V summarizes the paper and states how we continue with the work and possible future directions.

II. SECURITY IN EMBEDDED SYSTEMS

Security is an aspect that is often neglected in the design of embedded systems. However, the use of embedded systems for critical applications such as controlling power plants, vehicular systems control, and medical devices makes security considerations even more important. This is due to the fact that there is now a tighter relationship between safety and security in these systems.

Also because of the operational environment of embedded systems, they are prone to specific types of security attacks that might be less relevant for other systems such as a database inside a bank. Physical and side channel attacks [1] are examples of these types of security issues in embedded systems that bring along with themselves requirements on hardware design and for making systems tamper-resistant. Examples of side channels attack could be the use of time and power measurements and analysis to determine security keys and types of used security algorithms.

Increase in use and development of networked and connected embedded devices also opens them up to new types of security issues. Features and devices in a car that communicate with other cars (e.g., the car in front) or traffic data centers to gather traffic information of roads and streets, use of mobile phones beyond just making phone calls and for purposes such as buying credits, paying bills, and transferring files (e.g. pictures, music, etc.) are tangible examples of such usages in a networked environment.

Besides physical and side channel attacks, often mobility and ease of access of these devices also incur additional security issues. For example, sensitive information other than user data, such as proprietary algorithms of companies, operating systems and firmwares, are also carried around with these devices and need protection.

A. Implications of Introducing Security

Because of the constraints and resource limitations in embedded systems, satisfying a non-functional requirement such as security requires careful balance and trade-off with other properties and requirements of the systems such as performance and memory usage. Therefore, introducing security brings along its own impacts on other aspects of the systems. This further emphasizes the fact that security cannot be considered as a feature that is added later to the design of a system and needs to be considered from early stages of development and along with other requirements.

From this perspective, there are many studies that discuss implications of security features in embedded systems such as [1]. Considering the characteristics of embedded systems, major impacts of security features are on the following aspects: Timing and Performance, Power Consumption, Flexibility and Maintainability, and Cost.

Considering these points, the security code that is generated for an embedded system should be from a model that satisfies the aforementioned criteria. This means that the model should contain enough information to enable impact analysis of security features (such as timing and performance), and ensure that they are in line with the system specification before generating code from the model. In the scope of this work, we focus on the timing costs of security mechanisms that are important for schedulability analysis and performance of a system, particularly in real-time embedded systems.

III. AUTOMATIC PAYMENT SYSTEM EXAMPLE

Figure 1 shows internal interactions of a real-time embedded device in vehicles for automatic payment system in toll roads. The main goal in the design of this system is to allow a smoother traffic flow and reduce waiting times at tolling stations. This is an example of systems in which the impact of security features on timing properties are important and critical. The sequence diagram shows that when a payment station, through its camera, detects that a vehicle is approaching, it starts communicating with the vehicle and sends information such as the amount to pay to the vehicle. The vehicle, then shows this information to the driver through its User Interface

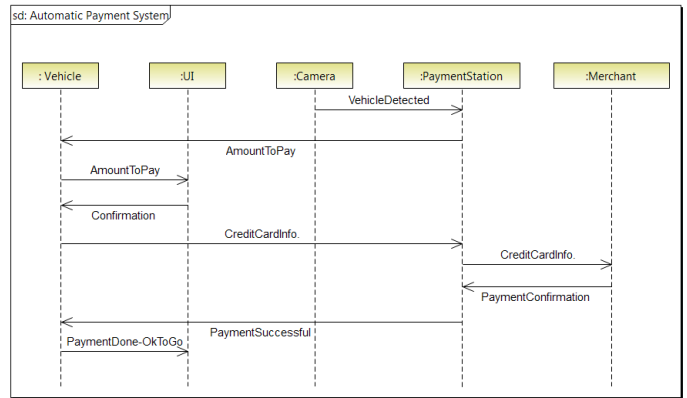


Fig. 1. Automatic Payment Systems for toll roads.

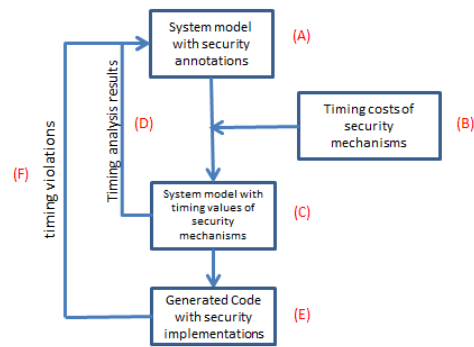


Fig. 2. Suggested Approach

(UI). Upon confirmation of this payment by the driver, the vehicle sends credit card information to the station through a secure wireless connection. However, in this system, not only there are several security requirement, but also we have timing requirements as well. For instance, there is a critical time window from the moment that the camera detects a vehicle until the time it reaches the tolling station. It is within this time window that a successful payment transaction should be done; otherwise, the vehicle has to stop.

To implement such a system while ensuring the satisfaction of timing requirements, it is necessary to take into account the timing costs of security mechanisms that are used to implement security requirements of the system. For example, there are operations such as the transfer of CreditCardInfo that not only require encryption to protect sensitive data, but also have constraints on their execution times and cannot just take any arbitrary amount of time to execute.

To achieve this, the approach depicted in Figure 2 is suggested.

To enable the generation of appropriate security implementations, with respect to the timing constraints of the system, the following challenges are identified:

- 1) Modeling security mechanisms with enough detail to enable both timing analysis on the model and generation of the code implementing them,
- 2) Obtaining timing costs of security mechanisms,

- 3) Generating code for security mechanisms and detecting possible timing violations of the generated code at run-time.

The first challenge is discussed in the following section. To get the timing costs of security mechanisms, we rely on studies such as [4] that have done such measurements. To solve the third challenge, some hints are provided in the last section, but we leave its thorough discussion and implementation as a future work.

IV. MODELING SECURITY MECHANISMS

In this section, we discuss how to model security mechanisms, namely confidentiality, for our example system.

A. Current Solutions for Modeling Security

There are several efforts on defining UML profiles for security. For example, SecureUML [5] focuses on modeling role-based access control. AuthUML [6] provides a framework for analysis of access control requirements. [7] introduces a set of stereotypes for specification of vulnerabilities that serve as guidelines for developers to avoid them during implementation. UMLsec [8] offers a broader range of security concepts and comes with an analysis tool. Article [9] tries to offer a solution for modeling security along with timing characteristics of the system using UMLsec and MARTE.

One main issue with these modeling profiles is that modeling of security requirements is often considered in separation from other requirements such as timing [9], [10]. One solution could be to combine security profiles with other profiles that enable modeling requirements of embedded real-time systems and their analysis such as MARTE. However, it should be noted that combining different UML profiles can be tricky as these profiles can have overlapping and conflicting semantics and notations. This issue can be even trickier remembering that most of available security profiles are limited in the sense that they usually focus on a certain aspect of security and several of them may need to be combined as well [10]. Supporting hardware modeling and hardware devices with built-in security mechanisms is also another issue that is important for evaluating different deployment scenarios and is often not covered in security profiles. We have discussed this issue with more details in [11].

Finally, to generate code that includes implementations of security mechanisms from a model of an embedded system, the modeling concepts for security should provide the necessary information to derive code. This level of information is equally important to enable certain types of analyses at model level such as performing schedulability analysis by taking into account execution times of security features (e.g. encryption/decryption) or energy consumption analysis. For example, execution time and energy consumption of a block cipher algorithm can vary depending on the used algorithm, number of rounds, key size and so on. Therefore, these influencing parameters are required to be annotated at the model to enrich and make analysis more accurate. However, many of the currently available security profiles do not provide

sufficient semantics to model and include the details necessary to perform these types of analyses and generate code.

B. Modeling Security Using MARTE

In this section, we discuss how MARTE modeling language can help to include timing costs of security mechanisms and include them in timing analysis of the system.

In order to alleviate the mentioned issues regarding security modeling in embedded systems, we propose extending MARTE with security concepts and building modeling semantics for security upon it. MARTE offers rich semantics for modeling non-functional requirements in real-time embedded systems and provides dedicated packages for schedulability and performance analysis. It also includes concepts for modeling deployment, hardware and annotating models with energy usage values. By extending MARTE with security concepts, it becomes possible to include impacts of security design decisions in the model for timing analysis, and evaluate their side effects before starting the implementation phase. Therefore, the code that will be generated from these models will better satisfy the requirements and constrains of an embedded system with less unknown and unmanaged side effects on other properties of the system such as timing, energy consumption, and memory usage.

Figure 3 shows part of our suggested MARTE extension for modeling block ciphers.

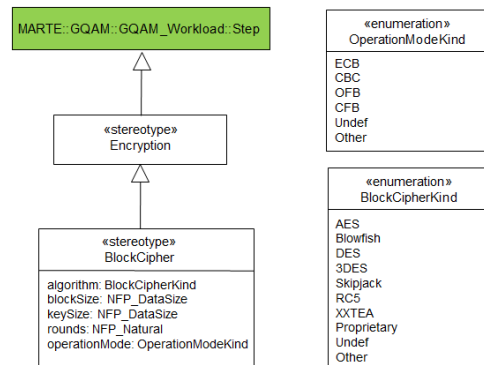


Fig. 3. Definition of BlockCipher stereotype

Using such concepts we can now annotate the operation of sending the CreditCardInfo, in the Payment System Example mentioned before, with the information that follows:

```

«BlockCipher» CreditCardInfo() {algorithm=AES
, blockSize=(128,bit), keySize=(128,bit), rounds=12,
operationMode=ECB}
    
```

C. Introducing Timing Costs of Security Mechanisms

So far, we have managed to annotate sensitive operations in the model, such as CreditCardInfo, with parameters (type of algorithm, blocksize, keysize,etc.) of the encryption algorithms that are selected to protect them. This information is not only required to generate code that implements each selected

encryption algorithm, but also enables us now to evaluate their timing costs at model level by using the result of studies such as [4] and [12] that have performed measurements of timing costs of encryption algorithms. We assume the existence of such measurements for the platform used in the automatic payment system example, in the form of Table I.

Algorithm	Key Size	BlockSize	Rounds	OperationMode	Execution Time (Bytes/Sec)
AES	128	128	10	ECB	490
AES	192	128	12	ECB	560
AES	256	128	14	ECB	710
...					

TABLE I
EXECUTION TIMES OF ENCRYPTION ALGORITHMS

Using this information, execution times of modeled encryption algorithms can be determined. In [11], we have discussed in more detail how the model can also be analyzed for energy costs of security mechanisms using a similar approach. The result would be similar to what follows:

```

«GaCommStep»      «BlockCipher»      CreditCardInfo()
{algorithm=AES    ,   blockSize=(128,bit),   keySize=(128,bit),
 rounds=10,       operationMode=ECB,   msgSize=(150,B),
 execTime=(306,ms,min,calc) }

```

GaCommStep is a MARTE concept which is a specialization of MARTE Step to describe communication workloads and is used in generic quantitative analysis contexts. Specification of execution time values are done here based on MARTE NFP concepts.

This way, the impact of security requirements on timing requirements in the system are identified. At this point, it is now possible to determine whether the chosen security mechanism is feasible considering specification and constraints on the allowed execution times. If not, blocksize, keysize, number of rounds, operationmode or even the size of the input message can be tweaked to balance security level with timing properties. This is done by iterating over steps A, B, C, and D of Figure 2. That is, timing costs for security mechanisms in the original model (A) are calculated resulting in a model with timing values for its security mechanisms using the MARTE concepts introduced above. These values are then checked against the timing specifications of the system. If violations are detected, the user modifies security mechanisms in the original model and goes through steps B, C and D again. After this phase, it is feasible to generate implementation of the defined security features for CreditCardInfo.

While, this approach seems to also enable energy consumption analysis on the model, this topic deserves a separate study; especially that detecting energy consumption violations later at runtime is a much bigger challenge than the detection of timing violations.

V. NEXT STEPS AND FUTURE WORK

In this paper, we presented the idea of generating security implementations from models of embedded systems. The challenges of designing secure embedded systems were identified.

We discussed impacts of security on other requirements on the system, namely timing requirements, and the importance of trade-off analysis among requirements to predict the side effects of the generated code. Therefore, to generate security implementations, it was realized that the main challenge is at the model level so that the generated code respects the constraints of embedded systems. We proposed using MARTE as the basis for modeling embedded systems to enable necessary analyses on security decisions before generating code for them. However, as pointed out, timing violations can still happen at runtime. Therefore, it is needed to relate requirements in the model to their corresponding implementations in the generated code, and report any timing violations back to the user at the model level. As a solution to develop this feature, we are investigating suitability of Java Modeling Language (JML) to annotate the code and define pre/post-conditions for generated methods as suggested in [13].

REFERENCES

- [1] P. Kocher, R. Lee, G. McGraw, and A. Raghunathan, "Security as a new dimension in embedded system design," in *Proceedings of the 41st annual Design Automation Conference*, ser. DAC '04, 2004, pp. 753–760, moderator-Ravi, Srivaths.
- [2] S. Colin and L. Mariani, "Run-time verification," in *Model-Based Testing of Reactive Systems*, ser. Lecture Notes in Computer Science, M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds., vol. 3472. Springer Berlin / Heidelberg, 2005, pp. 525–555.
- [3] MARTE specification version 1.0 (formal/2009-11-02), <http://www.omgmar.te.org>, Last Accessed: June 2011.
- [4] A. Nadeem and M. Javed, "A performance comparison of data encryption algorithms," in *First International Conference on Information and Communication Technologies, ICICT 2005.*, 2005, pp. 84 – 89.
- [5] T. Lodderstedt, D. A. Basin, and J. Doser, "Secureuml: A uml-based modeling language for model-driven security," in *Proceedings of the 5th International Conference on The Unified Modeling Language*, ser. UML '02. London, UK: Springer-Verlag, 2002, pp. 426–441.
- [6] K. Alghathbar and D. Wijesekera, "authuml: a three-phased framework to analyze access control specifications in use cases," in *FMSE '03: Proceedings of the 2003 ACM workshop on Formal methods in security engineering*. New York, NY, USA: ACM, 2003, pp. 77–86.
- [7] K. P. Peralta, A. M. Orozco, A. F. Zorzo, and F. M. Oliveira, "Specifying security aspects in uml models," in *First International Modeling Security Workshop*, ser. MODSEC08, Toulouse, France, September 2008.
- [8] J. Jürjens, "Umlsec: Extending uml for secure systems development," in *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*. London, UK: Springer-Verlag, 2002, pp. 412–425.
- [9] V. Thapa, E. Song, and H. Kim, "An approach to verifying security and timing properties in uml models," in *Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on*, 2010, pp. 193 –202.
- [10] R. J. Rodríguez, J. Merseguer, and S. Bernardi, "Modelling and Analysing Resilience as a Security Issue within UML," in *SERENE'10: Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems*. ACM, 2010, accepted for publication.
- [11] M. Saadatmand, A. Cicchetti, and M. Sjödin, "On the need for extending marTE with security concepts," in *International Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2011)*, March 2011.
- [12] J. Lee, K. Kapitanova, and S. H. Son, "The price of security in wireless sensor networks," *Computer Networks*, vol. 54, pp. 2967–2978, December 2010.
- [13] J. Lloyd and J. Jürjens, "Security analysis of a biometric authentication system using umlsec and jml," in *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 77–91.

Proposal for Ground Shipping High Volume of Data Parameter in Supersampling Unmanned Aircraft Through Radio Modem

Manuel Sánchez, Vicente Millet, Neves Seoane
INTA – National Institute of Aerospace Technology,
Madrid, Spain
{sanchezrum;milletecv;seoanevn@inta.es}@inta.es

Luis de-Marcos, José-Javier Martínez
Computer Science Department
University of Alcalá
Madrid, Spain
{luis.demarcos;josej.martinez@uah.es}

Abstract—In an unmanned aircraft, large volumes of data are generated by the various sensors installed on the aircraft. At critical moments such as take-off, landing, parachute openings, or when the aircraft performs sudden maneuvers, additional parameters besides the default need to be sampled in order to understand completely the behavior of the aircraft. We propose different alternatives for sending multiple data to land at sampling frequencies of up to 10 Hz at critical times. Preliminary results are presented for the most extreme case, that is using full RS-232 bandwidth for the six most important parameters resulting in 210 samples per second for each parameter.

Keywords—Information retrieval; unmanned aircraft; data processing; sensors; radio modem

I. INTRODUCTION

INTA is the Institute for Aerospace Technologies in Spain and flight tests have been part of INTA's activity since it was created in 1947. With the objective of upgrading such activities and modernizing its facilities, INTA created the Flight Test Area, Area de Ensayos en Vuelo (AEV) [1], AEV is responsible for providing flight test support for all current and future programs including RPV (Remotely Piloted Vehicle), Rocket Launches, Balloons and Missile Tests.

The problem is that during critical moments, data from several sensors needs to be transmitted at rates in excess of 10 Hz which is not possible given the limitation of bandwidth that radio modem communications presents. To solve this problem, software alternatives are considered. These are one-dimensional array based on differentiated values, two-dimensional array with a fix number of rows and columns and time stamp, two-dimensional array with time stamp and parameter identification label, and finally two-dimensional array with time stamp, parameter identification label, controlling the last value sent. The paper starts with a description of the problem followed by the methodology used to arrive at the solution. Finally, a few preliminary results with the obtained conclusions are shown.

II. STATE OF THE ART

Currently, INTA is working on various unmanned aircraft under development such as SIVA (*Integrated System for Aerial Surveillance*), ALO (*Lightweight Observation Air*

Vehicle), DIANA (*High speed target drone*) and HADA (*Morphing VTOL Aircraft*) among others. In large UAV's like SIVA (weight 300 Kg and wing span 5.81 m), data acquisition systems (DAQ) allow sending thousands of data samples per second of any parameter to ground in pulse code modulation (PCM) [2] format by using S-band telemetry frequency. However, in smaller UAV's like ALO shown in Figure 1 (weight 50 Kg and wing span 3.48 m), it is not possible to integrate a DAQ due to small payload. In such a case, it is necessary send data to ground using radio modems, with a frequency of ten samples per second for all sample parameters. The proposal presents alternatives to allow sending samples at a rate of more than ten times per second in critical periods for small UAVs that need to use radio modems. In such a way, it will be possible to know the aircraft behavior and validate it using simulation.



Figure 1. ALO unmanned aircraft

Initially, the data types of parameters used for shipping are 32-bits floating points or 16-bits integers. The input data for each sensor is written in a memory buffer and then sent to earth using a specific frequency (ten samples per second). Parameters are not grouped hierarchically and each one is sent using the same sample rate [3]. Some data parameters are sampled by the onboard computer at frequencies up to 450 samples per second, while other parameters are sampled at a lower rate (e.g., one per second for GPS).

The idea of storing data on the aircraft is not feasible due to the process for managing interruptions used by the operating system that could result in a possible loss of data in real time. The final storage of data on land is the only

feasible option. Fig. 2 shows a block diagram representative of the various elements that act on the aircraft, with the central part based on a control computer, along with their input chains, demand measures and associated communications.

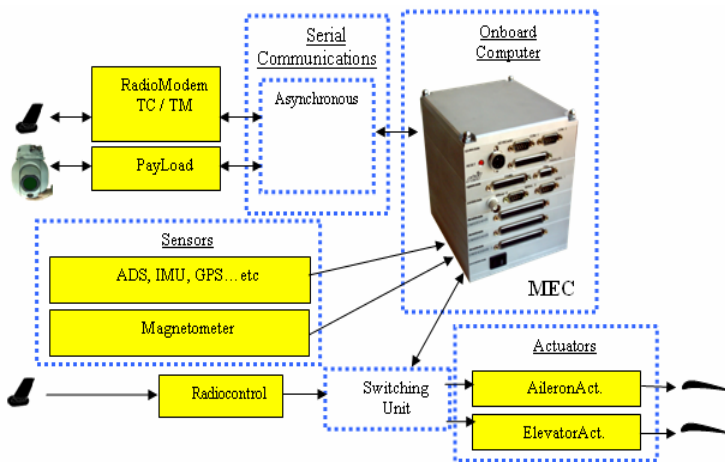


Figure 2. System onboard

III. METHODOLOGY

The objective is to investigate the features of the different models proposed, which will provide knowledge about what happens on certain aircraft sensors that suffer substantial alterations in its physical measures at critical moments. This process can be broken down into the following steps:

A. Analyze the initial conditions: parameters, frequencies of transmission, limits of communications, data storage and display; 280 bytes were sent in parameters 32 and 16 bits length, and 19 bytes of payload (camera) as the focus or zoom, which makes an approximate size of 300 bytes, every 100 milliseconds. This is a guaranteed rate for a RS-232, modem, but for reasons of data loss in communications (distance, weather conditions, etc.), it is not advisable to exceed this rate.

B. Identify critical moments and the variables concerned, establishing a proposal for grouping variables based on objective criteria. As an initial proposal, the parameters are grouped into three categories, the first (most important) contains those parameters that require a very high sampling and also with a high changing capacity on their values per time unit (e.g., acceleration and angular velocity). A second group includes those parameters that may require sampling rates of 10 Hz such as measured angles of attack, pitch, yaw and roll. The other parameters would be part of the group with lower necessities of sampling frequency.

C. Define strategies to establish optimal mechanisms for approaching the desired goal, which is to obtain more

information about certain parameters at critical moments. Experimental as well as quantitative methods will be used. A toolkit that approaches and solves the problem will be designed. Prototypes will be developed using a specific programming language for each alternative to allow for a comparative analysis of the different designs and techniques.

The techniques used for this proposal lie in the combination of the following fields or areas:

- Standards of measurement processes and data acquisition [4].
- Transmission of information via radio modem [5].
- Mechanisms of compression techniques based on data compression standards [6].

According to CVT (Current Value Table) technology, each sensor stores the sampled value on a cell, overwriting the previous value, and each sensor has a different sampling frequency. A process is activated and traverses all the cells, building an image of the values found at that moment. Using pointers and information about the order of the parameters (32-bit floats or 16-bit integers), the pointer moves through all the parameters to capture in a one-dimensional array the set of all values that are subsequently sent to land via radio modem, a transmission format of 8, n, 1 (8 data bits, no parity and one stop bit). This process is repeated ten times per second; when information is received on land, decoding is simply done in an analogous way. The proposed approach entails replacing the one-dimension array with a two-dimensional array with a variable number of rows and columns, depending on the different types of techniques. This array will also contain time stamps that will indicate the moment corresponding with the value sampled by the sensor. This is concept missing nowadays, because of the linearity used for the land consignment of the resulting array. The variability of a two-dimensional array can be done in real time, either automatically, so that it can be integrated into the onboard computer program of the flight to be done by the aircraft, or manually, by sending signals from the ground through the radio modem. Independently of the format of the cell in rows and columns, land transmitting will be done in the same way as a PCM (*Pulse Code Modulation*) stream, going through the array and sending the values byte by byte via RS-232.

The approach will start in a basic form and progress to increasing levels of complexity, reaching the best solutions for specific needs. For the early stage, the value of the parameters of the most important category are sent as along with the average between time units using functions specifically designed for this purpose (with optional insertion of timestamps). We can then proceed to defining a different array, mixing several values of the three categories, adding more values of the first category and less parameters of the last category. Each parameter will have a time stamp (either the measured values or the average between instances). Finally, at more complex stages, compression-based techniques for sending data values will

be considered (e.g., using differential values, similar to sending data using differential PCM).

IV. PRELIMINARY RESULTS AND CONCLUSIONS

As preliminary results, we present a graph obtained in the initial tests performed on a simulation with data array generation which corresponds to angular acceleration on the aircraft x body parameter [7] (cataloged on the first class or category). Fig. 3 presents the variation of this parameter in a wide period of the flight. Data circled is the basis for Fig. 4 that presents the values of the parameter every 10 milliseconds, thus offering more detail. Finally the resulting graph (Fig. 5) is obtained in the most extreme case, that is, using full RS-232 bandwidth for the six more important parameters, resulting in 210 samples per second for every parameter. It can be observed, that some parameter variations are not obtained using 10 samples per second. And it is necessary to know that during post-processing, these unobserved variations can appear using an increased data rate. Information about 6 parameters has been sent but there are 33 and therefore, there is not information about the other 27. Subsequent proposals should have the goal of sampling all the parameters, at a frequency rate depending on the category of the parameter, trying to find a balance between information loss for every parameter and the additional data variations obtained for more sensitive ones.

Further research is based on an approach using alternative methods, as to develop 2D array with different sample rate and timestamps, as explained earlier. At this second stage, it is necessary to use some techniques similar to those of a differential PCM for not-so-important important parameters: sending a first value with 32 bits of accuracy and the next ones using 16 or even 8 bits, not with the value but with an offset in relation to the first or previous record. In such a way some bits could be saved, so that more samples of important parameters could be obtained.

A final approach could be two-dimensional array with a fix number of rows and columns. Each row will also contain timestamps that will indicate the instant that corresponds with the value sampled by the sensor.

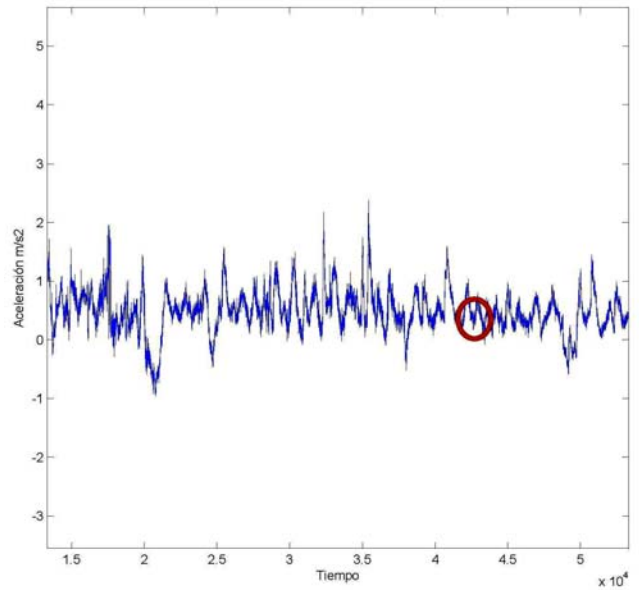


Figure 3. Acceleration in the x-axis (full flight)

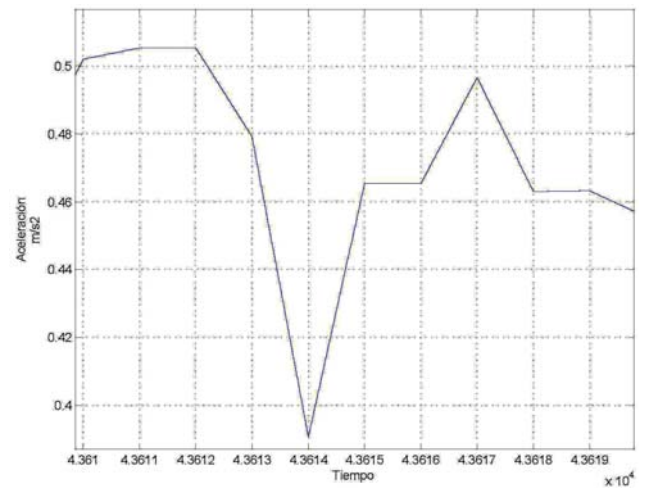


Figure 4. Extension of section marked in Figure 3

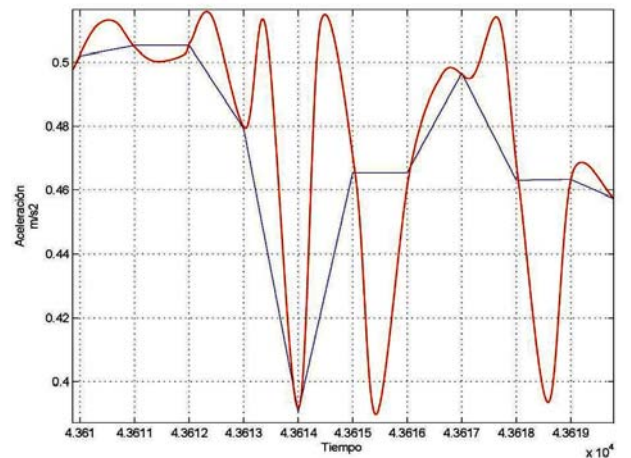


Figure 5. Values obtained

REFERENCES

- [1] Gonzalez, R., Millet, V., and Leon, R. "Mobile telemetry acquisitions system", International Telemetry Conference 1994, pp. 443-453.
- [2] U.S. Army, "IRIG Standard 106-86," Telemetry Standard Appendix C, Secretariat. Range Commanders Council, U.S. Army White Sands Missile Range, New Mexico 88002, chapter 4 pp. 1-12.
- [3] Parra, S. and Ángel, F. "Interfaz del NGFCS con la GCS." Madrid SIV/SPE/51CO/028/INTA/02, 2002, pp. 8-12.
- [4] U.S. Army, "IRIG Standard 119-06," Telemetry Applications Handbook, Secretariat. Range Commanders Council, U.S. Army White Sands Missile Range, New Mexico 88002. chapter 8 pp 12-16.
- [5] Electronic Industries Association, "EIA232E", Engineering Publications Office, pp. 32-45.
- [6] International Organisation for Standardisation MPEG-4 Overview – V.21, ISO/IEC JTC1/SC29/WG11, pp. 52-64.
- [7] Stevens, B. and Lewis, F. "Aircraft control and simulation", 2nd ed., Wiley-Interscience Publication, 1922, pp. 71-72.

The Smart Persistence Layer

Mariusz Trzaska

Software Engineering

Polish-Japanese Institute of Information Technology

Warsaw, Poland

mtrzaska@pjwstk.edu.pl

Abstract— We present an approach to solve the impedance mismatch problem caused by incompatibility between two models: object-oriented and relational ones. We believe that it cannot be unraveled by creating new Object-Relational Mappers (ORMs) like most of the software industry does. It is caused by some inherent differences between those two worlds. In our method we assume that both a programming language and a data source should be based on the same data model. Thus we propose a persistence layer for native data structures of a programming language. The presented idea is supported by a working prototype called the Smart Persistence Layer, which also supports extent management and bidirectional links. The prototype together with LINQ, the native query language for the .NET platform, formulates an easy-to-use yet powerful solution.

Keywords-Impedance mismatch; Databases mapping; Object-Relational Mappers; ORMs; Persistence; LINQ.

I. INTRODUCTION

The impedance mismatch is a negative software development phenomenon denoting severe incompatibility between two models: object-oriented and relational ones. It is caused by the fact that most modern software is implemented in object-oriented programming languages, but its data is persisted using relational databases. Such an approach forces the necessity of translating a rich object-oriented universe to a pretty simple relational world and vice versa.

In 2004, Ted Neward coined the phrase "Object/relational mapping is the Vietnam of Computer Science" [1]. His thesis was based on the observation that in the Vietnam and ORM cases there are less and less hope for success and unacceptable consequences of giving up. Two years later the phrase became famous thanks to Jeff Attwood who published the paper [2]. The paper mainly confirmed Neward's observations. One of the most important conclusions is choosing a single model both for the programming and data. Any other options are vulnerable to some level of the impedance mismatch.

This approach might be seen as too radical but in our opinion it is the only right choice. Contrary to the Attwood's preferences [2] we believe that the better choice is to select the object-oriented side rather than the relational one.

Unfortunately, a few years have passed since the phrase was coined, and nothing has changed on the battlefield. Even worse, it seems that nothing will change in the next few years. The software industry focuses on improving ORMs

rather than changing the approach to the problem. It looks like a situation where one is looking for a better and better medicine rather than eliminating the source of the illness. We believe that improving ORMs is questionable because there are too big discrepancies between the models and too big risk that attempts to match them will cut a lot from their functionalities. Usually, in such a cases and for large databases the object model is the victim: object-oriented qualities are reduced to minor (mostly syntactic) differences between the object and relational data schemas. The object model becomes a slave of the relational model. It is not possible to create a generic mapper, which will be able to automatically transform object-oriented queries addressing sophisticated object model into relational queries and commands (SQL), and vice versa. The main reason of that is the fact that probably there is no general algorithm that maps object-oriented queries and updates into SQL and still ensures good performance. In typical cases (our experience from other projects [3]) a mapper uses non-standard SQL features (e.g., traversing tables by cursors), thus the SQL query optimizer has no chances to work properly. Hence each case has to be manually designed by the programmer. In fact, it does not even matter how the mapping is to be defined: using a configuration file, a DSL or some other way. The result is still the same: the programmer has to spend his/her valuable time doing some repetitious and error-prone work.

The problem is not only related to mapping definitions by programmers. It is much more extensive and spreads on query languages, different types, semantics, etc.

There are opinions that solving the impedance mismatch problem should employ extending programming languages with declarative specification capabilities like JML [4] or Spec# [5]. Generally we do not agree with such a solution mainly because of the complexity, e.g., Spec# requires a dedicated compiler.

Our proposal is based on replacing both an ORM and a database with a data source native to a programming language. As a result, there is no impedance mismatch at all. The approach is supported by a working prototype for the .NET platform. The prototype provides a persistence layer and extent management for objects of a programming language.

The rest of the paper is organized as follows. To fully understand our motivation and approach some related solutions are presented in Section 2. Section 3 briefly discusses key concepts of our proposal and its

implementation. Section 4 contains sample utilizations of the prototype and simple benchmarks. Section 5 concludes.

II. RELATED SOLUTIONS

As we suggested previously, to reduce completely the impedance mismatch we need to leave the object model and to eliminate another data model. It means that both business logic and data store will be on the programming language's side or the database side. Both approaches have their advantages and disadvantages. We discuss them shortly.

A. *The Programming Language Side*

This approach requires that a business logic and a data source are implemented on the programming language side. It involves a dedicated data source, which is not only compatible with the programming language but fully native to it. The compatibility condition is quite common and means ability to work with a particular platform. However, it does not mean common models. The most obvious examples are relational databases and ORMs. Undoubtedly, the latter are more convenient for programmers but still require at least manual mappings.

The nativity condition is fulfilled when plain objects of a programming language are persisted using an additional tool. Usually the tool has to be an object-oriented database management system (ODBMS), i.e., db4o [6], [7] or Objectivity [8]. Both of them are mature solutions existing on the market for at least 10 years. However in some cases, using them could be too complicated. Thus, a more lightweight solution would be a better choice. Our proposal follows this idea. More information, comparing the db4o to our prototype could be found in Section 3.

The reference [9] provides a list of open source persistence frameworks for the MS .NET platform. Unfortunately, most of them are implemented as ORMs, which of course introduces some level of the impedance mismatch. We have found only two tools, which do not utilize a relational database: Bamboo.Prevalence [10] and Sisyphus [11]. However they usually require some special approaches, e.g., the command pattern utilized for data manipulation for the Bamboo and necessity of inheritance from a special class for the Sisyphus.

B. *The Database Side*

This solution utilizes the database model both for business logic and data. Thus it requires that the entire application is implemented in a database programming language. There are various DBMS and dedicated languages on the market, i.e., T-SQL, PL/SQL. Both of them have imperative functionality and PL/SQL has some object-oriented constructs. There are also fully object-oriented solutions like SBQL for the ODRA platform [12]. These seem more appropriate thanks to the more powerful and flexible model.

The ODRA (Object Database for Rapid Application development) is a prototype object-oriented database management system based on SBA (Stack-Based Architecture). The main motivation for the ODRA project is to develop new paradigms of database application

development. This goal is going to be reached mainly by increasing the level of abstraction at which the programmer works. ODRA introduces a new universal declarative query and programming language SBQL (Stack-Based Query Language), together with a distributed, database-oriented and object-oriented execution environment. Such an approach provides functionality common to the variety of popular technologies (such as relational/object databases, several types of middleware, general purpose programming languages and their execution environments) in a single universal, easy to learn, interoperable and effective to use application programming environment.

III. THE SMART PERSISTENCE LAYER

Programmers use databases for many reasons. One of the more important are persistence and a query language. A few years ago Microsoft introduced a query language called LINQ [13] to ordinary programming languages (e.g., C# and Visual Basic). The LINQ works with native collections of the programming language allowing querying them as regular databases. It is also supported by various ORM mappers including their own solution called Entity Framework [14]. Generally speaking, the mapper uses a relational database for storing data which, of course, causes some impedance mismatch (especially concerning inheritance).

Our approach is based on an observation: if we have a query language (LINQ) natively supported by the programming language, then we should use native data structures of the language as well. Such an approach guarantees that every bit of impedance mismatch simply disappears. Of course, in real case scenarios a persistency for the native data is required. At first glance it looks that such a mechanism already exists for modern programming languages and is called serialization. Unfortunately, it is not applicable as a replacement for databases. The main reason is the fact that the serialization every time stores the entire graph of objects. This behavior is caused by the way the serialization works: every saved object is valid, which means storing all connected objects, objects of connected objects and so on.

Our proposal focuses on delivering a persistency layer designed in a totally transparent way for the programmers. We do not want to make programmers use any kind of super classes or implementing special interfaces. The prototype is called The Smart Persistence Layer (SPL) and implemented for the MS .NET platform. However, it is possible to implement it for other platforms with the reflection capabilities, i.e., Java. In this case it would be possible to reuse significant parts of the source code and data files as well.

A. *The Basic Functionality*

The most basic functionality for a mapper is delivering an extent of objects belonging to a particular class. This could be achieved using many ways. For instance the db4o [8] uses the following code:

```
IList<Pilot> pilots =
db.Query<Pilot>(typeof(Pilot));
```

However in our prototype we have simplified that to:

```
IQueryable<Pilot> pilots =
db.GetExtent<Pilot>();
```

Please note that our method does not require the parameter, but the result is still strongly typed.

There is also a debate how objects belonging to different classes in the same inheritance's hierarchy should be treated. We believe that the extent of a super class must also contain all instances of subclasses. This approach guarantees that we can work on a higher level of abstraction (i.e., different subclasses of product processed just like products; see also Section 4). Of course, this relationship works only in one direction: extents of subclasses will not contain instances of super classes. Hence the above code returns a collection of objects belonging to the given class (as a type parameter) and all subclasses.

Another area related to an extent, which needs a clarification is how and when new objects will be incorporated into extent. Our proposal follows the following rules:

- an object could be added to an extent by executing by a programmer a dedicated method;
- every object, which is directly made persistent by a programmer is added to an appropriate extent.

If a programmer would like to achieve automatic adding to an extent, then the method could be executed in a constructor of a class. It is especially easy thanks to our designing decisions. We have utilized the C#'s extension method mechanism together with the default instance of the SPL. An extension method is a method adding a functionality to a class but defined outside the class. The listing 1 (due to readability all listing are located at the end of this paper) presents the mentioned method. Please note that the method's parameter is of type `object`, which means that any object could be added to an extent (and the extension method could be executed on any existing object). A dedicated logic adds a given object to appropriate extents (the current one and all super classes). This is performed based on the object's type. A similar extension method has been utilized for the `Save` operation, which persists a given object.

Another interesting concept is the default instance of our prototype layer. In case of many applications a persistence layer is available via a single object, i.e., a file stream or a DB instance/connection. Hence, we have introduced a concept of default instance, which is the first (and in many cases the only one) instance of the persistence object. The object has to be properly initialized at the very beginning. Otherwise, during accessing the default instance, appropriate exception would be thrown. This solution allows accessing the data without passing a reference to the object. This is also the case of the previously mentioned method adding an object to its extent.

Such an approach does not put any restraints on programmers i.e., implementing an interface or inheriting from a super class.

B. Bi-directional Associations

One of the key functionality of every data store is the ability for creating and persisting connections among objects. In our opinion, it is especially useful if the connections are bidirectional allowing navigation in both directions (i.e., from a product to its company and vice versa). Unfortunately, databases usually do not support the feature. According to [7] the db4o does not have it either. This is also the case of native references existing in popular programming languages (e.g., MS C#).

The implementation of the mentioned functionality is complicated especially if we would like to work with the POCO (Plain Old CLR Object) objects. This approach means that we cannot expect implementing a specified interface or functionality inherited from a super class. Another disadvantage of putting links into a super class would be problems with navigation using the LINQ.

Thus our goal was to design it as convenient as possible but still remembering that it would be extremely hard to find a perfect (totally transparent to a programmer) solution.

One of the approaches is generating classes based on same templates. This is the case of one of the options in the Microsoft Entity Framework [14]. However, this functionality requires some kind of support from a tool and in our opinion may not be useful for all programmers.

It seems that creating a bidirectional link requires defining the following data:

- role name,
- reverse role name,
- target object,
- reverse object.

We had to choose how and when to put them to minimize the amount of work required from a programmer. At the beginning we tried creating special annotations for classes. But it turned out that some data still has to be passed as string. After some research we came up with another solution, which spreads on two different levels (see Fig. 1).

The first one is a dedicated class parameterized with two types: target objects (`TTargetType`) and reverse object (`TReverseType`). Utilizing a parameterized class makes possible detecting some errors during a compilation time. The next level uses information passed to the constructor of the class. It takes a reverse attribute name, which will store the reverse link and an instance of the class, which should be the reverse target. The following listing presents the code, which should be placed inside a business class (see also Section 4).

```
ICollection<Tag> Tags = new
SplLinks<Tag, Product>("Products",
this);
```

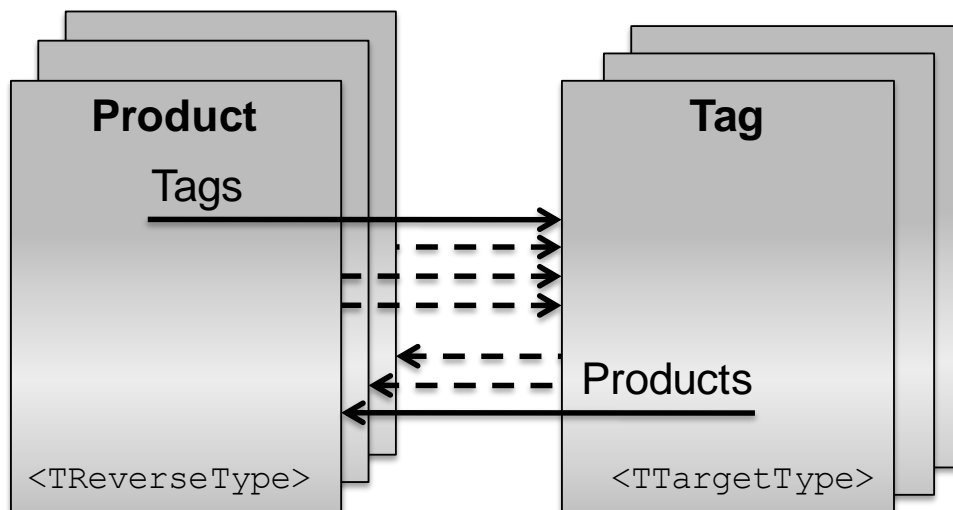



Figure 1. Explanation of the implemented bidirectional links mechanism.

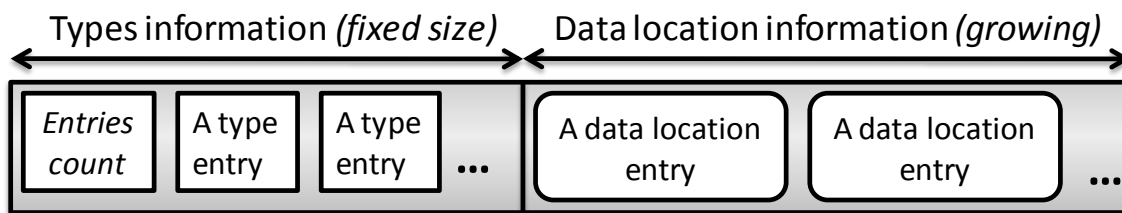


Figure 2. Structure of the file storing types and location information

It may look a bit complicated but it is created only once for each link. The `Splinks` class implements ordinary .NET interface for accessing collections thus using it is exactly the same as any other .NET collection. Creating a bidirectional link requires only executing a single `Add` method with the target object. The reverse connection will be created automatically based on previously defined data. Of course, all LINQ queries work as well.

C. The Transparent Persistence

The goal of the persistence process is to store data on some non-volatile media, usually in a disk file. In case of our solution we need to persist three types of data, namely:

- business content of the objects,
- location of the above,
- information about types (classes).

All of them can change and grow during the run-time. After some research we have decided to use two files: the first one will hold business information whereas the second the rest. Initially we thought about three files but the types information is usually quite small and repeatable thus can be stored at the beginning of the second file (Fig. 2). A programmer can define amount of the allocated space for the purpose. A default value is 1MB, which makes possible storing about 3000 entries. It is possible to use just one file

but at cost of more complicated design and possibly worse performance.

The single entry regarding the location of data (the *type entry* from Fig. 2) consists of:

- object identifier;
- identifier of its type;
- location in the data file where the object's content starts. This entry is updated every time when an object is saved;
- location in the index file where the location data starts.

The above information also exists in the memory to boost performance. It is saved to disk only as a backup and for reading objects purposes.

As mentioned previously we do not persist classes (types) in the file. Thus during an object initialization those classes have to be accessible by the .NET run-time (e.g., as standard DLL libraries).

The other file, with business data of persisted objects, can be read only using the location and types information. It is read at the very beginning. The current prototype reads all data to the memory. This could be a problem in some cases but modern computers are usually equipped with a lots of RAM. However, in the future versions we will probably introduce some kind of programmer's policy for defining this kind of behavior.

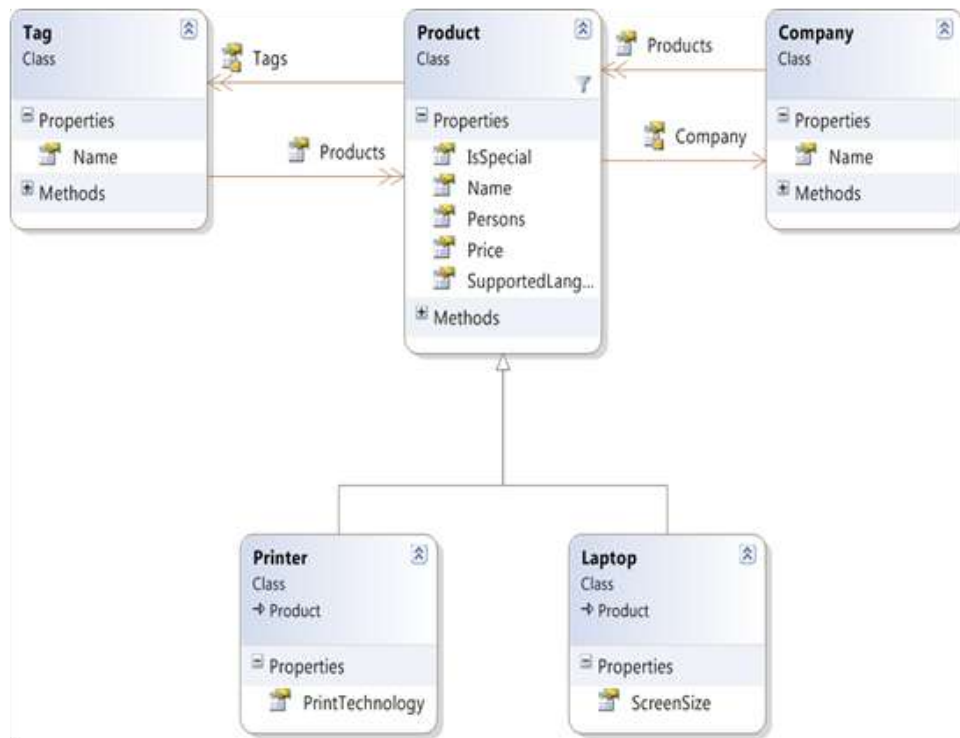


Figure 3. A class diagram of the sample implemented using the SPL.

The process of saving and reading objects intensively uses the reflection mechanism. Currently it is able to deal with atomic types, lists (classes implementing the `ICollection` interface), `ICollection` (see Section 3.B) and other types built using these invariants (see also the sample utilization in Section 4).

One of the problems related with links, which should be addressed, is persisting connected objects. When we would like to persist an object, how should we act with all referenced objects? There are different approaches, i.e., db4o [7] uses a concept called *update depth*. This is simply a number telling how many levels of connections should be saved. We have decided to follow another approach. When we save an object, all referenced unknown (not saved previously) objects are saved, no matter how deep they are. Thus the first execution could be costly, but the objects have to be saved anyway. All next updates will not save known objects. If a programmer wants to save them, then it has to be done directly by executing the `Save` method. The method should also be utilized every time a single object is modified (its content will be persisted in the file). This policy guarantees that persisting an object will not be costly.

IV. THE USE CASE AND SOME BENCHMARKS

Fig. 3 presents a class diagram of the sample created using our prototype implementation. It describes the following business case:

- Products have various properties including: a name, a price and a list of supported languages;
- Every product can be described using various tags;

- A company manufactures many products, but a product is related to a single company;
- There are various kinds of products with different properties. Printers contain information about utilized print technology and laptops store a screen size.

Although the presented case is quite simple, it contains different kinds of business information. Thus it allows verifying the usefulness of our approach.

Listing 2 contains the complete source code of the `Product` and `Tag` classes. The code, aside from normal C# functionality, together with the SPL provides full persistency, extents and query capabilities (thanks to the native LINQ). No additional configuration/mapping files, known from ORMs, nor special identifiers are required. Please note utilization of different types of data including the `SplLinks` class accessed using a standard C# interface (`ICollection`).

Similar simplicity can be observed on listing 3. A programmer creates instances of the `Product` and `Tag` classes, links them together (the `Tags` property) and persist (the `Save` method) using a few simple steps.

The important aspect of every data management system is its performance. We plan to perform detailed tests comparing our solutions to other approaches including ORMs and raw databases. Currently we have run some simple tests measuring speed of our prototype (the test computer configuration: Intel Core i7 2.93GHz, RAM: 8GB, Windows7 x64). The results are promising.

The test utilized two classes from the above business sample: the `Product` and `Company` (Fig. 3). They were connected using our bidirectional link. Table 1 presents times required by various operations.

TABLE I. RESULTS OF THE SIMPLE PERFORMANCE TESTS (ALL RESULTS ARE IN SECONDS; LESS IS BETTER)

Number of objects and the operation	Products: 50,000 Companies: 5,000 Total: 55,000 objects	Products: 100,000 Companies: 1,000 Total: 101,000 objects
Initializing the SPL	0.0180 s	0.0180 s
Generating and persisting data	17.3210 s	31.4018 s
Retrieving entire extent of Products	0.0100 s	0.0130 s
Retrieving entire extent of Companies	0.0040 s	0.0050 s
Opening file, reading all data and creating objects	21.9753 s	58.4323 s
Query Products for the price (LINQ)	0.0320 s	0.0550 s
Query Products with the specified Company's name (LINQ)	0.0120 s	0.0330 s

As it can be seen, the results are decent, especially for an early prototype. Please note short times for executing the LINQ queries, i.e., finding all products manufactured by a particular company took only 0.03s (for 100,000 products). It is probably caused by the fact that in the current prototype all data is kept in the RAM memory and a disk file is only a backup. That's why the time of opening the file and reading all data could be significant in case of bigger data sets (for 101,000 objects it is about 58 seconds). As we mentioned previously, we plan to add an option for loading data only when needed.

V. THE CONCLUSION AND FUTURE WORK

The impedance mismatch is a real problem experienced by many programmers for a very long time. In this paper, we have presented our approach to solve it. The idea is based on eliminating the causes rather than improving medicines (in this case various ORMs). We believe that the best method is to use the same coherent model both for programming and a data source. This could be achieved by providing a persistence layer and extent management for native objects created in a particular programming language.

The mentioned solution is even more useful if there is an existing query language natively supported by the

programming platform. This is the case of the .NET and the LINQ query language. The implemented prototype follows our proposal by adding persistency and extent functionality to standard C# objects. Moreover the functionality has been achieved without imposing on a programmer any special requirements regarding a super class nor interfaces.

Furthermore, our prototype adds functionality for easy-to-use bidirectional associations. They are usable as standard C# collections implementing the `ICollection` interface.

As a future work we would like to extend our prototype with some other useful functionalities associated with databases like indexes or transactions. However, we would like to implement them (in a way) preserving the lightness and flexibility of our solution.

Another field, which could be researched is performance. We are going to conduct dedicated tests comparing our prototype to other similar solutions like object-oriented databases or ORMs.

REFERENCES

- [1] Neward, T.: The Vietnam of Computer Science, <http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx>. Last accessed: 02-04-2011
- [2] Atwood, J.: Object-Relational Mapping is the Vietnam of Computer Science, <http://www.codinghorror.com/blog/2006/06/object-relational-mapping-is-the-vietnam-of-computer-science.html>, Last accessed: 02-04-2011
- [3] Kuliberda, K., Wiślicki, J., Adamus, R., and Subieta, K.: Object-Oriented Wrapper for Relational Databases in the Data Grid Architecture, w: On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops, Agia Napa, Cyprus, October 31 – November 4, 2005, Proceedings. LNCS 3762, Springer 2005, pp. 528-542
- [4] Chalin, P., R. Kiniry, J., T. Leavens, G., and Erik Poll. Beyond Assertions: Advanced Specification and Verification with JML and ESC/Java2. In Formal Methods for Components and Objects (FMCO) 2005, Revised Lectures, pages 342-363. Volume 4111 of Lecture Notes in Computer Science, Springer Verlag, 2006, pp. 342-363
- [5] Barnett, M., Rustan K., Leino M., and Schulte W.: The Spec# programming system: An overview. In CASSIS 2004, LNCS vol. 3362, Springer, 2004, pp. 144 - 152
- [6] Paterson, J., Edlich, S., and Rning, H.: The Definitive Guide to Db4o. Springer (August 2008), ISBN: 978-1430213772
- [7] db4o tutorial, <http://developer.db4o.com/Documentation/Reference/db4o-8.0/net35/tutorial>. Last accessed: 2011-04-02
- [8] The Objectivity Database Management System. <http://www.objectivity.com>. Last accessed: 2011-04-02
- [9] Open Source Persistence Frameworks in C#. <http://csharp-source.net/open-source/persistence>. Last accessed: 2011-04-02
- [10] Bamboo.Prevalence - a .NET object prevalence engine. <http://bbooprevalence.sourceforge.net/>. Last accessed: 2011-04-02
- [11] Sisyphus Persistence Framework. <http://sisyphuspf.sourceforge.net>. Last accessed: 2011-04-02
- [12] Adamus, R., Daczkowski, M., Habela, P., Kaczmarek K., Kowalski, T., Lentner, M., Pieciukiewicz, T., Stencel, K., Subieta, K., Trzaska, M., Wardziak, T., and Wiślicki, J.: Overview of the Project ODBA. Proceedings of the First International Conference on Object Databases, ICOODB

2008, Berlin 13-14 March 2008, ISBN 078-7399-412-9, pp. 179-197.

[13] Magennis, T.: LINQ to Objects Using C# 4.0: Using and Extending LINQ to Objects and Parallel LINQ (PLINQ).

Addison-Wesley Professional, ISBN-13: 978-0321637000 (2010)

[14] Lerman, J.: Programming Entity Framework: Building Data Centric Apps with the ADO.NET Entity Framework. O'Reilly Media, Second Edition, ISBN: 978-0-596-80726-9 (2010)

LISTING 1. AN EXTENSION METHOD ALLOWING ADDING AN OBJECT TO ITS EXTENT

```
public static class Helpers
{
    // ...
    public static void AddToExtent (this object objectToAdd)
    {
        NmoDatabaseManager.DefaultInstance.AddToExtent(objectToAdd);
    }
}
```

LISTING 2. A CODE USED FOR THE PRODUCT AND TAG CLASSES

```
public class Product
{
    public string Name { get; set; }
    public decimal Price { get; set; }
    public bool IsSpecial { get; set; }
    public IList<string> SupportedLanguages { get; set; }
    internal ICollection<Tag> Tags { get; set; }
    internal Company Company { get; set; }

    public Product() {
        Tags = new SplLinks<Tag, Product>("Products", this);
    }
}

public class Tag
{
    public string Name { get; set; }
    public ICollection<Product> Products { get; set; }

    public Tag() {
        Products = new SplLinks<Product, Tag>("Tags", this);
    }
}
```

LISTING 3. A CODE USED FOR PERSISTING INSTANCES OF THE PRODUCT AND TAG CLASSES

```
var tagSpecialOffer = new Tag() {Name="Special Offer"};
var product1 = new Product() {Name="Everyday Desktop VX5000", Price=799.0m,
    SupportedLanguages = new List<string>() {"en",
        "de", "pl"}};

product1.Tags.Add(tagSpecialOffer);
product1.Save();
```

UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems

Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin
 Mälardalen Real-Time Research Centre (MRTC)
 Mälardalen University, Västerås, Sweden
 {mehrdad.saadatmand, antonio.cicchetti, mikael.sjodin}@mdh.se

Abstract—Successful design of real-time embedded systems relies heavily on the successful satisfaction of their non-functional requirements. Model-driven engineering is a promising approach for coping with the design complexity of embedded systems. However, when it comes to modeling non-functional requirements and covering specific aspects of different domains and types of embedded systems, general modeling languages for real-time embedded systems may not be able to cover all of these aspects. One solution is to use a combination of modeling languages for modeling different non-functional requirements as is done in the definition of EAST-ADL modeling language for automotive domain. In this paper, we propose a UML-based solution, consisting of different modeling languages, to model non-functional requirements in telecommunication domain, and discuss different challenges and issues in the design of telecommunication systems that are related to these requirements.

Index Terms—Non-functional requirements; Telecommunication domain; UML modeling; Real-Time Embedded Systems; MDA

I. INTRODUCTION

The nature of embedded systems such as resource constraints, close integration and interaction with the environment through sensors and actuators (which can also incur requirements on safety), timing characteristics and lack of traditional user interfaces all bring with themselves requirements that make the design of these systems complicated [1]. Much of this complexity is due to handling a big range of different requirements, solving conflicts and finding the right balance and trade-offs among them. Especially non-functional requirements such as security usually cross cut organizational structures and development teams. Thus traditional functional decompositions do not suit them. However, compared to functional requirements not much work has been done on non-functional requirements and lack of proper methods and techniques for modeling of non-functional requirements and their integration into the development lifecycle are felt [2].

UML profile for Modeling and Analysis of Real-Time Embedded systems (MARTE) [3] is one of the recent and major efforts on modeling Real-Time Embedded Systems (RTES) and the non-functional properties in these systems. MARTE enables detailed modeling of RTES and facilitates their analysis. On the other hand, there is a big variety of systems in RTES domain and to cover the specific aspects and needs of each group of those systems (subdomains), a customized modeling approach is necessary. Such an approach

has been used in the automotive domain, leading to the definition of EAST-ADL profile [4] for modeling of vehicular systems.

This paper focuses on telecommunication systems and the aspects that modeling approaches for such systems should be able to cover regarding their non-functional requirements. We propose a UML-profiling approach consisting of features from different modeling languages to answer broader aspects in modeling non-functional requirements of telecommunication systems. One of these aspects is security. We will focus on security in this paper as an example for one of the intrinsic characteristics of telecommunication domain that is also not supported in EAST-ADL. Through an example, we show how it will be possible to model security requirements along with other aspects such as power, in one model while establishing traceability between high requirements and their refinements (lower level ones).

Regardless of the set of non-functional requirements that a subdomain in RTES has, modeling approaches for these systems should provide requirements traceability. This becomes even more important due to limited resources that systems in this domain have; while in other systems, it is usually a lot easier to add extra resources to the system such as additional memory and that way fulfill a requirement. Therefore, a more careful balance and trade-off analysis between requirements is necessary in order to satisfy all of them in RTES domain. Having traceability links among requirements and also between requirements and design artifacts facilitates to perform impact analysis and identify the effects a change on one requirement can have on other parts of the system.

To cover different aspects regarding non-functional requirements in telecommunication systems, we suggest a UML profiling solution consisting of concepts from SysML [5] for traceability, and MARTE for modeling general non-functional properties and their analysis. For security requirements, which are inherent in telecommunication domain but are not covered by MARTE, we adopt from available UML profiles for security, namely UMLsec [6]. Also since MARTE, SysML and UMLsec are UML profiles, they are faster for developers using UML to catch on and they also serve as a possible unifying factor between development departments. A comparison of different ways to define Domain Specific Languages (DSL) and the benefits of each approach are provided in [7], [8]. It is also important to note here that combining different UML

profiles is not a trivial task as it may seem and it can incur different problems such as semantic conflicts. These issues are discussed in an interesting work in [9].

The contributions of this paper can be summarized in the following points:

- Showing an approach on how to model non-functional requirements in telecommunication systems
- Identification of issues that should be taken into account in modeling those requirements and the modeling concepts to cover them

As a guideline we use our observations during a project we have done at Ericsson plus the results of other studies such as [2], [10] to describe the modeling challenges. In Section 2, we provide a deeper understanding of telecommunication systems, its characteristics and needs, and the problems observed around non-functional requirements in those systems. We discuss the related work and have a look at some modeling solutions in automotive domain in Section 3. In Section 4, we describe the ingredient concepts of our proposed approach for modeling non-functional requirements in telecommunication domain by highlighting some key relevant concepts offered in SysML, MARTE and UMLsec. Section 5 shows the application of the method as a usage example. In Section 6, we compare the features of our suggested approach with those of EAST-ADL in automotive domain, and finally, in Section 7 we summarize the work and suggest different areas that need to be studied as future work.

II. MOTIVATIONS

The observations and results in this section are achieved through collaboration with Ericsson engineers (Stockholm, Sweden) and gathered through several meetings with different teams, such as Radio Base Stations (RBS) development group, during CHES project [11] at Ericsson.

A. Telecommunication Systems

As a type of real-time embedded systems, telecommunication systems have specific characteristics, which incur certain requirements and prioritization of some requirements over the others. These systems need to be secure, are highly distributed, have a dynamic nature, require massive processing capacity and high availability (99.999% availability, which is sometimes referred to as *five nines*), and need to be scalable. The distribution in these systems can be regarded in two perspectives: the distribution inside one node (such as using multicore solutions and distribution of software functions among different processing units) and also the geographical distribution of nodes across different regions and the communication among them.

Typically, telecommunication networks consist of many different types of nodes such as Radio Base Stations (RBS), Radio Network Controllers (RNC), Media Gateways (MGW) and others that span across a big geographical area and communicate over different kinds of lines.

Regardless of the integration and interconnection of different nodes in the network, design of each node is a big complex

challenge in itself. For example, an RNC can easily contain between 500 to 700 CPUs, with software functions spanning across several CPUs. This number, however, is decreasing as new processors with higher capacities are produced. This reduction is important for the total cost, power consumption and heat generation of systems. As for functionality and services, in a typical telecommunication system a big number of connections should be established, routed and managed per second. Besides, cost calculation should also be done on them. Moreover, a typical telecommunication system can have a life span of about 20-30 years. Thus upgrade-ability and maintenance of such systems is also of great importance. Software upgrade should be done in such a way to have the least effect on the availability of the system. That is why requirements such as hot-swapping and plugging and the ability to perform restarts at different granularity levels (a single board, collection of boards or a complete node) are highly desirable and demanded in this domain.

B. Problems with Non-Functional Requirements

Due to the hierarchical and subsystem structure of telecommunication systems, first overall non-functional requirements are defined on the system and then they should be refined several times and in each step more concrete and design-decision information is added. However, not all requirements get refined, and as discussed in [10], this leads to weak traceability chains. What can happen is that the requirements that are defined on the system model are *consumed* (meaning that they are read and implemented in the system) and no explicit connection between the design artifact and the requirement leading to that design decision gets established. Also for verification, most of the requirements are tested on a reference configuration and then if some requirements are not met, changes are applied on the system model and again a reference configuration is built with the new requirements. Basically, there are two general issues with this current approach:

- Poor support for traceability of requirements to design artifacts
- The feedback loop for analysis of non-functional requirements takes much time and effort and the wish is to be able to perform verification of non-functional requirements at earlier phases

The organization in large companies usually have a hierarchical structure, which suits the actual system hierarchical structure as mentioned above. According to the study done in [2], this organizational structure matches the system structure well, as subsystems tasks and modules are allocated to specific departments and thus is more suitable for functional requirements. However, this is not the case for non-functional requirements. The problem as mentioned in [2] is that the autonomy of departments at the lowest levels of hierarchy makes management of non-functional requirements harder and that the decisions about these requirements should be done at higher levels of hierarchy and aligned and managed from top to down. This problem becomes more obvious with certain

types of requirements such as security, usability and user-interface characteristics, which should be aligned in different subsystems. Thus, non-functional requirements can easily cross-cut organizational structure of a company and therefore a methodology that works for functional decomposition may stop to work for non-functional requirements.

In such organizations, it also happens that different teams may have different interpretations and definitions for some non-functional requirements, which can cause problems for communication between the teams. On the other hand, this also means that people from different teams may talk about a specific requirement using different terms. If we can provide a consistent way of modeling non-functional requirements and a mechanism to establish associations between a requirement or a design artifact and its source requirement, such problems can be mitigated and detected more easily. Also as can be implicitly noticed from the discussion in previous section, there are many requirements that have conflict with others, and trade-off decisions to balance them need to be taken. However, these compromises and decisions, which may be made inside a subgroup, are somewhat unknown to upper levels and are only known by some engineers working in that section. For example, it is quite common that specific tweaking and settings in the code on bandwidth or memory usage are applied to ensure a certain level of balance between performance and maintainability of the system. Such decisions even if documented are hard to follow and track later on, especially for upper levels in the organizational hierarchy. On the other hand, some requirements that are decided on higher levels are lost in the transition to go to development teams in lower levels of hierarchy. This observation is also in alignment and confirmed by the study in [10], which states the problem as non-functional requirements "are not always available when needed". These issues can be alleviated by applying traceability (which can be traversed back and forth) between requirements and using a better form for representation and documentation of non-functional requirements.

III. RELATED WORK

Requirement Modeling: Telecommunication Standardization Sector (ITU-T) have offered several languages for system modeling in telecommunication domain. Each of these languages try to target different aspects and phases in system development. For example, Message Sequence Chart (MSC) is used for modeling asynchronous interaction scenarios. Specification and Description Language (SDL), which has both textual and graphical representations, uses block, process, channel and signal concepts to describe behavior in communicating real-time systems. At higher abstraction layers and for modeling requirements, ITU-T has suggested User Requirements Notation (URN). URN consists of two notations: Goal-oriented Requirement Language (GRL) to model goals and non-functional requirements and Use Case Maps (UCM) to describe functional scenarios. GRL is used to capture informal system goals, specification and rationals. We refer interested readers to ITU-T website [12] for more

information on these languages. Some efforts have been done to define these languages as UML profiles such as [13].

As for general UML-based approaches in RTES domain, MARTE with its expressive power and formal semantics enables capturing non-functional requirements in more formal ways and with necessary details for performing analysis earlier in system development phases. For system engineering, modeling general requirements and the relationships among them, SysML offers Requirements model, and semantics and notations for requirements traceability.

Modeling Security Requirements: There have been efforts on modeling and analysis of security aspects using UML to bring them into earlier phases of development. For example, SecureUML [14] focuses on modeling Role-Based Access Control (RBAC) by extending UML as a profile, while AuthUML [15] is a framework for analysis of access control in the specification phase and thus less suited for code generation. UMLsec on the other hand, uses stereotypes and tag values for modeling general security aspects such as secure links, connections, RBAC, secure information exchange, etc. to enable analysis and early automatic verification (which also matches our goal for early analysis of requirements). A comparison between SecureUML and UMLsec for modeling role-based access control is done in [16]. The UMLsec analysis tool suite can help to identify parts of the model that do not match a specified security requirement. This enables to perform a level of security analysis on the model and find inconsistencies before going into implementation phases. As for other works in this area, the study in [17], for example, introduces stereotypes to specify vulnerabilities so that developers can notice them and avoid in implementation. It also claims that these specifications can be used to generate test cases for security. Article [18] tries to merge Mandatory Access Control (MAC) and Discretionary Access Control (DAC) with RBAC. It is a good work for modeling access control aspects, but lacks other security aspects of UMLsec and their analysis. Doan and Demurjian [19], on the other hand, discuss security analysis based on RBAC and MAC in use-case and class diagrams. Houmb and Hansen [20] introduce SecurityAssessmentUML, which is intended to capture and document the results of risk (i.e., vulnerabilities, threats, etc.) identification and analysis. Discussion and comparisons of different UML-based security models can be found in the related work sections in [6], [14], [17]–[19].

Requirement Modeling in Automotive Domain: As an example of a UML-based domain-tailored approach, EAST-ADL has been developed in automotive domain for modeling software architecture and electronic parts of a system. By complementing and making use of general available modeling solutions in RTES domain, EAST-ADL tries to cover the specific requirements of automotive domain. It adopts concepts from UML, AADL [21] and SysML to provide modeling semantics aligned with AUTOSAR [22] specification. AUTOSAR focuses on lower design levels such as component model, software modules, control units, APIs and implementation parts of automotive systems.

For modeling requirements, EAST-ADL makes use of SysML requirements semantics and specializes them to match automotive domain (e.g., definition of timing, delay and safety requirements). However, it does not provide enough features to enable some analyses such as scheduling and timing verifications earlier than implementation phase [23]. There are studies such as [23] that suggest decorating EAST-ADL models with some features from MARTE such as timing and allocation packages to enable early scheduling analysis. TIMMO project [24] is one of the efforts using this idea to complement timing model of EAST-ADL for automotive domain. In general, EAST-ADL and its requirement model may not be appropriate and compatible as a whole for requirements in telecommunication domain. It does not cover security aspects, which are important for telecommunication systems, is aligned with EAST-ADL's specific abstraction levels, and is based on concepts like ECU, VehicleFeature, AutosarSystem, and Sensor which are not relevant for telecommunication systems. In order to better capture requirements of telecommunication systems that originate from their specific characteristics such as intensive performance demands, distribution, use of multi-core solutions, virtualization and hierarchical schedulers, etc. a tailored solution for this (sub)domain is required.

IV. SUGGESTED UML PROFILE

Adopting a model-based approach for the development of telecommunication systems helps to raise the abstraction level and cope with the design complexity. This also targets the challenge to shorten the feedback loop and enable analysis in earlier phases of development.

In this section, the key concepts that a desired UML profile for telecommunication systems should be able to offer are discussed. We explain traceability concepts from SysML, modeling general non-functional requirements with MARTE highlighting its relevant and interesting features for telecommunication domain and how to model security aspects along with an example of its analysis. Later in section 6, we compare the features of our suggested UML profile with EAST-ADL.

A. Modeling Traceability Using SysML

For modeling of requirements, SysML provides a specific diagram, which can be a solution to the issues regarding management of non-functional requirements of telecommunication systems identified in previous sections. An important feature of SysML is to represent requirements as first-class model elements. So requirements are included as parts of the system architecture and have semantics [25]. This also enables establishing relationships between requirements and other model elements showing, for example, design artifacts implementing and satisfying a requirement. It is possible to decompose requirements and create a hierarchy of requirements, which is needed to cope with the complexity of requirements faced in telecommunication domain. SysML provides different types of associations among requirements, which include: *copy*, *deriveReq*, *satisfy*, *verify*, *refine* and *trace*.

The counterpart of these associations are *derivedFrom*, *satisfiedBy*, *refinedBy*, *tracedTo*, *verifiedBy* and *master* properties that a requirement element can have. For example, *satisfiedBy* property of a requirement element contains the information of the model element that satisfies this requirement (counterpart of *satisfy* association). This way, SysML facilitates traversing back and forth between requirements and also model elements from high level departments in organizational hierarchy to lower level departments and development teams.

Another feature that SysML provides is requirements table. Requirements table provides traceability information for requirements in a single view, which is very helpful in managing the big number of versatile requirements in telecommunication systems. In this tabular representation of requirements, information such as requirements properties and types, dependency relationships with other elements/requirements and other information such as design rationale and test procedures may be included. By going through this table, it is possible to analyze the change (e.g., modification, deleting) effect of one requirement on other requirements in the systems. So basically, by providing different types of association and dependency and the tabular representation of requirements, SysML can answer problems identified for traceability and impact analysis of requirements in a complex and hierarchical telecommunication system. Moreover, by using stereotypes it is possible to extend SysML, which makes it very flexible to add new semantics such as new types of associations or requirements. An example of this extension is provided in [25], where three stereotypes for functional requirements, non-functional requirements and external interface are defined and used to model a system.

B. MARTE for Non-functional Requirements and Analysis Support

To represent the properties of non-functional requirements such as timing constraints in a formal way, MARTE provides rich modeling semantics. MARTE profile consists of different subpackages and in this section we try to identify packages and semantics in them, which serve to represent the type of non-functional requirements we identified in a telecommunication system.

MARTE NFP_Types, Value Specification Language (VSL) and the stereotypes defined in NFP package (Non-Functional Properties) help to define different non-functional properties specific to different domains. NFP package makes it possible to define percentage, dimensions, measurement precision and similar concepts for non-functional properties. Examples of basic NFP types already defined in MARTE type library can be power, frequency, duration, energy, weight, length, arrival-pattern (periodic,aperiodic, sporadic), price, etc. For time specifications, MARTE offers the time package and representation of time in MARTE can be in the form of a physical (continuous or discretized) or logical clocks (processor cycles, engine rotation, algorithmic steps...). The concept of multiform time provided in MARTE is very useful for telecommunication domain, which has already started heading for multi- and many-core solutions. The semantics to model the execution

platform (operating system, virtual machines, hardware) are packaged in Generic Resource Modeling (GRM), Software Resource Modeling (SRM) and Hardware Resource Modeling (HRM). With SRM it is possible to model concepts such as resources, services, concurrency and mutual exclusion features in a Real-Time Operating System (RTOS) as well as virtual machines, which are used in telecommunication systems. The stereotypes in HRM package enable modeling of processing units, different levels of memory, devices and their physical aspects such as layout in the system, power consumption, and heat dissipation. These concepts can be used to target non-functional requirements such as cost sensitivity, execution capacity, and environmental requirements (layout, size, structure, etc.).

An interesting feature provided in GRM is the modeling of primary and secondary schedulers, which enables modeling of systems having hierarchical schedulers. This is helpful for telecommunication domain in which, use of hypervisors and virtual operating systems on top of another operating system is common.

To model dependability requirements (reliability, availability, maintainability, safety), which is an important feature of telecommunication systems, there is a suggestion for an extension to MARTE, that is introduced in [26] as Dependability and Analysis Modeling (DAM) (sub)profile and offers relevant concepts such as threats (fault, failure, error, hazard, accident), maintenance, redundancy, etc.

C. Covering Security Aspects

Security in embedded systems is becoming more important and gaining greater attention. More mechanical parts are replaced by computer systems and the use of wireless technologies for communication between different units is becoming a dominant trend. In automotive domain for example, features such as traffic and accidents notification systems, built-in bluetooth devices and distance calculator between cars are representatives of such cases that require communication with other cars and devices. Such features along with electronic access controls (e.g., access to the vehicle internal bus and electronic locks) also open up the system for more security threats.

Due to the nature of systems in telecommunication domain, which naturally involve long distance communications and at a big level of distribution and scalability with many nodes and access points on the way, security aspects have always been an unavoidable part. A single telecommunication node such as a Radio Base Station (RBS) can serve different requests from different sources and these operations should be kept separate from each other keeping data intact and safe from interference. It becomes more critical when we add to the picture other services in the system such as call cost calculation for users and (recently) data traffic including images, emails, and other sensitive and personal information. However, in the design of a system, security considerations should not be considered as an add-on, but they should be taken into account from early phases of design.

UMLsec covers a broad scope and has a versatile tool suite for analysis. Using that we can complement modeling of security requirements as first class entities. With the help of SysML, relationships between them and other non-functional requirements and also design artifacts can be added and detailed non-functional properties using MARTE can be specified for them if necessary. So for example, it becomes possible to model nodes in a system as resources using MARTE GRM package, and then define necessary users, roles and communication security requirements between the nodes using UMLsec profile. The relation between these elements and the source requirement element incurring such security design can be established using SysML requirements concepts.

UMLsec offers several stereotypes such as *Internet*, *wire*, *LAN*, *encrypted* for physical links between nodes. These concepts can be applied on communication links in telecommunication systems. Each link type in UMLsec is defined as prone to different types of threats (*read*, *insert*, *delete*) from different attackers. For example, a link stereotyped as *LAN* or *wire*, has no threat from a default (external) attacker. However, an insider attacker can still pose *read*, *insert* and *delete* threats regarding the packets and information transferred on such a link. If it is needed to define new types of links, attackers or threats, UMLsec allows this. The new concepts can be defined in the UMLsec appropriate format in an XML file, so that the analysis tool can perform correct analysis based on these custom concepts on a model that makes use of them. In this sense, the analysis tool is flexible and extensible. *Secrecy* stereotype that is used on dependency relationships applies a secrecy/confidentiality requirement on the elements of dependency base class. This way, we specify that there is a secrecy requirement for the involved elements.

V. MODELING A SECURITY REQUIREMENT USING THE SUGGESTED PROFILE

In a typical 3G telecommunication network, different groups of Radio Base Station (RBS), Radio Network Controller (RNC) and Media Gateway (MGW) nodes are connected and communicate. When a Mobile Equipment (ME) wants to join the network, it starts communicating with RBS and authenticating itself to the system. Security operations such as key exchange take place through the communication path from the mobile equipment to the RNC. In our case study, we have two RBS 3202 nodes that communicate with an RNC. The output power requirements for RBS 3202 are as follows: **Req1:** Optimized Power 15W, Standard Power 20W, High Power 30W and Dual High Power 60W.

One of the security requirements that exist for the connection between the RBS and RNC is:

Req2: Data communication between RBS nodes and RNC should only be readable by inspection group.

The second requirement incurs that no one from outside and also inside of the network should be able to read the data traffic on the links between RNC and RBS except users in the inspection group. Thus the data should be encrypted using a specific key for this group. We try to violate this in our

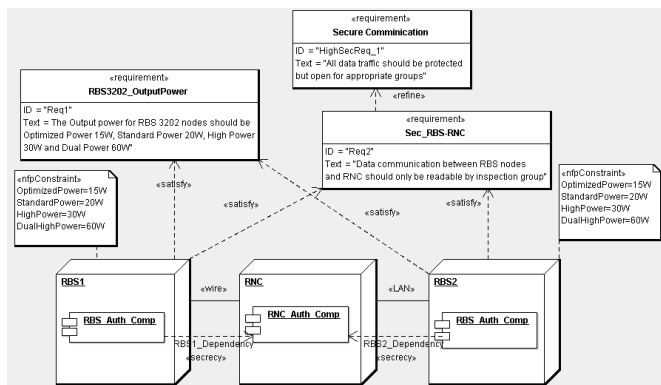


Fig. 1. Security Requirement on RBS Nodes

```

:::Against Default Attacker
====Here begins the verification
The name of the dependency is RBS2_Dependency
The stereotype of the communication link of the dependency RBS2_Dependency is LAN
The stereotype of the dependency is: secrecy
* The UML model satisfies the requirement of the stereotype secure links.
...
:::Against Insider Attacker
====Here begins the verification
The name of the dependency is RBS2_Dependency
The stereotype of the communication link of the dependency RBS2_Dependency is LAN
The stereotype of the dependency is: secrecy
* The UML model violates the requirement of the stereotype secure links, but
it has been fixed.
...
    
```

Fig. 2. Result from UMLsec Analysis Tool

example model by using unencrypted links and then perform analysis on the model.

As shown in Figure 1, the requirements and the relationships between them and design artifacts are modeled using SysML concepts. MARTE non-functional concepts (i.e., *nfp*, *nfpconstraint*, *PowerUnitKind* and *NFP_Power*) are used for modeling output power requirements of RBS nodes. Security concepts in our model are represented using UMLsec stereotypes. The link between RBS1 and RNC is marked with *wire* stereotype and the one between RBS2 and RNC is marked with *LAN* stereotype (in UMLsec *wire* and *LAN* are two different security stereotypes that can incur different security characteristics).

Doing analysis using UMLsec analysis tool on the model yields the result that is shown in Figure 2. The important part in this analysis output (marked with *) is that *LAN* and *wire* links are not readable by a default (external) attacker thus the model satisfies the secrecy requirement for this attacker type, but an insider attacker on *LAN* or *wire* can access the information and therefore the model violates the requirement. Although UMLsec has a general *encrypted* stereotype to label encrypted communications, it is also possible to define a custom stereotype for example as “*Uniquely encrypted by SIM ID*” and define different threats that different attackers can pose on these links such that only inspection group users can have access. Then we can use this stereotype on the links instead of *LAN* and *wire* that we used earlier, to create a model that satisfies the requirement and verify it with the analysis tool.

VI. DISCUSSION

As mentioned in the related work section, EAST-ADL is a modeling solution for automotive domain that is built using a similar approach to what we proposed here by adopting from several UML profiles. It is successfully accepted in the automotive domain and its usage together with AUTOSAR is gaining more momentum. In table I, a comparison of capabilities of our suggested solution using MARTE plus SysML and UMLsec against those of EAST-ADL is presented, with a focus on modeling concepts and features that are necessary for NFRs in telecommunication domain (e.g., processing capacity and memory consumption that are important for performance analysis). It summarizes the concepts we discussed and identified in previous sections. The star mark in the table is used to indicate that the feature is not enough/fully supported, such as the dependability modeling in our approach. However, it can be covered by using the DAM profile introduced earlier, which is built as an extension to MARTE. Modeling of time for schedulability analysis support in EAST-ADL needs also to be complemented (as is investigated in [24]).

TABLE I
COMPARISON OF THE SUGGESTED UML-BASED MODELING SOLUTION WITH EAST-ADL OF AUTOMOTIVE DOMAIN.

Modeling Feature	Our Approach	EAST-ADL
Generic NFRs (SysML Style)	✓	✓
Traceability of NFRs	✓	✓
Timing, Clock, Schedulability Support	✓	*
Memory consumption	✓	✗
Processing capacity	✓	✗
Power consumption	✓	✗
Virtual machines and hierarchical schedulers	✓	✗
Hardware platform	✓	✓
Multicore	✓	✗
Allocation and Deployment	✓	✓
Communication media	✓	✗
Safety	✗	✓
Security	✓	✗
Variability (product families)	✗	✓
Methodology (e.g., abstraction levels)	✗	✓
Dependability (e.g., fault, error...)	*	✓
Synchronization mechanisms	✓	✗
Arbitrary Non-Functional Properties	✓	✗
Component model	✓	(AUTOSAR)

From the table, it can be seen that by *tailoring* a UML profile for telecommunication systems based on the concepts in the three available profiles we discussed in this paper (MARTE, SysML and UMLsec), it is possible to better cover the requirements of telecommunication systems, than just re-using only EAST-ADL modeling semantics from automotive domain, which are tailored for the needs of systems in that domain. Security is one of the specific needs of telecommunication systems that is not supported by EAST-ADL and has not been in the main focus in automotive domain (so far). While on the other hand, safety requirements, which are very important for automotive systems, are explicitly supported in EAST-ADL. For differences between safety and security requirements, interested readers can refer to [27].

While in this paper, we discussed a UML-based solution by adopting and tailoring already existing profiles, other methods

of defining a specific language for modeling telecommunication systems are, of course, possible. However, although designing a domain specific language from scratch may match the needs of telecommunication systems better, it also implies the need to design dedicated modeling tools, and additional costs for training the users to learn the new language. On the other hand, some of the benefits of a solution based on UML are that many users are already familiar with UML, and thus, the learning curve is smaller. Also, there are already many tools for creating UML models which can be used 'out of the box' [7], [8]. One point to remember though is that, as mentioned before, combining different UML profiles can be problematic in some cases. For example, there is FlowPort both in SysML and MARTE. However, the semantics of FlowPort in SysML are different from those of MARTE. A systematic approach is suggested in [9] to ensure consistency in merging UML profiles.

Regarding the management of models, based on the features of the modeling tool, there can be several scenarios. For example, different models can be created for different aspects of the system. This can also help with the analysis, as one model for each type of analysis can be created. However, maintaining consistency between different models of the system and redundant information modeling are some of the challenges of this approach. Another scenario could be to have one single model for the system, and then have the modeling tool provide different views of the core model. This way, a user can just focus on the aspects of his/her interest in each view, while modifications are persisted into one single model representing the system. This method is under development in CHES project [11].

As for the analysis of the models, although this topic is not the main focus of this paper, but we provide some hints here. Basically, the process of analysis can be different for various analysis tools, and depending on which types of analysis are of interest for different end-users. In case of having just one single model of the system, if an analysis tool can ignore non-relevant model elements and perform analysis on the relevant parts, the model can be fed as input to the analysis tool directly. However, if non-relevant model elements may cause problems for the analysis, then it is possible to use model transformation to extract only the relevant ones into a new model appropriate as input for the analysis tool. Also, if the input model of any analysis tool has its own specific meta-model, then model transformation techniques can again be used to transform the original model into a new model conforming to the meta-model of the analysis tool.

VII. CONCLUSION AND FUTURE WORK

In this paper, we discussed several challenges in modeling non-functional requirements in telecommunication domain. We also suggested a modeling approach for representation of non-functional requirements and their properties in this domain. Our approach was to consider telecommunication systems as a subdomain of RTEs and therefore adopt from available modeling solutions for non-functional requirements

and their analysis that already exist in RTEs domain. Some concepts of MARTE that can cover the requirements of telecommunication systems were highlighted. For traceability aspects, SysML and the features it provides in establishing traceability in modeling of non-functional requirements were introduced. Finally, as a specific and intrinsic requirement in telecommunication domain, it was shown how it is possible to model and analyze security that is addressed in our suggested approach by adopting UMLsec. This way, we showed not only how it is possible to model different types of non-functional requirements, but also how model-based analysis can help with the need to perform analysis of non-functional requirements at earlier phases of development and therefore reduce time and cost. In CHES European project [11], we are developing a similar solution by using subsets from MARTE, SysML and DAM profile (without security considerations yet) to generate code for telecommunication systems (in this case, Ericsson platforms) considering and preserving non-functional requirements modeled using the mentioned subsets.

As further studies, it is necessary to augment the suggested approach in this paper, such as introducing it as part of a well-structured methodology similar to the methodology suggested in [28]. This methodology is more suited for automotive domain as it makes use of EAST-ADL and its abstraction levels. Applicability of the same concepts to telecommunication domain could be an interesting topic to investigate. Especially that EAST-ADL offers concepts for modeling *variability* requirements, which can be very useful in telecommunication domain for modeling product families, targeting cost-sensitivity non-functional requirements and performing cost analysis.

Also other challenges that exist regarding non-functional requirements in a model-based development approach can be guaranteeing and preservation of these requirements on the target platform, introducing runtime adaptability and re-configuration based on the requirements and handling their violations.

As a last note, in this paper we set the basis for a UML-based solution for telecommunication systems similar to EAST-ADL in automotive domain. While it was demonstrated how we can relate high-level and abstract representation of an NFR such as security with its lower level realizations and perform security analysis on it, a full scale solution needs contributions from different industrial partners active in the domain as has been done in the process of defining EAST-ADL and AUTOSAR.

REFERENCES

- [1] T. Henzinger and J. Sifakis, "The embedded systems design challenge," in *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*, August 2006.
- [2] A. Borg, A. Yong, P. Carlshamre, and K. Sandahl, "The bad conscience of requirements engineering : An investigation in real-world treatment of non-functional requirements," in *Third Conference on Software Engineering Research and Practice in Sweden (SERPS'03), Lund* :, 2003.
- [3] MARTE specification version 1.0 (formal/2009-11-02), <http://www.omgmarTE.org>, Accessed: August 2011.
- [4] EAST-ADL Specification V2.1, <http://www.atesst.org>, Accessed: August 2011.

- [5] OMG SysML Specification V1.2, <http://www.sysml.org/specs.htm>, Accessed: August 2011.
- [6] J. Jürjens, *Secure Systems Development with UML*, ISBN: 978-3-540-00701-2. Springer, 2005.
- [7] I. Weisemöller and A. Schürr, "A comparison of standard compliant ways to define domain specific languages." Berlin, Heidelberg: Springer-Verlag, 2008, pp. 47–58.
- [8] B. Selic, "A systematic approach to domain-specific language design using uml," in *Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, 2007.
- [9] F. Noyrit, S. Gérard, F. Terrier, and B. Selic, "Consistent modeling using multiple uml profiles," in *Model Driven Engineering Languages and Systems*, 2010.
- [10] A. Borg, M. Patel, and K. Sandahl, "Good practice and improvement model of handling capacity requirements of large telecommunication systems," in *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference*, Washington, DC, USA, 2006.
- [11] CHESSE Project: Composition with Guarantees for High-integrity Embedded Software Components Assembly, <http://chess-project.ning.com/>, Accessed: August 2011.
- [12] Telecommunication Standardization Sector (ITU-T), <http://www.itu.int/en/pages/default.aspx>, Accessed: June 2011.
- [13] M. R. Abid, D. Amyot, S. S. Somé, and G. Mussbacher, "A uml profile for goal-oriented modeling," in *Procs. of SDL'09*, 2009.
- [14] T. Lodderstedt, D. A. Basin, and J. Doser, "Secureuml: A uml-based modeling language for model-driven security," in *Proceedings of the 5th International Conference on The Unified Modeling Language*, ser. UML '02, 2002.
- [15] K. Alghathbar and D. Wijesekera, "authuml: a three-phased framework to analyze access control specifications in use cases," in *FMSE '03: Proceedings of the 2003 ACM workshop on Formal methods in security engineering*. New York, NY, USA: ACM, 2003, pp. 77–86.
- [16] A. Cenys, A. Normantas, and L. Radvilavicius, "Designing role-based access control policies with uml," in *Journal of Engineering Science and Technology Review*, vol. 2, no. 1, 2009, pp. 48–50.
- [17] K. P. Peralta, A. M. Orozco, A. F. Zorzo, and F. M. Oliveira, "Specifying security aspects in uml models," Toulouse, France, September 2008.
- [18] S. Demurjian, J. Pavlich-Mariscal, and L. Michel, "Enhancing uml to model custom security aspects," in *Proceedings of the 11th International Workshop. on Aspect-Oriented Modeling*, October 2007.
- [19] T. Doan and S. Demurjian, "A.: Mac and uml for secure software design," in *In: Proc. of 2nd ACM Wksp. on Formal Methods in Security Engineering*. ACM Press, 2004, pp. 75–85.
- [20] S. H. Houmb and K. K. Hansen, "Towards a uml profile for security assessment," in *Workshop on Critical Systems Development with UML*, 2003.
- [21] AADL, "The Architecture Analysis & Design Language: An Introduction ," <http://www.aadl.info/aadl/currentsite/aadlst.html>, Accessed: August 2011.
- [22] AUTOSAR Home Page, <http://www.autosar.org/>, Accessed: August 2011.
- [23] S. Ansi, A. Albinet, S. Tucci-Pergiovanni, C. Mraidha, S. Gérard, and F. Terrier, "Completing east-adl2 with marte for enabling scheduling analysis for automotive applications," in *Embedded Real Time Software and Systems Conference (ERTS)*, Toulouse, France, 2010.
- [24] TIMMO Project, <http://www.timmo.org/>, Accessed: August 2011.
- [25] M. dos Santos Soares and J. L. M. Vrancken, "Model-driven user requirements specification using sysml," *Journal of Software*, vol. 3, pp. 57–68, 2008.
- [26] S. Bernardi, J. Merseguer, and D. Petriu, "A dependability profile within marte," *Journal of Software and Systems Modeling*, 2009.
- [27] E. Albrechtsen, "Security vs safety," NTNU - Norwegian University of Science and Technology <http://www.iot.ntnu.no/users/albrecht/>, Accessed: May 2011.
- [28] A. Albinet, J.-L. Boulanger, H. Dubois, M.-A. Peraldi-Frati, Y. Sorel, and Q.-D. Van, "Model-based methodology for requirements traceability in embedded systems," in *Procs. of ECMDA'07*, Haifa, Israel, Jun. 2007.

A Maintenance Approach of a BJI Index Configuration

Saïd Taktak

University of Sfax, FSEGS Faculty, P.O.Box 1088
Miracle Laboratory
Sfax, Tunisia
Said.taktak@gmail.com

Jamel Feki

University of Sfax, FSEGS Faculty, P.O.Box 1088
Miracle Laboratory
Sfax, Tunisia
Jamel.Feki@fsegs.rnu.tn

Abstract— In data warehousing domain, OLAP (On Line Analytical Processing) queries are complex since they use several tables with huge cardinalities. Several optimization techniques have been studied in the literature as materialized views and bitmap join indexes (BJI). BJI indexes are useful to pre-calculate star joins in order to reduce the execution cost. Current approaches for the selection of BJI define a configuration that optimizes a beforehand definite workload of queries. However, this workload can evolve in time and is likely to make obsolete the configuration of index created. In order to take into account the evolution of a workload of queries, we propose, in this article, a maintenance approach for the recommendation of a new configuration of indexes. Our approach starts with an evaluation of the current configuration of indexes and then adapts it to the new workload of queries with an aim of guaranteeing the stability of performances. Queries of the new workload are directly extracted from log files. Furthermore, to validate our approach, we carried out a series of experimentations on a data warehouse created with the DWEB benchmark.

Keywords-data warehouse; bitmap join indexes; tuning.

I. INTRODUCTION

Due to the exponential increase in the volume of data, enterprises focus on the decisional information and, therefore move from a simple processing of data to a logical analysis of data. The data warehouse formalized in the early 90s by *Inmon* is the appropriate solution [1]. In fact, it is particularly designed to respond to complex decisional queries.

At the conceptual level, the data warehouse is usually modeled by a *star schema* that highlights the topic analyzed as a central *fact* (i.e., the fact table) which is composed of numerical attributes and connected to *dimensions* (i.e., dimension tables) representing the axes of analyses. The large volume of data manipulated by analytical queries (i.e., decisional) and the high number of tables to be joined raise the problem of performance [2]. In order to optimize these queries, taking intensive execution time, the data warehouse administrator (or designer) has to successfully achieve the physical design step [3]. This is why, he or she should select a set of optimization techniques that they consider pertinent to respond to the decision makers needs (i.e., expressed as a workload of queries defined beforehand). Several optimization techniques have been studied for relational data warehouses, some of which are inherited from traditional

databases. A Database management system (DBMS) offers techniques such as:

- Materialized views [4] improve the execution time of queries by pre-computing the most expensive operations such as joins and aggregations. Consequently, the execution of some queries OLAP requires only the access to one materialized view instead of its original data tables.
- Fragmentation [5] allows dividing the data of a DW into multiple partitions that can be accessed separately. It can be either vertical or horizontal (by projection or selection algebraic operators).
- Parallel query processing [6]. The query is divided into components that can be treated simultaneously. The results are combined and delivered to the customer as a single component.
- Advanced indexes [7], etc.

These optimization techniques can be used in an isolated manner (i.e., selected independently) or in a combined way (by exploiting the dependencies between them). The second way provides good results, because each technique can compensate the shortcomings of others.

In the data warehouse context, the indexing technique is an important issue due to the large volume of manipulated data and to the complexity of processed queries. The bitmap join indexes (BJI) pre-compute the joins between the fact and its dimension tables. A BJI index is defined on the fact table by using the values of one or several attributes belonging to dimension tables. This increases the number of possible indexes [8]. Several research studies have proposed optimal solutions to help the data warehouse administrator to select a BJI configuration that minimizes the execution cost of a given workload of queries.

However, the task of the administrator is more complicated than that. In fact it is not sufficient to establish the appropriate optimization technique, but it is more important to adjust the use of this technique in response to the occurring evolution on the data warehouse to avoid the performance degradation. This evolution may affect three things: (1) the schema and the content of the data warehouse tables, (2) the size of the memory space allocated to the optimization techniques selected, and (3) the workload of queries on which the selection of optimization technique has been established. In this paper, we propose an approach for the maintenance of a current BJI configuration (e.g., in use) in order to face the evolution of a workload of queries. This

evolution may concern many aspects as the frequency of access of the queries, the addition of new queries or even the deletion of existing ones, etc.

This article is organized into six sections: Section 2 recalls the definition of the BJI and illustrates it through an example. Section 3 presents a formalization of the selection problem of BJI and explores the existing approaches of indexing and their lacks. Section 4 is devoted to the presentation of our maintenance approach. Section 5 describes a set of experiments we have done. Section 6 concludes this article and enumerates some perspectives.

II. BINARY JOIN INDEX

A binary join index BJI ("bitmap join index") allows pre-joining the fact table with its dimension tables in a data warehouse. A BJI has the same number of tuples as its fact table and as many columns as the number of distinct values of the dimension attribute on which the BJI index is built [9]. The bit at row i and column j of the BJI index is set to 1 if the i^{th} tuple of the fact table can be joined with the tuple of the dimension table that has the value of the indexed attribute (i.e., in column j). Otherwise, this bit is set to zero.

Fig. 1 represents the *Product_Type* BJI built on the fact table *Sales* using the *Type* attribute of the dimension table *Product*.

Dimension Table: Product				Fact Table: Sales				Binary Join Index			
RID	PID	Label	Type	RID	PID	...	Amount	RID	T1	T2	T3
1	124	L1	T1	1	124		1213	1	1	0	0
2	137	L2	T1	2	137		23232	2	1	0	0
3	154	L3	T2	3	166		23244	3	0	0	1
4	166	L4	T3	4	154		4544	4	0	1	0
5	181	L5	T2	5	154		4544	5	0	1	0
6	209	L6	T1	6	166		444534	6	0	0	1
				7	166		33550	7	0	0	1
				8	181		6777	8	0	0	1
				9	209		6555	9	1	0	0
				10	209		4544	10	1	0	0

Figure 1. Bitmap join index (BJI).

In these tables, each tuple is identified with a unique identifier denoted RID (Row Identifier) generated by the DBMS. The index of Fig. 1 can be constructed by the following SQL statement:

```
CREATE BITMAP INDEX Product_Type
ON Sales (Product.Type)
FROM Sales S, Product P
WHERE S.PID = P.PID
```

The first tuple of the *Sales* table is joined with a tuple of the *Product* table corresponding to a *Type* product *T1*. Therefore, the bitmap corresponding to type *T1* of the first row of this index is set to 1 and then the remaining bits are set to zero.

Note that a BJI is particularly useful for star joins; like conventional binary indexes, it is very beneficial for *Count(*)* queries where the response to these requests requires only access to the binary index. No access to data

tables is necessary; we just need to count the number of 1 in the bitmap array that results of the requested operations. For instance, to determine the number of sales for products of *Type T2*, we count the number of 1 in column T2 of the BJI. For more complex queries (i.e., using several indexed attributes, the logical operators (e.g., AND, OR) are useful.

III. INDEX SELECTION PROBLEM

Index selection is a crucial step in the physical design of the data warehouse. It consists in building an index configuration to optimize the execution cost of a workload (of queries). This optimization can be realized respecting certain constraints, such as the storage space allocated to the index configuration or the cost of maintenance.

Generally, the algorithms proposed for index selection include three steps:

- (1) Identification of candidate attributes for indexing.
- (2) Pruning.
- (3) Construction of an index configuration.

During the first step, a set of candidate attributes can be built manually by the administrator according to his expertise, or automatically by using a queries parser. The step of pruning is necessary to reduce the number of candidate attributes for indexing; it is done by referring to certain criteria, for example, by eliminating high cardinality attributes or those belonging to small tables [10].

The third step builds progressively a final configuration of BJI respecting the constraints of execution and the storage cost. It is often done by selection algorithms (Glouton algorithms) or directed by data mining techniques [11] [12], or even genetic algorithms [13]. The quality of the generated index configuration is measured by its cost which is -calculated using the optimizer of the DBMS or by a mathematical cost model. All these approaches of BJI selection are applied in a static context (i.e., a workload defined beforehand); however, this workload may evolve in time and might yield obsolete its associated index configuration. To the best of our knowledge, the work in [14] is the only attempt that has tackled the problem of BJI dynamic selection after a workload evolution. It represents an extension to the approach of selecting BJI [15] which is based on data mining technique for the detection of frequent itemset. In fact, the proposed approach is to increment the frequent itemset by referring to a knowledge base that stores information (frequent itemset) of anterior executions. New frequent itemsets are analyzed to generate new candidate indexes; declined (now infrequent) itemsets correspond to indexes to be dropped.

The limited number of algorithms interested in the problem of dynamic index selection led us to propose a new approach. Our approach helps the administrator to maintain the existing index configuration during the evolution of queries executed on the data warehouse.

IV. MAINTENANCE APPROACH BY RECONFIGURING BJI INDEXES

In this section, we present our approach of BJI maintenance which reduces the execution's cost of a new

workload. Our approach is placed in a general framework, which allows illustrating the interest of BJI maintenance. Since the BJI's selection is based on a fixed set of queries, any change in this set may affect the existing BJI's configuration. Fig. 2 outlines the general architecture of our approach. Unlike the proposed approaches of BJI selection, it is not only based on the optimization process on a workload, but it also takes into consideration the existing BJI.

The objective of our approach is to assist the administrator to maintain an initial configuration of BJI during the evolution of the initial workload Q to a new one Q' , by proposing a new index configuration which ensures the reduction of Q' cost. The problem can be formalized as follows:

- A data warehouse composed of d dimension tables $D = \{D_1, D_2, \dots, D_d\}$ and a fact table F .
- A set of n BJI $I_{current} = \{BJI_1, BJI_2, \dots, BJI_n\}$ created referring to the initial workload Q .
- f_{ui} represents the frequency use of BJI_i for a given period P sufficiently significant, so that it covers a maximum of treatments.
- Q is a workload of m queries $Q = \{q_1, q_2, \dots, q_m\}$, extracted from the *DBMS* log file.
- f_{qi} frequency execution of query q_i .

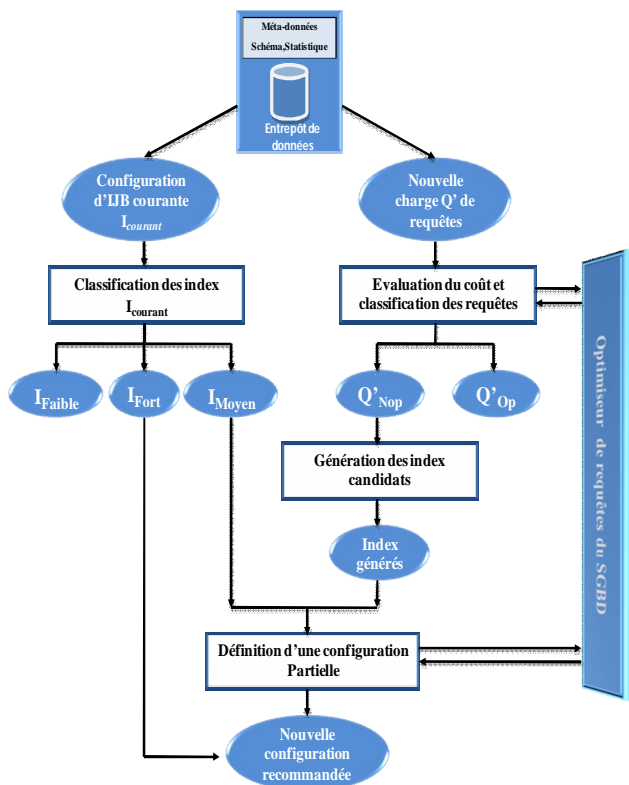


Figure 2. Maintenance approach architecture.

We detail in the following sections, the four steps illustrated in Fig. 2.

A. Classification of existing indexes

The current configuration of BJI was created specifically to optimize an initial workload Q of queries. The evolution of the workload Q to Q' can degrade the global performance of Q' . In fact, the new queries are not necessarily optimized compared to the current BJI configuration. In addition, some indexes may become unexploited if they are built to optimize queries actually infrequently executed.

To optimize the new workload Q' , we proceed to analyze the utility of the current BJI indexes ($I_{current}$) based on the actual frequency of use (fu) of each index during a period of time and, secondly, on the average frequency fm of all $I_{current}$ indexes used during the same period. This average frequency is calculated by formula (1):

$$fm = \frac{\sum_{i=1}^n fu_i}{n} \quad (1)$$

This frequency will allow us to classify the $I_{current}$ set in three subsets:

- I_{high} : represents the subset of indexes heavily used, that means, those whose frequency of use is higher than $3*fm/2$. The BJI belonging to this group are considered important and will be retained to be present in the new configuration BJI optimizing Q' .
- I_{medium} : represents the subset of indexes used moderately, those whose frequency of use is between $fm/2$ and $3*fm/2$. The relevance of these indexes will be reviewed during the index's selection step to decide whether to keep or reject.
- I_{low} : represents the subset of indexes slightly used, i.e., those whose frequency of use fu is less than $fm/2$. These indexes are rarely or completely unused when running queries from the initial workload. In addition, they occupy a memory space and require a maintenance cost without a justified usefulness. So it is better to remove these indexes and then recover their storage space.

Table I shows an $I_{current}$ set of 8 indexes to optimize a given initial workload Q . The total usage of all indexes is 80 and the average of their frequency of use fm is 10 ($= 80/8$).

TABLE I. EXAMPLE OF A SET OF BJI WITH THEIR FREQUENCY OF USE

Indexes	frequency of use
BJI_1	20
BJI_2	8
BJI_3	1
BJI_4	6
BJI_5	9
BJI_6	27
BJI_7	2
BJI_8	7
Total	80

According to our proposed classification, this set of indexes will be split into three subsets:

- $I_{high} = \{BJI_1, BJI_6\}$
- $I_{medium} = \{BJI_2, BJI_4, BJI_5, BJI_8\}$
- $I_{low} = \{BJI_3, BJI_7\}$

We find out that both indexes BJI_3 and BJI_7 are too little used by the initial workload and, therefore, can be dropped. The drop of indexes with a low-frequency usage optimizes the storage constraint, but it does not improve the global execution's performance.

In parallel, we study the new workload to examine whether it is optimized in respect to the current configuration otherwise we must change this configuration by adding new indexes. This requires the evaluation of the new workload (Q') execution costs and the identification of its non optimized queries in order to consider them when defining the new configuration. The next section details how we examine this workload.

B. Costs evaluation and queries classification

First, this step consists in evaluating the m queries of the workload Q' . In order to get closer to reality, the study of the evolution of this workload considers the queries actually executed on the data warehouse; we extract these queries from the log file. We note $Cost(q_i)$ the execution cost of query q_i , and I_{qi} all old indexes used by the query optimizer for q_i . Based on these two factors of classification, we subdivide Q' into two subsets of queries:

- Q'_{unop} : represents non optimized queries from Q' ; i.e., those who do not use any index during their execution or those who have a cost of performance higher than the average execution cost $CostAvg(Q')$ of the new charge Q' with:

$$CostAvg(Q') = \frac{\sum_{i=1}^m fq_i \times Cost(q_i)}{\sum_{i=1}^m fq_i} \quad (2)$$

$Q'_{unop} = \{q_i' \in Q' / Cost(q_i') > CostAvg(Q') \text{ or } I_{qi}' = \emptyset\}$. Queries of this set will be analyzed in the step called *candidate index generation* in order to define a new index configuration.

- Optimized queries (Q'_{op}) is the set of remaining queries ($Q'_{op} = Q' - Q'_{unop}$). They don't interfere in the definition of a new index configuration.

C. Generating candidate indexes

It is to generate indexes that improve execution performance of the new workload Q' , by focusing on all the non-optimized queries (Q'_{unop}). We realize this task throughout the following three steps:

- (1) Identification of indexable attributes for Q'_{unop} .
- (2) Construction of a BJI configuration per query.
- (3) Pruning of indexable attributes.

At the end of these three steps, which we detail below, we obtain a set of candidate indexes that will be evaluated (cf. Section D).

1) Identification of indexable attributes

Non optimized queries are handled by a syntactic analyzer to extract all attributes that may be carried for indexes. These attributes are those present in the WHERE,

GROUP BY and ORDER BY clauses of queries. For example, the indexable attributes issued from the following query are: City, Month and Type.

```
SELECT  AVG(amount)
FROM    Sales S, Customer C, Product P, Time T
WHERE   S.CID = C.CID AND S.PID = P.PID
AND     S.TID = T.TID AND T.Month = 'MARCH'
AND     C.City IN ('SFAX', 'SOUSSE')
AND     P.TYPE = 'TOY';
```

2) Construction of a BJI configuration per query

Referring to the attributes extracted in the previous step, we construct a matrix query-attribute where its lines represent queries of Q' and columns represent candidate attributes for indexing. The existence of an indexed attribute in a query is represented by the integer 1 and its absence by zero.

Fig. 3 is an example of matrix built on a workload of six queries and six indexable attributes noted A, B, C, D, E, F.

	A	B	C	D	E	F
Q1	0	1	0	1	0	0
Q2	0	1	1	0	0	0
Q3	1	0	1	0	1	0
Q4	0	0	1	1	0	0
Q5	0	0	1	0	1	1
Q6	0	0	1	0	0	0

Figure 3. Matrix Query-Attribute.

3) Pruning

The input of this step is the configuration of candidate indexes from the previous step. This index configuration is beneficial for the whole set of queries because each index has been defined to optimize queries separately. However, this configuration can be very large: it may include some attributes which are not suitable for indexing (high cardinality or belonging to small tables) [16], therefore their removal is recommended to reduce the cost of storage space for indexes. This removal operation is the *Pruning process*; it is based on a *Fitness* parameter introduced by [17] in the context of frequent patterns. We were inspired by this work to define a simplified formula applicable to each individual attribute. This *Fitness* parameter takes into consideration both the frequency of occurrence of attributes in queries and the table size.

$$fitness = sup_i \times \alpha_i \quad (3)$$

where sup_i represents the frequency of occurrence of attribute i in all queries and $\alpha_i = |Di|/|F|$.

Attributes having a Fitness parameter less than the threshold $minsup$ are not indexed; $minsup$ in $[0, 1]$ is defined as a parameter by the administrator.

The generated number of indexes is less than or equal to the number of the workload queries, firstly because some queries share the same indexes, secondly, it is unnecessary to generate indexes that already exist (belonging to an entire index of the current configuration $I_{current}$). This step generates

a set of candidate indexes created referring to Q'_{unop} . This set will serve together with the moderated used indexes (I_{medium}) to recommend a fairly new configuration.

D. Definition of a configuration to recommend

The union of the candidate indexes generated from the previous step with those belonging to the set of indexes moderately used (I_{medium}) forms the set of all indexes to be selected for the final configuration (Fig. 2). During this stage, the indexes that do not ensure an important gain of cost are considered to be useless and therefore eliminated. We use a Glouton algorithm as in [17] to select the best indexes among the n candidate indexes. The selection of an index configuration is an iterative approach by selecting, at each iteration, the index that most reduces the cost of queries execution as the storage space constraint is respected. Finally, the union of the set of selected candidate indexes with the heavily used ones forms the final index configuration proposed to the data warehouse administrator (Fig. 2).

V. EXPERIMENTAL EVALUATION

In order to test our approach, we made an experimental study on a data warehouse, we have generated under Oracle 11g, with the benchmark DWEB1 (Data Warehouse Engineering Benchmark) [18]. This data warehouse is composed of a fact table with 2.043.271 rows, and four dimension tables with 4000, 100, 10000 and 1000 rows. We used DWEB1 to generate a workload of 50 star join queries. Several types of query were considered: *Count* queries, queries using aggregate functions (*Sum*, *Avg*), queries with dimensional attributes in the SELECT clause, etc. We modified the cardinality of the attributes of the dimensions and certain clauses of query's restriction (WHERE clause attributes) to adapt them for indexing. Our approach of maintenance of BJI configuration is implemented in Visual Studio 2005 and a Core 2 Duo machine with 3 GB of RAM. We conducted a series of experiments that take place in three phases:

- Construction of an index configuration that optimizes our initial workload of 50 queries.
- Modification of this workload by following an increasing trend rate and computing the new cost before maintenance.
- Application of our approach of BJI configuration maintenance and calculation of the execution cost of the workload, after maintenance.
- Application of a second approach of maintenance (naïve) on the same workloads and comparison with the results of our proposed approach.

The execution of the initial 50 queries gave a total cost of input-output (I/O) equal to 173 918 without any index. After building an index configuration, the cost has been reduced to 138 231 thanks to the generation of 10 indexes (4 mono-attributes and 8 multi-attributes).

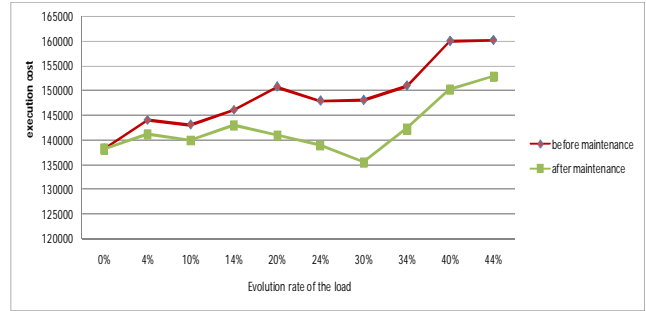


Figure 4. Evolution of the cost according to workload.

The next step is to study the effect of the initial query load evolution on overall performances. For this, we modified the workload several times, according to a variable rate of the charge evolution and keeping constant the number of queries to 50. This modification is to delete some queries and replace them with others of the same type (i.e., same tables and aggregate functions with different attributes of selection) to maintain constant the number of queries. The choice of query to remove is done randomly. The results (Fig. 4) show important performance degradation while the rate of the charge evolution increases. This degradation is due to the fact that newly added queries exploit, little or never, the initial index configuration (i.e., $I_{current}$). So, it's necessary to maintain this configuration. The curve of the execution cost before maintenance according to the evolution of the workload has generally an ascendant trend but we can see some times, zones of decline (Fig. 4) like for the evolution rate 24% and 30%. This can be due to the fact that some existing indexes (i.e., present in the initial configuration) are used by the newly inserted queries. These used indexes are generally mono-attribute. The application of our maintenance approach, based on the re-calculation of the initial BJI configuration, allows the reduction of the execution cost of the new workload compared to its execution cost before maintenance. According to Fig. 4, the gap between costs before (upper curve) and after (lower curve) the maintenance of indexes becomes more and more important when the rate of the workload evolution increases, which is very interesting in practice because the evolution increases as time goes on.

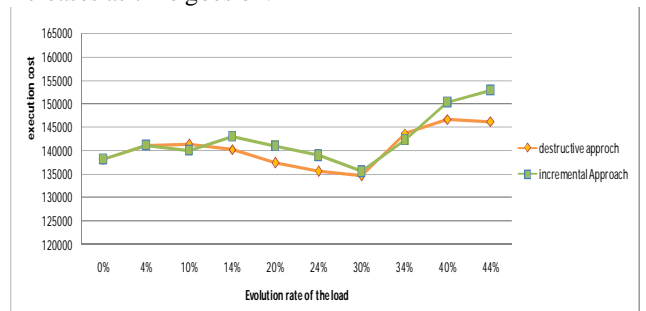


Figure 5. Incremental Approach Vs Destructive.

The experiments done until now approved the interest of applying our maintenance approach (incremental), by index reconfiguration, in order to reduce the execution cost of an

evolving query workload. We evaluate in this section another approach of maintenance (naive) whose principle is to remove all existing indexes and to create a new index configuration for each new workload. Certainly, this destructive approach may give good results in terms of cost of performance because it processes each workload independently from the existing one. Fig. 5 shows that the incremental maintenance approach gives results which are close to those obtained with the destructive approach, even equal in some cases, with the additional advantage of preserving the useful indexes and creating a very limited number of indexes; that constitutes a gain of time (creation of new indexes) and avoids fragmentation of disk space allocated to indexes.

In conclusion, our approach of incremental reconfiguration provides some stability in overall performance during the evolution of an initial workload.

VI. CONCLUSION

We proposed in this paper an approach to assist the administrator of the data warehouse to reconfigure BJI indexes, initially constructed on a workload of analytical queries, following its evolution. Our approach is incremental. It is characterized by the evaluation of the new workload compared according to the existing configuration of BJI in order to decide whether to recalculate a new configuration or to keep the old one. In practice, it proceeds to a classification of existing indexes in three categories: *low*, *medium* or *highly* used. Moreover, determining a new configuration eliminates the slightly used BJI and takes into consideration the old indexes (moderately and highly used) and the new workload to optimize. For the new query workload, we determine a set of candidate BJI indexes according to the conventional principle (extraction of indexable attributes, pruning, and construction of a BJI configuration per query). In order to test our approach, we have developed an iterative algorithm. It determines the cost of a query from its execution plan developed by the Oracle DBMS optimizer; it is to evaluate candidate indexes. The union of selected candidate indexes set and the highly used ones from the configuration to maintain forms the new index configuration proposed to the data warehouse administrator. We tested our approach on a data warehouse built with the benchmark DWEB by varying queries of the initial workload. The preliminary results are so encouraging. However, other experiments will be necessary for large scaling. Also, it would be interesting to study performance thresholds that trigger the recommendation process of reconfiguration.

REFERENCES

- [1] W. H. Inmon and R. D. Hackathorn, "Using the Data Warehouse", - USA : Wiley-QED Publishing, 1994.
- [2] T. Stöhr, H. Märtens and E. Rahm, "Multidimensional database allocation for parallel", Proceedings of the International Conference on Very Large Databases, 2000, pp. 273–284.
- [3] S. Chaudhuri and V. Narasayya, "Self-tuning database systems : A decade of progress", Proceedings of the 33rd International Conference on Very Large Databases, 2007, pp. 3–14.
- [4] M. Hung, M. Huang, D. Yang, and N. Hsueh, "Efficient approaches for materialized views selection in a data warehouse", Inf. Sci., vol. 177, pp. 1333–1348, March 2007.
- [5] S. Agrawal, V. Narasayya and B. Yang, "Integrating vertical and horizontal partitioning into automated physical database design", Proceedings of the ACM SIGMOD International Conference on Management of Data, 2004, pp. 359–370.
- [6] T. Stöhr, H. Märtens and E. Rahm, "Multidimensional database allocation for parallel", Proceedings of the International Conference on Very Large Databases, 2000, pp. 273–284.
- [7] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index Selection for OLAP", In Proceedings of the Thirteenth International Conference on Data Engineering (ICDE '97). IEEE Computer Society, Washington, DC, USA, 1997, pp. 208–219.
- [8] P. O'Neil and G. Graefe, "Multi-table joins through bitmapped join indices", SIGMOD Rec., vol. 24, pp. 8–11, September 1995.
- [9] P. O'Neil and D. Quass, "Improved Query Performance with Variant Indexes", in ACM SIGMOD International Conference on Management of Data (SIGMOD 1997). - Tucson, USA : 1997. - pp. 38–49.
- [10] K. Boukhalfa, L. Bellatreche and Z. Benameur, "Index de jointure binaires : Stratégies de sélection et étude de performances", 6^{ème} Journées Francophones sur les Entrepôts de Données et Analyse en Ligne (EDA10). - Jerba-Tunisie : 2010. - pp. 175–190.
- [11] K. Aouiche and J. Darmont, "Data mining-based materialized view and index selection in data warehouses", J. Intell. Inf. Syst., vol. 33, pp. 65–93, August 2009.
- [12] L. Bellatreche, R. Missaoui, H. Necir, and H. Drias, "A data mining approach for selecting bitmap join indices", Journal of Computing Science and Engineering 2, n° 1 (2008): 206–223.
- [13] J. Kratica, I. Ljubic et D. Tosic, "A Genetic Algorithm for the Index Selection Problem", (EvoWorkshops'03) The 2003 International Conference on Applications of Evolutionary Computing, 2003, pp. 281–291.
- [14] S. Azefack, K. Aouiche and J. Darmont, "Dynamic index selection in data warehouses", 4th International Conference on Innovations in Information Technology (Innovations 07), 2007.
- [15] K. Aouiche, J. Darmont, O. Boussaid, and F. Bentayeb, "Automatic Selection of Bitmap Join Indexes in Data Warehouses", 7th International Conference on Data Warehousing and Knowledge Discovery (DAWAK 05), 2005.
- [16] W. Dylan Bitmap Index – when to use it? [Online]// <http://dylanwan.wordpress.com/2008/02/01/bitmap-index-when-to-use-it/>. - 2008.
- [17] A. H. Necir, L. Bellatreche and R. Missaoui, "DynaClose : Une approche de fouille de données pour la sélection des index de jointure binaires dans les entrepôts de données", 3^{ème} Journées Francophones sur les Entrepôts de Données et Analyse en Ligne (EDA'07), 2007.
- [18] J. Darmont, F. Bentayeb and O. Boussaid, "Benchmarking data warehouses", International Journal of Business Intelligence and Data Mining, Vol. 2, 2007.

Software Cache Eviction Policy based on Stochastic Approach

Stoyan Garbatov and João Cachopo

Software Engineering Group

Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento, INESC-id

Lisbon, Portugal

stoyangarbatov@gmail.com and joao.cachopo@ist.utl.pt

Abstract — This work develops an innovative approach for guiding high-level software caches' eviction policy. The decision on which data to keep in the cache is made according to a stochastic analysis over the application data access behaviour. This approach shows it is possible to achieve high cache hit ratios with a reduced cache size. The effectiveness of the policy is tested and validated through the execution of two distinct benchmarks – the TPC-W and the oo7 benchmarks. The newly developed approach is flexible enough to be applied to any high-level software cache in an object-oriented system.

Keywords-software cache; stochastic approach; performance; data access.

I. INTRODUCTION

A cache is a small, high-performance memory-buffer abstraction used to store temporarily data that is deemed to be important for whatever operations may be taking place currently or in the near future. Most of the time, the data held by the cache originates from a (much) larger and (several orders of magnitude) slower medium, which is either the source or provides storage for the whole range of existing data. As a result, caches provide increased system performance by offering shorter access times to data, keeping the available processing units busy with work. The most common restriction of a cache, however, is that it cannot hold all the existing data. This may happen for several reasons – the cache may be physically unable to provide enough storage space for all the available information, or, even if there is enough space, it may be better to keep the size of the cache to a minimum because a bigger volume of data (being held in cache) usually leads to slower execution times of the lookup operations.

The success of caching mechanisms results from the “principle of locality”, which was first introduced by Denning [1]. The principle of locality, also known as locality of reference, has two basic variants, temporal and spatial. Over short periods of time, a program distributes its memory references non-uniformly over its address space, but the portions of the address space that are favoured remain largely the same for long periods of time. Temporal locality implies that the information that will be in use in the near future is likely to be already in use. Spatial locality states that the portions of the address space that are in use consist of a small number of individually contiguous segments of that address space. As a consequence, locality of space denotes

that the referenced locations of the program in the near future are likely to be near the currently referenced locations.

Optimizing the design of a cache revolves around four aspects: maximizing the probability of finding a piece of data in the cache (the hit ratio), minimizing the access time to information already in the cache (access time), minimizing the delay due to a cache miss, and minimizing the overheads of cache management, such as propagating modifications to the means that backs the cache, or dealing with consistency protocols (cache coherence).

The principles upon which the concept of caching is based are present in many contexts and situations. This makes it possible to employ them in a variety of different contexts to improve the system performance. Caching mechanisms can be divided into two main categories, namely hardware caching and software caching. Given that the main purpose of this work is to improve the hit ratios of a high-level software cache, hardware caches are not considered here.

Significant research has been carried out in software caching. As has been pointed out, the four major characteristics upon which a cache can be improved are its hit ratio, access times, speed at which update propagations are performed, and coherence. If we group existing research according to affinity with these four aspects, a trend becomes apparent – namely, most cache-related work concentrates on coherence, as can be seen in [2], [3], and [4]. At the same time, the hit ratio is an important property, especially for software caches. It has been systematically identified as being the main reason leading to poorer performance of software cache approaches, in comparison with their hardware counterparts, as has been reported in [4] and [5].

Bennet et al. [6] identified and classified several classes of shared data accesses, in the context of distributed shared memory systems. They proposed a number of memory coherence approaches tailored for these access categories and demonstrated that specialized approaches can significantly outperform general ones, whenever the expected type of access behaviour manifests itself in a consistent fashion.

Dash and Demsky [7] presented an innovative distributed transactional memory system that mitigates the effects of network latency by prefetching and caching domain objects. The authors developed several extensions to the Java programming language with the goal of allowing the use of a distributed transactional memory within any application that employs their system.

The objective of this work is to develop an innovative stochastic approach for guiding high-level software caches. It consists in using a guided cache policy to decide which data to keep in memory and which data may be discarded. The cache policy is guided because it adapts to the behaviour displayed by the application, and its goal is to provide the highest possible cache hit ratio, while keeping in memory (cache) the minimum amount of data.

The article has the following structure. Section II describes the system. Section III presents the results obtained through the benchmark execution and evaluates the system effectiveness. Finally, Section IV derives the concluding remarks.

II. SYSTEM DESCRIPTION

The system is composed of two parts: a stochastic access-prediction module and a high-level software cache. The access-prediction module is responsible for analysing the behaviour of the underlying application and in identifying the most common data access patterns performed. This information is subsequently used to guide the cache policy with the aim of improving its performance (at the level of its hit ratio). The software cache consists in a transparent data-storage component, responsible for supplying with data any request issued by the overlaying application, with the goal of improving the performance of an application.

A. Stochastic Behaviour Analysis

The stochastic behaviour analysis module is made up of three sub-modules: a code-injection module, a data-acquisition module, and a data-analysis module. An overview of their functionality is given here, while a detailed discussion of their implementation and behaviour may be found in [8], [9] and [10]. The model of Bayesian Updating, first presented in [8], is employed here for the stochastic behavioural analysis of the target application. An alternative model, based on discrete-time Markov Chains, may be seen in [9], whilst [10] deals with an Importance Analysis model.

The code-injection module is responsible for transforming the code of the target applications to inject the calls to the functionality present in the other modules. This code injection is performed in a completely automatic fashion by the system. It avoids the need for the application programmers themselves to perform any modifications whatsoever to their applications.

The data-acquisition module is responsible for acquiring behavioural data from the target application. This data describes how the application behaves, with regard to the data accesses that it performs. This module records which (application-domain) data is read and/or written, and in which contexts (methods, services, etc) this takes place.

Finally, the data-analysis module contains the implementation of the Bayesian Updating Inference model. This model corresponds to a stochastic approach for modelling the behaviour of the target application. The model uses as input the information collected by the data-acquisition module, about which domain data has been accessed by the application, and in which contexts. Initially, the input information is split into two sets of data. The first

of these data sets is designated as *prior* and contains information about the target system behaviour observed in the past. The second set is called *current* and includes more-recent behavioural information. It covers the time period defined between the moment at which the prior set ends, to the current point in time. Once these two sets have been established, the Bayesian Inference model uses the current data to "update" the posterior, generating thus a third set, called *posterior*. The posterior set corresponds to the prediction generated by the model. It describes the expected behaviour of the application, in the near future, in terms of the domain data that it is going to access. This is presented in terms of the probabilities of reading and writing domain data, depending on the contexts through which the application passes during its execution.

B. Software Cache

This section describes the implementation of the high-level software cache and its policy. It should be noted that the term "high-level" is used here in the sense that the objects being cached correspond to actual domain object instances, rather than a derivate of an SQL result set or some other lower-level abstraction. The mapping from the format used by the underlying persistence layer to the domain object instances manipulated by the application is taken care by the fenix-framework (Fernandes and Cachopo [11]).

The software cache implements the identity map design pattern, Fowler [12]. This pattern prevents duplicate loading of objects from the persistence layer. Consequently, if a requested datum has already been loaded from the persistence layer, then the identity map returns the same instance of the already instantiated object. If it has not been loaded yet, then the object is retrieved and stored in the map, before being returned to the request that demanded it.

This cache is implemented on top of Java's soft references. A normal Java reference, also known as a strong reference, guarantees that any object that is reachable through a chain of strong references is not eligible for garbage collection (GC). On the other hand, an object that is only weakly reachable is going to be discarded at the following cycle of the garbage collect. Soft references are not required to behave differently from weak references, but, in practice, softly-reachable objects are generally retained (in memory) provided that there is enough free space available to keep them there.

Continuing on to the implementation details, the high-level software cache keeps two collections into which it stores the loaded domain object instances. The first collection keeps soft references to all of its elements. This collection contains all the instances present in the cache. From the definition of a softly-referenced object, if the application does not hold a strong reference to them, the GC may discard them at will, if it deems it necessary to do so. However, they are usually kept in memory as long as it is not strictly necessary to evict them.

The second collection holds strong references to its elements, and guarantees that these can never be garbage collected, provided they remain in this collection. Objects being loaded into the cache are selectively added to the

strongly-referenced collection. The main idea behind this collection is to keep only object instances considered to be important for the execution of the application, and that should be kept in memory even when they are not currently being used. The decision of adding an element to this collection belongs to the caching policy, implemented as follows.

The cache policy manager considers, on an instance-by-instance basis, if a given datum should be placed in the strongly-referenced collection. It employs the results generated by the stochastic behaviour analysis module to infer how strongly referenced these should be. The main criterion is to consider the access probability of the type of data (class) to which an instance belongs. If this probability exceeds a certain threshold, then the datum is deemed critical for the application operation and is inserted into the strongly-referenced collection, besides being added to the softly-referenced collection. This approach may be complemented to take into account further restrictions, such as the available free memory, space limitations that the software cache should not exceed, or proportions of different domain data types kept in memory, among others.

Additionally, due to the fact that the stochastic analysis model is dynamic, it reveals any behavioural change that may eventually come to pass within the target application. This would bring about an updating of the expected application domain data access probabilities. Furthermore, it would lead to a change in the data types considered critical by the cache policy manager, which would be reflected in the contents of the strongly-referenced collection, resulting, ultimately, in a caching policy that can adapt itself to deal adequately with any behavioural patterns the application may exhibit during its life cycle.

III. RESULTS AND EVALUATION OF THE SYSTEM

For the validation of the system presented in this paper, we used two distinct benchmarks. The first of these is the TPC-W benchmark, which was introduced originally by Smith [13]. This benchmark specifies an e-commerce workload that simulates the activities of a retail store website, where emulated users can browse and order products from the website. The main evaluation metric is the WIPS – web interactions per second that can be sustained by the system under test. The TPC-W benchmark execution is characterised by a series of input parameters that control its behaviour. Among these is the type of workload simulated by the benchmark emulated browser clients. The results presented in the article are associated with the "Shopping Mix" workload, which is composed of 80% read operations and 20% write operations. Regarding the main control parameters, they are as follows: number of emulated browsers - 10; ramp-up time - 600sec; measurement interval - 1200 sec; ramp-down time - 300sec; number of items in the database - 100 000; think time - 0.01 (this value ensures that the emulated browsers wait between 0.07 sec and 0.007 sec before making a new request to the server). The emulated browsers and the benchmark application server were run on the same physical machine.

The second benchmark is the oo7, firstly presented by Carey et al. [14]. This benchmark is often used to assess the performance of object-oriented persistence mechanisms. It strives to present a broad set of operations, allowing the building of a comprehensive performance profile. The oo7 benchmark was designed to boast properties common to different CAD/CAM/CASE applications, although in its details it does not model any specific application. A run of the benchmark executes a series of traversals, updates, and query operations over the underlying object model, and the performance metric used is the time that these operations take to execute.

The results obtained with our proposed approach to implement a cache policy shall be presented next. We omit a more thorough analysis of the correct behaviour and precision of the predictions of the stochastic behaviour analysis module, because this has already been performed in [8]. There, it is demonstrated that the module is capable of predicting with high precision the types of data being accessed by the target application in the contexts through which it passes during its execution. The term context can be defined to correspond to a procedure, service, or any other abstraction deemed appropriate to describe the scope within which the current operation is taking place. Moreover, the module is capable of predicting not only the type of data (domain classes) that is most likely to be accessed in a given situation, but also the effectively accessed object fields. For the discussion presented here, only access probabilities at the level of domain class shall be considered.

As has been explained in Section II.B, whenever a domain object instance is loaded into cache, it is always added to the softly-referenced collection. Additionally, the cache policy manager uses the stochastic access prediction module to determine what is the global (at the level of the whole application) access probability of the type of object being loaded. If the access probability exceeds a certain threshold, then the object instance is also added to the strongly-referenced collection, ensuring that it cannot be garbage collected. This policy shall be referred to as the DAP (data access pattern) policy for the remainder of this article.

Due to the fact that it is through the strongly-referenced collection that the cache policy effectively controls which objects are kept in memory longer than their actual use by the application, all cache hit and miss ratio results presented next are computed based on the contents of the strongly-referenced collection only.

For evaluating the effects of employing the DAP policy, the resulting cache hit ratios are compared with those obtained by the use of three alternative cache policies.

The first of the alternative policies decides whether to insert a given domain object in the strongly referenced collection as a function of a randomly generated number. The random generator employs a uniform distribution.

The second policy adds objects to the strongly referenced collection whenever they are first loaded by the application into the cache – the first objects to be loaded into the cache are the first to be made strongly reachable (this policy shall

be henceforth referred to as “first load, first strongly referenced” – FLFSR).

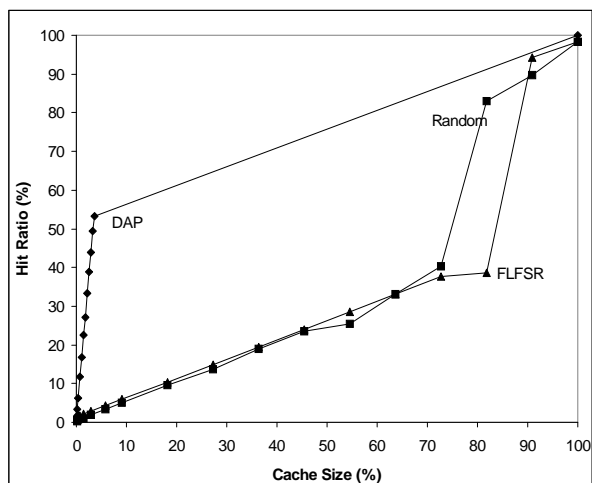


Figure 1. DAP, Random, and FLFSR policies - oo7

The third alternative corresponds to an LRU (least-recently used) policy. Its implementation is based on a synchronized and thread-safe version of the LRUMap structure of the Apache Commons Collections library. Because of the significantly different behaviour of an LRU policy, its comparison against the DAP policy shall be performed separately.

Furthermore, regardless of which policy is used, the strongly-referenced collection has an enforced maximum capacity. As such, for the Random and FLFSR policies, objects are inserted only if this capacity has not been reached. Generally, the dynamic nature of the DAP and LRU policies allows them to change the contents of the cache without exceeding the above threshold.

The results obtained from the execution of the oo7 benchmark for the DAP, Random, and FLFSR policies are shown in Fig. 1. The x-axis of the chart indicates the percentage of objects allowed to be strongly referenced in the cache, as a function of the total number of domain objects loaded into the cache during an execution of the benchmark. It has to be pointed out that due to the fact that both benchmarks access all of their domain objects during their operation, all of the existing persistent domain data ends up being accessed and cached during a single benchmark execution. The y-axis indicates the overall cache hit ratio achieved by a certain cache policy when the cache size is restricted to the value on the x-axis. Each of the dots presented in the graphs corresponds to the weighted average resulting from the measurements extracted from ten independent executions of a benchmark, for a given strongly-referenced cache size restriction.

As the results of the oo7 benchmark show, the DAP cache policy achieves better hit rate than both the Random and FLFSR cache policies, for the whole range of cache sizes. In particular, with only 3.6% of the total volume of domain data, the DAP policy achieves a hit rate of approximately 53%, whereas the Random and FLFSR

policies require caching 76% and 84%, respectively, of the total volume of existing data to achieve a similar hit rate.

An interesting observation regarding the results from the DAP policy is that the data considered as important according to the stochastic analysis module (and thus suitable to be placed in the strongly referenced collection in the cache) corresponds to 3.6% of the total volume of existing domain data. This explains two peculiarities of the results observed for this policy. The first of these is the high cache hit ratio achieved for the relatively low volume of cached data (3.6%). It confirms the belief that the most frequently used data for a given application corresponds to a relatively small set of data. The second is the lack of measurements in the range of 3.7% to 99% along the x-axis. According to the behaviour prediction module, besides the 3.6% of data considered very important for the operation of the application, there is no other domain data that is even closely as likely to be needed by the application. Consequently, the cache policy cannot place any additional information in the strongly referenced part of the cache.

The results for the DAP (uninterrupted curves) and LRU (dotted curves) policies are shown in Fig. 2. The x-axis of the chart corresponds to a logic time scale, where a single unit corresponds to the realization of 10,000 lookup operations in the cache. The y-axis indicates the accumulated cache hit ratio up to a given point in the logical time scale. We chose a logic time scale instead of a real time scale because even though the benchmark is deterministic and performs all operations in the same order, differences in the execution time from one benchmark run to another would cause the different sampled curves to compress or expand with regards to one another, resulting into a rather deformed diagram. With a logical time scale, all curves are in synch with one another.

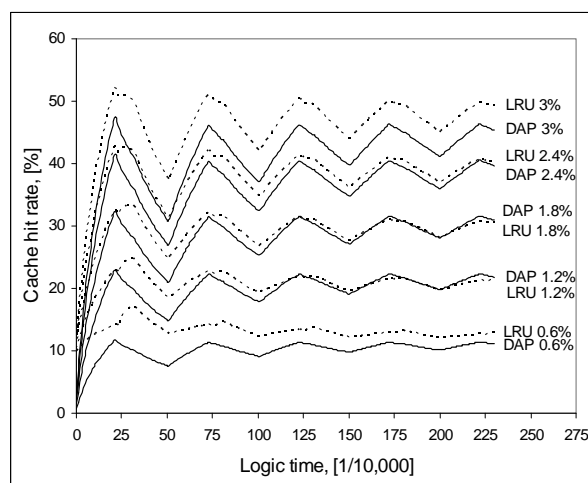


Figure 2. DAP and LRU policies - oo7

Fig. 2 presents five sample curves plotted for the DAP and LRU policies. Each of these is associated with a different cache size, corresponding to 0.6%, 1.2%, 1.8%, 2.4%, and 3% of the total volume of domain data. This relatively low percentage of domain data is due to the fact

that only a very small part of the domain data is highly likely to be accessed in run-time. This domain data accounts for a maximum of 3% of all existing domain data. The results show that the LRU cache policy presents a better cache hit ratio for the great majority of cases,. The differences in average hit ratios between the two policies vary from 1.13% for 1.2% cache size up to 5.03% for 3% cache size, all in favour of the LRU policy. The weighted average of all measurements is 2.55% cache hit ratio difference in favour of the LRU policy. Regarding the effect on overall benchmark performance, there were no observable differences between the DAP and LRU policies.

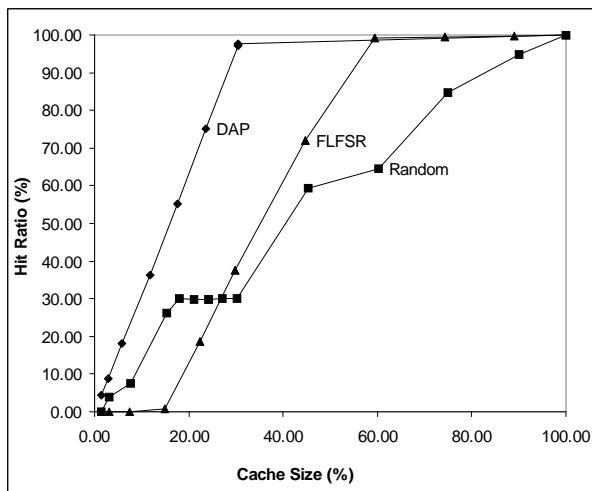


Figure 3. DAP, Random, and FLFSR policies - TPC-W

The results achieved from the execution of the TPC-W benchmark are discussed next. The hit ratio measurements for the DAP, Random and FLFSR cache policies can be seen in Fig. 3. The remarks to be made about these results are similar to the ones for the oo7 case, namely, the DAP cache policy presents cache hit ratios that are significantly better than the ones provided by the Random or FLFSR policies for any configuration.

Analysing the results of the DAP policy, we observe a practically linear growth in the hit ratio, starting from a hit ratio of 4.44% for cache size of 1.47% up to a hit ratio of 97.58% when the cache size corresponds to 30.42% of the domain data. The lack of measurements in the range of 30% to 100% of the cache size result from the same reasons presented for the oo7 benchmark – the domain data evaluated as important for the operation of the application corresponds to 30% of all of the existing domain objects; the remaining 70% of domain data are practically irrelevant, as they correspond to the remaining 2.42% of cache hit rate.

Considering the results for the uninformed caching policies, we are faced with a phenomenon not present in the oo7 benchmark results. This phenomenon consists in the existence of “plateaus” in the hit rate values achieved for a given range of cache sizes. For the Random policy, instances of this are the 30% hit rate in the range of 18% to 30% cache size and the 65% hit rate for 45% to 60% cache size. For the FLFSR policy, similar remarks are applicable to the 0% hit

rate in the range of [0%, 15%] cache size and the 99% hit rate for the range of [59%, 100%] of cache size. These plateau phenomena may be explained by the caching of domain data that is practically irrelevant, from the point of view of the application needs. This leads to an increase in the volume of cached data without any significant increase in hit rate, which is what the plateaus effectively correspond to.

The final set of results, comparing the DAP and the LRU policies, are shown in Fig. 4. In this case, the x-axis corresponds to a real time scale where the unit corresponds to 20 seconds, whilst the y-axis indicates the accumulated cache hit ratio observed up to a given point in the benchmark execution. There are three curves for each of the two policies, corresponding to cache sizes of 9%, 12%, and 15%. For the TPC-W benchmark, the LRU policy displays an even more accentuated advantage over the DAP policy, with regards to the hit ratio they achieve. In terms of differences between average hit ratios, the LRU policy leads with 9.2% for the 9% cache size, 13.8% for the 12% size, and 9.8% for the 15% size. This leads to an overall average hit ratio advantage of 10.9% in favour of the LRU policy. Yet, even though the average values give a clear advantage to the LRU policy, the observed behaviour for the LRU hit ratio is rather irregular, at least when compared to that of the DAP policy, whose results are very close to flat horizontal lines.

The most-significant difference between the two approaches in the case of the TPC-W benchmark (unlike what was seen for the oo7 benchmark) is the performance variations observed between the versions running with the DAP and the LRU policy. These variations are due only to the performance of the policy itself, rather than, for example, to the contents of the cache, because the contents of the real cache, which dictates the overall benchmark performance, is the same for both versions. Only the contents of the strongly-referenced collections are distinct and it is against those that the hit ratios are measured.

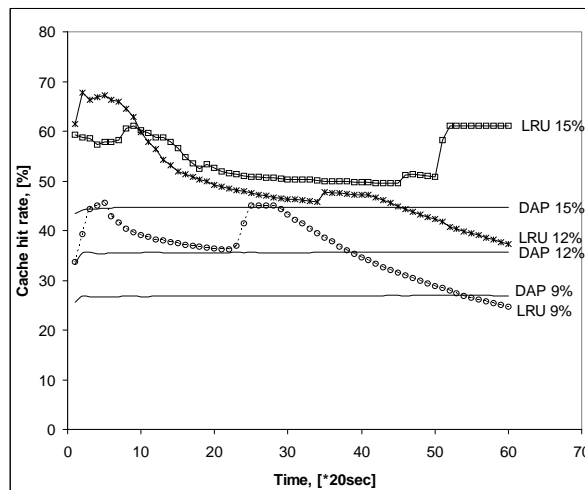


Figure 4. Using DAP and LRU policies with TPC-W

A comparison of the benchmark’s performance when using the DAP and the LRU policies is shown in Fig. 5. The x-axis indicates the number of emulated browsers (EBs)

employed for a given benchmark run and the y-axis shows the relative throughput gain achieved for a given number of EBs. The values shown correspond to the speedup relative to executing the benchmark version with LRU cache policy with 1 EB. These results show that both versions perform approximately the same amount of work for 2 EBs, which is approximately 100% more than the LRU version with 1 EB. However, as the number of EB increases, the results show a growing discrepancy in the benchmark performance between the LRU and the DAP policy versions. For four EBs, the LRU benchmark version performs approximately 210% more work than the baseline, whereas the DAP version manages over 310%. This difference is even more accentuated for 10 EBs, where the performance of the benchmark with the LRU policy has remained practically the same as the one from the 4 EBs configuration, whilst the DAP version has grown up to over 560%.

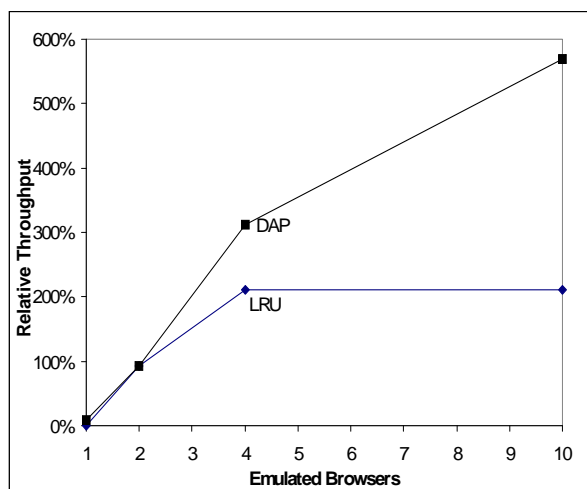


Figure 5. DAP and LRU throughput comparison, TPC-W

The most reasonable explanation for this phenomenon is that the synchronization present in the LRU policy implementation causes a bottleneck in multithreaded scenarios, leading to the poor performance gains observed in the results. Assuming this is the case, then the DAP policy would be the preferred alternative for situations where multithreading is common, while the LRU would be more appropriate for single threaded configurations.

IV. CONCLUSIONS

This paper presented a new approach for guiding the cache policy of a high-level software cache. This new approach employs a stochastic analysis based on Bayesian Updating Inference, which is responsible for predicting the behaviour of the target application, regarding its domain data needs. Based on the generated predictions, the cache policy is capable of deciding which domain objects are to be cached, leading to high cache hit rates with relatively low volumes of cached domain data.

The effectiveness of this approach was tested with two very different benchmarks – the TPC-W and the oo7 – by

comparing it against three different cache policies. The results illustrate the usefulness of employing dynamic adaptive approaches for guiding high-level software caches, by taking into consideration the behaviour of the target application.

ACKNOWLEDGMENT

This work was partially supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and by the Specific Targeted Research Project (STReP) Cloud-TM, which is co-financed by the European Commission through the contract no. 257784. The first author has been funded by the Portuguese FCT (Fundação para a Ciência e a Tecnologia) under contract SFRH/BD/64379/2009.

REFERENCES

- [1] Denning, P. J. and Schwartz, S. C., 1972, Properties of the working-set model, *Commun. ACM*, 15, (3), pp. 191-198.
- [2] Sandhu, H. S., Gamsa, B. and Zhou, S., 1993, The shared regions approach to software cache coherence on multiprocessors, *SIGPLAN Not.*, 28, (7), pp. 229-238.
- [3] Lilja, D. J. and Yew, P.-C., 1991, Combining hardware and software cache coherence strategies, *Proceedings of the 5th international conference on Supercomputing*, Cologne, West Germany, ACM, pp. 274-283.
- [4] Adve, S., Adve, V., Hill, M. and Vernon, M., 1991, Comparison of hardware and software cache coherence schemes, *SIGARCH Comput. Archit. News*, 19, (3), pp. 298-308.
- [5] Chen, T., Zhang, T., Sura, Z. and Tallada, M., 2008, Prefetching irregular references for software cache on cell, *Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization*, Boston, MA, USA, ACM, pp. 155-164.
- [6] Bennett, J., Carter, J. and Zwaenepoel, W., 1990, Adaptive software cache management for distributed shared memory architectures, *17th Annual International Symposium on Computer Architecture*, pp. 125-134.
- [7] Dash, A. and Demsky, B., Integrating Caching and Prefetching Mechanisms in a Distributed Transactional Memory, To Appear in *IEEE Transactions on Parallel and Distributed Systems*.
- [8] Garbatov, S., Cachopo, J. and Pereira, J., 2009, Data Access Pattern Analysis based on Bayesian Updating, *Proceedings of the First Symposium of Informatics (INForum 2009)*, Lisbon, Paper 23.
- [9] Garbatov, S. and Cachopo, J., 2010, Predicting Data Access Patterns in Object-Oriented Applications Based on Markov Chains, *Proceedings of the Fifth International Conference on Software Engineering Advances (ICSEA 2010)*, Nice, France, pp. 465-470.
- [10] Garbatov, S. and Cachopo, J., 2010, Importance Analysis for Predicting Data Access Behaviour in Object-Oriented Applications, *Computer Science and Technologies*, 1, pp. 37-43.
- [11] Fernandes, S. and Cachopo, J. A New Architecture for Enterprise Applications with Strong Transactional Semantics. Lisbon: INESC-ID / IST, 2011.
- [12] Fowler, M., 2003, *Patterns of enterprise application architecture*: Addison-Wesley Professional.
- [13] Smith, W. TPC-W: Benchmarking An Ecommerce Solution. Intel Corporation, 2000.
- [14] Carey, M., Dewitt, D. and Naughton, J., 1993, The OO7 benchmark, *ACM SIGMOD International Conference on Management of Data*, pp. 12-21.

Performance Simulation of a System's Parallelization

Markus Meyer*, Helge Janicke[†], Peter Trapp*, Christian Facchi* and Marcel Busch*

**Institute of Applied Research, University of Applied Sciences Ingolstadt, Germany*

{markus.meyer, peter.trapp, christian.facchi, marcel.busch.winf}@haw-ingolstadt.de

[†]*Software Technology Research Laboratory, De Montfort University, Leicester, United Kingdom*

heljanic@dmu.ac.uk

Abstract—A new approach to simulate the parallelization of a software function is presented in this paper. The parallelization's effects onto the system's performance prior to a costly realization of the parallelization are evaluated, leading to a more gain-oriented performance optimization. The presented approach defines a methodology to transform a single-threaded software function into a multi-threaded simulation. *CPU stubs*, simulating both, the performance and functional behavior, are applied to simulate the expected time slices. The proposed technique can estimate the expected performance gain for the whole system. A proof of concept is used to evaluate the proposed methodology and the simulation results are compared to a known parallel implementation of the algorithm. Initial results show our approach can be used to simulate the performance behavior of a parallelized system with high accuracy. In addition, the number of threads that result in the highest performance gain of the system is determined.

Keywords—*software performance optimization; performance simulation; parallelization; dynamic performance stubs*

I. INTRODUCTION

Dynamic performance stubs [1] can simulate various optimization levels of the component under study (CUS). *CPU stubs* [2], replace a software function (CUS) in the system under test (SUT) and model its CPU performance behavior. *CPU stubs* consist of two functionalities, the *simulated software functionality* (SSF), that recreates the functional behavior of the bottleneck, and the *performance simulation functions* (PSF) that simulate its performance behavior [3]. *PSF* are used to simulate different optimization levels and show the optimization's effect on the system's performance.

This paper presents a novel approach to the analysis of an identified bottleneck with respect to its parallelization potential. The key contribution of this work is that it allows for informed decision making to what degree the optimization of the bottleneck using parallelization techniques would impact on the overall performance of the system.

Section II shows the related work. The presented approach (Section III) is based on a simulation of the bottleneck. It is an efficient technique that can help to avoid wasted effort and costs associated with the parallelization of the affected system component. The described methodology is evaluated by a proof of concept in Section IV. The simulation of a system's parallelization can be used to estimate the achievable performance using parallelization techniques.

II. RELATED WORK

The concept of *dynamic performance stubs* was introduced in [1]. In [3], the usage of *CPU stubs* to determine the performance gain of a system's optimization is shown. Our approach, extends the *dynamic performance stubs* to investigate if this performance gain can be achieved by parallelization. According to Amdahl [4], not all instructions that are executed within a system can be performed concurrently. Amdahl shows that the maximum speedup of a parallelization is limited by the sequential part of an algorithm for an infinite number of processors. Gustafson [5] claims that the speedup that can be obtained by a parallel execution is not only limited by the remaining sequential part but increases linearly by the number of used processors. Both, [4] and [5], determine an upper bound of the speedup achievable with parallelization. In contrast, our approach simulates the anticipated system-wide performance benefit. In our case, the speedup described by Amdahl and Gustafson can not be used to estimate the runtime of the parallel parts of the algorithm without modification (see [6]).

The parallelization of a sequential algorithm requires thread management that reduces the achievable speedup [7]. Marinescu and Rice [8] introduced the concept of relative speedup taking sequential and duplicated work, communication and control and blocking into account. This paper's approach also considers the overhead created by parallelization. By simulating the parallelization, system influences such as the number of available processors and the current system load are included.

In [9], an approach to model the influences of the number of threads and processors on a system's performance using Solaris containers is introduced. In contrast, the presented approach simulates the parallelized software within the real system in order to get accurate performance measurements. To the best of our knowledge, no studies have evaluated the overall performance gain that can be achieved within the system using parallelization techniques.

III. SIMULATING PARALLELIZATION

This section presents an approach to simulate the parallelization of the system in order to obtain measurements on which decisions to use parallelization as an optimization

technique can be well-founded. Based on the results the degree of parallelization can be determined.

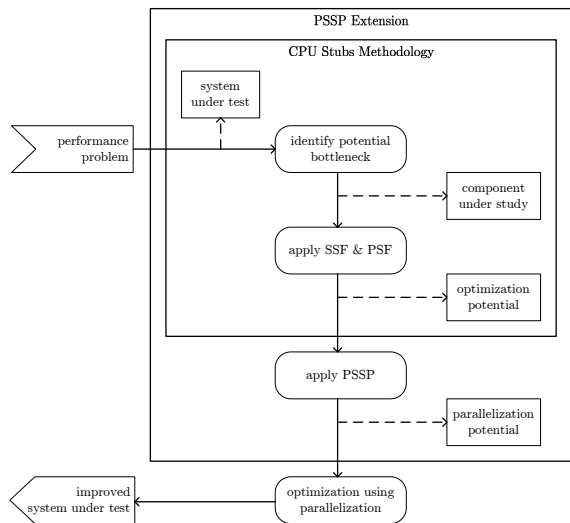


Figure 1. “Performance Simulation of a System’s Parallelization” (PSSP) - Extension of the CPU Stubs’ Methodology

Figure 1 depicts our approach. The *performance simulation of a system’s parallelization* (PSSP) extends the methodology of *CPU Stubs* [2]. *CPU Stubs* determine the optimization potential of a system that is affected by performance issues. A potential bottleneck (component under study, CUS) is identified within the system under test (SUT). The CUS is replaced by a *CPU Stub* simulating the functional and the performance behavior of the CUS. By varying the stub’s performance behavior, the *optimization potential* for this component is determined. The approach presented in this section extends the known methodology by the simulation of the system’s parallelization to decide whether the *parallelization potential* within the CUS is sufficient to reach the performance optimization goals. Hence, the presented approach helps to transform the measured overall optimization potential into a gain-oriented realization of the performance optimization.

A. Objectives

Depending on the accuracy of the known and measured original performance data, the simulation of the parallelization pursues the following objectives:

- **Performance potential:** Investigate if it is possible to gain performance within the system when parallelizing the CUS.
- **Determine the expectable performance gain:** Determine the performance gain that can be achieved by parallelizing the CUS. The system load and the available hardware resources limit the possible performance of the system. Decide whether the performance targets can be reached by a parallelization.

- **Determine the maximum overhead:** Parallelizing a sequential algorithm results in additional effort for thread management. The overhead needed to manage the parallelization reduces the achievable speedup.
- **Determine the optimal number of threads:** The number of used threads influences the overhead needed to manage the parallelization. The optimal number of threads that entails a short thread runtime in combination with a small amount of overhead is determined by test series.

B. Approach

The presented approach is based on the simulation of a concurrent implementation of the system component containing the bottleneck (CUS). It uses a simple fork and join mechanism to give a first estimation if parallelization is a valuable way to optimize the CUS. The introduced method applies the parallelization of the system using domain decomposition. Several threads are created within the simulation using the same thread runtime.

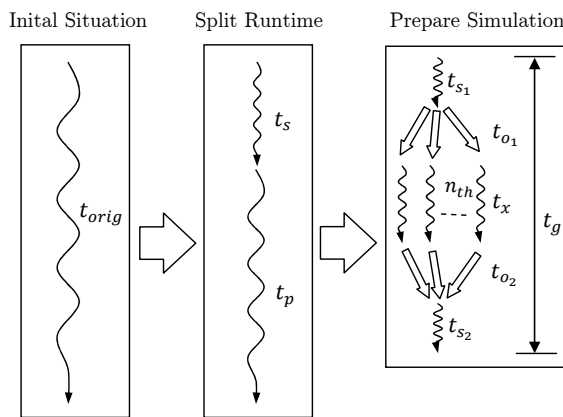


Figure 2. Steps Towards the Parallel Simulation

In Figure 2, the sequential time is broken down and replaced by a time estimation of the parallel computation.

Initial Situation A sequential implementation of the CUS is assumed. Its runtime t_{orig} is determined by measurements, e.g., using real time counters (see [10]).

Step 1: Algorithm Parallelization Potential The original runtime (t_{orig}) is split into a sequential part (t_s) and a parallel part (t_p) (see [4]), by carrying out data and control flow analysis (see [11], [12]).

In the next steps, the parts of the bottleneck that can be parallelized (t_p) as well as the sequential part (t_s) are prepared for multi-threaded execution. For that, the distribution of the sequential part, the threads’ runtime (t_x) and the additional work that is needed to manage the threads (t_o) are determined.

Step 2: Sequential Part The control flow analysis, applied in the first step of this approach, can be used to determine the distribution of the sequential part to the

beginning and the end of the simulation. The total sequential time remains constant at $t_s = t_{s1} + t_{s2}$.

Step 3: Threads' Runtime The parallel time (t_p) is converted into the threads' runtime (t_x) considering that:

- each thread has to execute a subset of the original instructions. Therefore, the determined parallel execution time (t_p) will also be split to the various threads. This has to be captured by the parallel simulation of the algorithm. The threads' runtime (t_x) is a function of the number of threads: $t_x(n)$ where $0 \leq n \leq n_{th}$.
- the algorithm in the bottleneck rarely has linear time complexity. The problem size remains constant for the test case and is equally split in the n_{th} threads. However, the threads' runtime does not change in a linear way for algorithms with non-linear time complexity when changing the problem size. Due to that, the threads' runtime not only depends on the number of created threads (n_{th}), but also on the parallelized algorithm's complexity.

There are two approaches to determine the threads' runtime. A complexity analysis of the CUS and the measurement of the algorithm's time behavior for various problem sizes can be performed:

Complexity Analysis: A complexity analysis of the CUS's algorithm is applied to determine the time complexity of the algorithm using Big-O-Notation (see [13]). Using the complexity, the threads' runtime t_x can be calculated depending on the number of threads. The following example assumes that an algorithm with a time complexity of $O(n^2)$ is used and that $n_{th} = 2$ threads are created within the simulation.

$$t_p \leq c * n^2 \quad (1)$$

Equation 1 shows the formula for the parallel execution time (t_p). The time complexity given in the Big-O-Notation defines an upper bound for the time. Therefore, the parallel execution time is less or equal to a constant time (c) multiplied by the given complexity.

This paper's approach uses domain decomposition to simulate the system's parallelization. This is realized by a divide-and-conquer strategy (see [13]). Thus, the presented approach uses the same algorithm but with an reduced problem size per thread to simulate the parallelization. As shown in Equation 2, the reduced problem size is used to calculate the execution time.

$$t_x \leq c * \left(\frac{n}{n_{th}}\right)^2 \leq c * \left(\frac{n}{2}\right)^2 \quad (2)$$

The combination of Equations 1 and 2 shows that in this example the threads' runtime (t_x) only depends on the parallel runtime (t_p). The upper bound $t_p = c * n^2$ of the parallel runtime is used as a pessimistic estimate to calculate the threads' runtime in Equation 3.

$$\frac{t_x}{t_p} \leq \frac{c * \left(\frac{n}{2}\right)^2}{t_p} \leq \frac{c * n^2}{4 * t_p} \leq \frac{t_p}{4 * t_p} \leq \frac{1}{4} \quad (3)$$

The upper bound of the calculated thread runtime is $t_x = \frac{t_p}{4}$.

Measured Time Behavior: Another approach to determine the threads' runtime is the measurement of the algorithm's time behavior. The CUS's algorithm is available and can be executed with different problem sizes capturing the various time behaviors of the algorithm. This data is used to determine the threads' runtime for a given problem size. The measurements' results strongly depend on the used input data, as best- or worst-case scenarios of the algorithms can be triggered. To get the expected timing data, the input data has to be chosen specifically corresponding to the original data.

Step 4: Parallelization's Overhead In addition to the threads' runtime (t_x) the time used to fork and join the threads as well as to split and combine the result of the problem has to be considered within the simulation as overhead $t_o = t_{o1} + t_{o2}$. The overhead is split into two sections: creation (t_{o1}) and synchronization (t_{o2}) of the multi-threaded part (Figure 2). The overhead depends on the concrete implementation. Two options need to be considered:

- 1) An idea about the implementation of the parallelization is available. Hence, the effort that is needed for the thread management can be estimated depending on the number of created threads.
- 2) Otherwise, it is possible to simulate the parallelization without an overhead. In this case, only the maximum overhead that can be introduced by the parallelization in order to gain performance can be determined. This reduces the problem to a thread-management and task distribution problem that can be solved more easily.

Depending on these considerations, the evaluation of the results focuses on different objectives (see Step 6).

Step 5: Simulation of the Parallelization The determined values of t_s , t_o and t_x are used to simulate the performance of a possible parallelization (Figure 2). Each of these elements has to be rebuilt as a *system influencing PSF* [2]. To build the multi-threaded behavior of the simulation, a method to generate the *multi-threaded PSF* from a textual description is introduced in [3]. In combination with the *SSF* [3], it is possible to simulate the expected performance behavior of a parallelization. The expected parallel execution time is $t_g = t_s + t_o + t_x$. This theoretical value does not consider any scheduling or runtime influences. The time t_g^{sim} , measured within the simulation under real conditions describes the real value of t_g . Consequently, the simulation takes system bottlenecks, as the number of available processors or other processes running on the system, into account.

Step 6: Evaluation of the Results Depending on the overhead estimation (Step 4) two results of the simulation can be evaluated:

- The performance gain by parallelization of the bottleneck is defined as $t_{gain} = t_{orig} - t_g^{sim}$. The simulation

is executed with a varying number of threads (n_{th}) to determine the optimal gain, leading to adjusted values for t_x and t_o . This is used to determine the optimal number of threads (n_{th}^{opt}).

- If the threads' administration overhead cannot be estimated (Step 4.2) the determination of maximum overhead is another objective. In this case the simulation rebuilds the sequential and parallel execution times, t_s and t_x and estimates the maximum overhead by $t_o^{max} = t_{orig} - t_g^{sim}$. If the parallelization and synchronization of the bottleneck can be performed in less than t_o^{max} , the optimization will result in a performance improvement for the bottleneck.

These results are used to decide whether parallelization can be applied as an optimization technique to achieve the performance targets.

IV. PROOF OF CONCEPT

This section evaluates the proposed methodology using a proof of concept. The threads' runtime is determined as described in Step 3. The measurements are used to simulate the parallelization for various numbers of threads. Finally, the simulation's results are compared to the performance data of the algorithm's parallel implementation. All measurements were performed on a FSC Amilo Si3655 Notebook with an Intel Core(TM)2 Duo P8400 CPU (Intel 64 architecture). As operating system Arch Linux, kernel version 2.6.34, is used. Binaries were build using GCC (version 4.5.1).

A. Overview

The example simulates the parallel execution of a sequential bubble sort algorithm (see [14]). A parallel implementation of this algorithm is used to validate the results of the simulation and uses the same implementation extended by a merge-sort to combine sorted sublists.

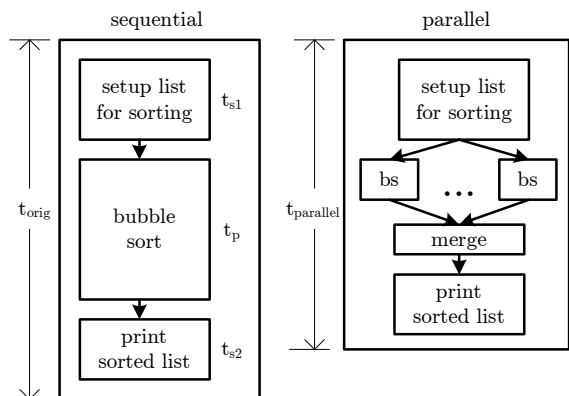


Figure 3. Sequential and Parallel Bubble Sort Algorithms

Figure 3 shows the timings of the bubble sort algorithms. Both algorithms have the same set up. The sequential algorithm processes the whole list, whereas in the parallel case,

the list is split into n_{th} parts that are sorted independently in different threads. The parallel execution additionally merges the sorted partial lists. Finally, both algorithms print the whole sorted list. To guarantee reproducibility of the test results, the used list is initialized with 4096 random integer numbers once and used for each test run.

B. Algorithm Parallelization Potential (Step 1)

The original runtime (t_{orig}) of the sequential algorithm was determined using the time stamp counters. Additionally, the runtime of the parallel part (t_p) and the sequential parts of the algorithm $t_s = t_{s1} + t_{s2}$ were measured as described in Step 1.

	avg[ms]	max[ms]	min[ms]	sqdcoeff of var
t_{orig}	103,66	103,87	103,48	4,06E-07
t_p	103,49	103,70	103,31	4,04E-07
t_{s1}	0,152	0,155	0,150	9,87E-05
t_{s2}	0,0177	0,0177	0,0175	1,63E-05

Table I
RUNTIME VALUES OF THE SEQUENTIAL BUBBLE SORT

Table I shows the measurements' average, maximum and minimum time of 20 samples taken in milliseconds. The squared coefficient of variation (see [15]) is applied to evaluate the deterministic behavior of the measured data.

C. Sequential Part (Step 2)

The sequential time t_s was measured (Section IV-B). Thus, t_{s1} and t_{s2} are known and can be simulated.

D. Threads' Runtime t_x (Step 3)

In the next step, the measured parallelizable time of the sequential algorithm (t_p) has to be converted into the threads' runtime (t_x). As described in Step 3 of Section III-B, there are two possibilities to estimate t_x , *complexity analysis* and *measurement*. To evaluate that both approaches can be used to determine the threads' runtime, their results are compared in this step. The complexity of the bubble sort algorithm is known as $O(n^2)$ [14]. In Section III-B, the calculation of the estimated thread runtime is shown by an example. To measure the time behavior of the used bubble sort algorithm, it is executed with different problem sizes.

Figure 4 presents the comparison of the determined parallel threads' runtime t_x . As described in Step 3 of Section III-B, the calculation of the thread's runtime using a complexity analysis (crosses) and its determination by measurements (circles) are drawn. The x-axis shows the number of threads, whereas, the y-axis depicts the time in milliseconds. In order to depict the differences of the two graphs, the y-axis is drawn logarithmically. As can be seen, both alternatives nearly provide the same results and can be used to give an estimation about the threads' runtime. As described in Step 3 of Section III-B, the complexity analysis provides an upper bound for the thread's runtime.

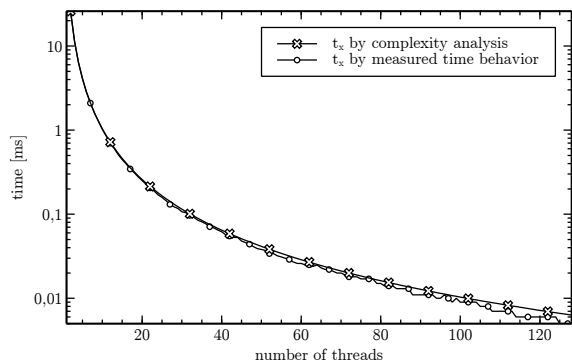


Figure 4. Determination of the Threads' Runtime

E. Parallelization's Overhead (Step 4)

The overhead that is needed for thread management has to be determined in this step. In this example, the parallel implementation of the algorithm is known. Due to that, the overhead values t_{o1} and t_{o2} can be estimated. The overhead to split the initialized list to the single threads is measured and set to $t_{o1} = 5, 2\mu s$, as the additional work that has to be done here is almost constant. The single bubble sort threads are just executed with sublists.

The work needed to synchronize the intermediate results influences the expected performance gain. As described in Section IV-A, a merge algorithm with a complexity of $O(n_{th} * n)$ is used to combine the results. In this example the overall problem size (n) remains constant. Thus, the time to merge the sublists only depends on the number of created threads. This overhead increases linearly and can be determined using the same methods as described for the calculation of the threads' runtime in Step 3 of Section III-B. In this proof of concept, it was determined to an average of $t_{o2} = 28\mu s * n_{th}$.

F. Simulation and Evaluation (Steps 5 & 6)

After all values have been determined, the simulation was executed. All measurements were performed with the number of threads increasing from 2 to 128.

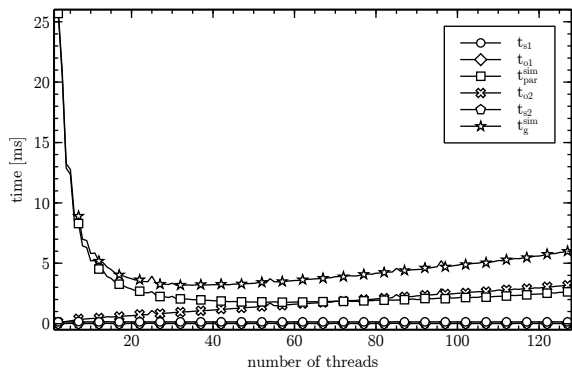


Figure 5. Simulation of the System's Parallelization

Figure 5 presents the simulation's results. The x-axis shows the number of created threads and the y-axis depicts the used time in milliseconds. The graph includes all the values introduced in Section III-B; the sequential parts t_{s1} and t_{s2} , the threads' overhead t_{o1} and t_{o2} , as well as the time needed to run all the created threads t_{par}^{sim} . Additionally, the overall simulation time (t_g^{sim}) is presented. The small variations included in the graph occur due to measuring inaccuracies.

The evaluation of the simulation resulted in a calculated performance gain of $t_{gain} = 100.48ms$ for the parallel execution with the optimal number of threads $n_{th}^{opt} = 35$.

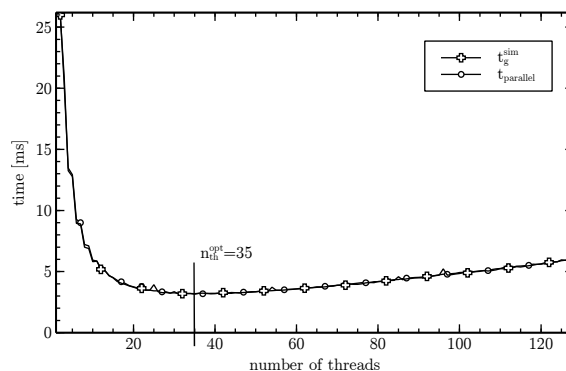


Figure 6. Comparison of Simulation and Parallel Implementation

Figure 6 presents a comparison of the overall simulation time t_g^{sim} and the total execution time $t_{parallel}$ of the parallel implementation. The y-axis plots the time in milliseconds against the number of used threads (x-axis). There are only small differences, probably caused by operating system interruptions. Clearly, the simulation of the parallelization has given an accurate estimation of the parallelization's results. Based on these initial results we are confident that the proposed methodology can be used to estimate the expected performance of a system's parallelization. The application of the presented approach provides a well founded estimation of the ideal number of threads without a realization of the parallel algorithm.

V. CONCLUSION AND FUTURE WORK

We presented a novel approach to simulate the effects of parallelization on the system's performance, by obtaining measurements on which decisions to use parallelization as an optimization technique can be based. This leads to a more gain-oriented performance optimization.

The presented approach converts a single-threaded bottleneck into a simulation of its parallel execution. Therefore, the expected bottleneck is split into several parts in order to simulate the parallel execution of the bottleneck. Especially, the part that can be parallelized is particularly investigated to determine the threads' runtime and the additional overhead that occurs when synchronizing the threads' results.

Complexity analysis and measurements are used to estimate the timing behavior of the parallel part. The identified time slices are rebuilt by *performance simulation functions*, and, thus, enable the performance simulation of the system's parallelization. The parallelization's expected performance benefit is determined by simulating varying parallelization degrees.

The described simulation provides more accurate knowledge on the achievable performance gain by parallelizing a component before spending the effort of a costly realization of the parallelization. And such, it enables a more gain-oriented performance optimization than a simple guess of the parallelization's effects. The approach is validated by a proof of concept using a bubble sort algorithm. The simulation's results estimate the expected performance of the parallel implementation with high accuracy. Additionally, the optimum number of threads that have to be created in order to achieve the maximum performance benefit is determined by the simulation. Please note that in a real environment a comparison of the expected performance gain and the parallel runtime behavior, as shown in the presented case study, is not possible until the realization of the parallelization.

In future work the introduced approach also has to be validated for other algorithms and in industrial case studies. Additionally, the CPU load will be taken into account as a further performance aspect. Upper bounds of CPU loads can be defined and help to decide whether parallelization of a system's component is a viable approach for the system's performance optimization.

The presented approach simulates a parallelization of the regarded bottleneck using domain decomposition. Functional decomposition as a parallelization technique will also be investigated, considering the simulation of threads with differing values of runtime and systems having several synchronization points of the threads during their execution. This has to be integrated with the presented approach.

A new methodology regarding the application of the performance simulation of a system's parallelization will be developed and evaluated by case studies. This enables the simulation of parallelization as a performance optimization technique. Thus, it reduces the optimization's effort and leads to a more gain-oriented performance optimization of the system.

VI. ACKNOWLEDGMENTS

The authors would like to thank the Software Technology Research Laboratory (STRL) of the De Montfort University, especially Francois Siewe and Hussein Zedan for providing the appropriate environment for research. This research has been funded by project grants from the German Federal Ministry of Education and Research (BMBF). Project: Perf-Boost, FKZ 17034X10.

REFERENCES

- [1] P. Trapp and C. Facchi, "Performance Improvement Using Dynamic Performance Stubs," Fachhochschule Ingolstadt, Tech. Rep. 14, Aug. 2007.
- [2] P. Trapp, M. Meyer, and C. Facchi, "Using CPU Stubs to Optimize Parallel Processing Tasks: An Application of Dynamic Performance Stubs," in *ICSEA '10*. IEEE Computer Society, 2010.
- [3] P. Trapp, M. Meyer, C. Facchi, H. Janicke, and F. Siewe, "Building CPU Stubs to Optimize CPU Bound Systems: An Application of Dynamic Performance Stubs," *International Journal on Advances in Software*, vol. 4, no. 1&2, 2011, accepted for publication.
- [4] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," in *AFIPS '67 (Spring)*. New York, NY, USA: ACM, 1967, pp. 483–485.
- [5] J. L. Gustafson, "Reevaluating Amdahl's Law," *Commun. ACM*, vol. 31, pp. 532–533, May 1988.
- [6] S. Krishnaprasad, "Uses and Abuses of Amdahl's Law," *J. Comput. Small Coll.*, vol. 17, pp. 288–293, December 2001.
- [7] C. Jeong and M. Shahsavari, "Performance Evaluation of Multithreading in Concurrent Programs," in *SoutheastCon, 2002. Proceedings IEEE*, 2002, pp. 7–9.
- [8] D. Marinescu and J. Rice, "Speedup, Communication Complexity and Blocking-a La Recherche du Temps Perdu," in *Parallel Processing Symposium, 1993., Proceedings of Seventh International*, apr 1993, pp. 712–721.
- [9] A. Siami Namin, M. Sridharan, and P. Tomar, "Predicting Multi-core Performance: a Case Study Using Solaris Containers," in *Proceedings of the 3rd International Workshop on Multicore Software Engineering*, ser. IWMSE '10. New York, NY, USA: ACM, 2010, pp. 18–25.
- [10] Y. Etsion and D. Feitelson, "Time Stamp Counters Library - Measurements with Nano Seconds Resolution," The Hebrew University of Jerusalem, Tech. Rep. 2000-36, 2000.
- [11] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, 2006.
- [12] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of Program Analysis*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999.
- [13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.
- [14] O. Astrachan, "Bubble Sort: an Archaeological Algorithmic Analysis," in *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, ser. SIGCSE '03. New York, NY, USA: ACM, 2003, pp. 1–5.
- [15] R. Srinivasan and O. Lubeck, "MonteSim: A Monte Carlo Performance Model for In-order Microarchitectures," *ACM SIGARCH Computer Architectur News*, vol. 33, no. 5, pp. 75–80, Dec. 2005.

Towards Executable Business Processes with the Problem Oriented Engineering Process Algebra

Dariusz Wojciech Kaminski
 Computing Department
 The Open University, UK
 dariusz.kaminski@gmail.com

Jon G. Hall
 Computing Department
 The Open University, UK
 J.G.Hall@open.ac.uk

Lucia Rapanotti
 Computing Department
 The Open University, UK
 L.Rapanotti@open.ac.uk

Abstract—The paper introduces a process algebra for business process models. The algebra is located within Problem Oriented Engineering, a framework for engineering design, and is based on a process pattern defined by Hall and Rapanotti by which Problem Oriented Engineering developments should be structured. The pattern is a generator for processes being composable in three ways: in sequence, in parallel and fractally. In explicating this process algebra, a machine readable and animatable CSP is used, which forms a semantic basis for the behaviour modelling of processes. The benefits of this algebra are: simplicity, support for business process analysis and synthesis, and explicit recognition of choices (and their impact) made by agents.

Keywords—process modelling; process algebra; Problem Orientation.

I. INTRODUCTION

Business process analysis and synthesis are greatly facilitated by executable models, such as Business Process Execution Language (BPEL), which can be represented graphically by Business Process Modeling and Notation (BPMN) [18]. Such approaches provide only the building blocks by which processes can be built, without any predictive capability of the properties of processes themselves.

A predictive model is, essentially, one with which “What if?” questions can be answered. An example question might be:

What if we asked a stake-holder at this point whether we have understood their problem well enough; would the developmental risk we face change?

Being able to answer such questions may allow us to distinguish from all those that solve a business problem, those business processes best match other criteria such as, for instance, the availability of resources or our attitude to risk.

In this paper, we provide a model that we have developed in the context of Problem Oriented Engineering (POE) [6], [7]. POE has been shown to have predictive capability in the design of artefacts, and we hope, with this paper, to begin working towards using POE’s predictive capabilities for business process design.

The paper is structured as follow. Section II provides a brief overview of business process modelling and Problem Oriented Engineering. Section III introduces and explains the POE Process Algebra, with an example in Section IV. Section V reflects on the contribution and its potential application.

II. BACKGROUND AND RELATED WORK

POE is motivated by a view of engineering as a problem solving activity [15]. POE provides many tools for solving problems, including the POE Process Pattern (PPP) [5], upon which the treatment of this paper is based. Simply put, the PPP orders a problem solving activity as iterations between exploring the problem and exploring the solution, with interleaved validation activities, whilst recognising that both problem and solution exploration can also be seen as problem solving activities.

Under the POE view, business processes are designed artefacts that solve business problems. We assume that for any business problem there will be many possible candidate business process solutions, each with different characteristics – here we consider cost (or resource use), and risk – in the solution space. This paper provides the business process designer with tools that allow the visualisation of business processes along a ‘risk/resource continuum’, each of which can then be compared to an organisation’s available resource and risk appetite, as illustrated in Figure 1.

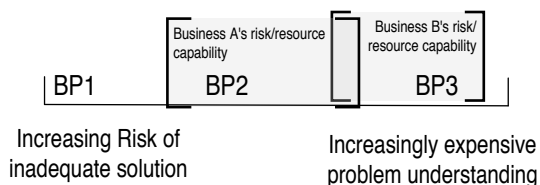


Figure 1. Two businesses A and B, each with different risk appetites and available resources, experience the same problem for which three candidate solutions – BP1, BP2 and BP3 – exist. Plotting each business process along the risk/resource continuum allows the most appropriate to be chosen.

Thus, whilst consistent with Aguilar-Savén’s definition [2]: “a business process is the combination of a set of activities

within an enterprise with a structure describing their logical order and dependence whose objective is to produce a desired result”, we provide hooks for the management of the following risk/resource trade-off [5]: although understanding the problem can be resource intensive in terms of developmental and stake-holder time, it can mitigate the risk of solving the wrong problem. For the purposes of this paper, we call this the “risk/resource trade-off assumption”.

A. Business process modelling

Many analytical frameworks have been considered for the classification of business process models. Wang *et al.*, for instance, [17] surveys BPEL4WS (BPEL for Web Services), BPMN, UML (Unified Modeling Language), XPDL (XML Process Definition Language), Petri Nets, IDEF0, and IDEF3.

Given their easy representation of concurrency and their explicit representation of state, Petri Nets have long been associated with the modelling of Business Processes, including the work of Van der Aalst [1], and that based on the Petri Box model of Best *et al.* Our model shares with Petri Nets a formal model in POE, but is focussed on on completeness of representation only on the accurate representation of problem solving steps that exist within a business process. Moreover, we aim for synthesis of business processes not just their modelling for animation.

Aguilar-Savén provides a comprehensive review of a number of business process modelling techniques and tools, and also proposes a framework to classify the techniques/models according to their purpose. The classification proposed in [2] is done according to the purposes (descriptive, analytical, enactable), and change model permissiveness (passive or active). The approach proposed in this paper aims to be active in regards to permissiveness, and depending on the context and modelling goals the purpose falls into all four categories.

Other classification emphasising the link between modelling, decision and planning capabilities, but also their relation to entities (time, resources, causality and authority), was provided by Macintosh [11].

Mentzas *et al.* [13] provide an evaluation of alternative approaches to business process modelling with workflows. One of the problems, as cited by Mentzas *et al.*, is the hardship in modelling exceptional tasks or processes, and their advice is to exclude such processes from process models, “due to uncertainty either in time or in the processing entities involved”.

Perhaps unsurprisingly, synthetic approaches to business processes are fewer in number: A formal definition of structured workflow in terms of activities was provided by Kiepuszewski *et al.* in [9], where it was shown how these can be composed to form arbitrary workflow models, but require the use of powerful verification engines to help detect whether a composed process is well-behaved.

B. Problem Oriented Engineering

Problem Oriented Engineering is a framework for engineering design, similar in intent to Gentzen’s Natural Deduction [16], presented as a sequent calculus. As such, POE supports rather than guides its user as to the particular sequence of design steps that will be used; the user choosing the sequence of steps that they deem most appropriate to the context of application. The basis of POE is the *problem* for representing *design problems* requiring designed solutions. *Problem transformations* transform problems into others in ways that preserve solutions (in a sense that will become clear). When we have managed to transform a problem to axioms¹ we have solved the problem, and we will have a designed solution for our efforts.

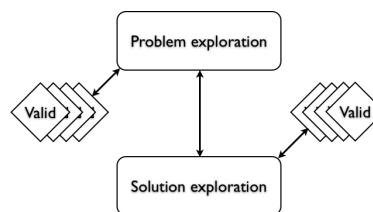


Figure 2. The POE Process Pattern: iteration between problem and solution exploration with interleaved validation (adapted from [5]).

A comprehensive presentation of POE is beyond the scope of this paper (but can be found in [6], [7]). For this paper it will be sufficient to consider the structure that POE suggests for problems solving steps that is illustrated in Figure 2 in which rectangles are resource consuming activities; diamonds indicate requests to stake-holders for validation either – on the left – of problem understanding, or – on the right – of a candidate solution.

The potential for looping in the POE process pattern concerns unsuccessful attempts to validate: unvalidated problem understanding will require problem rework as will an unvalidated solution. In this way, validation within the process has an impact on both (developmental) resources and risk: resource will vary *with* validation instances; risk varies inversely with validation instances.

C. Complex problem solving

Although the POE process pattern provides a structure for problem solving, in its raw form, a problem will only be solved when, after iteration, a validated problem is provided with a validated solution. This ‘bang-bang’ approach is suitable for simple problems, but is unlikely to form the basis of any realistically complex problem encountered in software engineering.

¹An *axiomatic problem* is a problem whose adequate, i.e., fit-for-purpose, solution is already known.

To add the necessary complexity, the POE process pattern combines with itself in three basic ways; in combination, it is again a process that can be combined. The three ways it can be combined are in sequence, in parallel and in a fractal-like manner, as suggested in Figure 3, and as described in the sequel.

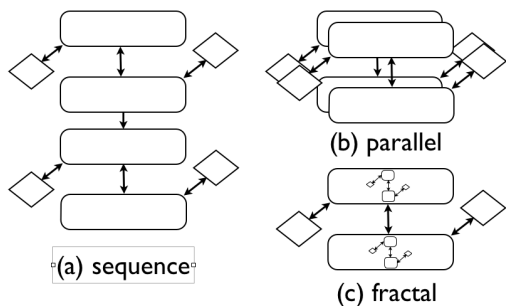


Figure 3. Problem solving can be performed (a) in sequence, and (b) in parallel. Under POE, problem and solution exploration are problems solving activities, so that *fractal* composition is also possible.

Briefly, in sequence, the POE process pattern models (more or less traditional) design processes in which existing structures, such as architectures, are used as structure in the solution space according to significant requirement and qualities, and according to developmental requirements.

If resources exist, parallel problem solving is possible. Of course, communications between those involved in parallel development is a non-trivial issue; we do not consider it here, though.

Fractal-like Design: [5] explains in detail how problem and solution exploration can be seen as problem solving processes: although we do not go into detail, essentially, the problem to be solved by problem exploration is to find the problem (or solution) description that satisfies the validating stake-holder. This allows us to embed within problem and solution exploration copies of the PPP, making the process self-similar in the sense that the whole structure resembles the parts it is made of.

Trusted processes: A POE process is *trusted* when there are no validating stake-holders. As argued in [5], trusted processes exist to ‘bottom out’ problem solving; given that no validator is involved they do not have a fractal form.

D. Validation

We wish to model the validation relationships that occur in organisations. In POE, we assume that there are two types of validation relationship: (i) *direct validation*, in which one actor determines whether the problem to solve has been understood or the solution is adequate; (ii) *delegated validation*, in which one actor – the delegator – delegates to one (or more) actors who then can validate either directly or through delegation. Of course, delegation does not transfer

the responsibility for the outcomes of a choice, so that at a future point the delegator may check that the choice has been made, and also that the outcome and justification of that choice is suitable.

A delegator can clearly not usefully delegate to themselves, nor can there be cycles in the delegator/delegatee relationship. The reflexive transitive closure of the relationship is, therefore, a partial order.

Thus, POE see validation as a *social choice* [3], which can be expressed as a *partially ordered set*, or *poset*, i.e., $(X, <)$, where X is the domain set and symbol $<$ is the delegation relation, which we call the *validation structure*, or $VStruct$.

Example 1: Consider three validators – v_1 , v_2 and v_3 – with v_i delegates to v_j , $v_i < v_j$, whenever $i > j$. The validation structure is the poset $(V, <)$ with:

$$V = \{v_1, v_2, v_3\}, \quad (v_1 < v_2), (v_2 < v_3)$$

We now turn to the modelling of business processes in POE.

III. POE PROCESS ALGEBRA

The aim is to define a process algebra to describe POE processes in terms of simple operations. These operations should allow for encoding and modelling of distinct exploration and validation activities. Firstly, they should provide compositions for the three basic ways in which POE processes combine. Secondly, the operations should be sufficient to express arbitrarily complex process models, whose structure and behaviour could be modelled and reasoned about.

To this end, we define the POE Process Algebra (PPA).

A process under PPA is formed under the following syntax:

$$P = \beta \mid P; P \mid P \parallel P \mid P \underset{V}{\bowtie} P$$

in which a basic POE Process, β , combines to produce sequence $P; P$, parallel $P \parallel P$, and fractal processes $P \underset{V}{\bowtie} P$, the latter in combination with validation structures².

The operators correspond to the possibilities for combination of POE Processes as described in Section II.

A. Executable semantics of PPA

We provide an executable semantics for PPA terms using machine readable CSP (CSP-M) [4], as implemented in *ProB* tool [10]. *ProB* was chosen as it provides for the animation and model-checking of the resulting CSP models and traces [8] on the trusted processes. A CSP semantics allows us to reason about processes in terms of the effects of basic process on resource, and on their relationship to validation, through fractal composition.

²The astute reader will note the lack of a choice composition, often included in process algebras that express computation. This is because the PPP describes the structure of the problem solving process and not the (creative) choices that are expressed therein: POE makes no comment on the creativity of the design process, that is contextually determined by the stake-holders whose notions of adequate apply.

We have already been able to use *ProB*'s model checking and animation functionality to test whether POE processes (encoded as POE Programs) execute and complete as expected. Examples follow.

B. POE Programs language syntax

We designed a language for encoding POE processes using the operators from PPA, and as a function over POE Programs in this language we created a set of tools to translate PPA encoded input to CSP-M.

The executable model for POE processes was implemented in the Ruby programming language [12]. The input POE Program is translated to CSP-M output, and this in turn can be directly used in *ProB* tool. Further technical details of the implementation are beyond the scope of this paper.

C. A CSP-M semantics of PPA

Our semantics is over the domain of CSP-M expressions and so we must associate with each POE Process expression a CSP-M term. Most choices for the semantics are made simple due to the process algebraic nature of the source and target languages: CSP is an algebra as is PPA. Below, we describe in detail only the more difficult encodings.

PPA trusted in CSP: Trusted POE processes are the building block from which others are built. Their defining characteristic is that they make no use of validators. As such they can be, essentially, any piece of CSP-M code.

Sequence: Sequential composition under PPA is, as might be expected, implemented using CSP's sequential operator ;. Simply:

```
Sequence(Left,Right) = Left ; Right
```

Parallel: Although there are other choices, in this initial semantics parallel composition under PPA is implemented using CSP's interleaving operator. Other choices would allow the processes involved to communicate with each other to, for instance, model the passing of documentation between them. Such details are left for a fuller description.

```
Parallel(Left,Right) = Left ||| Right
```

PPA fractal in CSP: Referring to Figure 3, fractal composition in our CSP-M implementation must accept four arguments: *Upper* for the problem exploration process, *Lower* for solution exploration process, and the two respective validation structures, which we will call *VSPID* (for Validation Struct for Problem) and *VSSID* (for Validation Struct for Solution). Our semantics simply places the CSP-M semantics of the operands together through CSP-M's interleaving operator.

```
Fractal(Upper,Lower,VSPID,VSSID) = (Upper |||
    ValidateUpper(VSPID) ||| Lower |||
    ValidateLower(VSSID))
```

Table I
SUMMARY OF KEY TERMS

Term	Description
Mortgage Servicing	Managing mortgage loans – interest accruals, billing, collecting due payments, redemption of loans, etc.
Financial Services Authority (FSA)	Regulatory body for financial institutions
(Mortgage) Servicing Software	Software used in servicing activities
Triage Document	A form used to report details of a production issue

IV. CASE STUDY EXAMPLE

As an example of the application of our algebra, we model the process described in [14], from which the following description is adapted.

The context of the study is a UK based subsidiary of an American financial organisation (the Company), with business, systems and technical analysts based in the UK, technical architects in the US and development staff in India. Relevant key terms and stake-holders are summarised in Tables I and II. The company supplies Mortgage Servicing Software package to its Client, a product that manages loan accounts once mortgage payments have been made by the Client's customers. The software facilitates business tasks such as payment calculation and processing, account queries, early redemption, correspondence, interest rate change and customer billing. The company also provides support and assists in the resolution of issues that arise during the use of the supplied software.

Recently, the company lost a number of subject matter experts but retains a contractual obligation to provide support to the Client to enhance and maintain the supplied software stack. This motivated the company to investigate through this study the capture of design rationale during Client's issue resolution. As many organisations face such losses, the success of our case study takes on increased importance. The case study also provided opportunities to consider how the application of POE techniques could improve the current issue resolution process. Process improvement is also an issue faced by many organisations.

A. Issue Resolution Process

When an issue is found in the Client's use of the Company's applications, a *Triage Document* is raised to describe the problem with information included that may assist in tracking down its cause. The reported issue is given a priority by the Client (*low, medium, high*) that governs the timeline for response and solutions, based on service-level agreements. Once the *Triage Document* is received by the Supplier's Production Support Team, an incident number is generated

Table II
SUMMARY OF KEY STAKE-HOLDERS

Stake-holder	Description
Client/Financial Institution	The mortgage institution managing customer mortgages
Customers	Patrons of the client organisation whose mortgages are being managed
(Servicing) Software Company	Company that supplies and maintains mortgage servicing software for the client. The case study is based in this organisation
Client Production Support Team	Client-side team tasked with resolving application software issues. Understand the workings of the application system and its platform, provide initial information on issues and communicate with business decision makers when questions arise of a business policy nature
Supplier Production Support Team (SPST)	Supplier-side team tasked with resolving issues with application software. In contact with CPST, assign work to the development and manage releases of solutions to the Client
Application Architect (AA)	Reviews a solution to assess if solution complies with standards
Product Assurance Team	Ensures the quality of the provided solution
Mort. Bus. Man. IT Management	Make final sign-off decision Make decision whether to implement solution in production
(Offshore) Development Team (DT)	Group of individual on the Supplier side tasked with developing software

and used to track the issue. The information is checked to see if it is sufficient for the investigation to progress.

Further discussions may be held between the Client and Supplier Production Support Teams to agree a) which issues lie with the application software and b) an approach for dealing with the issue. Additional clarification may be sought from the Client from which the issue report originated. The clarification may be in the form of screen shots of the application error, data extracts, event logs and example scenarios. When issues are agreed between CPST and SPST, they are analysed and solution approaches proposed by the development and architecture resources assigned to the issue. The proposed solutions are discussed with the CPST. Once agreement is reached on a solution approach, it is developed and tested. On completion of development and testing, the solution is packaged by the SPST with release notes and a test report, and delivered to the CPST. Subsequently, the CPST validate the delivered package, perform some further tests in collaboration with the Client, and may either return it for rework if it is unsatisfactory or implement it to the production systems if satisfied with the results.

B. The PPA Model

From this description, we identify four trusted processes, Report, Raise, Analyse and Deliver, and three fractal instances of Figure 2, here named F1, F2 and F3, with associated validation structures U; V1 and V2; W1 and W2. We

note that only the problem exploration part of F1 is validated. The process is illustrated in Figure 4.

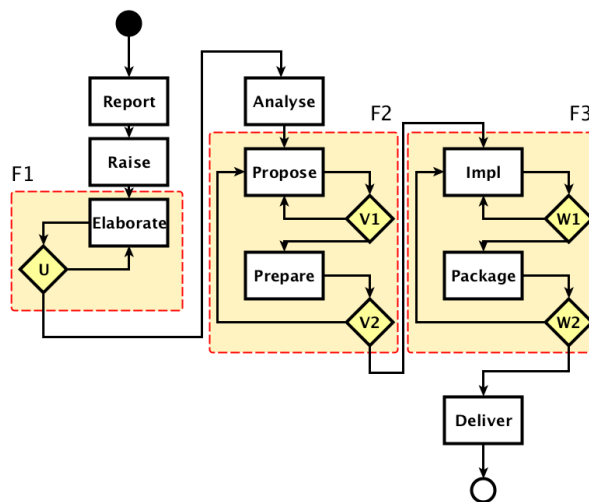


Figure 4. Processes; adapted from [14]

```

LET HiddenPV := (HiddenValidator)
LET U := (SPST1<CPST1<Client1)
LET V1 := (SPST2<AA1)
LET V2 := (CPST2<Client2)
LET W1 := (PA1)
LET W2 := (Client3<CPST3)

F1 := (ElaborateP{HiddenPV}>>{U}ElaborateS)
F2 := (Propose{V1}>>{V2}Prepare)
F3 := (Impl{W1}>>{W2}Package)
P := Report;Raise;F1;Analyse;F2;F3;Deliver
    
```

Our tool generates approximately 180 lines of CSP-M to model the case study process of Figure 4. The part of the process corresponding to the level of that figure is:

```

P = Sequence(Sequence(Sequence(Sequence
    (Sequence(Sequence(T(pReport),
    T(pRaise)), Fractal(T(pElaborateP),
    T(pElaborateS), vsHiddenPV,vsU)),
    T(pAnalyse)), Fractal(T(pPropose),
    T(pPrepare), vsV1, vsV2)),
    Fractal(T(pImpl), T(pPackage),
    vsW1,vsW2)), T(pDeliver))
    
```

V. CONCLUSIONS AND FUTURE WORK

We have described our current model of Hall and Rapanotti’s POE Process Pattern. The current model includes initial executable semantics of all aspects of the process, including fractal composition from which call-outs to validating stakeholders take place. Our model is encoded in a new process algebra, the POE Process Algebra (PPA), introduced here, and we have defined a semantic function over the algebra, which maps from CSP-M model that can be analysed in the ProB tool.

We have illustrated our PPA encoding on a business process that has appeared in the literature, presented a partial

CSP-M semantics of it as well as the ProB output of a partial exploration of its state space.

Our approach contributes to business process modelling by explicit recognition of choices that can be made by human agents, and with a devised language for representing POE Programs in the form of CSP-M, this approach provides syntax and semantics for behaviour modelling of business and validation processes in general.

Work that remains will consider how the executable model can be used to calculate resource usage of the process, and how risk and resources trade-off under, what we have termed, the risk/resource trade-off assumption of Section II. For, with such calculations, comes the possibility of systematic business process design in POE.

REFERENCES

- [1] W.M.P. van der Aalst, "Making Work Flow: On the Application of Petri Nets to Business Process Management." LNCS, J. Esparza and C. Lakos, Eds. Springer, 2002, vol. 2360, pp. 1–22.
- [2] R.S. Aguilar-Savén, "Business process modelling: Review and framework," *International Journal of Production Economics*, vol. 90, no. 2, pp. 129 – 149, 2004.
- [3] G. Brightwell and D. B. West, "Chapter 11: Partially ordered sets," in *Handbook of Discrete and Combinatorial Mathematics*, K. H. Rosen, J. G. Michaels, J. L. Gross, J. W. Grossman, and D. R. Shier, Eds. CRC Press, 2000.
- [4] M. Butler and M. Leuschel, "Combining CSP and B for Specification and Property Verification," in *FM 2005: Formal Methods International Symposium of Formal Methods Europe, Newcastle, UK, July 18-22, 2005. Proceedings*, LNCS, vol. 3582. Springer, 2005, pp. 221–236.
- [5] J.G. Hall and L. Rapanotti, "Assurance-driven design in Problem Oriented Engineering," *International Journal On Advances in Systems and Measurements*, vol. 2, no. 1, pp. 119–130, 2009.
- [6] J.G. Hall, L. Rapanotti, and M. Jackson, "Problem oriented software engineering: A design-theoretic framework for software engineering," in *Proceedings of 5th IEEE International Conference on Software Engineering and Formal Methods*. IEEE Computer Society Press, 2007, pp. 15–24.
- [7] J.G. Hall, L. Rapanotti, and M. Jackson, "Problem-oriented software engineering: solving the package router control problem," *IEEE Trans. Software Eng.*, 2008.
- [8] C.A.R. Hoare, *Communicating Sequential Processes*, ser. Series in Computer Science. Prentice-Hall International, 1985.
- [9] B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler, "On structured workflow modelling," in *CAiSE*, LNCS, B. Wangler and L. Bergman, Eds. Springer, 2000, vol. 1789, pp. 431–445.
- [10] M. Leuschel and M. Fontaine, "Probing the Depths of CSP-M: A New FDR-Compliant Validation Tool," in *ICFEM '08: Proceedings of the 10th International Conference on Formal Methods and Software Engineering, Kitakyushu-City, Japan*. Springer-Verlag, 2008, pp. 278–297.
- [11] A. Macintosh, "The need for enriched knowledge representation for enterprise modelling," in *AI (Artificial Intelligence) in Enterprise Modelling, IEE Colloquium on (Digest No.078)*, 7 1993, pp. 3/1 –3/3.
- [12] J. McAnally and A. Arkin, *Ruby in practice*. Manning Publications Co. Greenwich, CT, USA, 2008.
- [13] G. Mentzas, C. Halaris, and S. Kavadias, "Modelling business processes with workflow systems: an evaluation of alternative approaches," *International Journal of Information Management*, vol. 21, no. 2, pp. 123 – 135, 2001.
- [14] A. Nkwocha, J.G. Hall, and L. Rapanotti, "Design rationale capture in the globalised enterprise: An industrial study," in *Proceedings of Fifth International Conference on Software Engineering Advances (ICSEA 2010)*. IEEE, 2010, electronic proceedings.
- [15] G.F.C. Rogers, *The Nature of Engineering: A Philosophy of Technology*. Palgrave Macmillan, 1983.
- [16] M.E. Szabo, Ed., *Gentzen, G.: The Collected Papers of Gerhard Gentzen*. Amsterdam, Netherlands: North-Holland, 1969.
- [17] W. Wang, H. Ding, J. Dong, and C. Ren, "A comparison of business process modeling methods," in *Service Operations and Logistics, and Informatics, 2006. SOLI '06. IEEE International Conference on*, 21-23 2006, pp. 1136–1141.
- [18] S.A. White, "Introduction to BPMN," *IBM Corporation*, May 2004.

Optimal Functionality and Domain Data Clustering based on Latent Dirichlet Allocation

Stoyan Garbatov and João Cachopo

Software Engineering Group

Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento, INESC-id

Lisbon, Portugal

stoyangarbatov@gmail.com and joao.cachopo@ist.utl.pt

Abstract — This work presents a new approach for clustering domain data and application functionality, based on the Latent Dirichlet Allocation. The methodology, developed here, performs an optimal clustering by identifying input values that lead to the best possible clustering output. The optimal solutions are identified through the use of the Silhouette technique. A validation of the work is performed based on the TPC-W benchmark. The new approach is flexible enough to be applied to any object-oriented application where identifying meaningful clusters of its domain data and functionality is desired.

Keywords-clustering; Latent Dirichlet Allocation; stochastic model; Silhouette.

I. INTRODUCTION

The problem of clustering has been considered and analysed in many different disciplines' contexts, illustrating its relevance and usefulness in a variety of circumstances.

Clustering corresponds to an unsupervised classification of patterns (data, observations, etc) into sets or groups (clusters). Clustering algorithms organize pattern aggregates based on similarity criteria according to which these may be classified.

A pertinent situation requiring clustering would be in the context of large-scale object-oriented applications (e.g dynamic content web applications). There, it can be interesting to identify meaningful subsets of application functionality that display high affinity with regards to the domain data that is manipulated within their scope. If such information is available, then it may be feasible to carry out techniques such as load balancing or partial data replication to improve the application performance and scalability.

Based on what can be seen from recent research, there has been some effort spent in this area, but the great majority of these approaches display only partial automation in their mode of operation. Many approaches require user intervention at one, if not more, points of the analysis procedure, making the approaches more prone to errors and leading to non-optimal results, due to the subjectivity induced by the user interaction in the decision making process.

In contrast to these supervised approaches, we believe that a wholly automatic approach would lead to better results, by avoiding the problems identified above. This

paper describes the development and validation of a fully automated system capable of identifying the domain data manipulated during the execution of the target system's functionality and, based on that information, of performing optimal partitioning of the application's methods/services according to the domain data used within their runtime scopes. The partitioning of the application's functionality (represented by its services and/or methods) is performed by employing the Latent Dirichlet Allocation [1]. The optimality of the solutions is guaranteed through the use of the Silhouette technique, [2].

The article has the following structure. The related works are discussed in Section II. The description of the system is covered in Section III. The results and evaluation of the system are given in Section IV. The concluding remarks are presented in Section V.

II. RELATED WORK

Based on the nature of the work presented here, it is possible to identify two related research areas. The first one covers the development and analysis of clustering algorithms, whilst the second one encompasses works seeking to develop performance improvement techniques in the context of dynamic content web applications.

It was not possible to find any work that takes an at least comparable approach for the problem at hand. As such, the discussion of works strictly related to clustering algorithms will be restricted to the relevant references that are present in the system description.

It is important to discuss some of what has been done in the context of dynamic content web applications [3-8], so as to better appreciate the contribution of the current work. A rather thorough study and comparison of load balancing and scheduling strategies, for the type of applications identified above, can be seen in the work of Amza et al. [9].

The work of Elnikety et al. [7] introduced a memory-aware load balancing method for dispatching transactions to replicas in systems employing replicated databases. The algorithm uses information about the data manipulated in transactional contexts with the goal of assigning transactions to replicas so as to guarantee that all necessary data for their execution is in memory, thereby reducing disk I/O. For guiding the load balancing technique, the authors developed an auxiliary approach for estimating the volume and type of data manipulated during transactions. An additional

contribution of their work is an optimization designated *update filtering* for decreasing the overheads due to the propagation of updates between replicas.

The work of Amza et al. [5] presents a novel lazy replication technique, intended for scaling database backends of dynamic content site applications operating on top of computer clusters. The approach developed by Amza et al. is referred to as conflict-aware scheduling and provides throughput scaling and one-copy serializability. This technique exploits the fact that, in the context of database clusters, there is a scheduler responsible for processing all incoming requests. By making use of information regarding the domain data accessed within transactions, Amza et al. [3] developed a conflict-aware scheduler that provides one-copy serializability, as well as reducing the rate at which conflicts occur. This is achieved by guiding incoming requests to nodes based on the data access patterns that are expected to be performed during the execution of the associated transactions.

Gao et al. [6] developed an edge service replication architecture for e-commerce applications using application specific distributed objects. The authors exploit application specific behaviour to manage subsets of shared domain data through distributed objects. Higher system availability and efficiency is achieved by tolerating lower consistency among distributed objects.

Shen et al. [4] performed an analysis over the clustering of replicated services with high ratios of write operations. With their work, the authors developed an infrastructural middleware called Neptune, which allows the agglomeration and replication of a system's service modules. The middleware supports multiple alternative persistence mechanisms and is capable of maintaining consistency dynamically, independently of the location and availability of a particular replica.

Zuikėviciute and Pedone present in [8] a hybrid approach for conflict-aware load balancing for systems with database replication. The authors analyzed the effects of the often opposing requirements (from an engineering point of view) of maximizing transaction parallelism and minimizing conflict ratios. The work led to the development of a load balancing technique that finds a good compromise between parallelism and conflict minimization, accomplishing better results than approaches concentrating solely on one of the above requirements.

As can be seen from the above works, there are indeed very promising results for improving the performance and scalability of large scale applications, through the use of load balancing, replication techniques, adaptive scheduling, and other related approaches. Yet, there is still significant room for improving the full automation of existing solutions, both at the level of analyzing the behavior of target applications, as well as in the identification of meaningful functionality and domain data subsets on which the approaches are to be applied.

Thus, we believe that a system capable of performing a completely automated analysis of a target application's behavior (with regard to domain data manipulations performed in runtime), and of performing an optimal

clustering of the application's functionality and domain data (through the use of the current state-of-the-art multivariate clustering algorithm), would constitute an important contribution within this research area.

III. SYSTEM DESCRIPTION

The system developed with this work is composed of two parts: a data acquisition and analysis module and an optimizing clustering module. The first module is responsible for capturing the target application behaviour, for analysing it, and for generating predictions about what are the most likely domain data types to be needed by the application when it is in a specific execution context (e.g., method, service, etc). The full description of the implementation, functionality, and properties of this module has already been presented and discussed in detail in [10-12]. The prediction functionality is of no relevance for the work presented here. The key aspects of this module are that it provides the input necessary for the optimal clustering module, and that the data collection task performed is done with relatively low overheads (an average of 5-8% overheads in comparison with the original version of the target application performance), in an online fashion. Moreover, all modifications necessary for the acquisition of the behavioural data are performed in a completely automated manner by the system presented here.

The second module is responsible for identifying the optimal clustering of the target application's functionality and domain data, based on the data access pattern behaviour observed in runtime. For the clustering itself, we use the Latent Dirichlet Allocation algorithm, while the optimal clustering solution is guaranteed through the use of the Silhouette technique. Both of them shall be discussed in detail in the following subsections.

A. Latent Dirichlet Allocation

The data acquisition module is responsible for supplying the clustering module with the observed target application conduct. This corresponds to the application's domain data access behaviour, and is expressed in terms of the frequencies of the domain object manipulation operations observed when executing application *functionality*. For simplicity, the abstraction capturing this functionality shall be referred to as the *methods* of the application, but any other appropriate concept can be used instead (e.g., functions, services, etc).

When supplied with this input, the clustering module employs the Latent Dirichlet Allocation (LDA) algorithm, generating a probabilistic description of the contents of the clusters.

The decision of using LDA as the clustering algorithm was based on several factors. The first of these is the fact that LDA corresponds to the current state of the art in terms of clustering algorithms. Additionally, LDA consists in a three-level hierarchical Bayesian model. This shall be discussed in greater detail further on, but suffice it to say that LDA provides semantically richer results than other alternative methods, making it thus more useful for the purpose of the work presented here.

For this work, the contents of the clusters correspond to application methods that are strongly correlated in terms of the domain data manipulated within their runtime scopes. As such, the LDA will seek to populate the clusters in such a way as to maximize the intra-cluster similarity and minimize the inter-cluster similarity. This similarity is, once again, expressed in terms of the domain data used in the methods being clustered. It should be noted that the LDA, being a multivariate clustering model, provides a secondary result. This secondary result consists of a clustering of the application domain data. The cluster identities are the same as the ones for the application methods, with the difference that they are characterized by a stochastic description built-in function of the predominant domain data present in them.

The LDA does not estimate the optimal number of clusters that are to be found in the set of methods composing the target application. The number of clusters is supplied as input to the algorithm. The procedure for identifying the optimal value for the number of clusters shall be discussed at length in section III.B. In the remaining of this section, the theoretic bases of the LDA shall be considered.

Latent Dirichlet Allocation was developed and first presented by Blei et al. [1]. LDA can be generally described as a generative probabilistic model for collections of discrete data. In the probability analysis, a generative model corresponds to a model that can generate randomly observable data, based on some hidden parameters. The generative model specifies a joint probability distribution over observation and label sequences. Keeping this into account, LDA is a three-level hierarchical Bayesian model. The hierarchical Bayesian model corresponds to an elaborate model in modern Bayesian Analysis and allows the modelling of complex situations in a better way than simpler models.

Given data x and parameters \mathcal{G} , a simple Bayesian analysis starts with a prior probability (prior) $p(\mathcal{G})$ and likelihood $p(x|\mathcal{G})$ to compute a posterior probability:

$$p(\mathcal{G}|x) \propto p(x|\mathcal{G})p(\mathcal{G}) \quad (1)$$

The prior on \mathcal{G} depends, in turn, on other parameters φ that are not mentioned in the likelihood. So, the prior $p(\mathcal{G})$ must be replaced by a prior $p(\mathcal{G}|\varphi)$, and a prior $p(\varphi)$ on the newly introduced parameters φ is required, resulting in a posterior probability:

$$p(\mathcal{G},\varphi|x) \propto p(x|\mathcal{G})p(\mathcal{G}|\varphi)p(\varphi) \quad (2)$$

This procedure may be performed repeatedly, if any of the parameters employed up until now depends on additional parameters, requiring its own priors. The process terminates when priors that do not depend on any further unmentioned parameters have been reached.

The latent multinomial variables shall be referred to as clusters. The latent multinomial variables can be associated without any issue to different concepts.

In the LDA model, each collection item (e.g., method) is modelled as a finite random mixture over an underlying set of clusters. The clusters are modelled as an infinite mixture over a set of underlying cluster probabilities and are characterized by a distribution over domain data types. From the point of view of domain modelling, the cluster probabilities consist in an explicit representation of a method. The approximation inference techniques employed for LDA are based on variational methods and an estimation maximization algorithm for empirical Bayes parameter estimation.

LDA assumes the following generative process for each method m in an application A :

1. Choose $N \sim \text{Poisson}(\xi)$.
2. Choose $\theta \sim \text{Dir}(\alpha)$.
3. For each of the N data types m_n :
 - (a) Choose a cluster $z_n \sim \text{Multinomial}(\theta)$.
 - (b) Choose a data type m_n from $p(m_n|z_n,\beta)$, a multinomial probability conditioned on the cluster z_n .

There are a few simplifying assumptions made in this model, among which is that the dimensionality k of the Dirichlet distribution (and the dimensionality of the cluster variable z) is assumed known and fixed. The Poisson assumption is not crucial for any part of the model and other more appropriate method length distribution may be employed if deemed necessary.

A k -dimensional Dirichlet random variable θ can take values in the $(k-1)$ -simplex (a k -vector θ lies in the $(k-1)$ -simplex if $\theta_i \geq 0, \sum_{i=1}^k \theta_i = 1$), and has the following probability density on this simplex:

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^k \alpha_i)}{\prod_{i=1}^k \Gamma(\alpha_i)} \theta_1^{\alpha_1-1} \dots \theta_k^{\alpha_k-1} \quad (3)$$

where the parameter α is a k -vector with components $\alpha_i > 0$, and where $\Gamma(x)$ is the Gamma function. The Dirichlet distribution, as a distribution on the simplex, has several useful properties that make it easier to develop algorithms for inferring and estimating parameters for the LDA. The Dirichlet distribution belongs to the exponential family; it has finite sufficient dimensional statistics and is conjugate to the multinomial distribution.

Given the parameters α and β , the joint distribution of a cluster mixture θ , a set of N clusters z , and a set of N data types m is given by:

$$p(\theta, z, m|\alpha, \beta) = p(\theta|\alpha) \prod_{n=1}^N p(z_n|\theta) p(m_n|z_n, \beta) \quad (4)$$

where $p(z_n|\theta)$ is simply θ_i for the unique i such that $z_n^i = 1$. By integrating over θ and summing over z , the marginal distribution of a method is obtained:

$$p(m|\alpha,\beta) = \int p(\theta|\alpha) \left(\prod_{n=1}^N \sum_{z_n} p(z_n|\theta) p(m_n|z_n,\beta) \right) d\theta \quad (5)$$

Finally, taking the product of the marginal probabilities of single methods, the probability of the set of application methods is obtained:

$$p(D|\alpha,\beta) = \prod_{d=1}^M \int p(\theta_d|\alpha) \left(\prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn}|\theta_d) p(m_{dn}|z_{dn},\beta) \right) d\theta_d \quad (6)$$

A graphical representation of the probabilistic model of LDA can be observed in Fig. 1. As can be seen, the LDA representation has three levels. The parameters α and β are application-level parameters and are sampled once in the process of generating an application. The variables θ_d correspond to method-level variables, which are sampled once per method. Lastly, the variables z_{dn} and m_{dn} are variables at the domain data-level. These are sampled once for each domain datum per method.

With regards to the actual implementation employed for the work presented here, it is a Java port of the original LDA implementation presented by Blei et al. [1], with no modifications or extensions performed over the model itself here. The contribution of this work, regarding the use of LDA, resides in the new semantic interpretation given to the model and its associated concepts. This made possible the use of the LDA algorithm, for the first time, to the best of our knowledge, to perform clustering of an object-oriented application's functionality, based on the domain data manipulated within its scope.

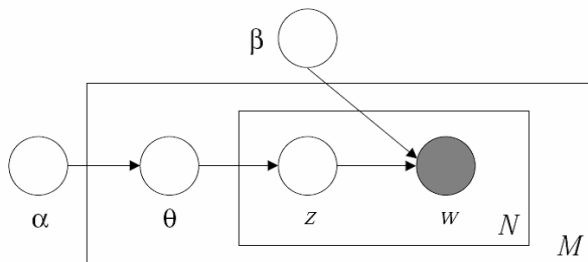


Figure 1. Graphical model representation of LDA

B. Optimal Clustering Solution

As indicated in the LDA model description, the algorithm does take some additional input parameters, apart from the occurrence frequencies of the data being modelled. These parameters are the number of clusters among which the data is to be split and the α coefficient value, which is also known as the Dirichlet parameter. The Dirichlet parameter controls the shape of the Dirichlet distribution and, subsequently, the likelihood of a given cluster being selected during the algorithm execution. In practice, high alpha values (close to 1) lead to many clusters being associated to each method, whereas a low value makes it so that few clusters are associated to each method.

As has been previously stated, what we intend with this work is an optimal clustering solution. This makes it necessary to find the additional input parameters' values that lead to the best clustering solutions. To evaluate the effects of the parameter values, we resorted to a well-known and recognized clustering model comparison technique. The technique is known as Silhouette, as reported by Rousseeuw [2]. Intuitively, good clusters have the property that cluster elements are close to each other and far from the elements of other clusters. The Silhouette technique captures this notion and provides an indicator value of how good a particular clustering is.

The Silhouette approach functions as follows. For each data element i , let $a(i)$ be the average dissimilarity between i and all other elements belonging to the same cluster. The approach is independent of the dissimilarity criteria, allowing any appropriate measure to be employed. The value of $a(i)$ can be considered as a measure of how well the element i is matched to the cluster. The smaller the value of $a(i)$, the better the matching is.

Afterwards, for every cluster where i does not belong, an average measure of dissimilarity is calculated, between the data elements of the cluster and i . The minimum of these dissimilarity measures is denoted by $b(i)$. The cluster to which $b(i)$ is associated with is called the "neighbouring cluster" of i , because it is the second best cluster where i could be placed, from among all available clusters. Based on this, $s(i)$ can be defined as:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \quad (7)$$

where $-1 \leq s(i) \leq 1$. When $s(i)$ is close to 1, this means that the datum i is properly clustered. When $s(i)$ is close to -1, the interpretation is that i would have been better placed in its neighbour, instead of the cluster where it is currently placed. If $s(i)$ is close to 0, then it means that the datum is

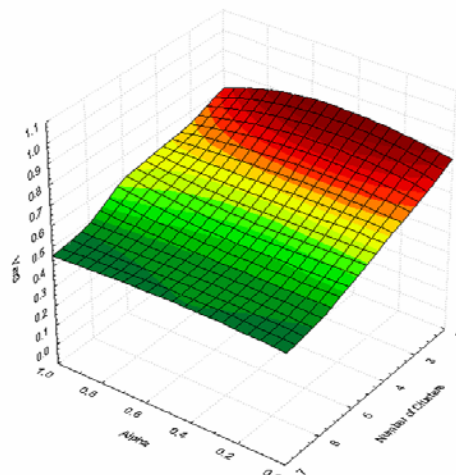


Figure 2. Silhouette coefficient values

placed somewhere "in between" the two clusters.

The average $s(i)$ of all the data placed in a cluster is an indicator of how tightly grouped all cluster data is. The average $s(i)$ for all clusters is a measure of how properly the data has been clustered.

To find the optimal values of the input parameters for the LDA, our system calculates the average $s(i)$ from several executions of the LDA algorithm for every combination of input parameters, within their valid range of values. Once the $s(i)$ coefficients are available for all the evaluated scenarios, the pair of input values which produced the closest to 1 $s(i)$ corresponds to the optimal input scenario that leads to the best possible clustering.

Regarding the similarity measure employed to calculate the $s(i)$, it is based on the gamma values generated by the LDA itself. The gamma values indicate the affinity between the data and the clusters where the data is placed. These affinity coefficients are normalized so that the sum of their values equals 1 for a given method. The dissimilarity measure $a(i)$ of a given application method is set to 1 minus the normalized gamma value for the associated best cluster, whereas the $b(i)$ is set to 1 minus the normalized gamma of the second best cluster.

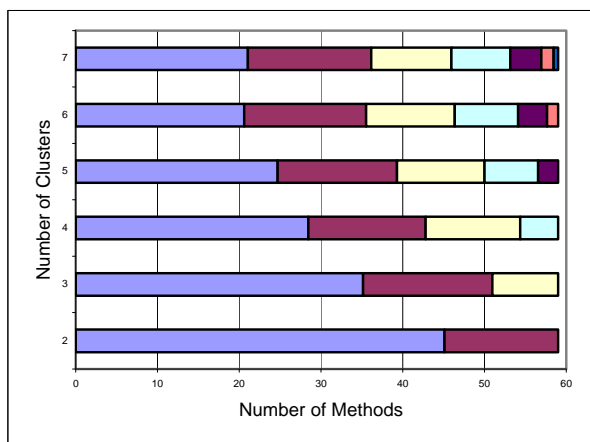


Figure 3. Distribution of average number of methods per cluster

IV. RESULTS AND EVALUATION OF THE SYSTEM

The TPC-W benchmark was selected to serve as a test-case for the demonstration of the new approach presented in this paper. The TPC-W benchmark was introduced by Smith [13]. This benchmark specifies an e-commerce workload that simulates the activities of a retail store website, where emulated users can browse and order products from the website. This particular benchmark was chosen for two main reasons. First of all, it has a reasonably rich application domain model and functionality. Secondly, due to the fact that the benchmark fits well with the type of applications that are most likely to benefit from optimizations that employ the results generated by the system developed with the current work. As previously stated, such optimizations would

include dynamic load balancing schemes, conflict-aware approaches for partial or full data replication approaches, among others.

The Silhouette coefficients achieved for the evaluated range of values for the input parameters of the LDA algorithm, when applied to the methods and domain data accessed within them, for the TPC-W benchmark, can be seen in Fig. 2.

The z axis represents Silhouette coefficients, where the valid range of values is [-1,1]. Every point of the surface plotted in Fig. 1 corresponds to an average calculated from 20 independent LDA executions with the same combination of input values. This was done in order to have representative results of the non-deterministic behaviour of the LDA model.

As can be seen from Fig. 2, the input parameter controlling the number of clusters has been varied from 2 to 7, whilst the alpha parameter was varied from 0.01 to 1. Even though there are 59 benchmark methods within which domain data accesses take place, the number of clusters has been varied only up to 7 because, even though the LDA algorithm takes as input the maximum number of clusters among which the methods are to be partitioned, the algorithm decides by itself what is the optimal solution, within the possibilities given by its actual input parameters. Consequently, it is possible for the effective clustering result to consist in a solution where only a portion of the maximum number of clusters have been allocated any elements. This is an increasingly frequent occurrence as the maximum number of clusters increases, and, to a smaller degree, for the lower possible limit of clusters as well.

By analysing the results depicted in Fig. 2, we may conclude that, with regards to the maximum number of clusters, the best Silhouette coefficients (closest to 1) are those associated to 2 and 3 clusters. Regarding the optimal alpha values, even though they do not seem to exert a significant influence over the Silhouette coefficients, the best results are achieved when alpha is in the range of]0.4, 0.5[. The sensitivity analysis study performed by Park in [14] reached the same conclusion, with regards to the effect of the optimized alpha value on the general quality of the clustering results.

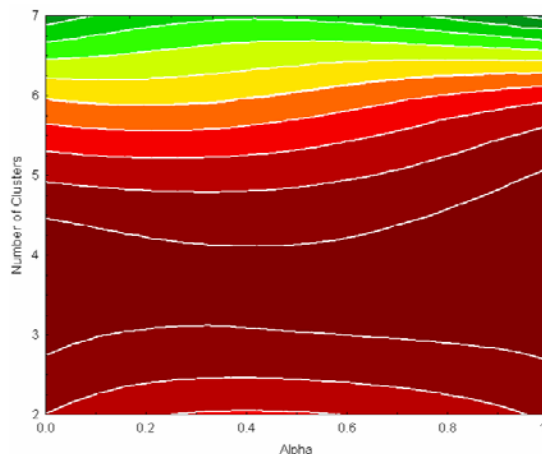


Figure 4. Effective clustering

The most commonly observed number of methods per cluster can be seen in Fig. 3. Each histogram bar was calculated as the average of 20 independent LDA executions, for the same total number of clusters. The x axis represents the number of methods present in a given cluster, while the y axis indicates the total number of clusters among which the data has been partitioned. These results show that, as the total number of clusters increases, the "new" clusters tend to be very small. This is an indicator that a lower total number of clusters is more appropriate, where the number of methods per cluster is more balanced.

A summary of the effective ratio of non-empty clusters can be seen in Fig. 4. The chart represents a 2D projection of the tri-dimensional surface describing the dependency between effective cluster number ratio and the LDA control parameters. The ratio has been calculated as the number of non-empty clusters divided by the maximum number of clusters supplied as input. The highest clustering ratios are achieved for 3 to 4 clusters and alpha values in the range of]0.4, 0.5[.

Combining the results of Silhouette coefficients with the effective cluster number ratio, we can deduce that the input parameter values that most consistently lead to the best clustering results are alpha in the]0.4, 0.5[range and a total of 3 clusters.

V. CONCLUSIONS

This work presented an innovative approach for clustering domain data and application functionality. The algorithm employed is the current state of the art multivariate Latent Dirichlet Allocation. The methodology performs an optimal clustering by fitting input control parameters so as to achieve the best possible clustering result. The optimal solutions are identified through the use of the Silhouette technique. A demonstration of system's capabilities is done based on the TPC-W benchmark. The approach is flexible enough to be applied to any object-oriented application where identifying meaningful clusters of its domain data and functionality is desired.

ACKNOWLEDGMENT

This work was partially supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and by the Specific Targeted Research Project (STReP) Cloud-TM, which is co-financed by the European Commission through the contract no. 257784. The first author has been funded by the Portuguese FCT (Fundação

para a Ciência e a Tecnologia) under contract SFRH/BD/64379/2009.

REFERENCES

- [1] Blei, D. M., Ng, A. Y. and Jordan, M. I., 2003, Latent dirichlet allocation, *Journal of Machine Learning Research*, 3, pp. 993-1022.
- [2] Rousseeuw, P. J., 1987, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, *Journal of computational and applied mathematics*, 20, pp. 53-65.
- [3] Challenger, J., Iyengar, A., Witting, K., Ferstat, C. and Reed, P., 2000, A publishing system for efficiently creating dynamic web content, *IEEE*, Vol. 2, pp. 844-853 vol. 842.
- [4] Shen, K., Yang, T., Chu, L., Holliday, J. A. L., Kuschner, D. A. and Zhu, H., 2001, Neptune: Scalable replication management and programming support for cluster-based network services, *USENIX Association*, pp. 17-29.
- [5] Amza, C., Cox, A. L. and Zwaenepoel, W., 2003, Conflict-aware scheduling for dynamic content applications, *USENIX Association*, pp. 6-20.
- [6] Gao, L., Dahlin, M., Nayate, A., Zheng, J. and Iyengar, A., 2003, Application specific data replication for edge services, *ACM*, pp. 449-460.
- [7] Elnikety, S., Dropsho, S. and Zwaenepoel, W., 2007, Tashkent+: Memory-aware load balancing and update filtering in replicated databases, *ACM SIGOPS Operating Systems Review*, 41, (3), pp. 399-412.
- [8] Zuikėviciute, V. and Pedone, F., 2008, Conflict-aware load-balancing techniques for database replication, *ACM*, pp. 2169-2173.
- [9] Amza, C., Cox, A. L. and Zwaenepoel, W., 2005, A comparative evaluation of transparent scaling techniques for dynamic content servers, *IEEE*, pp. 230-241.
- [10] Garbatov, S., Cachopo, J. and Pereira, J., 2009, Data Access Pattern Analysis based on Bayesian Updating, *Proceedings of the First Symposium of Informatics (INForum 2009)*, Lisbon, Paper 23.
- [11] Garbatov, S. and Cachopo, J., 2010, Importance Analysis for Predicting Data Access Behaviour in Object-Oriented Applications, *Computer Science and Technologies*, 1, pp. 37-43.
- [12] Garbatov, S. and Cachopo, J., 2010, Predicting Data Access Patterns in Object-Oriented Applications Based on Markov Chains, *Proceedings of the Fifth International Conference on Software Engineering Advances (ICSEA 2010)*, Nice, France, pp. 465-470.
- [13] Smith, W. TPC-W: Benchmarking An Ecommerce Solution. Intel Corporation, 2000.
- [14] Park, L. and Ramamohanarao, K., 2009, The sensitivity of latent dirichlet allocation for information retrieval, *Machine Learning and Knowledge Discovery in Databases*, pp. 176-188.

Formal Parsing Analysis of Context-Free Grammar using Left Most Derivations

Khalid A. Buragga
 College of Computer Sciences and I. T.
 King Faisal University
 Hofuf, Saudi Arabia
 Email: kburagga@kfu.edu.sa

Nazir Ahmad Zafar
 Department of Computer Science
 King Faisal University
 Hofuf, Saudi Arabia
 Email: nazafar@kfu.edu.sa

Abstract—Formal approaches are useful to verify the properties of software and hardware systems. Formal verification of a software system targets the source program where semantics of a language has more meanings than its syntax. Therefore, program verification does not give guarantee the generated executable code is correct as described in the source program. This is because the compiler may lead to an incorrect target program due to bugs in the compiler itself. It means verification of compiler is important than verification of a source program to be compiled. In this paper, context-free grammar is linked with Z notation to be useful in the verification of a part of compiler. At first, we have defined grammar, then, language derivation procedure is described using the left most derivations. In the next, verification of a given language is described by recursive procedures. The ambiguity of a language is checked as a part of the parsing analysis. The formal specification is analyzed and validated using Z/Eves tool. Formal proofs of the models are presented using powerful techniques, that is, reduction and rewriting of the Z/Eves.

Keywords—parsing analysis; context-free language; formal specification; Z notation; verification.

I. INTRODUCTION

Formal methods are mathematical-based approaches used for specifying, proving and verifying properties of software and hardware systems [1]. The process of formal verification means applying the mathematical techniques to verify the properties ensuring correctness of a system. Formal verification of software systems targets the source program where the semantics of the programming language gives a precise meaning to the programs to be analyzed. On the other hand, program verification does not give guarantee that the generated executable code is correct as described by the semantics of the source program. This is because the compiler may lead to an incorrect target program because of bugs in the compiler and it can invalidate the guarantees obtained by formal methods to source program. It means the verification of compiler is more important than verification of source program to be compiled.

The design, construction and exploitation of a fully verifying compiler will remain a challenge of twenty first century in the area of computer science. The main functionality of compiler is to translate a source code understandable by programmers to an executable machine code correctly and efficiently. Although compiler is a mature

area of research but it needs further investigation, as mentioned above, because bugs in the compiler can lead to an incorrect machine code generated from a correct source program. That is why design and construction of a bug free compiler is an open area of research. Further, as executable code generated by the compiler is tested and if bugs are detected it might be due to the source program or compiler itself. This has led to verification of compiler that proves automatically that a source program is correct before allowing it to be run.

In this paper, parsing analyzing of language is presented using Z notation by left most derivations, which will be useful in our ongoing project on verification of compiler. Another objective of this research is linking context-free grammar with formal techniques to be useful in development of automated computerized systems. Currently, it is not possible to develop a complete software system using a single formal technique and hence integration of approaches is required. Although integration of approaches is a well-researched area [2][3][4][5][6][7][8], but there does not exist much work on formalization of context-free languages. Dong et al. [9][10] have described the integration of Object Z and timed automata. Constable has proposed a constructive formalization of some important concepts of automata using Nuprl [11][12]. A relationship is investigated between Petri-nets and Z in [13]. An integration of B and UML is presented in [14][15]. W. Wechler has introduced some algebraic structures in fuzzy automata [16]. A treatment of fuzzy automata and fuzzy language theory is discussed in [17]. Some important concepts of algebraic theory and automata are given in [18].

In [19], preliminary results of this research were presented by linking context-free grammar and Z notation. In this paper, first, formal definition of context-free grammar is given. Then a derivation procedure is described by replacing non-terminal with a string of terminal and non-terminals. The derivation procedure is extended to a sequence of derivations to derive a string form a given string using production rules of the context-free grammar. Then parsing analysis is described for word by left most derivations resulting a parsing tree. The parsing analysis for a language is specified by introducing recursion using derivations used in generation of a word. Next, ambiguity of a word is checked by specifying if there exists more than two left most derivation trees for a given words. The same concept is formalized for the language to check if it is ambiguous. The

formal specification is analyzed and validated using Z Eves tool set. The major objectives of this research are:

- Identifying and proposing an integration of context-free grammar and formal methods to be useful in verification of the compiler
- Providing a syntactic and semantic relationship between Z and context-free grammar
- Proposing and developing an approach for supporting an automated tools development

Rest of the paper is organized as: in Section 2, an introduction to formal methods is given. In Section 3, an overview of context-free grammar and its applications is provided. Formal construction of models of context-free grammar is given in Section 4. Formal analysis for validating the models is presented in Section 5. Finally, conclusion and future work are discussed in Section 6.

II. FORMAL METHODS

Formal methods are approaches based on mathematical techniques and notations used for describing and analyzing properties of software and hardware systems. These formal techniques are based on discrete mathematics such as sets, logic, relations, functions, graphs, automata theory and higher order logic. Formal methods may be classified in terms of property and model-oriented methods [20].

Property oriented methods are used to describe software in terms of properties, constraints or invariants that must be true. Model-oriented methods are used to construct a model of a system [21]. Formal methods are being applied successfully to improve quality by means of describing and specifying software systems in a well-precise and structured manner. Although there are various tools, techniques and notations of formal methods but at the current stage of their development, it needs an integration of formal techniques and traditional approaches for the complete design, description and construction of a system.

Z notation is a popular specification language in formal methods used at an abstract level. The Z is a model-oriented approach based on set theory and first order predicate logic [22]. Usually, it is used for specifying behavior of sequential programs of systems by abstract data types. In this paper, Z is selected to be linked with context-free language because of a natural relationship which exists between both of these approaches. The Z is based upon set theory including standard set operators, for example, union, intersection, comprehensions, Cartesian products and power sets. On the other hand, the logic of Z is formulated using first order predicate calculus. The Z is used in our research because it allows organizing a system into its smaller components known as schemas. The schema defines a way in which the state of a system can be described and modified. A promising aspect of Z is its mathematical refinement that is a verifiable stepwise transformation of an abstract specification into an executable code. Once formal specifications in Z are written, it can be refined into implemented system by a process of series of stepwise mathematical refinements.

III. APPLICATIONS OF CONTEXT-FREE GRAMMAR

The context-free grammar (CFG) is important in design and description of a programming language and its compiler. Initially, formalism of CFG was developed by Chomsky who described linguistics in a grammatical form and converted into mathematical models providing a precise and simple mechanism of description of languages. The context-free grammars allow a simple and an efficient way of parsing the algorithms. Using the grammar, it can be determined whether a particular pattern can be generated and the way of generation is also determined.

Inclusion of empty string is always required for completeness of a language. All context-free grammars cannot generate the empty string. If a grammar generates the empty string then it is needed to include some rules generating the empty string. Every context-free grammar without null production has an equivalent grammar in Chomsky Normal Form (CNF). Here by equivalence we mean that both the grammars generate the same language. The CNF grammar is important both in theoretical and practical point of view, it can be constructed from a given context-free grammar. By using CNF, it can be decided for a given string if it can be accepted in polynomial time algorithm. Context-free grammars contain both the decidable and un-decidable problems. Deciding for a grammar that it accepts the language of all the strings is an example of un-decidable problem which can be proved by reduction by linking it with the Turing machine. Deciding whether two context-free grammars describe the same language is another example of the un-decidability.

On the other hand, context-free languages have their own limitations. Some of the operators which are well-defined in many other models of automata theory do not behave well in case of the context-free grammar. For example, the intersection of two context-free languages, in general, is not context-free. Similarly, the complement of a context-free language is not context-free one. However, union, concatenation and Kleene star operators produce context-free languages when applied to it.

Context-free grammar can be applied to many areas of diversity, for example, robotics, speech recognition, software engineering, and software maintenance [23]. The applications of CFG in the area of pattern recognition increase the accuracy of patterns to be recognized. This is because it can provide a higher level of abstraction by defining the semantics of patterns as compared to its other counterparts of specification, for example, strings and regular expressions. This semantic analysis can be used to reduce the false identification of the patterns [24]. Further, the applications of pattern matching can be observed everywhere from language processing to networks.

In automatic speech recognition system, the spoken words can be generated by a context-free grammar using dynamic programming algorithms. As an example of application of CFG in the area software engineering, the components in a source code are recognized and re-generated using context-free grammar [25]. As the output of parsing are larger and less-ambiguous and have meaning of

the structures in a sentence, therefore, for question answering and interactive voice response systems, the use of context-free grammar can highly be effective and useful in such kind of systems and applications [26][27].

IV. FORMAL ANALYSIS OF CONTEXT-FREE GRAMMAR

Context-free grammar is a 4-tuple (V, Σ, R, S) where:

- V is a finite set of non-terminal called variables representing different types of clauses in a sentence.
- The Σ is a finite set of terminals and final contents of a string or sentence are based on it.
- The third one R is the start variable used to represent the whole string or a sentence.
- The last one S is a relation consisting of set of all the productions or rules of the grammar.

Every production is of the form: $S \rightarrow t$, where S is a non-terminal consisting of a single character or symbol and t is a string which may contain only terminals or non-terminals or combination of both. Further, t might be an empty string. The notations, $S \rightarrow t$, are called productions or rules which are applied one after other producing a parse tree. The tree ends with terminals called leaves and each internal node is a non-terminal which produces one or more further nodes. The left hand side of a production rule of a context-free grammar is always a single non-terminal. Because all rules only have non-terminals on the left hand side and it can easily be replaced with the string on the right hand side of this rule.

Further the context in which the symbol occurs is therefore not important and hence the grammar is called context-free grammar. It is to be noted that context-free grammar is always recognized by finite state machines having a single infinite taps. For keeping track of nested units, the current parsing state is pushed at the start of the unit and it is recovered at the end.

In this section, formal analysis of CFG is presented using Z notation. We start with the definition of context-free grammar which is a 4-tuple as defined above. R in the tuple is a relation from V to $(V \cup \Sigma)^*$ such that $\exists w \in (V \cup \Sigma)^*$, $S \in V$ and $(S, w) \in R$. The symbol $*$ represents to any combination of characters of V and Σ .

In the specification of CFG, we define the sets of non-terminal by V and terminal by Σ . The set of terminals and non-terminals together denoted by $vandt$ and alphabets of the grammar are of type $\text{seq } X$. The sequence of elements of X , $\text{seq } X$, denotes the set of all sequences containing terminals and non-terminals. The notation for rules is defined by the relation between V and $\text{seq } X$. The production rules are defined by the relation denoted by $rules$. Further there exists exactly one rule, $(s0, w) \in rules$ where $s0$ is the start non-terminal and w is string s of type $\text{seq } X$. With these definitions, a formal definition of context-free grammar is given in terms of a schema CFG . The variables are given in first part and constraints are defined in the second part of schema. The V , X and Σ are defined as sets at an abstract level of specification.

$[X]; V == X;$

$\Sigma == X$

<i>CFG</i>
variables: $F V$ terminals: $F \Sigma$ vandt: $F X$ rules: $V \leftrightarrow \text{seq } X$ s0: V
<hr/> variables \cap terminals = $\{\}$ vandt = variables \cup terminals dom rules \subseteq variables $\forall s: \text{seq } X \mid s \in \text{ran rules} \cdot \text{ran } s \subseteq \text{vandt}$ s0 \in variables $\exists w: \text{seq } X \mid (s0, w) \in \text{rules}$

Invariants:

- The terminals and non-terminals are disjoint sets.
- The entire set of alphabets is union of terminals and non-terminals.
- The domain of $rules$ relation is a subset of variables.
- The set of elements in the range of $rules$ relation are defined based on members of alphabets .
- The variable $s0$ must be an element of $variables$.
- There exists at least one rule which contains start variable on the left hand side of it.

A. Producing Left Most Derivations

In this section, we describe the formal left derivations procedure using the production rules. The substitution can be performed recursively to derive new string from a given string of terminal and non-terminal. First, we specify the process of generating a string using a single production by the schema *LeftDerivation* given below. In the specification, $s1$ and $s2$ are two strings of type $\text{seq } X$. We say $s1$ yields $s2$ if $\exists a \in V$ and $b, s3, s4 \in \text{seq } X$ such that $s1 = s3 \hat{\ } \langle a \rangle \hat{\ } s4$ and $s2 = s3 \hat{\ } b \hat{\ } s4$. It is to be noted that a is an element in set of variables, the ranges of sequences $b, s3, s4$ are subsets of $vandt$, (a, b) is a production rule.

<i>LeftDerivation</i>
CFG drives: $\text{seq } X \leftrightarrow \text{seq } X$
<hr/> $\forall s1, s2: \text{seq } X \mid \text{ran } s1 \subseteq \text{vandt} \wedge \text{ran } s2 \subseteq \text{vandt}$ $\cdot (s1, s2) \in \text{drives} \Rightarrow (\exists a: V; b: \text{seq } X; s3, s4: \text{seq } X$ $\mid a \in \text{variables} \wedge \text{ran } b \subseteq \text{vandt}$ $\wedge (a, b) \in \text{rules} \wedge \text{ran } s3 \subseteq \text{terminals}$ $\wedge \text{ran } s4 \subseteq \text{vandt} \cdot s1 = s3 \hat{\ } \langle a \rangle \hat{\ } s4 \wedge s2 = s3 \hat{\ } b \hat{\ } s4)$

Now, we describe a sequence of left derivations using the approach of single left derivation defined above. The derivation procedure is described below and is denoted by the schema *LeftDerivations* which is an extension of schema

LeftDerivation. It describes the generation from a string of terminals or non-terminals to another string of the same type. In the specification, two strings are considered denoted by $s1$ and $s2$ as in the schema which uses the *LeftDerivation* recursively by introducing a sequence $s3$ of sequences representing an order of the derivations.

<i>LeftDerivations</i>
<i>LeftDerivation</i> $drivess: \text{seq } X \leftrightarrow \text{seq } X$
$\forall s1, s2: \text{seq } X \mid \text{ran } s1 \subseteq \text{vandt} \wedge \text{ran } s2 \subseteq \text{vandt}$ <ul style="list-style-type: none"> • $(s1, s2) \in drivess$ $\Rightarrow (\exists s3: \text{seq } (\text{seq } X)$ <ul style="list-style-type: none"> $1 \leq \# s3 \wedge (\forall ss: \text{seq } X \mid ss \in \text{ran } s3 \cdot \text{ran } ss \subseteq \text{vandt})$ • $(s1, s3\ 1) \in drives$ $\wedge (\forall i: \mathbb{N} \mid i \in 2 \dots \# s3 \cdot (s3\ (i - 1), s3\ i) \in drives)$ $\wedge (s3\ (\# s3), s2) \in drives)$

B. Verification of Language Generated From CFG

In this section, verification of a language generated from a context-free grammar is done. First, verifying procedure of a word is defined then it is extended to the whole language. For this purpose, the formal procedure is described in the schema *WordLeftDerivation* given below. The schema *LeftDerivations* and a *word* are given as input to the schema and it is checked if the *word* can be generated from the CFG using the procedure *WordLeftDerivation* defined above. The symbol, $?$, is used to represent that word is an input variable. In the predicate part of the schema, first, it is checked that all alphabets of the word must be from the set of terminal of CFG. Secondly, it is verified that in the derivation of the word the first production used contains the start variable (non-terminal) on the left hand side of the production.

<i>WordLeftDerivation</i>
<i>LeftDerivations</i> $word?: \text{seq } \Sigma$
$\text{ran } word? \subseteq \text{terminals}$ $(\langle s0 \rangle, word?) \in drivess$

To verify a language, the approach of word verification is used. In the schema *LanguageLeftDerivation* described below, the schema *LeftDerivations* and *language* are given as input and is checked if the *language* can be generated from the CFG by using universal quantifier.

<i>LanguageLeftDerivation</i>
<i>LeftDerivations</i> $language?: \mathbb{P} (\text{seq } \Sigma)$
$\forall w: \text{seq } \Sigma \mid w \in language? \cdot \text{ran } w \subseteq \text{terminals}$ $\forall w: \text{seq } \Sigma \mid w \in language? \cdot (\langle s0 \rangle, w) \in drivess$

C. Checking Ambiguity of Language

In this section, ambiguity of a context-free language is verified. The language is ambiguous if there is a word for which there exists at least two parsing trees based on leftmost derivations. First, verification procedure for a word is defined if it is ambiguously generated using the schema *AmbiguousWord* given below. The schema takes *LeftDerivations* and a *word* as input and checks if the *word* has more than one left most derivations.

<i>AmbiguousWord</i>
<i>LeftDerivations</i> $word?: \text{seq } \Sigma$
$\text{ran } word? \subseteq \text{terminals}$ $(\langle s0 \rangle, word?) \in drivess \Rightarrow (\exists s3, s4: \text{seq } (\text{seq } X) \mid s3 \neq s4$ <ul style="list-style-type: none"> $\wedge 1 \leq \# s3 \wedge 1 \leq \# s4 \wedge \text{ran } s3 \subseteq \text{ran } rules$ $\wedge \text{ran } s4 \subseteq \text{ran } rules \cdot (\langle s0 \rangle, s3\ 1) \in drives$ $\wedge (\forall i: \mathbb{N} \mid i \in 2 \dots \# s3 \cdot (s3\ (i - 1), s3\ i) \in drives)$ $\wedge (s3\ (\# s3), word?) \in drives \wedge (\langle s0 \rangle, s4\ 1) \in drives$ $\wedge (\forall i: \mathbb{N} \mid i \in 2 \dots \# s4 \cdot (s4\ (i - 1), s4\ i) \in drives)$ $\wedge (s4\ (\# s4), word?) \in drives)$

To verify if the language is ambiguous, the verification procedure of a word is reused. In the schema *AmbiguousLanguage* described below, it is checked if there exists any word having more than one derivations by using the universal quantifier. If this is the case the given language is ambiguous.

<i>AmbiguousLanguage</i>
<i>LeftDerivations</i> $language?: \mathbb{F} (\text{seq } \Sigma)$
$\forall w: \text{seq } \Sigma \mid w \in language? \cdot \text{ran } w \subseteq \text{terminals}$ $\exists w: \text{seq } \Sigma \mid w \in language? \cdot (\langle s0 \rangle, w) \in drivess$ $\Rightarrow (\exists s3, s4: \text{seq } (\text{seq } X) \mid s3 \neq s4 \wedge 1 \leq \# s3 \wedge 1 \leq \# s4$ <ul style="list-style-type: none"> $\wedge \text{ran } s3 \subseteq \text{ran } rules \wedge \text{ran } s4 \subseteq \text{ran } rules \cdot (\langle s0 \rangle, s3\ 1) \in drives$ $\wedge (\forall i: \mathbb{N} \mid i \in 2 \dots \# s3 \cdot (s3\ (i - 1), s3\ i) \in drives)$ $\wedge (s3\ (\# s3), w) \in drives \wedge (\langle s0 \rangle, s4\ 1) \in drives$ $\wedge (\forall i: \mathbb{N} \mid i \in 2 \dots \# s4 \cdot (s4\ (i - 1), s4\ i) \in drives)$ $\wedge (s4\ (\# s4), w) \in drives)$

V. MODEL ANALYSIS

There does not exist any computer tool which may guarantee about complete correctness of a computer model. Therefore, even the specification is written using any of the formal languages it may contain potential hazardous or errors. It means an art of writing a formal specification never assures that the developed system is consistent, correct and complete. On the other hand, if the specification is checked and analyzed with the computer tool support it certainly increases the confidence over the system to be developed by

identifying the potential errors, if exist, in syntax and semantics of the formal description. The Z/Eves is one of the most powerful tools which can be used for analyzing the formal specification written by Z notation. A snapshot of the formal specification using Z/Eves tool is presented in Figure 1. The first column on left most of the figure shows a status of the syntax checking and the second one presents the status of proof correctness. The symbol ‘Y’ shows that specification is correct syntactically and proof is correct while the symbol ‘N’ stands that errors are identified. In schemas, it is checked that specification is correct in syntax and has a correct proof.

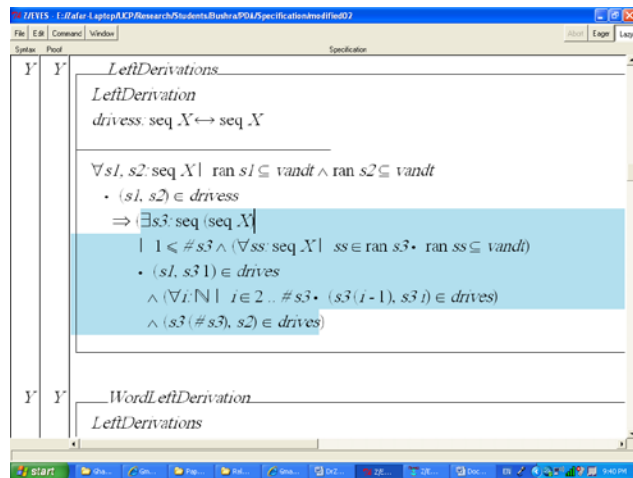


Figure 1. Snapshot of the Model Analysis.

The results of the formal specification are presented in the Table 1. The schema name represents the name of the schemas described for specification. These schemas are analyzed by using the model exploration techniques provided in the Z/Eves tool. The symbol “Y” in column 2 indicates that all the schemas are well written and proved automatically. Similarly, domain checking, reduction and proof by reduction are represented in column 3, 4 and 5, respectively. The symbol “Y*” describes that the schemas are proved by performing reduction on the predicates to make the specification meaningful.

TABLE I. RESULTS OF MODEL ANALYSIS

Schema Name	Syntax Type Check	Domain Check	Reduction	Proof
CFG	Y	Y	Y	Y
LeftDerivation	Y	Y	Y*	Y
LeftDerivations	Y	Y	Y*	Y
WordLefDerivation	Y	Y	Y	Y
LanguageLefDerivation	Y	Y	Y	Y
AmbiguousWord	Y	Y	Y*	Y
AmbiguousLanguage	Y	Y	Y*	Y

VI. CONCLUSION AND FUTURE WORK

An efficient and correct translation from a programming language to machine language is an open issue in the area of computer science and this task is usually done by compilers. Errors in the compiler can lead to incorrect machine code from a source program even the source is correct and verified. Therefore, design and construction of correct compiler is more important than verifying the source programs. If the compiler is formally verified it gives guarantee that the executable code generated behaves exactly as described in the source program. In this paper, formal procedure of identification and analysis of ambiguities is done which is a real challenge in parser development. We know it is an un-decidable problem but this exercise is useful for applying it to a simple compiler for academic purpose, which can be extended to formally verify the compiler.

Both regular expressions and context-free grammars are widely used in construction of the compiler. Regular expressions are not powerful enough and are used to identify token from the source program while syntax is checked by the context-free grammar. The design of a compiler can be benefited by transforming context-free grammar to Z specification because Z notation being abstract in nature and having computer tool support enhances reliability and correctness providing a context in which important properties of the system can be formally analyzed and verified. Further, formal specification helped us to make it possible describing precise, unambiguous and easier to understand the resultant model.

An approach is developed by linking context-free grammar with Z notation defining a relationship between fundamentals of these techniques. It is observed that a natural relationship exists between these approaches. This linkage will be useful in verification of compiler in addition to many other applications. At first, we have described the structures of CFG using Z then formal description of derivation process from a sequence of terminals and non-terminals is presented. Further, a procedure of derivations is described by identifying the productions to be used in this process. Then formal models are defined to check the generation of the words and language from the context-free grammar. Finally, ambiguity of the language is verified by using the left most derivations. Formal proofs of the relationship are presented under certain assumptions. The specification is verified and validated using Z/Eves tool.

An extensive survey of existing work was done and explored before initiating this research. Some interesting work [28][29][30][31][32][33] [34][35][36] was found but our work and approach are different because of conceptual and abstract level integration of Z and CFG. Few of the benefits of Z are listed as follows. Every object is assigned a unique type providing useful programming practice. Several type checking tools exist to support the specification. The Z/Eves is a powerful tool to prove and analyze the specification used in this research. The rich mathematical notations made it possible to reason about behavior of a specified system more rigorous and effectively.

Formalization of some other concepts, useful in compiler verification, is under progress and will appear soon in our future work. Further, we have taken some assumptions for simplicity of construction. In future work, a more generic formal integration will be proposed after relaxing such assumptions.

REFERENCES

- [1] C. J. Burgess, "The Role of Formal Methods in Software Engineering Education and Industry," Technical Report, University of Bristol, UK, 1995.
- [2] H. Beek, A. Fantechi, S. Gnesi, and F. Mazzanti, "State/Event-Based Software Model Checking," *Integrated Formal Methods*, Springer, vol. 2999, pp. 128-147, 2004.
- [3] O. Hasan and S. Tahar, "Verification of Probabilistic Properties in the HOL Theorem Prover," *Integrated Formal Methods*, Springer, vol. 4591, pp. 333-352, 2007.
- [4] F. Gervais, M. Frappier, and R. Laleau, "Synthesizing B Specifications from EB3 Attribute Definitions," *Integrated Formal Methods*, Springer, vol. 3771, pp. 207-226, 2005.
- [5] K. Araki, A. Galloway, and K. Taguchi, "Integrated Formal Methods," *Proceedings of the 1st International Conference on Integrated Formal Methods*, Springer 1999.
- [6] B. Akbarpour, S. Tahar, and A. Dekdouk, "Formalization of Cadence SPW Fixed-Point Arithmetic in HOL," *Integrated Formal Methods*, Springer, vol. 2335, pp. 185-204, 2002.
- [7] J. Derrick and G. Smith, "Structural Refinement of Object-Z/CSP Specifications," *Integrated Formal Methods*, Springer, vol. 1945, pp. 194-213, 2000.
- [8] T. B. Raymond, "Integrating Formal Methods by Unifying Abstractions," Springer, vol. 2999, pp. 441-460, 2004.
- [9] J. S. Dong, R. Duke, and P. Hao, "Integrating Object-Z with Timed Automata," pp 488-497, 2005.
- [10] J. S. Dong, et al., "Timed Patterns: TCOZ to Timed Automata," *The 6th International Conference on Formal Engineering Methods*, pp 483-498, 2004.
- [11] R. L. Constable, P. B. Jackson, P. Naumov, and J. Uribe, "Formalizing Automata II: Decidable Properties," Technical Report, Cornell University, 1997.
- [12] R. L. Constable, P. B. Jackson, P. Naumov, and J. Uribe, "Constructively Formalizing Automata Theory," *Foundations of Computing Series*, MIT Press, 2000.
- [13] M. Heiner and M. Heisel, "Modeling Safety Critical Systems with Z and Petri nets," *International Conference on Computer Safety, Reliability and Security*, Springer, pp. 361-374, 1999.
- [14] H. Leading and J. Souquieres, "Integration of UML and B Specification Techniques: Systematic Transformation from OCL Expressions into B," *Asia-Pacific Software Engineering Conference*, pp. 495-504, 2002.
- [15] H. Leading and J. Souquieres, "Integration of UML Views using B Notation," *Proceedings of Workshop on Integration and Transformation of UML Models*, 2002.
- [16] W. Wechler, "The Concept of Fuzziness in Automata and Language Theory," *Akademic-Verlag*, Berlin, 1978.
- [17] N. M. John and S. M. Davender, "Fuzzy Automata and Languages: Theory and Applications," *Chapman & HALL, CRC*, 2002.
- [18] M. Ito, "Algebraic Theory of Automata and Languages," *World Scientific Publishing Co.*, 2004.
- [19] N. A. Zafar, S. A. Khan, and B. Kamran, "Formal Procedure of Deriving Language from Context-Free Grammar," *International Conference on Intelligence and Information Technology*, vol. 1, pp. 533-536, 2010.
- [20] M. Brendan and J. S. Dong, "Blending Object-Z and Timed CSP: An Introduction to TCOZ," *Proceedings of 20th International Conference on Software Engineering*, pp. 95, IEEE Computer Society, 1998.
- [21] J. M. Spivey, "The Z Notation: A Reference Manual," *Englewood Cliffs, NJ, Prentice-Hall*, 1989.
- [22] J. M. Wing, "A Specifier, Introduction to Formal Methods," *IEEE Computer*, vol. 23 (9), pp. 8-24, 1990.
- [23] J. A. Anderson, "Automata Theory with Modern Applications," *Cambridge University Press*, 2006.
- [24] H. C. Young, J. Moscola, and J. W. Lockwood, "Context-Free Grammar based Token Tagger in Reconfigurable Devices," *Proceedings of International Conference of Data Engineering (ICDE/SeNS)*, pp. 78, 2005.
- [25] M. v. d. Brand, A. Sellink, and C. Verhoef, "Generation of Components for Software Renovation Factories from Context-Free Grammars," *Conference on Reverse Engineering*, pp. 144-153, 2001.
- [26] M. Balakrishna, D. Moldovan, and E. K. Cave, "Automatic Creation and Tuning of Context-Free Grammars for Interactive Voice Response Systems," *Proceedings of IEEE NLP-KE '05*, pp. 158 - 163, 2005.
- [27] L. Pedersen and H. Reza, "A Formal Specification of a Programming Language: Design of Pit," *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, pp. 111-118, 2008.
- [28] D. P. Tuan, "Computing with Words in Formal Methods," Technical Report, University of Canberra, Australia, 2000.
- [29] S. A. Vilkomir and J. P. Bowen, "Formalization of Software Testing Criterion," *South Bank University, London*, 2001.
- [30] A. Hall, "Correctness by Construction: Integrating Formality into a Commercial Development Process," *Praxis Critical Systems Limited, Springer*, vol. 2391, pp. 139-157, 2002.
- [31] B. A. L. Gwandu and D. J. Creasey, "Importance of Formal Specification in the Design of Hardware Systems," *School of Electron & Electr. Eng., Birmingham University*, 1994.
- [32] D. K. Kaynar and N. Lynch, "The Theory of Timed I/O Automata," *Morgan & Claypool Publishers*, 2006.
- [33] D. Jackson, I. Schechter, and I. Shlyakhter, "Alcoa: The Alloy Constraint Analyzer," *Proceedings of The 22nd International Conference of Software Engineering (ICSE'2000)*, pp. 730-733, 2000.
- [34] D. Aspinall and L. Beringer, "Optimisation Validation," *Electronic Notes in Theoretical Computer Science*, vol. 176, pp. 37-59, 2007.
- [35] S. Briaisa and U. Nestmann, "A Formal Semantics for Protocol Narrations," *Theoretical Computer Science*, vol. 389, pp. 484-511, 2007.
- [36] L. Freitas, J. Woodcock, and Y. Zhang, "Verifying the CICS File Control API with Z/Eves: An Experiment in the Verified Software Repository," *Science of Computer Programming*, vol. 74, pp. 197-218, 2009.

Functional Complexity Measurement: Proposals and Evaluations

Luigi Lavazza

Dipartimento di Informatica e Comunicazione
Università degli Studi dell'Insubria
Varese, Italy
luigi.lavazza@uninsubria.it

Gabriela Robiolo

Departamento de Informática
Universidad Austral
Buenos Aires, Argentina
grobiolo@austral.edu.ar

Abstract — Several definitions of measures that aim at representing the size of software requirements are currently available. These measures have gained a quite relevant role, since they are one of the few types of objective data upon which effort estimation can be based. However, traditional Functional Size Measures do not take into account the amount and complexity of elaboration required, concentrating instead on the amount of data accessed or moved. This is a problem, when it comes to effort estimation, since the amount and complexity of the required data elaboration affect the implementation effort, but are not adequately represented by the current measures, including the standardized ones. Recently, a few approaches to measuring aspects of user requirements that are supposed to be related with functional complexity and/or data elaboration have been proposed by researchers. The authors of this paper have also proposed a measure of the functional complexity as specified in user requirements. In this paper we take into consideration some of these proposed measures and compare them with respect to their ability to predict the development effort, especially when used in combination with COSMIC measures of functional size.

Keywords-Functional size measurement; Function Points; COSMIC function points; effort estimation; functional complexity measurement.

I. INTRODUCTION

COSMIC function points [8][12] are growingly used for measuring the functional size of applications, i.e., to measure the size of functional user requirements. The measure of functional size is typically used to drive the estimation of the development effort. To this end, effort models require several inputs in addition to the functional size, including the complexity of the software to be [3][7]. In fact, problem complexity is recognized as one of the elements that contribute to the comprehensive notion of software size [9].

The need to account for software complexity when estimating the development effort does not depend on the functional size measurement method used: for instance, when more traditional measures of the functional size –like IFPUG function points [12]– are used, complexity has to be accounted for as well.

Actually, both COSMIC and IFPUG function points fail to represent the amount and complexity of data elaboration required. COSMIC function points concentrate on the measure of the data movements, neglecting the data

elaboration. More precisely, the model of software used by the COSMIC method –illustrated in Figure 1–includes data elaboration, but no indication on how to measure it is provided. The COSMIC measurement manual [8] simply assumes that every data movement accounts for some amount of data elaboration, and that such amount is proportional to the number of data movements, so that by measuring data movements one measures also data manipulation.

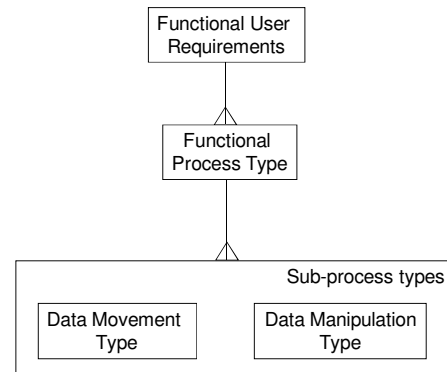


Figure 1. The COSMIC generic software model.

Before proceeding, it is useful to spend some words on the fact that throughout the paper we treat the terms “complexity” and “amount of data elaboration” as synonyms. This is due to the fact that complexity is an inherently elusive concept, and also to the fact that at the functional requirements level it is not clear what should be the difference between the amount and the complexity of data elaboration: for instance, in many cases, complexity is considered proportional to the number of alternatives in a process execution, but this number is also clearly related to the size of the process.

When dealing with effort estimation, the most popular methods require an evaluation of the complexity of the application. Currently such evaluation is of a purely qualitative nature. For instance, COCOMO II [7] provides a table that allows the user to evaluate complexity on an ordinal scale (from “very low” to “extra high”) according to five aspects (control operations, computational operations, device-dependent operations, data management operations, user interface management operations) that have to be evaluated in a qualitative and subjective way: e.g., the characterization of computational operations corresponding

to the “Nominal” complexity is “Use of standard math and statistical routines. Basic matrix/vector operations” [7].

It is quite clear that it would be greatly beneficial to replace such subjective and approximate assessment of complexity with a real measure, based on objective and quantitative evaluations, since this would enable the construction of more objective and accurate models of effort.

Several different possible measures of functional complexity were proposed. For instance, in [5] the number of inputs and outputs, the number of decision nodes, the sum of predicates of all decision nodes, the depth of decision tree and the length of paths are considered as possible indicators of complexity.

In [6], Cao et al. propose the usage of the number of data groups (NOD), the number of conditions (NOC) and entropy of system (EOS). They also study how these measures (also in combination with COSMIC FP) are correlated with the development effort.

Another measure of complexity, the Paths, was defined on the basis of the information typically available from use case descriptions [21]. The measure of the complexity of use cases is based on the application of the principles of McCabe’s complexity measure [18] to the descriptions of use cases in terms of scenarios. In fact, use cases are usually described giving a main scenario, which accounts for the ‘usual’ behaviour of the user and system, and a set of alternative scenarios, which account for all the possible deviations from the normal behaviour that have to be supported by the system. Robiolo and Orosco [21] apply to the use case textual descriptions the same measure applied by McCabe to code. Every different path in a given use case scenario contributes to the measure of the use case’s complexity. The definition of Paths conforms to several concepts enounced by Briand et al. [4]: Paths represent “an intrinsic attribute of an object and not its perceived psychological complexity as perceived by an external observer”, and they represent complexity as “a system property that depends on the relationship between elements and is not an isolated element’s property”. A detailed description of the Paths measure and its applicability to use cases described in UML can be found in [15].

Previous work showed that effort models that take into consideration complexity measures are more precise than those based on the functional size only. In particular, the authors of this paper showed that development effort correlates well with COSMIC function points and Path [15], and that the inclusion of a Path-based complexity measure improves the models based on size, whatever size measure is used (IFPUG Function Points, CFP, or even Use Case Points) [16].

In this paper we enhance the dataset used in [16] with some measures that represent potential complexity dimensions, build effort estimation models that exploit these measures, and discuss the precision of fit of these models.

The results of the measurements and analyses reported in the paper contribute to enhancing the knowledge of how it is possible to measure functional complexity at the requirements level, and what is the contribution of such measure to effort estimation.

II. THE EXPERIMENTAL EVALUATION

In the research work reported here, we used measures that are conceptually very close to those proposed in previous studies [5][6]. However, we did not stick exactly to the previous proposals, essentially for practical reasons. We used Paths instead of NOC because both measures capture essentially the same meaning, and the measures of Paths were already available. Similarly, we used the number of data groups instead of NOD, because –having measured the size of the applications in CFP, the documentation on the data groups was already available, thus the measurement could be performed very easily.

Finally, we decided to use another “by product” of CFP measurement, namely the number of functional processes, as a simplified measure of size.

A. The Dataset

In order to evaluate the measures mentioned above with respect to their usability as effort predictors, we collected all such measures for a set of projects. We could not use data from the best known repositories –such as the PROMISE or ISBSG– because they do not report the size of each project according to different FSM methods; moreover, the Paths measure is very recent, and no historical data exist for it.

TABLE 1. THE DATASET

ProjID	Actual effort	Path	CFP	Func. Proc.	Data groups	Pers. DG
P1	410	71	143	39	21	7
P2	473.5	73	118	28	15	9
P3	382.4	60	109	24	15	12
P4	285	49	74	25	14	8
P5	328	34	48	12	17	7
P6	198	35	67	10	15	7
P7	442.02	50	81	16	12	6
P8	722.65	97	115	27	19	10
P9	392	83	105	24	22	11
P10	272	42	73	21	9	9
P11	131	18	51	13	5	5
P12	1042	118	85	30	29	12
P13	348	32	46	12	12	6
P14	242.5	68	96	26	18	9
P15	299.76	33	54	12	12	4
P16	147	20	53	14	15	4
P17	169	17	30	5	10	6

We measured 17 small business projects, which were developed in three different contexts: an advanced undergraduate academic environment at Austral University, the System and Technology (S&T) Department at Austral

University and a CMM level 4 Company. The involved human resources shared a similar profile: advanced undergraduate students who had been similarly trained worked both at the S&T Department and at the CMM level 4 Company. All the selected projects met the following requisites:

- a) Use cases describing requirements were available.
- b) All projects were new developments.
- c) The use cases had been completely implemented, and the actual development effort in PersonHours was known.

The dataset is reported in TABLE 1. Note that we distinguished the number of persistent data groups (column *Pers. DG*) from the total number of data groups, which includes also transient data groups. Our hypothesis is that persistent data groups are more representative of the amount of data being handled by the application.

B. Analysis of the dataset using log-log transformations

As a first approach to evaluating the correlation of effort with other measures, we used linear regression after log-log transformation, as is usually done in studies concerning effort (see for instance COCOMO [3][7]).

We started by checking the correlation between effort and CFP. The results are not very good: after eliminating outliers, we got a model featuring adjusted $R^2 = 0.335$.

Then we moved to univariate analysis of the correlation between Effort and each variable mentioned in TABLE 1:

- Path [Path]
- COSMIC Function Points [CFP]
- Functional Processes [FPr]
- Data Groups [DG]
- Persistent Data Groups [PDG]

We also systematically tested the correlation between effort and the following *density* measures:

- Path per Functional Process [Path/FPr]
- Path per CFP [Path/CFP]
- Data Groups per Functional Process [DG/FPr]
- Data Groups per CFP [DG/CFP]
- Persistent Data Groups per Functional Process [PDG/FPr]
- Persistent Data Groups per CFP [PDG/CFP]

These density measures introduce the concept of complexity per size unit. The complexity of a system is a property that depends on the relationships among system's elements [4]. So, the measures listed above represent the density of relationships among elements per unit size. As size units we adopted both the fine grained CFP and the coarse grained number of functional processes. In fact, the number of functional processes is suggested as a reasonable approximation of the size in CFP in [8].

Quite interestingly, we got significant models only based on variables involving Paths. The results are synthetically reported in TABLE 2. For each model, we have also assessed the precision of the fit by using what are considered the de facto currently used goodness-of-fit indicators in Empirical Software Engineering, i.e., the Mean Magnitude of Relative Error (MMRE) and the percentage of data points whose actual effort falls within 75% and 125% of the estimated value (pred(25)) and the error range.

In TABLE 2 are reported only the models that satisfy the applicability conditions of linear regression (e.g., the residuals are normally distributed), are statistically significant (e.g., their p-value is < 0.05), and have coefficient of determination (Adjusted R^2) sufficiently high (> 0.6).

TABLE 2. CORRELATIONS WITH EFFORT (LOG-LOG UNIVARIATE REGRESSION)

Var.	Adj. R^2	p-value	Outl.	MMRE	Pred(25)	Error range
Path	0.79	$< 10^{-5}$	2	22.7	70.6	-35%..82%
Path/FPr	0.73	$< 10^{-3}$	5	37.2	58.8	-48% ..169%
Path/CFP	0.65	$< 10^{-4}$	0	24.1	52.9	-43% ..66%

The regression line of the model representing Effort vs. Paths –which appears as the best univariate model– is illustrated in Figure 2.

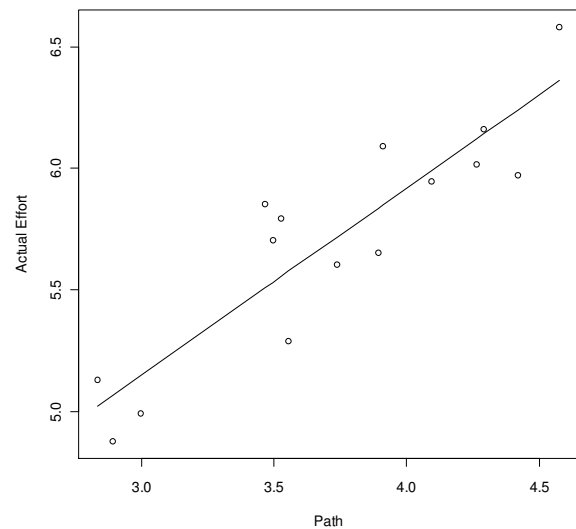


Figure 2. Effort vs. Path: log-log regression line.

The distribution of relative residuals is given in Figure 3.

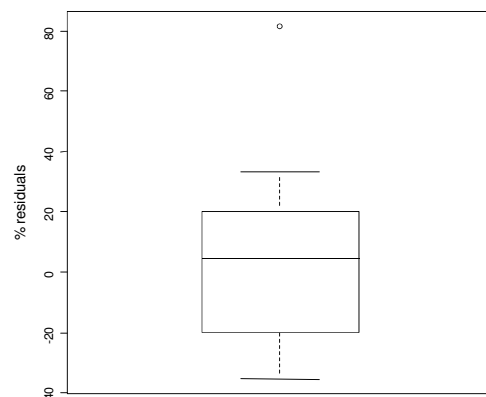


Figure 3. Log-log regression of effort vs. Path: distribution of relative residuals.

We then proceeded to the analysis via multiple regression. Again, we systematically tested the correlation of Effort with any combination of the aforementioned variables.

The statistically significant models obtained are reported in TABLE 3.

TABLE 3. CORRELATIONS WITH THE ACTUAL EFFORT (LOG-LOG MULTIPLE REGRESSION)

Var.	Adj. R ²	Pr(> t)	Outl.	MMRE	Pred(25)	Error range
FPr, Path /FPr	0.71	< 10 ⁻³	1	19.4	70.6	-37%.. 67%
FPr, Path /CFP	0.78	< 10 ⁻³	0	18.7	82.4	-31% .. 80%
Path/FPr, PDG/FPr	0.64	< 10 ⁻³	1	22.9	64.7	-45% .. 53%
Path/CFP, DG/FPr	0.69	< 0.03	1	22.6	64.7	-30% .. 73%
Path/CFP, DG/CFP	0.72	<0.02	1	21.3	64.7	-41%.. 69%
Path/CFP, PDG/FPr	0.75	<0.02	0	18.5	70.6	-35%.. 74%
Path/CFP, PDG/CFP	0.80	< 10 ⁻²	0	17.8	82.4	-36%.. 72%
DG, Path/FPr, DG/FPr	0.75	< 10 ⁻²	0	18.6	70.6	-29%.. 76%
Path/FPr, DG/CFP, PDG/FPr	0.67	<0.05	3	23.9	58.8	-66%.. 79%

It is quite interesting to see that none of the obtained models uses size in CFP as an independent variable. On the contrary, most of the other variables (including size expressed as number of Functional Processes, computation density, amount of data and data density) can be used to build valid and significant models.

It is also interesting to see that these models appear quite good both in terms of their ability to explain the variation of effort depending on the variation of the size and complexity measures (as indicated by the values of the adjusted R²) and in terms of precision of the fit (as indicated by MMRE, pred(25) and the relative error range).

Although it is quite clear that some models appear better than others, e.g., with respect to precision of fit and adjusted R², it is not so obvious which one is best.

A possible way for identifying the best model is by comparison of the relative absolute residuals (since we are considering the ability to predict effort, we have to look at relative absolute residuals, since an error of, say, two PersonMonths can be irrelevant or very important, depending on the total effort). The models that feature the highest values of the adjusted R² are those based on

- a) Paths
- b) Path per CFP and PersistentDataGroups per CFP
- c) Functional Processes and Path per CFP

The boxplots representing relative absolute residuals of these models are reported in Figure 4 .

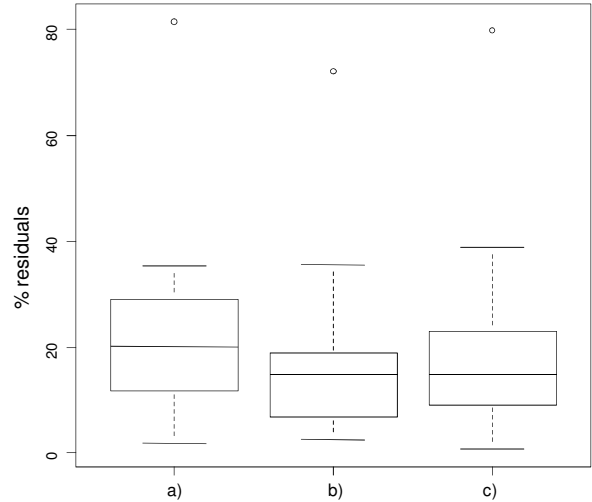


Figure 4. Model comparison: relative absolute residuals.

The comparison of boxplots does not allow selecting a model as clearly the best, although it seems that the univariate model is a bit less precise than both the other two models. In order to evaluate whether a model can be elected the best, Kitchenham et al. [14] suggest to use paired tests of the absolute residuals. We then proceeded to compute the paired tests. We used t-tests when appropriate (i.e., when the distributions were close to normal) and the Wilcoxon signed rank test otherwise. Also the paired tests did not indicate a clear winner. Therefore, we must conclude that further research is needed to understand if it is possible to build a model that explains in the best possible manner the dependency of effort from size and complexity measures.

C. Analysis of the dataset using plain linear regression

Having performed the analysis on log-log transformed data, we checked if valid and significant models can be built using ordinary least squares (OLS) linear regression, i.e., without log-log (or any other) transformation.

We found that a linear model linking Effort and Paths exists: it features adjusted R² = 0.71, p-value < 10⁻³, MMRE = 23.5%, Pred(25) = 58.8%, Error range = -33% .. 81%.

The models involving two independent variables are summarized in TABLE 4.

TABLE 4. CORRELATIONS WITH THE ACTUAL EFFORT (OLS MULTIPLE REGRESSION)

Var.	Adj. R ²	Pr(> t)	Outl.	MMRE	Pred(25)	Error range
CFP, Path /CFP	0.82	< 10 ⁻³	4	18.5%	76.5%	-20%.. 84%
FPr, Path/CFP	0.64	< 10 ⁻²	3	20%	76.5%	-31%.. 76%

It is interesting to note that in this case the best model involves the usage of a size measure (CFP) and a complexity density measure (Paths/CFP).

III. DISCUSSION

The only models based on a single variable that feature adjusted R^2 greater than 0.6 involve either Path or Path density. Anyway, Path seems to be better than both Path per CFP and Path per Functional Processes as far as R^2 , MMRE and Pred(25) are concerned. The reason why Path appears as a good predictor of effort is probably that this measure summarizes the needed information concerning both size (a la COSMIC) and amount of required elaboration.

Concerning models using two independent variables, we can observe that they appear of similar precision (e.g., MMRE ranges from 17.8% to 23.9%).

As already mentioned, there is no statistically evidence that any of these models features a better fitting than univariate models.

Also in these models, Path per CFP appears as an independent variable in several good models, together with

- the density of data (DataGroups per CFP, DataGroups per FunctionalProcess, PersistentDataGroups per CFP or PersistentDataGroups per FunctionalProcess);
- the number of functional processes.

Interestingly, the persistent data groups (a concept close to unweighted data functions in Function Point Analysis) appear to be a better predictor than the whole number of data groups (i.e., including transient ones).

Path per Functional Process provides –as Path per CFP– good models in combination with the density of data (PersistentDataGroups per FunctionalProcess) or the number of functional processes.

It should be noted that we found some models based *exclusively* on density (such as the second and third in TABLE 2 or the seventh in TABLE 3). These models are rather unexpected, as they say that the size of the programs is not important at all. This result is probably due to the fact that the variation of size was relatively little in the set of projects that we analysed. Additional research is needed to explore this point.

Finally, Path per Functional Process appears also as an argument in models featuring three independent variables. So, the complexity density (i.e., Paths divided by a size measure) appears in *all* the models.

When considering models obtained via OLS regression (i.e., without log-log transformation) we find again an elaboration density measure (Path per CFP), this time in combination with a size measure (CFP or Functional Processes).

IV. RELATED WORK

A few attempts to account for data elaboration in FSM have been done.

3D Function Points [22] consider three dimensions of the application to be measured: Data, Function, and Control. The Function measurement considers the complexity of algorithms; and the Control portion measures the number of major state transitions within the application.

Bernárdez et al. [2] measured the cyclomatic complexity of a use case in order to validate the use case definition, while Levesque [17] measured the conditions of inputs in a

sequential diagram in order to add the concept of complexity to the COSMIC method.

Bashir and Thomson [1] used traditional regression analysis to derive two types of parametric models: a single variable model based on product complexity and a multivariable model based on product complexity and requirements severity. Generally, the models performed well according to a number of accuracy tests. In particular, product complexity explained more than 80% of variation in estimating effort. They concluded that product complexity as an indicator for project size is the dominant parameter in estimating design effort.

Our results are in agreement with those by Bashir and Thomson, in fact several of our models explain 80% (or just slightly less) of the variation of effort.

Hastings and Sajeev [11] proposed a Vector Size Measure (VSM) that incorporates both functionality and problem complexity in a balanced and orthogonal manner. VSM is used as the input to a Vector Prediction Model (VPM) which can be used to estimate development effort early in the software life cycle. The results indicate that the proposed technique allows for estimating the development effort early in the software life cycle with errors not greater than 20% across a range of application types.

Our results are in accordance with the consideration expressed by Morasca on the definition of measures [19] as it appears that the notion of complexity may be represented by taking into account several basic indicators (size, control flow, data, ...) that can be used individually (i.e., without the need to build a derived measure defined as a weighted sum) in estimation models.

Finally, Gencil and Demirors [10] point out that we still need a new Base Functional Component (BFC) Types for the boolean operations of Functional User Requirements, which are often not considered to be algorithmic operations, but which are related to complexity. This point of view highlights the necessity of considering the complexity of elaboration required in FSM, and they suggested introducing as a new BFC type which differs from authors' proposal.

V. CONCLUSIONS

The work reported here moves from the consideration that development effort depends (also) on the complexity or the amount of computation required, but no suitable measure has emerged as a reliable way for capturing such complexity. In fact, very popular methods like COCOMO II [3][7] still use just an ordinal scale measure for complexity, based on the subjective evaluation performed by the user.

We approached the problem of measuring the required functional complexity by considering (a subset of) the approaches presented in the literature, and testing them on a set of projects that were measured according to the COSMIC FSM.

The results of our analysis do not allow us to draw definite conclusions about the best set of measures to use for effort estimation. However, we observed that all the most significant models obtained were based on a notion of computation density, which is based on the measure of Paths

[21], i.e., the number of computation flows in functional processes.

Since Paths are quite easy to measure [15] and appear as good effort predictors, we suggest that future research on COSMIC based effort estimation takes into consideration the possibility of involving a Path based measure of functional complexity.

We plan to continue experimenting with measures of functional complexity. Since in this type of experimentations a critical point is the difficulty to get measures, we kindly invite all interested readers that are involved in effort estimations to perform functional complexity measurement and share the data with us and the research community.

ACKNOWLEDGMENT

The research presented in this paper has been partially funded by the IST project QualiPSo [20], sponsored by the EU in the 6th FP (IST-034763), the project “Metodi e tecniche per l’analisi, l’implementazione e la valutazione di sistemi software” funded by the Università degli Studi dell’Insubria, and the Research Fund of School of Engineering of Austral University.

REFERENCES

- [1] Bashir, H. and Thomson, V. Models for estimating design effort and time. Elsevier. Design Studies, vol.22, n.2, 2001.
- [2] Bernárdez B., Durán A., and Genero M. Empirical Evaluation and Review of a Metrics-Based Approach for Use Case Verification. Journal of Research and Practice in Information Technology, vol. 36 n. 4, 2004.
- [3] Boehm, B.W., Horowitz, E., Madachy, R., Reifer, D., Clark, B.K., Steece, B., Winsor Brown, A., Chulani S., and Abts, C. Software Cost Estimation with Cocomo II. Prentice Hall, 2000.
- [4] Briand L.C., Morasca S., and Basili V.R. Property-Based Software Engineering Measurement. IEEE Transactions on Software Engineering, Vol. 22, 1996.
- [5] Cao, De Tran, Lévesque, G., and Abran, A. From Measurement of Software Functional Size to Measurement of Complexity, ICSM 2002, Montreal, Canada, 2002.
- [6] Cao, De Tran, Lévesque, G., and Meunier, J-G. A Field Study of Software Functional Complexity Measurement, 14th International Workshop on Software Measurement, IWSM/METRIKON’04, Berlin, 3-5 November 2004.
- [7] COCOMO II Model Definition Manual. http://csse.usc.edu/csse/research/COCOMOII/cocomo_downloads.htm
- [8] COSMIC – Common Software Measurement International Consortium, 2009. The COSMIC Functional Size Measurement Method - version 3.0.1 Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761: 2003), May 2009.
- [9] Fenton, N.E. Software Metrics: A Rigorous Approach. Chapman and Hall, London, 1991.
- [10] Gencel, C. and Demirors, O. Functional Size Measurement Revisited. ACM Transactions on Software Engineering and Methodology, 17(3), 2008.
- [11] Hastings, T. and Sajeev, A. A Vector-Based Approach to Software Size Measurement and Effort Estimation. IEEE Transactions on Software Engineering, vol.27 n.4, 2001.
- [12] ISO/IEC19761:2003, Software Engineering – COSMIC-FFP – A Functional Size Measurement Method, ISO.
- [13] ISO/IEC 20926: 2003, Software engineering – IFPUG 4.1 Unadjusted functional size measurement method – Counting Practices Manual, International Organization for Standardization, Geneva.
- [14] Kitchenham, B., Pfleeger, S.L., McColl, B., and Eagan, S., An empirical study of maintenance and development accuracy, Journal of Systems and Software, vol. 64, 2002.
- [15] Lavazza, L. and Robiolo, G. Introducing the Evaluation of Complexity in Functional Size Measurement: a UML-based Approach, 4th International Symposium on Empirical Software Engineering and Measurement - ESEM 2010, Bozen, 16-17 September 2010.
- [16] Lavazza, L. and Robiolo, G., The Role of the Measure of Functional Complexity in Effort Estimation, 6th International Conference on Predictive Models in Software Engineering (PROMISE 2010), Timisoara, Romania, 12-13 September 2010.
- [17] Levesque G., Bevo V., and Cao, De Tran, Estimating Software size with UML Models. In Proceedings of the 2008 C3S2E conference, ACM International Conference Pro-ceeding Series, vol. 290, 2008.
- [18] McCabe, T.J. A complexity measure. IEEE Transactions on Software Engineering, vol.2, n.4, 2005.
- [19] Morasca, S. On the use of weighted sums in the definition of measures. ICSE Workshop on Emerging Trends in Software Metrics (WETSoM '10), Cape Town, South Africa, May 04, 2010.
- [20] QualiPSo project portal. <http://www.qualipso.eu/>
- [21] Robiolo, G. and Orosco, R. Employing use cases to early estimate effort with simpler metrics. Innovations Syst. Softw. Eng, vol.4, 2008.
- [22] Whitmire, A., An Introduction to 3D Function Points, Software Development, vol. 3 n.4, 1995.

Design Patterns for Model Transformations

Kevin Lano
 Dept. of Informatics
 King's College London
 London, UK
 Email: kevin.lano@kcl.ac.uk

Shekoufeh Kolahdouz-Rahimi
 Dept. of Informatics
 King's College London
 London, UK
 Email: shekoufeh.kolahdouzrahimi@kcl.ac.uk

Abstract—Model transformations are a central element of model-driven software development. This paper defines design patterns for the specification and implementation of model transformations. These patterns are commonly recurring structures and mechanisms which we have identified in many specific transformations. In this paper we show how they can be used together to support an overall development process for model transformations from high-level specifications to executable Java implementations.

Keywords — Design patterns; model transformations; UML.

I. INTRODUCTION

Design patterns for software development were introduced by Gamma et al [5]. Subsequently, many hundreds of patterns have been identified, including patterns for specialised forms of development such as enterprise information systems [3]. Patterns for model transformations were proposed by [1]. In this paper, we consider further patterns, based on a large number of case studies which we have carried out or analysed. These patterns are inter-related and can be used together to support the development of transformations from high-level specifications as sets of constraints, to executable implementations in Java. They have been incorporated into our transformation environment, UML-RSDS [10].

Section II describes related work, Section III defines a general development process for model transformations. Section IV describes specification patterns, Section V describes implementation patterns and Section VI gives conclusions.

II. RELATED WORK

General design patterns can be used for model transformations. For example, the Builder and Abstract Factory patterns are directly relevant to transformation implementation, in cases where complex platform-specific structures of elements must be constructed from semantic information in a platform-independent model, such as the synthesis of J2EE systems from UML specifications. The Visitor pattern can be used for model-to-text transformations [4]. The Model-view-controller pattern is relevant for change-propagating model transformations, where changes to the source model are propagated to the target (view).

Patterns specific to model transformations have been identified and used previously. In [2], specifications of the conjunctive-implicative form (Section IV) are derived from model transformation implementations in triple graph grammars and QVT, in order to analyse properties of the transformations, such as definedness and determinacy. This form of specification is therefore implicitly present in QVT and other transformation languages.

In [14], [15] the concept of the conjunctive-implicative form was introduced to support the automated derivation of transformation implementations from specifications written in a constructive type theory.

In [1], a transformation specification pattern is introduced, *Transformation parameters*, to represent the case where some auxiliary information is needed to configure a transformation. This could be considered as a special case of the auxiliary metamodel pattern (Section IV). An implementation pattern *Multiple matching* is also defined, to simulate rules with multiple element matching on their antecedent side, using single element matching. We also use this pattern, via the use of multiple \forall quantifiers in specifications and multiple *for* loops at the design level to select groups of elements.

Our work extends previous work on model transformation patterns by combining patterns into an overall process for developing model transformation designs and implementations from their specifications. The patterns are an essential part of the UML-RSDS development process for model transformations.

III. DEVELOPMENT PROCESS FOR MODEL TRANSFORMATIONS

In this section, we outline a general development process for model transformations specified as constraints and operations in UML. We assume that the source and target metamodels of a transformation are specified as class diagrams, S and T , respectively, possibly with OCL constraints defining semantic properties of these languages.

For a transformation τ from S to T , there are three separate predicates which characterise its global properties, and which need to be considered in its specification and design [10]:

- 1) *Asm* – assumptions, expressed in the union language \mathcal{L}_{SUT} of the source and target metamodels, which can be assumed to be true before the transformation is applied. These may be assertions that the source model is syntactically correct, that the target model is empty, or more specialised assumptions necessary for τ to be well-defined. These are preconditions of the use case of the transformation.
- 2) *Ens* – properties, usually expressed in \mathcal{L}_T , which the transformation should ensure about the target model at termination of the transformation. These properties usually include the constraints of T , in order that syntactic correctness holds. For update-in-place transformations, where the source and target languages are the same, *Ens* may refer to the pre-state versions of model data.
- 3) *Cons* – constraints, expressed in \mathcal{L}_{SUT} , which define the transformation as a relationship between the elements of the source and target models, which should hold at termination of the transformation. Update-in-place transformations can be specified by using a syntactically distinct copy of the source language, for example by postfixing all its entity and feature names by *@pre*.
Cons corresponds to the postconditions of the use case of the transformation.

We can express these predicates using OCL notation, this corresponds directly to a fully formal version in the axiomatic UML semantics of [8]. Together these predicates give a global and declarative definition of the transformation and its requirements, so that the correctness of a transformation may be analysed at the specification level, independently of how it is implemented.

The following should be provable:

$$Cons, \Gamma_S \vdash_{\mathcal{L}_{SUT}} Ens$$

where Γ_S is the semantic representation of the source language as a theory.

Development of the transformation then involves the construction of a design which ensures that the relationship *Cons* holds between the source and target models. This may involve decomposing the transformation into *phases* or sub-transformations, each with their own specifications. By reasoning using the weakest-precondition operator [] the composition of phases should be shown to achieve *Cons*:

$$\Gamma_S \vdash_{\mathcal{L}_{SUT}} Asm \Rightarrow [activity]Cons$$

where *activity* is the algorithm of the transformation. Each statement form of the statement language (Chapter 6 of [8]) has a corresponding definition of [].

IV. SPECIFICATION PATTERNS

In this section we describe characteristic patterns for the specifications of model transformations.

A. Conjunctive implicative form

Synopsis: To specify the effect of a transformation in a declarative manner, as a global pre/post predicate, consisting of a conjunction of constraints with a $\forall \Rightarrow \exists$ structure.

Forces: Useful whenever a platform-independent specification of a transformation is necessary. The conjunctive-implicative form can be used to analyse the semantics of a transformation, and also to construct an implementation.

The pattern typically applies when S and T are similar in structure, for example in the UML to relational database mapping of [13], [10], the source structure of *Package*, *Class*, *Attribute* corresponds to the target structure of *Schema*, *Table*, *Column*.

Solution: The *Cons* predicate should be split into separate conjuncts C_n each relating one (or a group) of source model elements to one (or a group) of target model elements:

$$\forall s : S_i \cdot SCond_{i,j} \text{ implies } \exists t : T_{i,j} \cdot LPost_{i,j} \text{ and } GPost_{i,j}$$

where the S_i are source entities, the $T_{i,j}$ are target model entities, $SCond_{i,j}$ is a predicate on s (identifying which elements the constraint should apply to), and $LPost_{i,j}$ defines the attributes of t in terms of those of s . $GPost_{i,j}$ defines the links of t in terms of those of s .

Figure 1 shows a schematic structure of this pattern.

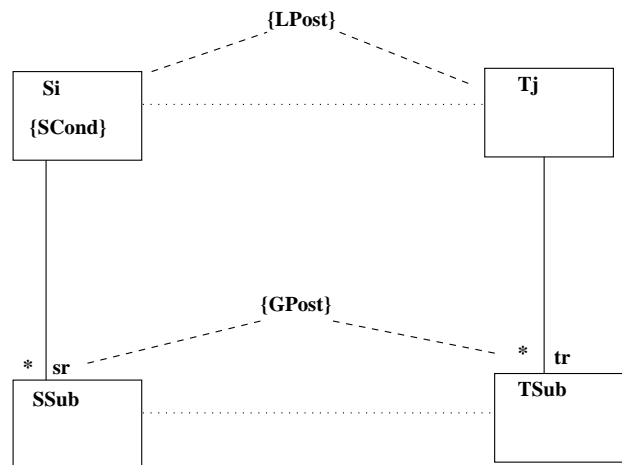


Figure 1. Conjunctive-implicative form

We distinguish three cases of constraints C_n :

- 1) Type 1 constraints: $rd(C_n) \cap wr(C_n) = \{\}$ where rd is the read frame and wr the write frame of the constraint: the set of features and entities which it (conceptually) reads and updates.
- 2) Type 2 constraints: $S_i \notin wr(C_n)$ and $rd(SCond) \cap wr(C_n) = \{\}$ but $rd(C_n) \cap wr(C_n) \neq \{\}$.
- 3) Type 3 constraints: all other cases.

For type 2 or type 3 constraints, suitable metrics are needed to establish termination and correctness of the derived transformation implementation: There should exist a measure $Q : \mathbb{N}$ on the state of a model, such that Q is decreased on each step of the transformation (application of a constraint to a particular domain element), and with $Q = 0$ being the termination condition of the transformation.

There are also special cases of the pattern for *entity splitting*, when the data of one source entity is used to produce the data of several target entities, and *entity merging*, when data from several source entities is used to produce the data of a single target entity.

Consequences: The *Ens* properties should be provable directly from the constraints: typically by using the *Cons* constraints that relate the particular entities used in specific *Ens* constraints.

Implementation: Implementation can be either by the phased creation or recursive descent implementation patterns (Section V). For phased creation the constraints can be individually implemented as phases, with different strategies being used for each type of constraint.

Individual constraints C_n :

$$\forall s : S_i \cdot SCond \text{ implies } \exists t : T_j \cdot LPost \text{ and } GPost$$

are examined to identify which implementation strategy can be used to derive their design. This depends upon the features and objects read and written within the constraint (Table I).

Constraint type	Implementation choice
Type 1 constraint	Approach 1: single for loop $\text{for } s : S_i \text{ do } s.op()$
Type 2 constraint	Approach 2: while iteration of for loop.
Type 3 constraint	Approach 3: while iteration of search-and-return for loop

Table I
DESIGN CHOICES FOR CONSTRAINTS

Code examples: A large example of this approach for a migration transformation is in [9]. The UML to relational mapping is also specified in this style in [10].

A simple example of the pattern is the specification of the three-cycles graph analysis in Section IV-C.

B. Recursive form

Synopsis: To specify the effect of a transformation in a declarative manner, as a global pre/post predicate, using a recursive definition of the transformation relation.

Forces: Useful whenever a platform-independent specification of a transformation is required, and the conjunctive-implicative form is not applicable, because an explicit description of the transformation relation as a single relation between the source and target models cannot be defined.

Solution: The *Cons* predicate should be split into separate disjuncts each relating one (or a group) of source model elements to one (or a group) of target model elements:

$$\exists s : S_i \cdot SCond_{i,j} \text{ and } \exists t : T_{i,j} \cdot Post_{i,j}$$

where the S_i are source entities, the $T_{i,j}$ are target model entities, $SCond_{i,j}$ is a predicate on s (identifying which elements the constraint should apply to), and $Post_{i,j}$ defines the mapping $\tau(s)$ of s in terms of s, t and other mapping forms $\tau(s')$ for some s' derived from s .

There should exist a measure $Q : \mathbb{N}$ on the state of a model, such that Q is decreased on each step of the recursion, and with $Q = 0$ being the termination condition of the recursion (no rule is applicable in this case). Q is an abstract measure of the time complexity of the transformation, the maximum number of steps needed to complete the transformation on a particular model. For quality-improvement transformations it can also be regarded as a measure of the (lack of) quality of a model.

Consequences: The proof of *Ens* properties from *Cons* is more indirect for this style of specification, typically requiring induction using the recursive definitions.

Implementation: The constraints can be used to define a recursive function that satisfies the specification, or an equivalent iterative form. The constraints can also be used to define pattern-matching rules in transformation languages such as ATL [6] or QVT [12].

Code examples: Many computer science problems can be expressed in this form, such as sorting, searching and scheduling. Update-in-place transformations, which usually employ a fixpoint iteration of transformation steps, can be specified using this pattern. For example, a transformation to remove multiple inheritance from a class diagram can be specified by constraints:

$$\begin{aligned} &(\exists c : Class; g : c.generalization \cdot \\ & \quad c.generalization \rightarrow size() > 1 \text{ and} \\ & \quad \exists a : Association \cdot a.end1 = c \text{ and} \\ & \quad \quad a.end2 = g.general \text{ and} \\ & \quad \quad a.multiplicity1 = ONE \text{ and} \\ & \quad \quad a.multiplicity2 = ZEROONE \text{ and} \\ & \quad \quad g.isDeleted()) \text{ or} \\ & (\forall c : Class \cdot c.generalization \rightarrow size() \leq 1) \end{aligned}$$

In this case

$$Q(smodel) = \sum_{c:Class \text{ non root}} (c.generalization \rightarrow size() - 1)$$

C. Auxiliary metamodel

Synopsis: The introduction of a metamodel for auxiliary data, neither part of the source or target language, used in a model transformation.

Forces: Useful whenever auxiliary data needs to be used in a transformation: such data may simplify the transformation definition, and may permit a more convenient use of the transformation, eg., by supporting decomposition into sub-transformations. A typical case is a query transformation which counts the number of instances of a complex structure in the source model: explicitly representing these instances as instances of a new (auxiliary) entity may simplify the transformation.

Solution: Define the auxiliary metamodel as a set of (meta) attributes, associations, entities and generalisations extending the source and/or target metamodels. These elements may be used in the succedents of *Cons* constraints (to define how the auxiliary data is derived from source model data) or in antecedents (to define how target model data is derived from the auxiliary data).

Figure 2 shows a typical structure of this pattern. The auxiliary metamodel simplifies the mapping between source and target by factoring it into two steps.

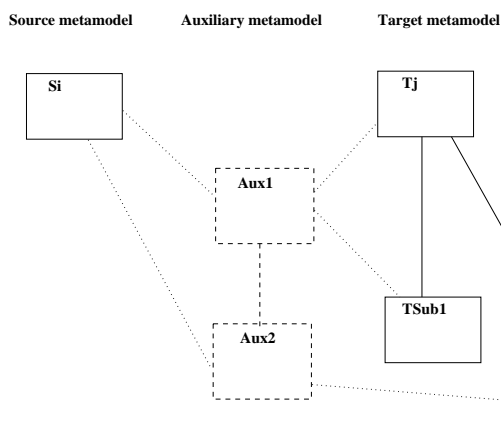


Figure 2. Auxiliary metamodel structure

Consequences: It may be necessary to remove auxiliary data from a target model, if this model must conform to a specific target language at termination of the transformation. A final phase in the transformation could be defined to delete the data (cf. the construction and cleanup pattern).

Code example: An example is a transformation which returns the number of cycles of three distinct nodes in a graph. This problem can be elegantly solved by extending the basic graph metamodel by defining an auxiliary entity *ThreeCycle* which records the 3-cycles in the graph (Figure 3).

The auxiliary language elements are shown with dashed lines.

The specification *Cons* of this transformation then defines how unique elements of *ThreeCycle* are derived from the graph, and returns the cardinality of this type at the end

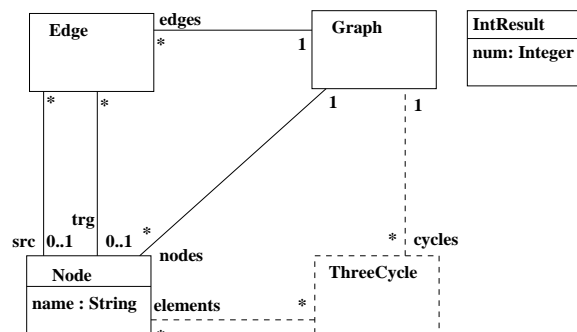


Figure 3. Extended graph metamodel

state of the transformation:

$$\begin{aligned}
 (C1) : & \\
 \forall g : Graph \cdot \forall e1 : g.edges; e2 : g.edges; e3 : g.edges \cdot & \\
 e1.trg = e2.src \text{ and } e2.trg = e3.src \text{ and } & \\
 e3.trg = e1.src \text{ and } & \\
 (e1.src \cup e2.src \cup e3.src) \rightarrow size() = 3 \text{ implies } & \\
 \exists_1 tc : ThreeCycle \cdot & \\
 tc.elements = (e1.src \cup e2.src \cup e3.src) & \\
 \text{and } tc : g.cycles &
 \end{aligned}$$

$$\begin{aligned}
 (C2) : & \\
 \forall g : Graph \cdot \exists r : IntResult \cdot r.num = g.cycles \rightarrow size() &
 \end{aligned}$$

The alternative to introducing the intermediate entity would be a more complex definition of the constraints, involving the construction of sets of sets using OCL *collect*.

Tracing is another example, which is often carried out by using auxiliary data to record the history of transformation steps within a transformation.

This pattern is referred to as *intermediate structure* in [4].

Related patterns: This pattern extends the conjunctive-implicative and recursive form patterns, by allowing constraints to refer to data which is neither part of the source or target languages.

D. Construction and cleanup

Synopsis: To simplify a transformation specification by separating it into a phase which constructs model elements, followed by a phase which deletes elements.

Forces: Useful when a transformation needs to create and delete elements of entities. For example, because an auxiliary metamodel is being used, whose elements must be removed from the final target model.

Solution: Separate the creation phase and deletion phase into separate sets of constraints, usually the creation (construction phase) will precede the deletion (cleanup). These can be implemented as separate transformations, each with a simpler specification and coding than the single rule.

Consequences: The pattern leads to the production of intermediate models (between construction and deletion) which may be invalid as models of either the source or target languages. It may be necessary to form an enlarged language for such models.

Code examples: An example is migration transformations where there are common entities between the source and target languages [11]. A first phase copies/adapts any necessary data from the old version (source) entities which are absent in the new version (target) language, then a second phase removes all elements of the model which are not in the target language. The intermediate model is a model of a union language of the source and target languages.

Another example are complex quality improvement transformations, such as the removal of duplicated attributes [7]. These can involve addition and removal of elements in a single step, and can be re-expressed more simply by separating these actions into successive steps.

Another implementation strategy for this pattern is to explicitly mark the unwanted elements for deletion in the first phase, and then to carry out the deletion of marked elements in the second phase.

V. IMPLEMENTATION PATTERNS

In this section we define patterns to organise the implementation of model transformations.

A. Phased creation

Synopsis: Construct target model elements in phases, ‘bottom-up’ from individual objects to composite structures, based upon a structural dependency ordering of the target language entities.

Forces: Used whenever the target model is too complex to construct in a single step. In particular, if an entity depends upon itself via an association, or two or more entities are mutually dependent via associations. In such a case the entity instances are created first in one phase, then the links between the instances are established in a subsequent phase.

Solution: Decompose the transformation into phases, based upon the *Cons* constraints. These constraints should be ordered so that data read in one constraint is not written by the same or a subsequent constraint, in particular, phase $p1$ must precede phase $p2$ if it creates instances of an entity $T1$ which is read in $p2$.

Figure 4 shows the schematic structure of this pattern.

Consequences: The stepwise construction of the target model leads to a transformation implementation as a sequence of phases: earlier phases construct elements that are used in later phases.

Implementation: The constraints are analysed to determine the dependency ordering between the target language data and entities. $T1 < T2$ means that a $T1$ instance is used in the construction of a $T2$ instance. Usually this is because

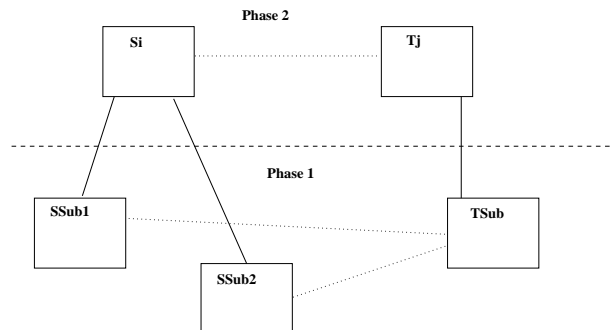


Figure 4. Phased creation structure

there is an association directed from $T2$ to $T1$, or because some feature of $T2$ is derived from an expression using $T1$ elements.

If the order $<$ is a partial order (transitive, antisymmetric and irreflexive) then the corresponding ordering of phases follows directly from $<$: a phase that creates $T2$ instances must follow all phases that create $T1$ instances, where $T1 < T2$. However, if there are self-loops $T3 < T3$, or longer cycles of dependencies, then the phases creating the entities do not set the links between them, instead there must be a phase which follows all these phases which specifically sets the links.

Code examples: The *ThreeCycle* example illustrates the simple case. Here $ThreeCycle < IntResult$, so the phase implementing $C2$ must follow that for $C1$.

B. Unique instantiation

Synopsis: To avoid duplicate creation of objects in the target model, a check is made that an object satisfying specified properties does not already exist, before such an object is created.

Forces: Required when duplicated copies of objects in the target model are forbidden, either explicitly by use of the $\exists_1 t : T_j \cdot Post$ quantifier, or implicitly by the fact that T_j possesses an identifier (primary key) attribute.

Solution: To implement a specification $\exists_1 t : T_j \cdot Post$ for a concrete class T_j , test if $\exists t : T_j \cdot Post$ is already true. If so, take no action, otherwise, create a new instance t of T_j and establish $Post$ for this t .

In the case of a specification $\exists t : T_j \cdot t.id = x$ and $Post$ where id is a primary key attribute, check if a T_j object with this id value already exists: $x \in T_j.id$ and if so, use the object ($T_j[x]$) to establish $Post$.

Consequences: The pattern ensures the correct implementation of the constraint. It can be used when we wish to share one subordinate object between several referring objects: the subordinate object is created only once, and is subsequently shared by the referrers. There is, however, an additional execution cost of carrying out checks for existing elements.

Implementation: The executable ‘update form’ in Java of $\exists_1 t : T_j \cdot Post$ for a concrete class T_j is:

```
if (qf) { }
else
{ uf }
```

where qf is the query form of $\exists_1 t : T_j \cdot Post$, and uf is its update form.

The pattern is used in a number of model transformation languages, such as QVT-R, to avoid recreating target elements with required properties. In QVT-R it is known as the ‘check before enforce’ strategy.

Related patterns: Object Indexing can be used to efficiently obtain an object with a given primary key value in the second variant of the pattern.

C. Object indexing

Synopsis: All objects of a class are indexed by a unique key value, to permit efficient lookup of objects by their key.

Forces: Required when frequent access is needed to objects or sets of objects based upon some unique identifier attribute (a primary key).

Solution: Maintain an index map data structure $cmap$ of type $IndType \rightarrow C$, where C is the class to be indexed, and $IndType$ the type of its primary key. Access to a C object with key value v is then obtained by applying $cmap$ to v : $cmap.get(v)$.

Figure 5 shows the structure of the pattern. The map $cmap$ is a qualified association, and is an auxiliary metamodel element used to facilitate separation of the specification into loosely coupled rules.

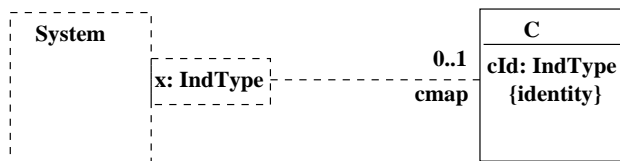


Figure 5. Object indexing structure

Consequences: The key value of an object should not be changed after its creation: any such change will require an update of $cmap$, including a check that the new key value is not already used in another object.

Implementation: When a new C object c is created, add $c.ind \mapsto c$ to $cmap$. When c is deleted, remove this pair from $cmap$. To look up C objects by their id, apply $cmap$.

In QVT-R the pattern is implemented by defining *key* attributes by which objects can be uniquely identified.

VI. CONCLUSION

We have described four specification patterns and three implementation patterns, which can be used together within a development process for model transformations. These

have been implemented within the UML-RSDS toolset. Other patterns which are widely used in model transformations are the *Recursive descent* pattern, where an implementation is structured as a series of recursive operations, using the hierarchical structure of source and target language entities [13].

ACKNOWLEDGMENT

This paper describes work carried out in the UK HoRT-MoDA project, funded by EPSRC.

REFERENCES

- [1] J. Bevizin, F. Jouault, J. Palies, *Towards Model Transformation Design Patterns*, ATLAS group, University of Nantes, 2003.
- [2] J. Cabot, R. Clariso, E. Guerra, J. De Lara, *Verification and Validation of Declarative Model-to-Model Transformations Through Invariants*, Journal of Systems and Software, preprint, 2009.
- [3] J. Crupi, D. Alur, D. Malks, *Core J2EE Patterns*, Prentice Hall, 2001.
- [4] K. Czarniecki, S. Helsen, *Feature-based survey of model transformation approaches*, IBM Systems Journal, vol. 45, no. 3, 2006, pp. 621–645.
- [5] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
- [6] F. Jouault, I. Kurtev, *Transforming Models with ATL*, in MoDELS 2005, LNCS Vol. 3844, pp. 128–138, Springer-Verlag, 2006.
- [7] K. Lano, *Class diagram rationalisation case study*, Dept. of Informatics, King’s College London, 2011.
- [8] K. Lano (ed.), *UML 2 Semantics and Applications*, Wiley, New York, 400 pages, 2009.
- [9] K. Lano, S. Kolahdouz-Rahimi, *Migration case study using UML-RSDS*, TTC 2010, Malaga, Spain, July 2010.
- [10] K. Lano, S. Kolahdouz-Rahimi, *Model-driven development of model transformations*, ICMT 2011, June 2011.
- [11] K. Lano, S. Kolahdouz-Rahimi, *Specification of the GMF migration case study*, TTC 2011.
- [12] OMG, *Query/View/Transformation Specification*, ptc/05-11-01, 2005.
- [13] OMG, *Query/View/Transformation Specification*, annex A, 2010.
- [14] I. Poernomo, *Proofs as model transformations*, ICMT 2008.
- [15] I. Poernomo, J. Terrell, *Correct-by-construction Model Transformations from Spanning tree specifications in Coq*, ICFEM 2010.

Component-oriented Software Development with UML

Nara Sueina Teixeira

Graduate Program in Computer Science
Federal University of Santa Catarina - UFSC
Florianópolis, Brasil
E-mail: narasueina@inf.ufsc.br

Ricardo Pereira e Silva

Department of Informatics and Statistics - INE
Federal University of Santa Catarina - UFSC
Florianópolis, Brasil
E-mail: ricardo@inf.ufsc.br

Abstract— This paper proposes to automate the process of structural and behavior analysis of component-oriented software fully specified in UML. The structural specification uses component, class and deployment diagrams, and the behavior specification, state machine diagram. The produced structural analysis tool analyzes a connection between pairs of components at a time. The produced behavioral analysis tool considers the behavior of the system as a whole, leading to behavioral specification of the application automatically from the machine state of each connected component. It is performed the conversion of the state machines of the individual components and of the application to Petri nets in a transparent manner to the user. The behavioral assessment is done by analyzing Petri net properties, considering the context of the components. Analysis results are produced without demand effort, allowing early location of design problems.

Keywords-Component-oriented development; structural compatibility analysis; behavioral compatibility analysis; UML; Petri Nets.

I. INTRODUCTION

For the component-based software development approach, software construction consists in an interconnection of a collection of units: the components. “A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment” [1]. “A component interface (CI) is a collection of service access points, each one with a defined semantics” [2]. The latter establishes the services required and provided by a component, not considering implementation details.

Some research efforts suggest the automation of component compatibility analysis evaluating their CIs. Dias and Vieira [3] use the Argus-I tool integrated to the SPIN tool [4] for the component compatibility analysis, in which specifications are produced in ADL (Architecture Description Language) and state machine diagram is converted to PROMELA [5]. The architectural analysis considers the "super state model", but the authors do not detail how it is generated.

Chouali and Souquières [6] use refinement in B to prove the compatibility between two interfaces, through the tool AtelierB [7]. The CI specification is converted to the formal method B and consists of a data model associated with each component provided and the required interface. The interoperability does not cover behavioral aspects, therefore it does not assess the feasibility of the component-based application.

Mouakher, Lanoix and Souquières [8] improved the approach [6] by adding an interface protocol, described in PSM (Protocol State Machine), to the CI specification and proposing adapters when incompatible interfaces were identified. However, the analysis is also performed between two connected interfaces, disregarding problems associated with the whole set of application components. In [6] and [8] the notion of component port is not treated.

Bracciali, Brogi and Channel [9] describe the interface of components through IDLs (Interface Description Language) and they use a subset of Lambda Calculus to represent the behavior of components. This low level solution becomes difficult to be applied to describe complex systems.

The component compatibility analysis should be performed based on the CI specification and must consider three distinct aspects: structural, behavioral, and functional. “The structural aspect concerns the static features of a component and corresponds to the set of required and provided operation signatures of the CI. The behavioral aspect defines constraints in the invocation order of provided and required operations. The functional aspect describes what the component does, not necessarily going into details of its implementation” [10].

The lack of a widely accepted standard for the specification of CI makes the analysis of compatibility between components difficult and hence, their reuse. The second version of UML, called henceforth UML [1] provides mechanisms to deal with components, but does not establish a standard for complete specifications.

In a previous publication [11] were proposed ways of specifying component-oriented software and CI, in which the specification is based on the object-oriented paradigm and uses only UML diagrams. For the CI structural specification component and class diagrams are used and for the CI behavioral specification is utilized the state machine diagram. Thus, each component has its own state machine (SM) representing its externally observable behavior – being this observable behavior the sequence of required and provided operations performed during the component’s operation. The organization of components of an application is described by using the deployment diagram.

This paper proposes the automation of the component’s compatibility analysis process from the component-based software specification [11]. The approach used in this paper is implemented in the current version of the SEA environment [10] [12] [13], which uses UML. SEA is a development environment in which the object-oriented paradigm is used for production and use of reusable software

artifacts. Some tools were built in this environment to automate the analysis of structural and behavioral compatibility.

The structural analysis tool (SAT) handles at each time, pairs of connected ports. The behavioral analysis tool (BAT) considers not only the individual behavior of each component, but also the behavior of the system as a whole. It involves the entire set of application’s interconnected components at the same time. In this work, the application SM is automatically obtained from the union of the SMs of each connected component.

For the behavioral analysis, the UML state machine diagram is converted in Petri net (PN). This conversion is done automatically in a transparent way to the user, who does not need any knowledge of this modeling technique. Behavioral problems are identified through the interpretation of PN properties, considering the component context. The conversion method (of SM to PN) used in this study is similar to that proposed in [14], however, it only handles PN of the ordinary kind and presents particularities of the treaty context.

Functional compatibility analysis consists in evaluate if the execution steps of an operation are in agreement with the need of the component that invokes the operation. This kind of analysis is not automatable and is beyond the scope of this work.

The following sections are organized as: Section II presents concepts related to OCEAN / SEA, and Section III presents the approach to specify component-based software. In Section IV, the automated structural compatibility analysis is described, while in Section V, the behavioral analysis is presented. Software specification and analysis are supported by the tools inserted in SEA environment. Section VI presents how the evaluation of the produced tools occurred. The article ends with conclusions, in Section VII.

II. OCEAN/SEA IMPLEMENTATION

OCEAN [10] is an object-oriented framework for the domain of the software development environments. From this framework, SEA environment, a software development support, was built.

The software development using SEA starts with the production of a UML design specification. In this environment, a design specification is an object that aggregates models and concepts (that are objects) and includes relationships between these objects. Each kind of UML diagram is defined as a class related to the proper diagram elements, that is, to the classes that model the diagram elements.

In the SEA environment, tools are also defined as classes and they are related to one or more kinds of specification – the ones that can be handled by these tools. The tools can be produced to be accessed by a menu or to be automatically called in a specific situation.

Tools of an OCEAN-based environment are produced by means of framework extension (subclassing). There are three kinds of tool: editors (such as a diagram editor), converters (such as a code generator), and analyzers. The analyzers read

a design specification without changing it and produce reports with the specification features. The tools SAT and BAT are analyzers.

III. SPECIFICATION OF COMPONENT-BASED SOFTWARE

A. Structural Specification

The structural specification concerns to all the operation signatures of the CI. “The CI refers to the portion of the component responsible for communicating with its external environment. Taking into account the nomenclature of UML, the CI is composed by a port collection, each one associated to one or more UML interfaces” [11].

In this approach, producing the CI structural specification requires the specification of all interfaces associated with the component (in class diagram) and the definition of the component ports, associating required or provided interfaces to each of them in component diagram.

With the establishment of the interfaces related to the component ports through realization or dependency relationship, it becomes possible to check what operations are provided or required from a component’s port.

Figure 1 illustrates the structural specification of a hypothetical component, CompB, made in the SEA environment. At the right side there is the class diagram with the interfaces; at the left side, a component, in a component diagram, that is related to the declared interfaces.

In the SEA environment, the connection between components is made in the deployment diagram, linking the ports of connected components. Figure 2 illustrates a deployment diagram with a hypothetical software artifact consisting of the interconnection of three components. All the components must be declared in component diagram and all the interfaces, in class diagram.

B. Behavioral Specification

The CI behavioral specification sets restrictions on the invocation order of operations provided and required by the component. In this approach, the behavioral specification is represented by a UML state machine diagram. The basic idea is that each state represents a situation that occurs during the operation of a component, which is characterized by the operations required and provided that can be performed at the time. Each transition leaving a state represents the execution of an operation – provided or required – that can leave the component in the same state or lead to another state. Some conventions have been established:

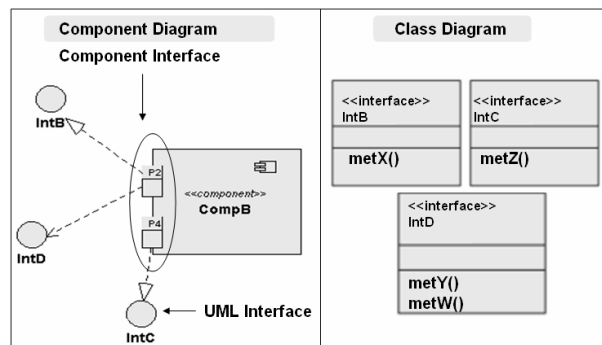


Figure 1. Component structural specification in the SEA environment.

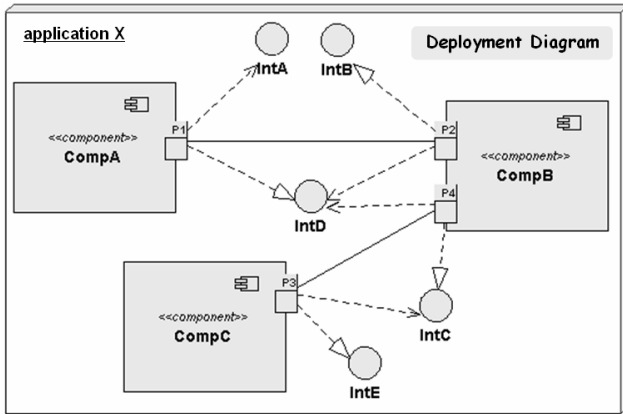


Figure 2. Software artifact consisting of the interconnection of components CompA, CompB, CompC.

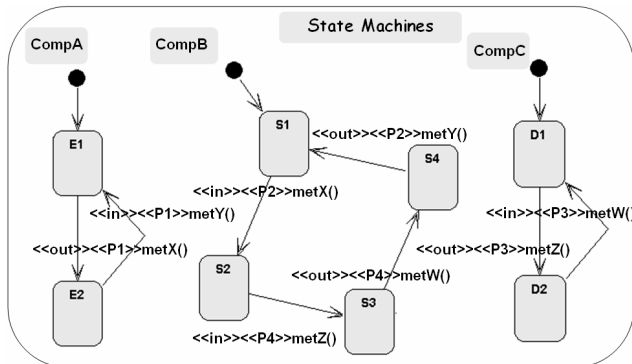


Figure 3. Behavioral specification of components CompA, CompB and CompC.

- The state identifiers are combinations of letters and numbers, which only differentiate a state of the others (the transitions are the elements that define the semantics of the model).

-The transitions are labeled according to the following convention: <direction> <port> <operation>, where <direction> may be <<out>> for the operations invoked by the component and <<in>> for provided operations.

Figure 3 illustrates the SMs of the components CompA, CompB and CompC (mentioned in Figure 2).

IV. AUTOMATION OF STRUCTURAL COMPONENTS'S COMPATIBILITY ANALYSIS

Figure 4 illustrates the SAT performance. Its purpose is to perform structural analysis, which consists in the following actions:

A. Structural Specification Consistency Analysis

The structural specification consistency analysis verifies if the system is specified with all restrictions set forth in approach, such as:

- All components are specified in a component diagram with at least one port associated to each one.
- Each port is associated with at least one required or provided interface.
- Each interface referenced in the component diagram is described in a class diagram.

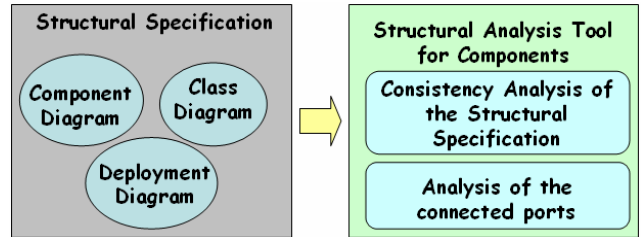


Figure 4. Structural Analysis Tool of the SEA environment (SAT).

- Each interface defined in the class diagram has at least one declared operation.

- At least two components are connected in a deployment diagram.

B. Connected Port Analysis

The structural compatibility is evaluated for each pair of connected ports of the application components. "The set of required operations by a port includes the operations of all interfaces related to that port by dependency. These operations should be provided by the port on the other side of the connection through its set of provided operations, in other words, the set of operations of all interfaces related to that port by realization" [11]. Otherwise, structural incompatibility is identified in the connection.

The analysis of the connected ports compares, for each pair of connected ports, the operations required in the port of a component with the operations provided by the port of the other component attached to it, considering operation name, return type, number of parameters and parameter type.

At the end of the analysis, SAT reports the results, with the structural incompatibilities found.

V. AUTOMATION OF THE COMPONENT BEHAVIORAL COMPATIBILITY ANALYSIS

Figure 5 illustrates the BAT operation in the SEA environment. In this approach, the behavioral analysis of components involves the following actions:

A. Behavioral Specification Consistency Analysis

The behavioral specification consistency analysis checks whether the specification complies with the restrictions established in the approach, such as those mentioned in Section III-B.

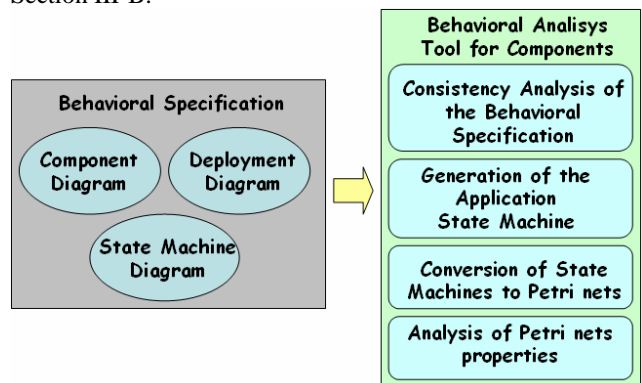


Figure 5. Behavioral Analysis Tool of the SEA environment (BAT).

For each analyzed SM, a behavioral specification evaluation report is generated, assessing the respective component, with the found errors.

B. Generation of the application behavioral specification

“Behavioral compatibility is observed between components if the restrictions on the operation invocation order of the required and provided operations established in each component are compatible with the other components connected to it. This type of evaluation involves the whole set of connected components” [11].

The method for generating the application SM was proposed in a previous work [11] and consists of:

1. Identifying pairs of related transitions. Two transitions are related if they involve interconnected ports and execution of the same operation, which is required on one side and provided by the other side;

2. Inserting fork and join pseudostates (a single syntactic element) that synchronizes the related transitions of the different machines. This link will convert the set of SMs in a single one – the component-based application SM – and synchronizes an operation invocation with its execution;

3. Synchronize the transitions of the initial pseudostates of various machines with a fork pseudostate (inserting a single initial pseudostate for the application SM). This step preserves the initial state of all SMs.

From this algorithm, the application SM will include the states of all involved components. Figure 6 illustrates the SM (automatically generated by BAT) of the application, consisting of the interconnection of components CompA, CompB and CompC, illustrated in the Figures 2 and 3.

C. Conversion of State Machines in Petri Nets

The user of the SEA environment manipulates only UML diagrams to specify component-based software. The SMs are converted into the corresponding PNs automatically, in a completely transparent way to the user, who never see PN diagrams.

The algorithm for conversion of the SMs in PNs is summarized in the following steps:

1. For each state of the SM, create a place in the PN.

2. Identify the states related to the initial pseudostate and mark the corresponding places with a token at each PN.

3. For each SM transition not related to another, create a transition and connect it with arcs to its input and output places (it applies to the SM transitions of the individual components and the application SM transitions corresponding to unconnected ports).

4. For each set of SM transitions related to a fork/join in the application, create a transition with a set of arcs connecting it to their respective input and output places.

Figure 7 illustrates the PN obtained from the conversion of the SM showed in Figure 6.

D. Petri net properties analysis

The Pipe analyzer tool – Platform Independent Petri net Editor 2, version 2.5 [15] – was integrated to the SEA environment, with adaptations and extensions. Given a PN,

the analyzer, through state enumeration, reports whether or not it has a certain property. The interpretation of each property is done for the treated context. The following properties are considered:

1. Safeness: the PNs that represent component-based applications must be safe. Otherwise, it denotes behavioral error.

2. Reversibility: in this study, the conclusion that a PN that represent component-based applications is not reversible causes a warning which should be evaluated by the user.

3. Deadlock: a deadlocked PN characterizes a behavioral error. This can occur for two reasons: one is because the restrictions associated to the execution order of the operations, established by a component, are not respected by other components connected to it. Another reason is the occurrence of unconnected port(s) in one or more application components. It occurs when the component requires or provides operations, through this port, which are essential to its operation.

4. Liveness: an alive PN representing a component-based application characterizes a behavioral specification without errors. However, the absence of this property does not necessarily denote behavioral error. A not alive PN may have almost alive or dead transitions and is the user's responsibility to assess whether or not this is a behavioral compatibility problem.

5. Almost alive transitions: this characteristic leads to a warning, because it is necessary that the user evaluates if the unavailability of an operation, at a certain moment of the execution, is a behavior compatibility problem.

6. Dead transitions: this feature also requires the user evaluation, that is, if the permanent unavailability of an operation is a problem for the application.

7. Transition invariants: In the analysis of the application PN, the invariants are identified and compared with the invariants of the individual component PNs, because possible cyclic sequences of operations of a component may not be possible when it is connected to others.

The analysis of the PN properties is made for both application PN and individual component PNs. The interpretation that occurs to this case is the same as the application PN, except for the property deadlock: considering the specification of an individual component, deadlock means modeling inconsistency. It is necessary to compare situations that occur in the component behavior, but that no longer occur in the application, when the component is connected to others.

VI. PROPOSED APPROACH EVALUATION

Two emphases have been adopted in the evaluation process: the tools's ability to identify errors and suspicious situations (reported as warnings) and the appropriateness of the analysis approach. The evaluation of the implemented tools was performed with small applications, with a maximum of ten components. Specifications without errors and specifications with purposely inserted errors were treated by the analysis tools in order to evaluate all situations in which they should work.

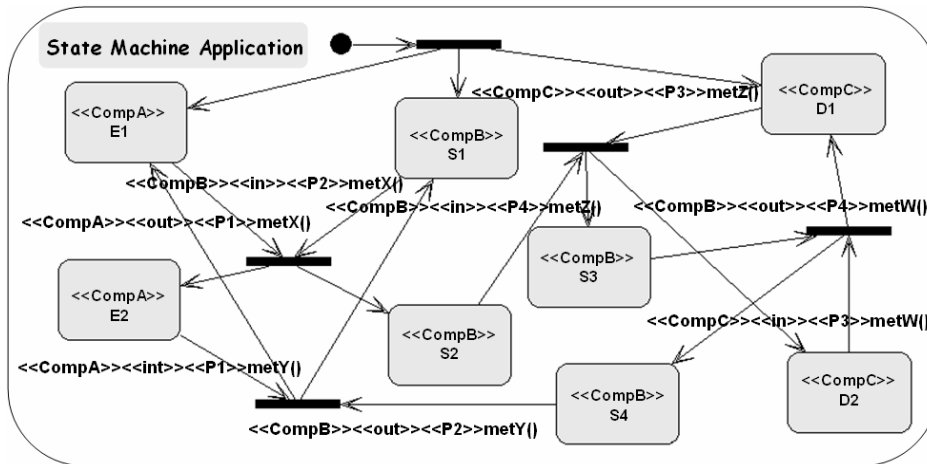


Figure 6. Behavioral specification corresponding to the application of the figure 2.

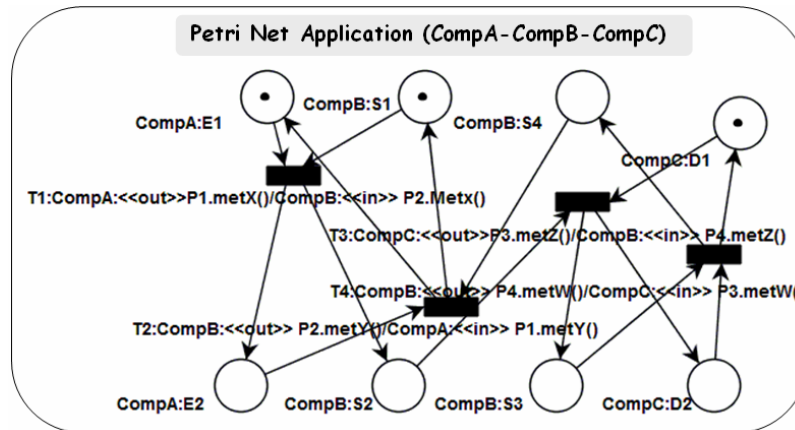


Figure 7. Petri net obtained from the conversion of the SM showed in Figure 6.

The analysis approach showed to be adequate when comparing their results with the conclusions of not automatic analysis. The tools were not submitted to stress test. The following are some analysis examples.

Figure 8 shows an exaple of a structural analysis report with error detection – in this case, operation not provided and problems with parameters. Figure 9 shows an example of behavioral analysis report with errors due to unconnected port, that is, a deadlock caused by the need of operation invocation in an unconnected port. Figure 10 shows another example of behavioral analysis report with warnings due the possible changes that may occur in the component behavior when it becomes part of a component connection. In this case, possible service execution cycles of an individual component not be preserved when it is connected to other components. Besides that, operations always available in the components become temporarily unavailable in the application that contains the components.

Based on reports like the ones showed, the user can make decisions and define corrective action related to the component and application specifications. For situations that represent warnings, the user must evaluate whether or not they mean a problem for the application.

VII. CONCLUSION AND FUTURE WORKS

This paper has presented an automatic procedure for the structural and behavioral compatibility analysis. The approach was implemented in the SEA environment, using tools embedded in it.

Component and class diagrams have been used for the CI structural specification. The behavioral aspect is defined using the state machine diagram. The component organization is defined using the deployment diagram.

The structural analysis tool evaluates whether the operations required on one side of the connection are provided by the component on the other side.

The behavioral analysis tool generates the application SM automatically. All SMs are converted into PNs, which are analyzed and interpreted in the given context.

The main advantages of this proposal are: the specification is made with just a single language, UML; the application behavioral specification is generated automatically, reducing the design effort; the behavioral analysis considers the behavior of individual components and application, comparing them and identifies errors and suspicious situations reported as warnings.

```

Model Analysed: Component
Type Analysis: Structural Compatibility
-----
Analyzing Component [CompA] port [P1]
connected Component [CompB] port [P2]
- Service [metX(par:String): String].
  Provided as [metX(par1:Integer): String].
  Divergent type of parameters.
- Service [metK(): String].
  Service NOT provided.
-----
(...)
-----
Analyzing Component [CompC] port [P3]
connected Component [CompB] port [P4]
- Service [metZ(): Boolean].
  Service [metZ] provided ok.
-----
Analyzing Component [CompB] port [P4]
connected Component [CompC] port [P3]
- Service [metY(): String].
  Service NOT provided.
- Service [metW(par1:String; par2: Integer): String].
  Provided as [metW(par1:String): Integer].
  Return type differs.
  Number of parameters differs.
-----
Structural Analysis with Errors
-----

```

Figure 8. Connected Port Analysis Report with error.

```

Model Analysed: Component
Type Analysis: Behavioral Compatibility
-----
(...)
-----
Behavior Analysed: Application
-----
- Error: Application locked due to unconnected ports.
- Port [P4] Component [CompB] is not connected with the following
  methods provided essential for its operation: [metZ]
- Port [P4] Component [CompB] is not connected with the following
  methods requireds waiting for connection: [metW]
- Cycles of the service execution
  Not identified.
-----
Behavioral Analysis with Errors
-----

```

Figure 9. Behavioral Analysis Report with error.

The developed tools automate the proposed analysis approach and the tests have shown the ability to automatically locate structural and behavioral errors.

As future work, we highlight the need of assessing the produced automatic support in the development of larger applications, the development of automated support to assist the creation of component adapters and to find alternatives to assess functional compatibility. Thus, we expect the possibility of producing component-oriented software specification more accurately, less prone to error, and improve its quality.

REFERENCES

[1] Object Management Group. Unified Modeling Language: Superstructure version 2.4. Available in: <<http://www.omg.org/spec/UML/2.4/Superstructure/Beta2/PDF>>. Access: 20 January 2011.

[2] C. Szyperski, Component Software: beyond object-oriented programming, 2.ed. Boston, EUA: Addison-Wesley Professional, 2002.

[3] M. S. Dias, and M.E.R. Vieira, "Software Architecture Analysis based on Statechart Semantics" in *Proceedings of the 10th International Workshop on Software Specification and Design*. [S.1]: IEEE Computer Society, 2000.p.133.ISBN 0-7695-0884-7.

[4] SPIN. Available in: <<http://spinroot.com/spin/whatispin.html>>. Access: 10 November 2010.

```

Model Analysed: Component
Type Analysis: Behavioral Compatibility
-----
Behavior Analysed: Component [CompP]
-----
- Component specified ok.
- Cycles of the service execution
  C1: metq(), metb()
  C2: metr(), metv()
-----
Behavior Analysed: Component [CompQ]
-----
- Component specified ok.
- Cycles of the service execution
  C3: metq(), metp()
  C4: metr(), metv()
-----
Behavior Analysed: Component [CompL]
-----
- warning: Component not restartable.
- warning: Component services temporarily available.
  Services: [metg] provided by the port [P2].
- Cycles of the service execution
  C5: metb(), metq(), metp()
-----
Behavior Analysed: Application
-----
- warning: Application not restartable.
- warning: Application services temporarily available.
  services: [metb] required by the component [CompP] at the port [P1] and
  provided by the component [CompL] at the port [P2].
  [metg] required by the component [CompP] at the port [P1] and
  provided by the component [CompL] at the port [P2].
  [metq] required by the component [CompL] at the port [P4] and
  provided by the component [CompQ] at the port [P3].
  [metp] required by the component [CompL] at the port [P4] and
  provided by the component [CompQ] at the port [P3].
- Cycles of the service execution
  C6: metr(), metv()
-----
Comparative table of component services
-----
Met/Comp| CompP | CompQ | CompL | Application
-----
metq()|available (C1) | - | temp.available | temp.available
metb()|available (C1) | - | available (C5) | temp.available
metr()|available (C2) | available (C4) | - | available (C6)
metv()|available (C2) | available (C4) | - | available (C6)
metq()| - | available (C3) | available (C5) | temp.available
metp()| - | available (C3) | available (C5) | temp.available
-----
warning: cycles [C1] [C3] and [C5] presents in the components and absent
in the application.
-----
Behavioral Analysis with warnings
-----

```

Figure 10. Behavioral analysis report with warnings.

[5] PROMELA. Process or Protocol Meta Language. Available in: <<http://www.dai-arc.polito.it/dai-arc/manual/tools/jcat/main/node168.html>>. Access: 10 November 2010.

[6] S. Chouali, M. Heisel, and J. Souquières, "Proving Component Interoperability with B Refinement," in *International Workshop on Formal Aspect on Component Software*, H. R. Arabnia and H.Reza, Eds. CSREA Press, 2005, to appear in ENCTS 2006.

[7] Atelier-B. Available in: <<http://www.atelierb.eu/index-en.php>>. Access: 10 November 2010.

[8] I. Mouakher, A. Lanoix, and J. Souquières, "Component Adaptation: Specification and Verification," in *Proceedings of the International Workshop on Component-Oriented programming, (WCOP)*. 2006

[9] A. Braccialia, A. Brogi, and C. Canal, "A formal approach to component adaptation," in *Journal of Systems and Software* Volume 74, Issue 1, 1 January 2005, Pages 45-54.

[10] R. P e Silva, "Suporte ao Desenvolvimento e Uso de Frameworks e Componentes," PhD Dissertation , Porto Alegre, UFRGS/II/PPGC, march 2000.

[11] R. P e Silva, Como Modelar com UML 2, Florianópolis: Visual Books, 2009. ISBN: 978-85-7502-243-6.

[12] A. Coelho, "Reengenharia do Framework OCEAN," M.Sc. Thesis, Florianópolis, UFSC. 2007.

[13] T. C. de S. Vargas, "Suporte à Edição de UML 2 no Ambiente SEA," M.Sc. Thesis, Florianópolis, UFSC. 2008.

[14] J. A. Saldhana and S. M. Shatz, "UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis," in *International Conference on Software Engineering and Knowledge Engineering*. Proc. of the Int. Conf. On Software Eng. and Knowledge Eng. (SEKE), Chicago, 2000.

[15] Pipe. Platform Independent Petri net Editor 2, versão 2.5. Available in: <<http://pipe2.sourceforge.net/>>. Access: 20 January 2011.

Metrics in Distributed Product Development

Maarit Tihinen and Päivi
Parviainen

Software Technologies
VTT Technical Research Centre of
Finland
maarit.tihinen@vtt.fi
paivi.parviainen@vtt.fi

Rob Kommeren

Digital Systems & Technology
Philips,
The Netherlands
r.c.kommeren@philips.com

Jim Rotherham

Project Management Office
Symbio,
Finland
jim.rotherham@symbio.com

Abstract— Nowadays the products are increasingly developed globally in collaboration between subcontractors, third party suppliers and in-house developers. However, management of a distributed product development project is proven to be more challenging and complicated than traditional one-site development. From the viewpoint of project management, the measurements and metrics are important activities for successful product development. This paper is focused on describing a set of metrics that is successfully used in industrial practice in distributed product development. Based on the experiences, the reasoning for selecting these metrics was similar: they are easy to capture and can be quickly calculated and analysed on a regular interval. One of the most important reasons for choosing these metrics was that they were aimed especially to provide early warning signals, i.e., means to proactively react to potential issues in the project. This is especially important in distributed projects, where specific means to track project status are needed.

Keywords-metrics; measurements; global software development; distributed product development

I. INTRODUCTION

Globally distributed software development enables product development to take place independently of the geographical location of the individuals or organizations. In fact, nowadays the products are increasingly developed globally in collaboration between subcontractors, third party suppliers and in-house developers [1]. In practice distributed projects struggle with the same problems than single-site projects including problems related to managing quality, schedule and cost. Distribution only makes it even harder to handle and control these problems [2][3][4][5]. These challenges are caused by various issues, for example, less communication – especially informal communication – caused by distance between partners, and differences in background knowledge of the partners. That's why, in distributed projects the systematic monitoring and reporting of the project work is especially important, and measurement and metrics are an important means to do that effectively.

Management of a distributed product development project is more challenging than traditional development [6].

Based on an industrial survey [7], one of the most important topics in the project management in distributed software development is detailed project planning and control during the project. In global software development (GSD), this includes, e.g., dividing work by sites into sub-projects, clearly defined responsibilities, dependencies and timetables, along with regular meetings and status monitoring.

The main purpose of measurements and metrics in software production is to create means for monitoring and controlling and this way to provide support for decision making [8]. Traditionally, the software metrics are divided into process, product and resource metrics [9]. In the comprehensive measurement program, all these dimensions should be taken into consideration while interpreting measurement results, otherwise, the interpretation may lead to wrong decisions or incorrect actions. Successful measurement program can prove to be an effective tool for keeping on top of development effort, especially, for large distributed projects [10]. However, many problems and challenges have been identified that reduce and may even eliminate all interests to the measurements. For example, not enough time is allocated for measuring and metrics during a project, or not enough benefit is visibly gained by the project doing the measurement work (e.g., data is useful only at the end of project, not during the project). In addition, the "metric enthusiasts" may define too many metrics making it too time consuming. Thus, it's beneficial [10] to define core metrics to collect across all projects to provide benchmarking data for projects, and to build on measures that come naturally out of existing processes and tools.

This paper is focused on describing a metrics set that are successfully used in distributed product development. The main purpose of the paper is to offer a set of essential metrics with experiences of their use. The amount of the metrics is knowingly kept as limited as possible. Also, the metrics should be such, that they provide online information during the projects, in order to enable fast reaction to potential problems during the project. The metrics and experience presented in the paper are based on metrics programs of two companies, Philips and Symbio. Royal Philips Electronics is a global company providing healthcare, consumer life-style

and lighting products and services. Digital Systems & Technology is a unit within Philips Research that develops first of a kind products in the area of healthcare, well-being and lifestyle. The projects follow a defined process and are usually distributed over sites and/or use subcontractors as part of product development. Symbio Services Oy provides tailored services to organizations seeking to build tomorrow's technologies. Well-versed in a variety of software development methodologies and testing best practices, Symbio's specialized approaches and proprietary processes begin with product design and stem through globalization, maintenance and support. Symbio has built a team of worldwide specialists that focus on critical areas of the product development lifecycle. Currently Symbio employs around 1400 people and their project execution is distributed between sites in the US, Sweden, Finland and China.

The paper is structured as follows. Firstly, an overview of related work – literature studies and their limitations related to measurements and metrics of distributed product development – is introduced in Section II. Then, proposed metrics are presented using Rational Unified Process (RUP) [11] approach as a framework. After that, industrial experiences of using the metrics are discussed. Finally, the conclusions are drawn in Section V.

II. MEASUREMENTS IN GSD

There are several papers that discuss globally distributed software engineering and its challenges, for example, [10], [12] and [13]. Also, metrics in general and for specific aspects have been discussed in numerous papers and books for decades. However, little global software development (GSD) literature has focused on metrics and measurements or even discusses the topic. Da Silva et al. [6] report similar conclusion based on analysis of DSD literature published during 1999 – 2009: they state as one of their key finding that the “vast majority of the reported studies show only qualitative data about the effect of best practices, models, and tools on solving the challenges of distributed software development (DSD) project management. In other words, our findings indicate that strong (quantitative) evidence about the effect of using best practices, models, and tools in DSD projects is still scarce in the literature.”

The papers that have discussed some metrics for GSD usually focus on some specific aspect, for example, Korhonen and Salo [13], discuss quality metrics to support defect management process in a multi-site organization. Simmons and Ma [14] discuss a software engineering expert system (SEES) tool where the software professional can gather metrics from CASE tool databases to reconstruct all activities in a software project from project initiation to project termination. Misra [15] presents a cognitive weight complexity metric (CWCM) for unit testing in a global software development environment. Lotlikar et al. [16] propose a framework for global project management and governance including some metrics with main aim to support work allocation to various sites. Peixoto et al. [12] discuss effort estimation in global software development, and one of their conclusions is that “GSD projects are using all kinds of

estimation techniques and none of them is being considered as proper to be used in all cases that it has been used”, meaning, that there is no established technique for GSD projects.

Some effort has also been invested in defining how to measure success of GSD projects [17], and these metrics mainly focus on cost related metrics and are done after project completion. The focus of this paper is to discuss metrics for monitoring ongoing GSD projects and that way identify needs for corrective actions early.

A. Traditional metrics and project characteristics

Software measurements and metrics have been discussed since 1960's. The metrics have been classified many different ways, for example, they can be divided into basic and additional metrics [18] where basic metrics are size, effort, schedule and defects, and the additional metrics are typically metrics that are calculated or annexed from basic metrics (e.g., productivity = software size per used effort). The metrics can be divided also into objective or subjective metrics [18]. The objective metrics are easily quantified and measured, examples including size and effort, while the subjective metrics include less quantifiable data such as quality attitudes (e.g., excellent, good, fair, poor). An example of the subjective metrics is customer satisfaction. Furthermore, software metrics can be classified according to the entities of product, processes and resources [9]. Example metrics of product entities are size, complexity, reusability and maintainability. Example metrics of process entities are effort, time, number of requirements changes, number of specification/coding faults found and cost. Furthermore, examples of resource entities are age, price, size, maturity, standardization certification, memory size or reliability. These classifications, various viewpoints and the amount of examples merely prove how difficult the selection of metrics really can be during the project.

In addition to different ways of metrics classification, development projects can also be classified. Typically, the project classification is used as a baseline for further interpretation of the metrics and measurements. For example, all kind of predictions or comparison should be done within the same kind of development projects, or the differences should be taken into account. Traditional project characteristics are, e.g., size and duration of a project, type of a project (development, maintenance, operational lifetime etc.), project position (contractor, subcontractor, internal development etc.), type of software (hardware-related software development, application software, etc.) or used software development approaches (agile, open source, scrum, spiral-model, test driven development, model-driven development, V-model, waterfall model etc.). Furthermore, different phases of development projects have to be taken consideration while analyzing gathered measurement data.

B. Metrics and measurements during product development

A phase of lifecycle of development project affects to the interpretation of the metrics. Thus, in this paper, proposed metrics are introduced by using commonly known approach of software development Rational Unified Process (RUP). RUP is a process that provides a disciplined approach to

assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget [11].

The software lifecycle is divided into cycles, each cycle working on a new generation of the product. RUP divides one development cycle in four consecutive phases [11]: 1) inception phase, 2) elaboration phase, 3) construction phase and 4) transition phase. Furthermore, there can be one or more iterations within each phase during the software generation. The phases and iterations of RUP approach are illustrated in following Figure 1.

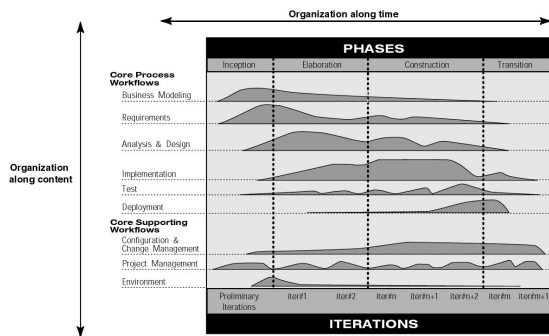


Figure 1. Phases and Iterations of RUP approach [11].

From a technical perspective the software development is seen as a succession of iterations, through which the software under development evolves incrementally [11]. From measurement perspective this means that some metrics can be focused on one or two phases of the development cycle, and some can be continuous metrics that can be measured in all phases, and can be analysed, e.g., in iterations.

C. Measurements and metrics in GSD

Software measurement is defined by [19] as follows: “The software measurements is the continuous process of defining, collecting and analysing data on the software development process and its products in order to understand and control the process and its products and to supply meaningful information to improve that process and its products”. In the daily software development work, the measurements are still seen as unfamiliar or even an extra burden for projects. For example, project managers feel it as time consuming to collect metrics for the organization (e.g., business-goal-related metrics) while they need to have metrics that are relevant to the project. Furthermore, they have impressed that there has not been budgeted enough time for measurements, and that’s why it’s really difficult to get approval from stakeholders for this kind of work [10].

Globally distributed development generates new challenges and difficulties for the measurements. For example, the gathering of the measurements data can be problematic because of different development tools or their versions, work practices with related concepts can vary by project stakeholders or reliability of the gathered data can vary due to cultural differences, especially, in subjective

evaluations. In addition, distributed projects are often so unique (e.g., product domain and hardware-software balance vary, or different subcontractors are used in different phases of the project) that their comparison is impossible. Thus, the interpretation of measurements data is more complicated in GSD than one site projects. That’s why it’s recommended to select moderate amount of metrics. In this paper we will present a set of metrics to use during GSD. Also industrial experiences about the metrics will be discussed.

The common metrics (effort, size, schedule etc.) are also applicable for GSD projects. However, special attention may be needed in training the metrics collection, to ensure common understanding of them (e.g., used classifications). Also, as measurements also tend to guide people’s behavior, it’s important to ensure that all are aware of the purpose of the metrics (i.e., not to measure individual performance), specifically in projects distributed over different cultures.

III. EXAMPLES OF INDUSTRIAL PRACTICES

In this Section the metric set used in the companies is introduced. The metrics are introduced according to the RUP phases where the metric is seen most relevant to measure. For each metric, a name, a notation and a detailed definition is introduced. The main goal is to offer a useful, yet a reasonable amount of metrics, for supporting the on-time monitoring of the GSD projects. Thus, the indicators are supposed to be leading indicators rather than lagging indicators, for example, planned / actual schedule measurements should be implemented as milestone trend analysis: measure the slip in the first milestone and predict the consequences for the other milestones and project end.

A. Metrics for Inception Phase

During the inception phase, the project scope has to be defined and the business case has to be established. The business case includes success criteria, risk assessment, and estimate of the resources needed, and a phase plan showing dates of major milestones. Inception is the smallest phase in the project, and ideally it should be quite short. Example outcomes of the inception phase are a general vision document of the core project's requirements, main constraints, an initial use-case model (10% -20% complete), and a project plan, showing phases and iterations [20]. Proposed metrics to be taken consideration in this phase are introduced in Table I.

TABLE I. METRICS FOR THE INCEPTION PHASE

Metric	Notation	Definition
Planned Schedule	$D_{PLANNED}$	The planned Date of delivery (usually the completion of an iteration, a release or a phase)
Planned Personnel	$\# FT_{PLANNED}$	The planned number of Full Time persons in the project at any given time
Proposed Requirements	$\# Reqs$	The number of proposed requirements.

The metrics Planned Schedule and Planned Personnel are mostly needed for comparison with actual schedule and

personnel, in order to identify lack of available resources as well as delays in schedule quickly. The amount of Proposed Requirements tells about the progress of the product definition.

B. Metrics for Elaboration Phase

During the elaboration phase a majority of the system requirements is expected to capture. The purpose of the phase is to analyze the problem domain, establish a sound architectural foundation, develop the project plan, and eliminate the highest risk elements of the project. The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the construction phase. Example outcomes of the elaboration phase are a use-case model (at least 80% complete), a software architecture description, supplementary requirements capturing the non-functional requirements and any requirements that are not associated with a specific use case, a revised risk list and a revised business case, and a development plan for the overall project. Proposed metrics to be taken consideration in this phase are introduced in Table II.

TABLE II. METRICS FOR THE ELABORATION PHASE

Metric	Notation	Definition
<u>Schedule:</u> Planned /Actual Schedule	$D_{PLANNED}$ D_{ACTUAL}	The planned/actual Date of delivery (usually the completion of an iteration, a release or a phase)
<u>Staff:</u> Planned /Actual Personnel	$\#FT_{PLANNED}$ $\#FT_{ACTUAL}$	The planned/actual number of Full Time persons in the project at any given time
<u>Requirements</u> -Proposed -Accepted -Not implemented	$\#Reqs_{PROP.}$ $\#Reqs_{ACCEP.}$ $\#Reqs_{NOT_IMPL.}$	The number (#) of - proposed requirements - reqs accepted by customer - not implemented reqs
<u>Tests</u> -Planned	$\#Tests_{PLANNED}$	The number (#) of - planned tests
<u>Documents:</u> -Planned -Proposed -Accepted	$\#Docs_{PLANNED}$ $\#Docs_{PROPOSED}$ $\#Docs_{ACCEPTED}$	The number (#) of planned /proposed /accepted documents to be reviewed during the project.

The metrics related to requirements, tests and documents indicate the technical progress of the project from different viewpoints. Staffing metric may explain deviations in the expected progress vs. the actual progress, both from technical as well as from schedule viewpoint. Note that those metrics that are more relevant to measure by iterations (e.g., effort and size) are introduced later (in Section E).

C. Metrics for Construction Phase

Construction is the largest phase in the project. During the phase, all remaining components and application features are developed and integrated into the product, and all features are thoroughly tested. System features are implemented in a series of short, time boxed iterations. Each iteration results in an executable release of the software. Example outcomes of the phase consist of a software product integrated on the adequate platforms, user manuals, and a description of the current release. Proposed metrics to be taken consideration in this phase are introduced in Table III.

Note that those metrics that are continuously measured are introduced later (in Section E).

TABLE III. METRICS FOR THE CONSTRUCTION PHASE

Metric	Notation	Definition
Planned /Actual Schedule Planned /Actual Personnel	$D_{PLANNED}$ D_{ACTUAL} $\#FT_{PLANNED}$ $\#FT_{ACTUAL}$	Defined in the elaboration phase.
<u>Requirements:</u> -Proposed -Accepted -Not implemented -Started -Completed	$\#Reqs_{PROP.}$ $\#Reqs_{ACCEP.}$ $\#Reqs_{NOT_IMPL.}$ $\#Reqs_{STARTED}$ $\#Reqs_{COMPLETED}$	The number (#) of - proposed requirements - reqs accepted by customer - not implemented reqs - reqs started to implement - reqs completed
<u>Change Requests:</u> -New CR -Accepted -Implemented	$\#CRs_{NEW}$ $\#CRs_{ACCEPTED}$ $\#CRs_{IMPL.}$	The number (#) of - identified new CR or enhancement - CRs accepted for implementation - CRs implemented
<u>Tests:</u> -Planned -Passed -Failed -Not tested	$\#Tests_{PLANNED}$ $\#Tests_{PASSED}$ $\#Tests_{FAILED}$ $\#Tests_{NOT TESTED}$	The number (#) of - planned tests - passed tests - failed tests - not started to test
<u>Defects</u> -by Priority: e.g., Showstopper, Medium, Low	$\#Dfs_{PRIORITY}$	The number (#) of - defects by Priority during the time period
<u>Documents:</u> -Planned -Proposed -Accepted	$\#Docs_{PLANNED}$ $\#Docs_{PROPOSED}$ $\#Docs_{ACCEPTED}$	Defined in the elaboration phase.

The metrics related to requirements, tests and documents indicate the technical progress of the project from different viewpoints. Metrics related to changes indicate both on the stability of the project technical content, and can explain schedule delays, and unexpected technical progress. Defect metrics tell both of the progress of testing, as well as maturity of the product.

D. Metrics for Transition Phase

The final project phase of the RUP approach is transition. The purpose of the phase is to transfer a software product to a user community. Feedback received from initial release(s) may result in further refinements to be incorporated over the course of several transition phase iterations. The phase also includes system conversions, installation, technical support, user training and maintenance. From measurements viewpoint the metrics identified in the phases relating to schedule, effort, tests, defects, change requests and costs are still relevant in the transition phase. In addition, customer satisfaction is generally gathered in the transition phase.

E. Metrics for Iterations

Each iteration results in an increment, which is a release of the system that contains added or improved functionality compared with the previous release. Each release is accompanied by supporting artifacts: release description, user’s documentation, plans, etc. Although most iterations will include work in most of the process disciplines (e.g.,

requirements, design, implementation, testing) the relative effort and emphasis will change over the course of the project. Proposed metrics to be taken consideration in this phase are introduced in Table IV.

TABLE IV. METRICS FOR ITERATIONS

Metric	Notation	Definition
<u>Effort:</u> -Planned Effort -Actual Effort	$E_{PLANNED}$ E_{ACTUAL}	The planned/actual effort required of any given iteration of the project.
<u>Size:</u> -Planned size -Actual size	$SIZE_{PLANNED}$ $SIZE_{ACTUAL}$	The planned /actual size of each iteration can be measured as SLOC (Source Lines of Code), Points or any other commonly accepted way.
<u>Cost:</u> -Budgeted -Expenditure	$COST_{BUDGET}$ $COST_{ACTUAL}$	The budgeted cost /actual expenditure for any given iteration.
<u>Velocity:</u> -planned /actual story points	$\#PTS_{PLAN}$ $\#PTS_{ACT}$	How many story points are planned to be /actually implemented of any given iteration of the project.
<u>Productivity:</u>	$\frac{E_{ACTUAL}}{\#PTS_{ACTUAL}}$	Use effort per actually implemented story points for each sprint /iteration

All of these metrics provide indication of the project progress and reasons for deviations should be analysed. These metrics should be analysed together with other metrics results (presented in Tables I-III) in order to gain comprehensive picture of the status.

IV. EXPERIENCES AND DISCUSSION

The metrics presented in previous section were common for both of the companies. Although the metrics were chosen independently by both companies, the reasoning behind choosing these metrics was similar. An important reason was to come from a re-active into a pro-active mode, i.e., to introduce ‘early warning’ signals for the project and management. Specifically these metrics have been chosen as they indicate a well-rounded view of status in the various engineering disciplines and highlight potential issues in the project. This creates real possibilities to act proactively based on signals gathered from various engineering viewpoints. This is especially important in GSD, where information of project status is not readily available but needs special effort, distributed over sites and companies. Accordingly, the metrics set can be seen as a ‘balanced score card’, on which management can take the right measures, balancing insights from time, effort (e.g., staffing), cost, functionality (requirements) and quality (tests) perspective.

An important aspect was also that the metrics are easy of capture and that they can be captured from the used tools “for free”, or can be quickly calculated at regular intervals. Costs and budgets are good examples of metrics that can be easily captured from the tools. This is also important from GSD viewpoint, as automated capturing reduces the chance of variations caused by differences in recording the metrics

data in different sites. Neither of the companies use metrics based on lines-of-code as they did not find it to be a reliable indicator of progress, size or quality of design.

As can be seen, the metrics are quite similar as in single site development. However, the metrics may be analysed separately for each site, and comparisons between sites can thus be made in order to identify potential problems early. Also, while interpreting or making decisions based on the measurement results the distributed development implications need to be taken into account. Distributed development requires ‘super-balancing’: how to come to the right corrective action if for instance, in one side the % of not accepted requirements is high, and in the other side the # of passed tests is lagging behind. Distributed development may also affect the actual results of the measurements. For example, relating to subjective metrics, such as effort estimation, differences between backgrounds of the people (e.g., cultural or work experience) in different sites may affect the result.

The companies also use the measurement results to gain insight into why a measure varies between similar single site and multi-site projects in order to try to reduce potential variances. This also partially explains the use of the same metrics as single-site development. Furthermore, the challenges in communication and dynamics of distributed teams mean that working practices need to be addressed continuously. However, in addition to metrics results, paying close attention and acting on feedback from retrospectives is as important, if not more important than drawing strong conclusions from metrics alone.

Currently, both companies are in process of revamping their metric usage, but feel confident that these metrics are the right ones. Easy implementation and by that easy acceptance is the most crucial thing to get these metrics as established practice within the company.

Both companies are careful in introducing new metrics, as it’s well known that too many metrics leads to overkill and rejection by the organization, and does not provide the right insights and indication for control measures. However, a potential measurement to be added to the set specifically from distributed development viewpoint, could be measurements related to time spent idling, i.e., waiting for something, and the time blocked because of the impediments elsewhere in the team as these affect productivity and highlight when a team is not performing. These additional metrics should be focused on measuring the project performance, especially task and team performance in GSD.

V. CONCLUSION

The management of the increasingly common distributed product development project is proven to be more challenging and complicated than traditional one-site development. Metrics are seen as important activities for successful product development as they provide means to effectively monitor the project progress. However, defining useful, yet reasonable amount of metrics is challenging, and

there is little guidance available for a company to define metrics for its distributed projects.

Globally distributed development generates new challenges and difficulties for the measurements. For example, the gathering of the measurements data can be problematic because of different development tools or their versions, work practices with related concepts can vary by project stakeholders or reliability of the gathered data can vary due to cultural differences, especially, in subjective evaluations. Furthermore, especially interpretation and decision-making based on the measurement results require that the distributed development implications are taken carefully into consideration.

This paper focused on describing a set of metrics that is successfully used in industrial practice in distributed product development. These metrics, are aimed especially to provide means to proactively react to potential issues in the project, and are meant to be used as a whole, not interpreted as single information of project status.

The metrics presented in the paper were common for both of the companies. Based on experiences, the reasoning for selecting these metrics was similar: they are easy to capture and can be quickly calculated and analysed at regular interval. Also, one of the most important reasons was that these metrics were aimed especially to provide means to proactively react to potential issues in the project. The balancing insights from time, effort, cost, functionality and quality was also seen as very important aspect.

ACKNOWLEDGMENT

This paper was written within the PRISMA project that is an ITEA 2 project, number 07024 [21]. The authors would like to thank the support of ITEA [22] and Tekes (the Finnish Funding Agency for Technology and Innovation) [23].

REFERENCES

- [1] J. Hyysalo, P. Parviainen, and M. Tihinen, "Collaborative embedded systems development: Survey of state of the practice," 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS 2006), IEEE, 2006, pp. 1-9.
- [2] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," In Proceedings of Future of Software Engineering FOSE '07, IEEE Computer Society, 2007, pp. 188-198.
- [3] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "Distance, dependencies, and delay in a global collaboration," In Proceedings of the ACM Conference on Computer Supported Cooperative Work, ACM, 2000, pp. 319-328.
- [4] M. Jiménez, M. Piattini, and A. Vizcaíno, "Challenges and improvements in distributed software development: A systematic review," *Advances in Software Engineering*, 2009, pp. 14.
- [5] S. Komi-Sirviö and M. Tihinen, "Lessons learned by participants of distributed software development," *Knowledge and Process Management*, vol. 12, (2), 2005, pp. 108-122.
- [6] F. Q. B. da Silva, C. Costa, A. C. C. França, and R. Prikładnicki, "Challenges and solutions in distributed software development project management: A systematic literature review," In Proceedings of International Conference on Global Software Engineering (ICGSE2010), IEEE, 2010, pp. 87-96.
- [7] S. Komi-Sirviö and M. Tihinen, "Great challenges and opportunities of distributed software development - an industrial survey," 15th International Conference on Software Engineering and Knowledge Engineering (SEKE2003), San Francisco, USA, 2003, pp. 489-496.
- [8] V. R. Basili, "Software modeling and measurement: The Goal/Question/Metric paradigm," 1992.
- [9] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co. Boston, MA, USA, 1998.
- [10] M. Umarji and F. Shull, "Measuring developers: Aligning perspectives and other best practices," *IEEE Software*, vol. 26, (6), 2009, pp. 92-94.
- [11] P. Kruchten, *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 2004.
- [12] C. E. L. Peixoto, J. L. N. Audy, and R. Prikładnicki, "Effort estimation in global software development projects: Preliminary results from a survey," In Proceedings of International Conference on Global Software Engineering, IEEE Computer Society, 2010, pp. 123-127.
- [13] K. Korhonen and O. Salo, "Exploring quality metrics to support defect management process in a multi-site organization - A case study," In Proceedings of 19th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2008, pp. 213-218.
- [14] D. B. Simmons and N. K. Ma, "Software engineering expert system for global development," In Proceedings of 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), IEEE, 2006, pp. 33-38.
- [15] S. Misra, "A metric for global software development environment," In Proceedings of the Indian National Science Academy 2009, pp. 145-158.
- [16] R. M. Lotlikar, R. Polavarapu, S. Sharma, and B. Srivastava, "Towards effective project management across multiple projects with distributed performing centers," In Proceedings of IEEE International Conference on Services Computing (CSC'08), IEEE, 2008, pp. 33-40.
- [17] B. Sengupta, S. Chandra, and V. Sinha, "A research agenda for distributed software development," In Proceedings of the 28th International Conference on Software Engineering, ACM, 2006, pp. 731-740.
- [18] K. H. Möller and D. J. Paulish, *Software Metrics: A Practitioner's Guide to Improved Product Development*. Institute of Electrical & Electronics Engineer, London, 1993.
- [19] R. Van Solingen and E. Berghout, *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill, 1999.
- [20] P. Kruchten, "A rational development process," *CrossTalk*, vol. 9, (7), 1996, pp. 11-16.
- [21] PRISMA, *Productivity in Collaborative Systems Development*, PRISMA project (2008-2011) homepage, URL: <http://www.prisma-itea.org/> (Accessed 1.6.2011).
- [22] ITEA 2, *Information Technology for European Advancement*, ITEA 2 homepage, URL: <http://www.itea2.org/> (Accessed 1.6.2011).
- [23] Tekes, the Finnish Funding Agency for Technology and Innovation, Tekes homepage. URL: <http://www.tekes.fi/eng/> (Accessed 1.6.2011).

Edola: A Domain Modeling and Verification Language for PLC Systems

Hehua Zhang
School of Software, KLISS, TNLIST
Tsinghua University
Beijing, China
Email: zhanghehua@gmail.com

Ming Gu
School of Software, KLISS, TNLIST
Tsinghua University
Beijing, China
Email: guming@tsinghua.edu.cn

Xiaoyu Song
Dept. ECE
Portland State University
Oregon, USA
Email: song@ee.pdx.edu

Abstract—Formal modeling and verification of PLC systems become paramount in engineering applications. The paper presents a novel PLC domain-specific modeling language Edola. Important characteristics of PLC embedded systems, such as reactivity, scan cycling, real-time and property patterns, are embodied in the language design. Formal verification methods, such as model checking and automatic theorem proving, are supported in Edola modeling. The TLA⁺ specification language constitutes an intermediate language layer between Edola and the verification tools, enhancing a large degree of reusability. A prototype IDE for Edola and its seamless integration of a model checker TLC and an automatic theorem prover Spass are implemented. A case study illustrates and validates the applicability of the language.

Keywords—domain-specific modeling language; formal verification; PLC; TLA⁺.

I. INTRODUCTION

Programmable Logic Controllers (PLCs) are widely used in industry for embedded systems [1]. A PLC interacts with its environment, following a so-called scan cycling mechanism. It starts with inputting environmental data, then performs a local computation, and finally outputs the results to the environment [2]. With their increasing use, PLC systems become more and more complex. Formal modeling and verification becomes paramount in PLC engineering applications to ensure the correctness.

There are several expressive formal modeling languages that has been adopted in the modeling and verification of PLC systems, such as timed automata [3], timed Petri net [4], SMV [5] and TLA⁺ [6].

Many PLC modeling work focus on a high level of abstraction, so that a small model can be obtained for verification. However, the characteristics like scan cycling are not considered, and a wide gap exists between the abstract models and their PLC implementations. To get a suitable level of abstraction to model PLC systems, their characteristics like reactivity, scan cycling, real-time and property patterns should be embodied in the modeling language. Although the existent formal modeling languages are powerful, the characteristics of PLC applications are not directly supported.

In this paper, we presented a novel PLC domain-specific modeling language Edola, which provides notations for

better understanding and easier modeling of applications in the PLC domain. Formal verification methods, like model checking and automatic theorem proving are supported in Edola modeling. Edola provides a suitable level of abstraction to model PLC systems, which can express features of PLC systems and also rule out unnecessary details. We adopt the TLA⁺ specification language as an intermediate layer between the Edola language and the verification tools, to enhance a large degree of reusability. With the inherent logic of TLA⁺, it is possible to verify an Edola model with a state based method like model checking and also a logic based reasoning method like theorem proving. A prototype IDE for Edola has been implemented, which provides both the user-interface for modeling and the seamless integration of two verification tools: TLC [7] and Spass [8].

The paper is organized as follows. We introduce the syntax, the intuitive semantics of Edola in Section III. The formal semantics is illustrated in Section IV. Section V explains the verification method of Edola models, including the transformation rules and the optimization strategies taken in the procedure. Section VI introduces the prototype IDE tool. A case study is illustrated in Section VII to validate the applicability of the language. Finally, we conclude our work in Section VIII.

II. AN EXAMPLE: FIRE-FIGHTING PLC CONTROL SYSTEM

To better explain the language Edola and its tool, we first introduce a fire-fighting PLC control system which is used in ship docks. We will take part of the case now and then, to explain the syntax, semantics and our design considerations of Edola.

This running case is a system used to fight fire that may happen at ship docks. It operates the fire-fighting cannons under the control of a user and displays information about the current operating state. The cannons are used in some specified fire cases and are connected with several valves. When there is a fire-fighting request, the user can control the equipments in the control panel. A possible designed control panel with two cannons and two fire-fighting cases are sketched in Figure 1. The preparatory steps of the operations are as follows: (1) powering up the system; (2)

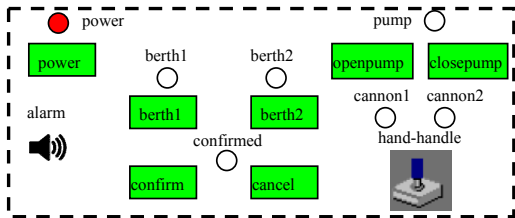


Figure 1. A sketch of the control panel of the system

opening the pump; (3) selecting the firing place (case); (4) confirming the selection; (5) controlling the direction of the cannon by the hand-handle. After the fire fighting, the user proceeds as follows to shut down the system: (6) closing the pump; (7) canceling the current fire-fighting case; (8) restarting the system for another fire-fighting case or powering down the system to finish. Note that because of the technic requirement, only one cannon can be used at the same time. On the other hand, the system should consider the case when the controlled devices or the communication network go wrong, and the users should be alarmed. In this example, we consider a typical alarm, that is, when the PLC program sends the command to open the pump, while the pump does not open in 5 seconds, the beeper in the control panel rings 3 seconds and the control system goes to the initial state.

III. THE SYNTAX OF EDOLA

An outline of the Edola syntax is shown in Figure 2. Details are omitted for the sake of space limitation. The Edola language is composed by modules. The main body includes module extensions, static declarations, dynamic definitions and verification requests. An EXTENDS statement can extend standard modules like Naturals, Reals in Edola or user-defined modules.

The auxiliary symbols are declared by a series of Edola formulas (**GeneralDef**). Taking the fire-fighting case as an example, we can use $Direction == \{“up”, “down”, “left”, “right”, “none”\}$ to denote the possible moving directions of a cannon, where *Direction* is defined as a enumeration type with 5 elements.

Constant declarations start with **CONSTANT**, denoting the parameters of a module. The declared parameters cannot be changed in the latter dynamic definitions.

Variable declarations are classified by the input variables (**INPUTVAR**), output variables (**OUTPUTVAR**) and system variables (**SYSTEMVAR**), according to the PLC scan cycling mechanism. Input variables denote the environment of a PLC software, including the commands from users by the control panels and the signals from the physical devices. The values of input variables are unchanged during a single scan cycle. Output variables denote the output signals of a PLC software, controlling the moving of physical devices or

```

EDOLA-module ::= AtLeast4("-") MODULE ModuleName (AtLeast4("-")
  ( nil | EXTENDS CommaList( Name) )
  GeneralDef
  Declarations
  ActionDef
  Constraints
  ( nil | Properties)
  AtLeast4("=")

GeneralDef ::= nil | ( formula )*
Formula ::= LeftF '=' Exp
LeftF ::= Name | Name "(" CommaList(ID) ")"
Declarations ::= ConstDeclarations VarDeclarations
ConstDeclarations ::= nil | CONSTANT CommaList(OpDec)
OpDec ::= ConstName | ConstName "(" CommaList("_") ")"
VarDeclarations ::= INPUTVAR varDecList
  OUTPUTVAR varDecList
  SYSTEMVAR varDecList
ActionDef ::= INIT formula
  ACTION ActionList
ActionList ::= ( Formula )+
Constraints ::= EnvConstraint
  ( nil | TimeConstraints )
EnvConstraint ::= ENV TOTAL
  | ENV ( Formula )*
TimeConstraints ::= TIME ( Duration | Interval | Delay | Deadline | Timeout | Waituntil ) +
Properties ::= PROP PropName ":"
  ( Respond | Compete | Sequence | Priority | Inv | ActInv )+
Respond ::= ( nil | Quantif ) RESPOND "(" Actname, SysStateExp, EnvStateExp ")"
Quantif ::= (A|E) Name \in SetName
.....

```

Figure 2. The excerpt of the Edola syntax

displaying the status of the system in control panels. System variables are used by the PLC software to implement the controlling functionalities. The values of system variables are usually changed during a scan cycle. The variables are defined by their name and type. For example,

$$\begin{aligned}
 & \text{INPUTVAR } realPump \in \text{BOOLEAN} & (1) \\
 & \text{OUTPUTVAR } alarm \in \text{BOOLEAN}, \\
 & \quad s_handle \in [Cannon \rightarrow Direction] \\
 & \text{SYSTEMVAR } state \in SysState
 \end{aligned}$$

declares an input boolean variable *realPump*, which denotes the opening state of the water pump; an output boolean variable *alarm* to denote the beeper rings or not, the other output variable *s_handle* representing the control commands to the water cannons, which is an array represented in the functional style. A system variable *state* is also declared to represent the current state of the PLC software.

The dynamic definitions describe how the PLC software works in a specified environment. The behaviors of PLC software are defined by an initial state and a series of actions in Edola. The initial state is represented by the keyword **INIT**

following an Edola formula, to assign the initial value for all the variables. The keyword ACTION starts the definition of a series of actions. Each action is represented by an Edola formula, to define the next-state action of PLC software in the current system state. Edola permits the definition of parameterized actions. For example, the following formula

$$\begin{aligned}
 & \text{SelectCase}(i) == & (2) \\
 & (\forall j \in \text{UnlockedButton} : j \neq i \Rightarrow u_button[j] \\
 & \quad i \in \text{FireCase} \\
 & \wedge u_button[\text{"power"}] \\
 & \wedge s_sysState \in \{\text{"pumpopened"}, \text{"selected"}\} \\
 & \wedge u_button[i] \\
 & \wedge s_buttonLight' = [j \in \text{ButtonLight} \mapsto \\
 & \quad \text{IF } j \notin \text{FireCase} \text{ THEN } s_buttonLight[j] \text{ ELSE} \\
 & \quad \text{IF } j = i \text{ THEN TRUE ELSE FALSE}] \\
 & \wedge s_sysState' = \text{"selected"}
 \end{aligned}$$

defines the on-fire case selecting action, which means that, for any fire case, if the power is on, the system is in the expected state, only the button for fire case i is pushed down, the case i is successfully selected. Only the corresponding light for case i is set to on and the system state is modified accordingly. The type of the parameter should be specified in the formula definition, which will be checked by the Edola compiler.

PLC applications are reactive, thus the environment should also be specified besides the behaviors of a PLC software. Edola provides two possibilities for environmental modeling. Users can define the specific behaviors of the environment by a series of formulas starting with the keyword ENV or use the keyword ENV TOTAL to leave the environment modeling work to Edola compiler. In the latter case, the compiler will generate a complete environmental model automatically, which covers all the possibilities of the environmental inputs.

When there are time constraints on system behaviors, they can be described in Edola by the part starting with the keyword TIME . We provide several time operators for describing the constraints on an action (*Duration*) or on the interval between actions (*Interval*), respectively. Four advanced operators *Delay*, *Deadline*, *Timeout* and *WaitUntil* are also supported for the usability. For example, the opening pump time limit 5 can be represented in Edola by applying the *Timeout* operator on the action *OpenPump* and the action *BeeperRing*: `TIMEOUT (OpenPump, BeeperRing, 5)`.

The verification requests are represented in Edola by a series of properties, and start with the keyword PROP . The given properties should be checked whether they are satisfied by the PLC software behaviors under the specified environment and the requested time constraints. Edola provides six property patterns: the responding properties (with the keyword RESPOND), the competing properties (COMPETE), the sequential properties (SEQUENCE), the priority properties (PRIORITY) and two patterns more general: state

invariants (STATEINV) and action invariants (ACTINV). For example, in a fire-fighting application, the correctness property

$$\begin{aligned}
 & \text{CannonUsedOnlyOne} : & (3) \\
 & \backslash A i \backslash in \text{Cannon}, j \in \text{Cannon} : \\
 & \text{COMPETE } (i \neq j, \text{Selected}[i], \text{Selected}[j])
 \end{aligned}$$

denotes the competing requests among selection of cannons: at any moment, at most one cannon can be selected. Note that all the provided property patterns are safety properties, which denotes in general that something bad will never happen.

IV. THE FORMAL SEMANTICS OF EDOLA

In this section, we give the formal semantics of Edola by the transformational method with the specification language TLA⁺.

A. Preliminaries of TLA⁺

TLA⁺ [9] is a formal specification language based on the Temporal Logic of Actions TLA, first-order logic and Zermelo-Fränkel set theory. It is un-typed, abstracted and widely used in the high-level specification of concurrent and reactive systems.

The characteristic form of the TLA⁺ specification of a transition system is a formula of the form $Spec \triangleq Init \wedge \square[Next]_{vars} \wedge L$, where $vars$ is a tuple containing all state variables of the system. The first conjunct *Init* describes the possible initial states of the system. The second conjunct of the specification asserts that every step (i.e., every pair of successive states in a system run) either satisfies *Next* or leaves the term $vars$ (and therefore all state variables) unchanged. The third conjunct L is a temporal formula stating the liveness conditions of the specification, and in particular can be used to rule out infinite stuttering.

B. Module extensions and static declarations

The semantics of an Edola module is given by a TLA⁺ module. The EXTENDS and CONSTANT statements of Edola are assigned the same semantics with the ones in TLA⁺. The input variables, output variables and system variables are explained by the variables declarations in TLA⁺ together with a type invariant to be ensured, since TLA⁺ is an un-typed language.

C. Dynamic definitions

The definition of the PLC behaviors in Edola includes the INIT part and the ACTION part. The two components are illustrated by the corresponding TLA⁺ components, intuitively. The INIT part in Edola corresponds the initial state formula in TLA⁺, which is composed by the whole INIT definition in Edola with the conjunction of the initial value of the variable *aux*, that is, zero. An action definition

in the ACTION part of Edola corresponds an action definition in TLA⁺, with the conjunction of an UNCHANGED statement.

The value changing of the variables in TLA⁺ is total, which requests the explicit statements of all the variables that are unchanged. This feature is good for mathematical reasoning, but is unintuitive and tedious for writing a model [10]. Edola then possesses the advantages on both modeling and mathematical reasoning by the transformational semantics. The semantics of the complete Edola specification (based on TLA⁺) depends on the time description. We will explain it in the end of this section.

The semantics of the environmental model lies on whether it's a total one generated by the Edola compiler or not. The former is illustrated by an TLA⁺ action *EnvInput* which changes the input variables randomly on condition of the type invariant is satisfied. The latter corresponds to the TLA⁺ action definitions, similar to the ones for PLC actions, except that the UNCHANGED statement denoting the unchanged value for the input variables that are not defined instead. The transformation for ENV TOTAL is shown in Table I.

Table I
THE FORMAL SEMANTICS OF ENV TOTAL IN EDOLA

Edola definitions	The formal semantics with TLA ⁺
ENV TOTAL	$EnvInput \triangleq \bigwedge_{i \in n_i} invar'_i \in ValRange_i$

A real-time TLA⁺ module *RealTimeNew* is provided to interpret the time operators in Edola. The time in Edola is logical and continuous, which was interpreted with a real type variable *now* in TLA⁺. A time constraint in Edola is then interpreted on an action denoting the constraints of its enabling time with the running of real time *now*. Each time pattern in Edola corresponds a defined time action in *RealTimeNew*. The details of the time module are introduced in our work [11].

Finally, we define the formal semantics of the Edola dynamic behaviors. When there isn't any time constraint in a Edola module, the whole Edola specification is interpreted as the formula $SpecName \triangleq Init \wedge \square[Next]_{vars}$, where *SpecName* is needed in TLA⁺, so it is generated by the Edola compiler according to the module name. *Init* is the formula name used for defining the initial state in Edola and *Next* is defined by

$$\begin{aligned}
 Next \triangleq & \vee \wedge aux = 0 \\
 & \wedge EnvInput \wedge UNCHANGED SOV \\
 & \wedge aux' = 1 \\
 & \vee \wedge aux = 1 \\
 & \wedge SystemAction \wedge UNCHANGED IV \\
 & \wedge aux' = 0.
 \end{aligned} \tag{4}$$

EnvInput denotes the environment model we introduced above. *SOV* denotes the system variables and the output variables declared in Edola, while *IV* denotes the declared input variables. *SystemAction* defines a complete set of possibilities for PLC responses, which includes all the expected actions defined in the ACTION part, and the case when none of them are enabled.

When there are time constraints in an Edola module, the Edola specification is interpreted in a module *RTModule*, which extends a *FuncModule* for the functional modeling. The specification is then illustrated by $SpecName \triangleq BigInit \wedge \square[BigNext]_{RTvars} \wedge RTL$, where the functional semantic interpretations of Edola like *Init*, *Next* are same as the former introduced ones, but encapsulated in the module *FuncModule*. The initial state *BigInit* and the next-state action *BigNext* of the timed specification is composed by the functional parts and the settings of the variable *now* and the *n* timers t_1, \dots, t_n appeared in the TIME part of the Edola model.

D. Verification requests

The property definitions provided in Edola are illustrated by the property definitions with temporal logic of actions (TLA) in TLA⁺ language. The excerpt of the translation is shown in Table II.

Table II
THE FORMAL SEMANTICS OF PROPERTY DEFINITIONS IN EDOLA

Edola Property definitions	The formal semantics with TLA ⁺
RESPOND (<i>Act</i> , <i>EnvS</i> , <i>SysS</i>)	$\square(EnvS \wedge \neg SysS \Rightarrow \neg(ENABLED Act))$
COMPETE (<i>Cond</i> , <i>S</i> ₁ , <i>S</i> ₂)	$\square(Cond \Rightarrow \neg(State_1 \wedge State_2))$
SEQUENCE (<i>Act</i> , <i>SysS</i>)	$\square[Act \Rightarrow SysS]_{vars}$
PRIORITY (<i>Act</i> , <i>SysS</i>)	$\square(SysS \Rightarrow (ENABLED Act \wedge \neg ENABLED(OtherActs)))$
STATEINV (<i>SysS</i>)	$\square(SysS)$
ACTINV (<i>Act</i>)	$\square(Act)_{vars}$

V. THE VERIFICATION OF EDOLA MODELS

Providing the automatic verification support for the Edola language is important to improve its usability. Model checking and automatic theorem proving are the two dominant automatic verification methods. The Edola compiler implements the support for both model checking and automatic theorem proving, with the intermediate language TLA⁺. To make the verification procedure pragmatically efficient, we took two major optimization strategies in the transformation procedure.

First, when the environmental model is specified by the ENV TOTAL keyword, the compiler will generate the formula for *EnvInput* with the input clearing action *ClearEnvInput* added, which resets the values of all the input variables to the initial value. According to the PLC scan cycling mechanism, PLC gathers the new values of its environment at the beginning of each cycle, so the values of the input

variables in the last cycle are discarded. It ensures the logical correctness of adding a *ClearEnvInput* action in the end of each cycle. The new definition of *Next* saves much state space and search space for verification and thus improves the efficiency.

Second, when all the verification requests are functional properties (which means that none of the timer variables appears in the PROP part of Edola), even though some time constraints are described in the TIME part, the compiler will generate a un-timed TLA⁺ model for it. The reason is that the timed model *RTModule* refines its functional part *FuncModule*: $RTModule \Rightarrow FuncModule$, so that for each property P , if it holds on the functional model, say $FuncModule \Rightarrow P$, with the transitivity of logic implication, it also holds for the timed model *RTModule*. As a result, we can check the functional properties on the functional part instead of the complete one, thus a better space and time cost can be saved in the verification.

The model checking of the Edola model is achieved by the transformation and the TLC model checker. As to the automatic theorem proving support, we can prove inductive invariants with the reasoning rule in TLA⁺:

$$\frac{Init \Rightarrow P, \quad P \wedge Next \Rightarrow P', \quad P \wedge v = v' \Rightarrow P'}{Init \wedge \square[Next]_v \Rightarrow \square P}. \quad (5)$$

The verification is then reduced to the first-order logic level without temporal operators.

VI. THE EDOLA TOOL

We implemented a prototype IDE to model and verify PLC systems with the Edola language, see Figure 3. The tool includes an editor to write the model, and a compiler to check, transform and verify the model. It is implemented with Java 1.6. The interface is developed with Netbeans IDE 6.7 and the compiler is implemented with the scanner/parser generator JavaCC 4.2 with the JJTree preprocessing functionality. The compiler implements syntax and semantic checking, and then the seamless integration of the model checker TLC and the automatic theorem prover Spass to verify an Edola model.

Beside the general semantic checking same with other language compilers, the Edola compiler provides also semantic checking specific to PLC applications. We check whether the actions defined in ACTION part are possible to execute one by one. If an action can never be executed, an alarm information is provided. We also check whether the disjunction of conditions for all the actions defined in ACTION part is TRUE. If not, an alarm is provided.

The semantic checking and the later model checking and automatic theorem proving provides strong verification of Edola models. The checking procedure is completely automatic.

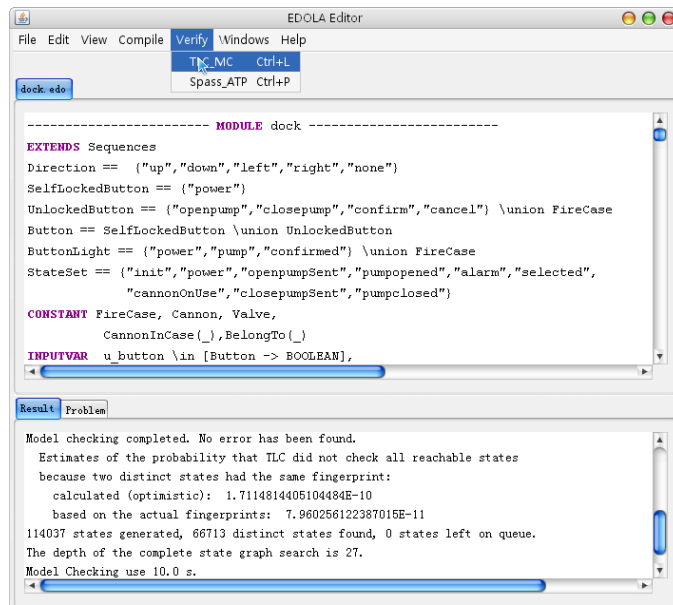


Figure 3. The prototype IDE of Edola

VII. CASE STUDY

The Edola language and its IDE has been applied in several medium-scale PLC applications, like the answering machine problem, the steeves control in a theater and the fire-fighting controls in a dock. In this section, the case about a fire-fighting PLC system used for the docks is chosen and presented to further illustrate and validate the Edola language and its tool. We introduce the Edola model, the TLC model checking and the Spass automatic theorem proving of it, respectively.

In the Edola model, we set the physical connections of the fire-fighting system as parameters. They are declared as: *FireCase* for the set of fire cases, *Cannon* for the set of used cannons, *Valve* for the set of valves, *CannonInCase*($_$) denoting which cannon is used in which fire case, and *BelongTo*($_$) representing which valve belongs to the connection of which cannon.

The operations are described in the ACTION part of Edola by a series of actions: *PowerUp*, *OpenPump*, *SelectCase*(i), *Confirm*, *HandleControl*, *ClosePump*, *Cancel* and *PowerDown*. The environmental model is chosen to be generated automatically through the keyword ENV TOTAL in the Edola model. No time constraint is needed in the simplified case.

The 8 requested properties are specified in the TOPROVE part. For example,

$$ClosePumpNotRespond : RESPOND (ClosePump, \quad (6) \\ s_sysState = \text{"cannonOnUse"}, u_button[\text{"closepump"}]).$$

asserts that the action *ClosePump* is disabled unless the

user has pressed the button *closepump* as well as the current system state is “cannonOnUse”.

Model checking technique can be used to verify the finite instances of a parameterized model. As a result, when choosing TLC model checker as the verification tool in the Edola IDE, a window is popped up to config the parameters. We instantiate the model by 2 fire cases, 2 cannons, and 4 valves. The TLC model checker is then called automatically to check the 8 properties. It generates 66, 713 different states in total and verifies that all the 8 properties hold in 10.0 seconds.

We can also try to prove the 8 properties with the integrated automatic theorem prover Spass. It can prove the properties (if and only if they are inductive invariants) directly on a parameterized Edola model, without the need of instantiation. The result of proving the 8 properties are shown in Table III. The two popular verification methods complement each other and provide the powerful verification capability for Edola.

Table III
THE SPASS PROVING RESULT OF THE 8 PROPERTIES IN EDOLA IDE

Properties	Spass result	Time
1. ClosePumpNotRespond	Proof found.	12.4s
2. SelectCaseNotRepond	Proof found.	10.6s
3. CaseSelectOnlyOne	Proof found.	1m33s
4. CannonUsedOnlyByOne	Proof found.	39.2s
5. ValveMutex	Proof found.	18m33s
6. OpenPumpAfterPower	Proof found.	1m37s
$OpenPump \Rightarrow s_buttonLight["power"]$	Completion found.	17.2s
$\square Inv$	Proof found.	1m26s
$OpenPump \wedge Inv \Rightarrow s_buttonLight["power"]$	Proof found.	10.3s
7. SelectAfterOpenPump	Proof found.	1m49s
8. ClosePowerAlwaysRespond	Proof found.	16.2s

VIII. CONCLUSION AND FUTURE WORK

In this paper, we presented a novel PLC domain-specific modeling language Edola. It provides useful notations to denote the features of PLC like reactivity, scan cycling, real-time and property patterns. As a result, with the Edola language, we can get a better understanding and easier modeling of PLC applications. It is noteworthy that both the two popular automatic verification methods: model checking and automatic theorem proving are supported in Edola modeling. To implement this functionality, we adopt the TLA⁺ specification language as the intermediate language between Edola and the verification tools, enhancing a large degree of reusability. A prototype IDE for Edola has been implemented, which provides the user-friendly interface for modeling and the seamless integration of two tools TLC and Spass for verification.

As to the future work, we will enrich the Edola language with module compositions, action priorities, etc. to increase its expressiveness. The support of other verification tools like

the model checker UPPAAL [12] and the theorem prover CVC3 [13] will also be considered.

ACKNOWLEDGMENT

This research is supported in part by NSFC Programs (No.91018015, No.60811130468) and 973 Program (No.2010CB328003) of China.

REFERENCES

- [1] R.W. Lewis. *Programming industrial control systems using IEC 1131-3, volume 50 of Control Engineering Series*. The Institution of Electrical Engineers, Stevenage, United Kingdom, 1998.
- [2] F. Bonfatti, P.D. Monari, and U. Sampieri. *IEC 1131-3 Programming Methodology*. CJ International, Fontaine, France, 1999.
- [3] R.Wang, X.Song, and M. Gu. Modelling and verification of program logic controllers using timed automata. *IET Software*, 4:127–131, 2007.
- [4] Hehua Zhang, Ming Gu, and Xiaoyu Song. Modeling and analysis of stage machinery control systems by timed colored Petri nets. In *Proceedings of the 3rd International Symposium on Industrial Embedded Systems, (SIES 2008)*, pages 103–110, 2008.
- [5] G. Canet, S. Couffin, J. J Lesage, A. Petit, and Ph. Schnoebelen. Towards the automatic verification of PLC programs written in instruction list. In *Proceedings of IEEE International conference on Systems, Man and Cybernetics (SMC'2000)*, pages 2449–2454, 2000.
- [6] Hehua Zhang, Stephan Merz, and Ming Gu. Specifying and verifying plc systems with TLA+. In *Proceedings of the 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2009)*, pages 293–294, 2009.
- [7] Yuan Yu, Panagiotis Manolios, and Leslie Lamport. Model checking TLA+ specifications. In *Proceedings of Correct Hardware Design and Verification Methods (CHARME'99)*, volume 1703, pages 54–66. Springer Verlag, 1999.
- [8] The SPASS homepage: <http://www.spass-prover.org/index.html>[Accessed: 13 Aug. 2011].
- [9] Leslie Lamport. *Specifying Systems*. Addison-Wesley, 2002. See also <http://research.microsoft.com/users/lamport/tla/tla.html>.
- [10] Leslie Lamport and Lawrence C. Paulson. Should your specification language be typed. *ACM Trans. Program. Lang. Syst.*, 21(3):502–526, 1999.
- [11] Hehua Zhang, Ming Gu, and Xiaoyu Song. Specifying time-sensitive systems with TLA+. In *34th Annual IEEE International Computer Software & Applications Conference (COMPSAC 2010)*, pages 425–430, 2010.
- [12] The UPPAAL homepage: <http://www.uppaal.com/>[Accessed: 13 Aug. 2011].
- [13] The CVC3 homepage: <http://www.cs.nyu.edu/acsys/cvc3/>[Accessed: 13 Aug. 2011].

A Practical Method for the Reachability Analysis of Real-Time Systems Modelled as Timed Automata

Abdeslam En-Nouaary and Rachida Dssouli
ECE Department, Concordia University
Montreal, Canada
 {ennouaar,dssouli}@ece.concordia.ca

Abstract—Real-time systems (RTSs) interact with their environment under time constraints. Such constraints are so critical because any deviation from the specified deadlines might have severe consequences on both the human lives and the environment. To develop reliable RTSs, formal methods should be used along the development life cycle. Verification is one of these formal methods, which aims at ensuring that the system is correct before its deployment. This paper presents a new verification method for the reachability analysis of real-time systems modelled as timed automata (TA) [1]. The paper basically addresses two main issues: are all the transitions of the system executable? Are all the locations reachable from the initial location of the system? In order to answer these questions, our method uses a metric that gives the minimum delay between any state and all the transitions leaving that state.

Keywords-Real-Time systems, Formal Methods, Timed Automata, Verification, Reachability Analysis.

I. INTRODUCTION

Over the past two decades, many researchers have been investigating the verification and validation of real-time systems with different backgrounds. As a result, several verification methods have been devised to make sure that the system functions properly before its deployment. These verification techniques attempt to check if the specification of the system satisfies some desirable functional and performance properties. All the verification and validation techniques rely on the use of formal models to describe the behaviour of the systems being investigated (see for instance [2], [3], [4], [5], [6], [7], [8], [9]). In the case of real-time systems, timed automata model [1] is intensively used by researchers to develop verification and testing techniques. Although, existing verification methods and tools (see for instance [2], [3], [4], [5]) provide successful results for RTSs, most of them suffer from the state explosion problem and are a bit complicated to use. This is mainly due to the fact that most of the proposed techniques are based on either the region graph [1] or the zone graph [10] as semantics for timed automata. So, the need for practical verification and validation methods still exists.

In this paper, we present a new method for the reachability

analysis of RTSs modelled as timed automata. We are basically addressing two main issues: are all the transitions of the system executable? Are all the locations reachable from the initial location of the system? In order to answer these questions, our method uses a metric that gives the minimum delay between any state and all the transitions leaving that state. Our method presents two advantages. On the one hand, it automatically calculates on the fly the paths that ensure the reachability of the transitions and locations. On the other hand, it avoids the costly operation of constructing the region graph of the timed automata.

The remainder of this paper is organized as follows. Section 2 presents the timed automata model and its related concepts. Section 3 introduces our contributions. Section 4 concludes the paper and presents future work.

II. BACKGROUND

This section presents the definitions and concepts required for introducing our method. We basically present the timed automata model and the related theoretical results illustrated with simple examples.

Definition 1: Timed Automata (TA)

A TA A is a 5-tuple (Σ, L, l_0, C, T) , where :

- Σ is a finite set of inputs and output messages. In this paper, inputs begin with "?" while outputs start with "!".
- L is a finite set of locations. A location represents the "status" of the system after the execution of a transition. The term location is used instead of the term "state" because the latter is used to define the operational semantics of the TA.
- $l_0 \in L$ is the initial location where the execution of the TA starts.
- C is a finite set of clocks, all initialized to zero in l_0 . A clock is a time variable that counts how much time has elapsed since the clock was (re-)initialized to zero.
- $T \subseteq L \times \Sigma \times \Phi(C) \times \mathcal{P}(C) \times L$ is the set of transitions, where $\Phi(C)$ and $\mathcal{P}(C)$ denote the set of clock guards and the power set of C , respectively.

A transition in a TA, denoted by $t : l \xrightarrow{m,G,R} l'$, consists of a source location l (i.e., $source(t) = l$), an input or output

message m , a clock guard (or time constraint) G , which should hold to execute the transition, a subset of clocks R to be reset when the transition is fired, and a destination location l' (i.e., $destination(t) = l'$). Each clock in R ($R \subseteq C$) is used to record, when not reinitialized to zero, how much time has elapsed since the execution of the transition. Such clocks are mainly used to set clock guards between the transition where they are reset and future transitions.

A sequence of consecutive transitions that starts at a location l and ends at a location l' is called a path from l to l' ; we write $path(l, l') = t_1.t_2...t_n$, where $t_i \in T$ for $1 \leq i \leq n$, and $source(t_1) = l$, $destination(t_n) = l'$ and $source(t_i) = destination(t_{i-1})$ for $2 \leq i \leq n$. Since paths in TA are made of transitions with clock guards that could be conflicting (i.e., they cannot be satisfied by the same values of clocks) one can easily see that a path might not be executable. Hence, finding an executable path from one location to another requires, as explained later on in this paper, a systematic approach and a deeper investigation as to how messages and clock values should be chosen to fire the transitions of the system.

We assume that the transitions in a TA are instantaneous (i.e., they don't take time to execute). Also, the clock guards of the transitions are supposed to be conjunctions of atomic formulas of the form $(b_1 op_1 x op_2 b_2)$, where $x \in C$, $(op_1, op_2) \in \{<, \leq, =\}$, and b_1 and b_2 are natural numbers. Multiple clocks are used in the TA to express time constraints between more than two transitions. Each clock, $x \in C$, in a TA takes real number values and has a bounded domain $[0, B_x] \cup \{\infty\}$, as stated by Springintveld et. al. [11], where B_x is the largest integer constant appearing in the time constraints over clock x in the automaton. This means that each clock x is relevant only under the integer constant B_x , and all the values of x greater than B_x are represented by ∞ ; Hence, we write : $\forall \epsilon > 0, B_x + \epsilon = \infty$ and $\infty + \epsilon = \infty$.

For a clock x and a clock guard G of a transition in a TA, we define the projection of G over x , written $Proj(G, x)$, by the condition $(b_1 op_1 x op_2 b_2)$ in G , obtained by removing the conditions over all the clocks except x ; if clock x is not involved in G then $Proj(G, x) = true$.

Example 1: Figure 1 shows a TA for a specification of a simple telephone system. The system waits for the user to hang up, get the dial tone and dial two digits $Digit_1$ and $Digit_2$; then the system issues the output $Connect$, to indicate that the connection has been established and the user can start talking. At the end, the user lifts the phone to allow the system to go back to its initial location. The behaviour of the system is subject to several time constraints. On the one hand, the user should type the first digit 1 to 3 time-

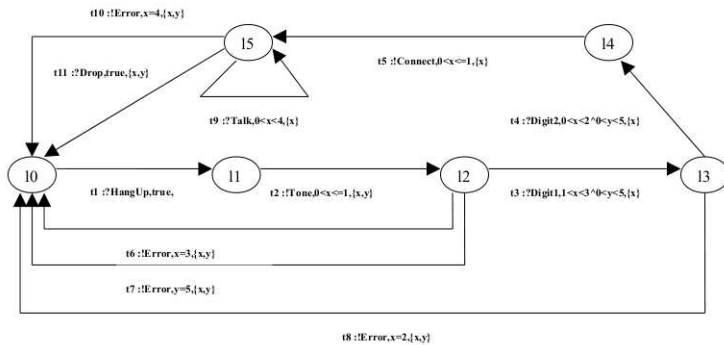


Figure 1. An Example of TA.

units after getting the tone and the second digit no more than 2 time-units after the first digit; the dialling operation (from getting the tone until dialling the last digit) should not exceed 5 time-units. On the other hand, the system must respond with the signal $Tone$ within 1 time-unit after the user hangs up, and with $Connect$ within 1 time-unit after the last digit has been typed. Whenever the time constraint of an input is not respected, the system times out, issues an error message and goes back to its initial location.

The TA model introduced thus far is an abstract model because it does not explain the execution of the system it describes. The executions, also called the operational semantics or the region graph of the TA, can be informally stated as follows. The TA starts at its initial location with all clocks initialized to zero. Then, the values of the clocks increase at the same speed and measure the amount of time elapsed since the last (re-)initialization. At any time, the TA can execute a transition $l \xrightarrow{m, G, R} l'$ if the input/output message m takes place, its current location is l , and the values of its clocks satisfy the clock guard G . After this transition, all the clocks in R are reset and the TA changes its location to l' . To formalize the operational semantics of the TA, we need to define the concepts of clock valuations and states for the TA.

Definition 2: Clock valuations

Let $A = (\Sigma, L, l_0, C, T)$ be a n -clocks TA (i.e., an TA with n clocks), $\mathbf{R}^{\geq 0}$ be the set of non-negative real numbers.

- A clock valuation of A (or over C) is a function $v : C \rightarrow [\mathbf{R}^{\geq 0} \cup \{\infty\}]^n$, which assigns a positive value to each clock $x \in C$. A clock valuation is simply the binding of clocks to their actual values. In this paper, a clock valuation is represented by a vector $(v_{x_1}, v_{x_2}, \dots, v_{x_n})$, where $v(x_i) = v_{x_i}$ is the value of clock x_i , $1 \leq i \leq n$. The set of all clock valuations of A is referred to by $V(C)$.
- For any clock valuation $v \in V(C)$ and any non-negative real number d , $v + d$ is a clock valuation that assigns the value $v(x) + d$ to each clock $x \in C$. $v + d$

is the clock valuation reached from v by letting time elapse by d time units.

- For any clock valuation $v \in V(C)$ and any subset of clocks $R \subseteq C$, $[R := 0]v$ is a clock valuation that assigns the value 0 to each clock $x \in R$ and $v(x)$ to any other clock (i.e., $y \in C$ and $y \notin R$). $[R := 0]v$ is the clock valuation obtained from v by resetting the clocks in R when a transition $l \xrightarrow{m,G,R} l'$ is executed.
- A clock valuation $v \in V(C)$ satisfies the clock guard G of a transition $l \xrightarrow{m,G,R} l'$, denoted by $v \models G$, if and only if G holds under v .

Informally speaking, a clock valuation is an interpretation of clocks, which allows us to know at any time the value of each clock used in the TA. In other words, a clock valuation can be used to determine how much time has elapsed since the execution of each transition that has last reinitialized a clock. The combination of a clock valuation and a location defines a state of the TA. The formal definition of such states follows.

Definition 3: States of the TA

Let $A = (\Sigma, L, l_0, C, T)$ be a TA.

- A state of A is a pair (l, v) consisting of a location $l \in L$ and a clock valuation $v \in V(C)$. Intuitively, a state of A is a configuration that indicates the current location of A and the current value of each clock used in A .
- The initial state of A is the pair (l_0, v_0) , where $v_0(x) = 0$ for each clock $x \in C$. Intuitively, the initial state of A is the configuration of A in the beginning of its execution (i.e., the location is l_0 and all clocks are set to 0 as stated in Definition 1).
- The set of states of A is denoted by $S(A)$.

Example 2: To illustrate the concepts of clock valuations and states, let us consider again the TA of Figure 1. The number of clocks in this TA is 2, namely clocks x and y . So, a clock valuation, here, consists of assigning a non-negative real number or ∞ to each of the clocks x and y . Examples of such clock valuations are $v_0 = (0, 0)$, $v_1 = (\frac{1}{4}, \frac{1}{4})$ and $v_2 = (\frac{1}{2}, \frac{3}{2})$. The set of the states of this TA is the set of all the pairs obtained by combining the locations and the clock valuations of the TA. Examples of such states are $s_0 = (l_0, v_0)$, $s_1 = (l_1, v_1)$ and $s_2 = (l_3, v_2)$, where l_0 , l_1 , and l_3 are the locations of the TA, and v_0 , v_1 , and v_2 are the clock valuations explained in this example.

Formally, the operational semantics of a TA is described by a state machine $M = (S, s_0, A, T)$, where S is the set of states of the TA, s_0 is the initial state, A is the set of actions, and T is the set of transitions. The actions of M are made up of the input and output messages of the TA as well as the time delays (i.e., $A = \Sigma \cup [\mathbf{R}^{\geq 0} \cup \{\infty\}]^n$). Hence, there are

two categories of transitions in M : The explicit transitions on input and output messages, and the implicit transitions on time delays. The explicit transitions are obtained from the transitions of the TA and they describe the interactions of the system with its environment. The explicit transitions do not contain time constraints because the clock valuations of their source states do satisfy their clock guards. On the other hand, the delay transitions describe the progression of time but they do not appear in the transitions of the TA.

The operational semantics of timed automata helps us define the concepts of traces for real-time systems as follows. A trace for a real-time system is a sequence of input and output messages as well as time delays that starts at the initial state of the system and ends at one of its reachable state. It basically reflects an execution of the system on some input and output messages when the clock guards of the corresponding transitions are satisfied by the values of the clocks upon the occurrence of the messages. For instance, the trace $?m_1.\frac{1}{2}.?m_2.3.!m_3$ means that when the system starts its execution it immediately accepts the input message m_1 , waits $\frac{1}{2}$ time-unit before accepting the input message m_2 , and then waits 3 time-units before responding with an output message m_3 .

III. OUR METHOD FOR THE REACHABILITY ANALYSIS OF TIMED AUTOMATA

This section introduces our method for the reachability analysis of real-time systems modelled as timed automata. The two main issues dealt with in this paper are: the reachability of the locations from the initial location of the system and the executability of the transitions of the system. The objective of the former issue is to check if every location of the system is reachable from its initial location. However, the objective of the latter issue is to check if every transition of the system is executable. To address these issues, we propose a metric that determines the minimum delay between each state and each of its outgoing transitions. We also present an algorithm to implement the metric in order to decide the reachability issues automatically. The minimum delay between a state and a transition represents the minimum waiting time required at the state in order to execute the transition. It basically reflects the point of time right after the execution of the transition was impossible (i.e., as soon as the transition becomes executable). Formally, the minimum delay between a state $s = (l, v)$ and a transition $t = l \xrightarrow{m,G,R} l'$, written $delay_{min}(s, t)$, is calculated as follows:

$$delay_{min}(s, t) = \text{Max}_{x \in C} \{0, delay_{min}(v(x), \phi(x))\},$$

where:

- $\phi(x) = Proj(G, x)$ is the projection of the transition's guard over the clock x , as explained in Section II,
- $v(x)$ is the value of the clock x at state s , and
- $delay_{min}(v(x), \phi(x))$ is the minimum waiting time at state s for clock x to satisfy its time constraint $\phi(x)$:

$$delay_{min}(v(x), \phi(x)) = \begin{cases} m_1 - v(x) + \epsilon \text{ if } \phi(x) \text{ is} \\ (m_1 < x \leq m_2) \\ m_1 - v(x) \text{ if } \phi(x) \text{ is} \\ (x = m_1) \text{ or } (m_1 \leq x \leq m_2) \\ 0 \text{ if } \phi(x) \text{ is true} \end{cases}$$

ϵ is a small positive real value chosen by the designer. It is a parameter, which helps him/her specify how far from the lower bound of the open clock guard the transition should be executed.

Example 3: Let us consider again the telephone system of Figure 1 and suppose that the designer chooses $\epsilonpsilon = \frac{1}{4}$. For the states $s_0 = (l_0, (0, 0))$ and $s_1 = (l_2, (\frac{1}{4}, \frac{1}{4}))$, and the transitions t_1 and t_3 , we have:

- $delay_{min}(s_0, t_1) = Max\{0, delay_{min}(0, true), delay_{min}(0, true)\} = Max\{0, 0, 0\} = 0$.
- $delay_{min}(s_1, t_3) = Max\{0, delay_{min}(\frac{1}{4}, 1 < x < 3), delay_{min}(\frac{1}{4}, 0 < y < 5)\} = Max\{0, \frac{1}{2}, 0\} = \frac{1}{2}$.

Now, we explain how we address the reachability issues aforementioned. To deal with the reachability of the locations of the system, we proceed as follows. Let $A = (\Sigma, L, l_0, C, T)$ be a TA and l be a location of A (i.e., $l \in L$). l is said to be reachable from the initial location of the TA if and only if there exists an executable path, $path(l_0, l)$, from l_0 to l in A . There are at least three different ways to find an executable path from the initial location l_0 to another location l in the TA:

- The first method consists of first extracting all the paths from l_0 to l and then choosing the shortest one (i.e., the path with the least number of transitions).
- The second method consists of, as the first method, extracting all the paths from l_0 to l and then choosing randomly one of them.
- The third method consists of extracting on the fly only one path from l_0 to l according to some metrics that minimize either the number of the transitions in the path or the time it takes to execute the path.

The first two methods have the disadvantage of being costly because they have to extract all the paths from l_0 to l . Moreover, the chosen path (either the shortest one or the randomly selected one) might not be executable because of the conflicting clock guards of its transitions and hence the resulting path could be useless. The third method is less costly than the two others because it does not rely on the extraction of all the paths from l_0 to l . However, if the minimization adopted is with regard to the number of the transitions in the selected path then we will have no guarantee about the executability of the path, all as for the first and the second methods. Hence, the best way to choose and ensure an executable path from l_0 to l is to extract it on the fly by minimizing the time it takes

to execute the path. This can be done by using the metric $delay_{min}()$ introduced so far. More precisely, to get an executable path $path(l_0, l)$, we have to start at the initial state (l_0, v_0) and calculate the minimum time delay to execute each transition leaving l_0 and decide which one should be added to the path. Then, we compute the resulting states and repeat the process on the new states until we reach l .

Similarly, to address the executability of a transition we have to find an executable path from the initial location to the source location of the transition, and use the minimum time delay in order to calculate at least one time point that makes the transition executable from the last reached state from the path. In order to ensure an executable path from l_0 to the source location of the transition, we follow the same process described previously when dealing with the reachability of the locations of the TA. Regarding the time point that makes the transition executable, we calculate it using the minimum time delay between the last reached state in the path for the transition, and the transition being checked. Formally speaking, let $A = (\Sigma, L, l_0, C, T)$ be a TA and $t = l \xrightarrow{m, G, R} l'$ be a transition of A . t is said to be executable if and only if:

- $source(t)$ is reachable from the initial location l_0 and the resulting state is $s = (source(t), v)$, and
- $(v + delay_{min}(s, t))$ satisfies the clock guard of t (i.e., $(v + delay_{min}(s, t)) \models G$).

It should also be noted that a location l is reachable if and only if there exists $t \in T$ such that $destination(t) = l$ and t is executable. Likewise, if a location l is not reachable then all the transitions leaving l are non-executable (i.e., for all $t \in T$ such that $source(t) = l$ the transition t is non-executable).

The algorithm used to check the reachability of the locations and transitions of the TA is shown below. The algorithm takes as input the TA and returns a Boolean value for each location and each transition that says whether or not the location (respectively the transition) is reachable (respectively executable). The algorithm starts by calculating the initial state of the TA and initializing all the variables to be used, namely RS (the set of reachable states) and HS (the set of the handled states among RS). Then, it goes through all the states in RS and handles all the outgoing transitions from each of these states. Indeed, for each reachable state, the algorithm checks all of the outgoing transitions from the location of the state and verifies if they are executable by calculating the minimum delay between the state and each of the transitions. If the clock guard of a transition is satisfied by the clock valuation of the current state plus the minimum delay calculated previously then the transition (respectively its destination

location) is marked as being executable (respectively reachable), and the resulting state is calculated and added to RS if it is not already there. When the algorithm terminates the handling of all the reachable states (i.e., all the states in RS), it goes through all the locations and transitions to check if they have been marked so far. If a location has not been marked then the location is declared unreachable. Likewise, if a transition has not been marked then the transition is declared non-executable. It should be noted that the algorithm does not construct the region graph [1] of the TA but calculates on the fly only one state for each transition in the TA. That state is obtained using the metric $delay_{min}(s,t)$ introduced in the beginning of this section. Hence, the proposed approach has the advantage of being scalable and rapid compared to existing methods that are based on the construction of either the region graph [1] or the zone graph [10] of the TA. We implemented the algorithm in Java and promising results are obtained for specifications with different sizes. The presentation of the tool and the analysis of the experimentation results are left for a future publication.

Algorithm 1: Our Algorithm for the Reachability Analysis of a TA.

```

ReachabilityAnalysis(INPUT: TA)
 $s_0 \leftarrow (l_A^0, v_0)$  // ( $v_0(x) = 0$  for every clock  $x$  in the TA).
 $RS \leftarrow s_0$ . //  $RS$  is the set of reachable states of the TA.
 $HS \leftarrow \emptyset$ . //  $HS$  is the set of handled states of the TA.
while ( $RS \neq HS$ ) do
    Pick one state ( $s : (l, v) \in RS$ ) not yet processed.
    Add  $s$  to  $HS$ .
    foreach ( $transition\ t : l \xrightarrow{m,G,R} l'$  in the TA) do
        Calculate  $\delta = delay_{min}(s,t)$ .
        if ( $(v + \delta) \models G$ ) then
            Add the state ( $l', [R := 0](v + \delta)$ ) to  $RS$  if not yet there.
            Mark the location  $l'$  and the transition  $t$  in the TA as they are reached.
    foreach ( $location\ l \in L$ ) do
        if ( $l$  is not marked) then
             $l$  is not reachable
    foreach ( $transition\ t \in T$ ) do
        if ( $t$  is not marked) then
             $t$  is not executable
    
```

The complexity of the algorithm is $\Theta(|L| \times |T|)$, where $|L|$ is the number of locations and $|T|$ is the number of transitions. Indeed, the algorithm goes through all the reachable states whose number is at most equal to $|L| \times |T|$.

Each reachable state is handled only once and requires the processing of only the transitions leaving the location of the state. By adding up the number of these iterations we get an order of $\Theta(|L| \times |T|)$.

Example 4: Let us consider again the telephone system of Figure 1. By applying our algorithm, with $\epsilon = \frac{1}{4}$, we get the results shown in Figure 2. The first table gives for each location if it is reachable or not while the second table determines for each transition if it is executable or not. When a location (respectively, a transition) is reachable (respectively executable) the tables show one of the traces that make it possible. By examining the results in Figure 2, one can easily see that all the locations (respectively all the transitions) of the system modelled in Figure 1 are reachable (respectively executable). For each reachable location, the corresponding trace is obtained by extracting an executable path from the initial location to the location. Similarly, for each executable transition the corresponding trace is obtained by extracting an executable path from the initial location to the source location of the transition plus the time delay and the message to execute the transition. The executability of any path is ensured based on the metric $delay_{min}()$, introduced so far. For instance, the location l_4 is reachable and the transition $t_4 : l_3 \xrightarrow{?Digit_3, 0 < x < 2 \wedge 0 < y < 5, \{x\}} l_4$ is executable because the trace $?HangUp.1.!Tone.\frac{5}{4}.?Digit_1.\frac{1}{4}.?Digit_2$ makes it possible for the system to move from its initial state $(l_0, (0, 0))$ to $(l_4, (0, \frac{3}{2}))$; the corresponding executable path then is $t_1.t_2.t_3.t_4$.

Example 5: Let us now change the specification of Figure 1 by changing the clock guards of the transitions t_3 t_4 and t_8 to $((2 < x < 3) \wedge (0 < y < 3))$, $((2 < x < 3) \wedge (0 < y < 3))$ and $(x = 3)$, respectively. Is it really easy to guess the reachability analysis of the new system? It is not that simple! By applying our algorithm, with $\epsilon = \frac{1}{4}$, we can see that the transitions t_4 , t_5 , t_9 , t_{10} and t_{11} are non-executable, and the locations l_4 and l_5 are non-reachable. Let us say why. The minimum executable path to reach the location l_3 (the source location of the transition t_4) is $t_1.t_2.t_3$ and the corresponding trace is $?HangUp.1.!Tone.\frac{9}{4}.?Digit_1$. Hence, the state that should be considered to execute the transition t_4 is $s_4 = (l_3, (0, \frac{9}{4}))$, which gives $delay_{min}(s_4, t_4) = \frac{9}{4}$. But, by adding $\frac{9}{4}$ to $(0, \frac{9}{4})$ (i.e., the clock valuation of s_4) we obtain the clock valuation $(\frac{9}{4}, \frac{18}{4})$ that does not satisfy the clock guard $((2 < x < 3) \wedge (0 < y < 3))$ (i.e., the clock guard of t_4). Hence, the transition t_4 is non-executable and since t_4 is the unique transition between l_3 and l_4 then the location l_4 is not reachable. Consequently, all the transitions leaving l_4 are non-executable, namely the transition t_5 . Since t_5 is the unique transition between l_4 and l_5 then the location l_5 is not reachable and all the transitions leaving l_5 become non-executable, namely t_9 , t_{10} and t_{11} .

Loc.	Reach. (Y/N)	Corresponding Trace
l_0	Y	ϵ (the empty sequence)
l_1	Y	?HangUp
l_2	Y	?HangUp.1.!Tone
l_3	Y	?HangUp.1.!Tone. $\frac{5}{4}$.?Digit ₁
l_4	Y	?HangUp.1.!Tone. $\frac{5}{4}$.?Digit ₁ . $\frac{1}{4}$.?Digit ₂
l_5	Y	?HangUp.1.!Tone. $\frac{5}{4}$.?Digit ₁ . $\frac{1}{4}$.?Digit ₂ .1 .!Connect

Trans.	Exec. (Y/N)	Corresponding Trace
t_1	Y	?HangUp
t_2	Y	?HangUp.1.!Tone
t_3	Y	?HangUp.1.!Tone. $\frac{5}{4}$.?Digit ₁
t_4	Y	?HangUp.1.!Tone. $\frac{5}{4}$.?Digit ₁ . $\frac{1}{4}$.?Digit ₂
t_5	Y	?HangUp.1.!Tone. $\frac{5}{4}$.?Digit ₁ . $\frac{1}{4}$.?Digit ₂ .1 .!Connect
t_6	Y	?HangUp.1.!Tone.3.!Error
t_7	Y	?HangUp.1.!Tone.5.!Error
t_8	Y	?HangUp.1.!Tone. $\frac{5}{4}$.?Digit ₁ .2.!Error
t_9	Y	?HangUp.1.!Tone. $\frac{5}{4}$.?Digit ₁ . $\frac{1}{4}$.?Digit ₂ .1 .!Connect. $\frac{1}{4}$.?Talk
t_{10}	Y	?HangUp.1.!Tone. $\frac{5}{4}$.?Digit ₁ . $\frac{1}{4}$.?Digit ₂ .1 .!Connect.4.!Error
t_{11}	Y	?HangUp.1.!Tone. $\frac{5}{4}$.?Digit ₁ . $\frac{1}{4}$.?Digit ₂ .1 .!Connect.?Drop

Figure 2. The Reachability Results for the TA in Figure 1.

IV. CONCLUSION

We presented in this paper a new verification method for the reachability analysis of real-time systems modelled as timed automata. Our method addresses the reachability of the locations and transitions of the system by calculating a trace that allows the system to go from its initial state to the location or transition being investigated. To this end, the method uses a metric that gives the minimum delay between any state and all the transitions leaving that state. Our method has at least two advantages. On the one hand, it automatically calculates on the fly the paths that ensure the reachability of the transitions and locations. On the other hand, it avoids the costly operation of constructing the region graph of the TA, which makes the method more scalable than the others. To help us quantify precisely the gain of the method with respect to existing methods, we implemented the method to conduct more experimentation on TA specifications with different sizes. The tool and the analysis of the experimentation results will be discussed in a future paper.

We are currently working on two extensions of the

proposed method. On the one hand, we would like to make it incremental to adjust to successive evolutions of the specification either when designing the system the first time or later when maintaining the system. On the other hand, we are investigating the possibility of adopting the incremental method to the area of testing real-time systems modelled as TA.

REFERENCES

- [1] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.
- [2] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages –. Springer-Verlag, 1995.
- [3] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In *4th. DIMACS Workshop on Verification and Control of Hybrid Systems*, New Brunswick, New Jersey, October 1995.
- [4] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.
- [5] K.G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Proceedings of Fundamentals of Computation Theory*, pages –, Dresden, Germany, August 1995.
- [6] Osmane Koné, Patrice Laurecot, and Richard Castanet. On the Fly Test Generation for Real-Time Protocols. In *International Conference on Computer Communications and Networks, Lafayette, Louisiana, USA*, pages 378–387, 1998.
- [7] Brian Nielsen and Arne Skou. Automated Test Generation from Timed Automata. In *5th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems FTRTFT'98, Lyngby, Denmark*, pages 59–77, September 1998.
- [8] A. En-Nouaary, R. Dssouli, and F. Khendek. Timed Wp-Method: Testing Real-Time Systems. *IEEE Transactions on Software Engineering*, 28(11):1023–1038, November 2002.
- [9] A. En-Nouaary. A Scalable Method for Testing Real-Time Systems. *Software Quality Journal, Springer*, 16(1):3–22, March 2008.
- [10] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of Timed Transition Systems. pages 340–354, 1992.
- [11] J. Springintveld and F. Vaandrager. Minimizable Timed Automata. In B. Jonsson and J. Parrow, editors, *Proceedings of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems*, Uppsala, Sweden, volume 1135 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

Reverse Engineering of Graphical User Interfaces

Work partially supported by FCT under contract PTDC/EIA/66767/2006

Inês Coimbra Morgado, Ana C. R. Paiva
 Department of Informatics Engineering,
 Faculty of Engineering, University of Porto,
 Porto, Portugal
 {ei07040, apaiva}@fe.up.pt

João Pascoal Faria
 Department of Informatics Engineering,
 Faculty of Engineering, University of Porto
 INESC Porto
 Porto, Portugal
 jpf@fe.up.pt

Abstract—This paper describes a dynamic reverse engineering approach and the correspondent tool, ReGUI, developed to reduce the effort of obtaining visual and formal models of both the structure and the behaviour of a software application’s graphical user interface.

This paper describes the tool’s architecture, the exploration process it follows, the outputs it generates and the rules used to generate a Spec# model, which can be used in the context of Model-Based Graphical User Interface Testing. The case study presents the results obtained by applying the tool to the Microsoft Notepad application.

Keywords—ReGUI, Reverse Engineering, GUI testing

I. INTRODUCTION

This research work is part of a wider ongoing project called AMBER iTest. The main goal of this project is to “develop a set of tools and techniques to automate specification based Graphical User Interface (GUI) testing, solving the shortcomings found in previous work, and show their applicability in industrial environments” [1]. Model-Based Testing (MBT) can contribute to increase the systematisation and automation of the testing process. However, the manual construction of a formal model (required as input by MBT techniques) is a too time consuming and error prone activity. The challenge to be tackled in this research work is the automatic construction of part of the software model using reverse engineering techniques, easing the process of creating visual and formal models. To build these models, both structural and behavioural information are required. This information is extracted by the ReGUI tool. The visual models help to quickly understand the GUI. The formal model is written in Spec# [2] and it is necessary to automatically generate test cases inside the AMBER iTest project.

Once extracted, the formal model needs to be verified, completed and validated. This process is of the utmost importance in order to ensure the model describes the intended behaviour. In addition, the extracted model may reveal errors that must be fixed. In that case, the model should be updated in order to describe the intended behaviour and identify, later on, the conformance errors with the application under test. If this validation process is not performed, the extracted model may describe the implemented behaviour, which may be different from the intended behaviour, and be useless as a test oracle. The validated model is then used by the Spec Explorer Tool

[3] to generate test cases. These tests are afterwards run over the GUI of the application under test using the GUI Mapping Tool [4].

This paper is divided as follows. Section II describes the state of the art on reverse engineering and Section III presents the developed tool, ReGUI, focusing on its architecture, functioning and artifacts produced. Finally, Section IV presents a case study and Section V presents some conclusions about this research work, along with the limitations of the approach.

II. STATE OF THE ART

“Reverse engineering is the process of analysing a subject system to create representations of the system at a higher level of abstraction” [5]. This representation is usually presented as a model, which can help to better understand an application, can be used by a code generation process to change the platform of legacy systems and can be used to check if the system has the required properties. There are two types of reverse engineering: static and dynamic, depending on whether the model is extracted from the source code or from the program in execution, respectively [6]. Both approaches follow the same three main steps: collect the data, analyse it and represent it in a legible way, and both allow obtaining information about control and data flow [7].

A. Static Reverse Engineering

Static reverse engineering tries to extract information about an application through its source code or through its byte code. Static reverse engineering techniques may be useful during the development of a software system as a way of ensuring the correctness of the implementation or as a way of being aware of the current stage of the development [8].

There are several studies on static reverse engineering [9], [10], [11], [12], [13]. Bouillon et al. implemented a set of derivation rules in *ReversiXML* [9] to enable model extraction from web pages. Instead of extracting a single model it is also possible to extract several models of different abstraction levels or perspectives and it is even possible to obtain models in different abstraction levels from a previously extracted one. In order to support this, some graph grammars were implemented in *TransformiXML* [9], [14].

Vanderdonck et al. describe a reverse engineering process, which enables the extraction of a model from a web appli-

cation, *VAQUISTA* [10]. This method was developed in order to enable the automatic migration of the web application into other platforms, such as pocket computers or mobile phones.

B. Dynamic Reverse Engineering

Dynamic approaches extract information from the Application Under Analysis (AUA) in run-mode. Unlike static techniques, dynamic approaches are able to extract information about concurrent behaviour, code coverage and memory management [6]. Initially, the data is collected by running the AUA under a debugger or a profiler. There are several strategies to analyse and represent this data [7].

Even though dynamic approaches are not as common as static ones, there are still some important works, which need to be mentioned.

Shehady et al. propose a reverse engineering method to automate part of the interface testing activity. It extracts the user interface’s model representing it as a Variable Finite State Machine, which is later on transformed into a Finite State Machine (FSM) for testing purposes [15]. On top of the FSM, the W_p algorithm [16], which assumes the FSM is fully specified, is applied. This algorithm generates tests, which allow the identification of any discrepancies between the FSM and a model specifying the expected output values. The error diagnosis process is manual.

Chen and Subramaniam developed *VESP* (Visual Environment for manipulating test SPecifications) that works on GUI based applications in Java [17]. The *VESP*’s purpose is to obtain a FSM representation of a GUI coded in Java. Black box test cases [18] are generated from the FSM and afterwards executed on the GUI of the AUA. One aspect that differentiates this approach from more common processes is that the graphical environment provided enables the tester to modify the test specification by modifying the FSM itself, without needing to know any internal representation details.

Atif Memon developed a framework, *GUITAR*, which generates and runs test cases on a GUI, using both reverse engineering and model-based testing techniques [19]. *GUI Ripper*, a component of this framework, extracts a GUI model, representing the structure of the GUI as a Forest (graph, which relates the different windows to be opened in the AUA) and the behaviour as an event flow graph (EFG, graph, which relates the different events, which may take place in the AUA) and as an integration tree (tree, which relates the different components of the AUA) [20]. These are used by the remaining framework tools to generate and run the tests.

C. Conclusions

Some information can only be extracted using a dynamic reverse engineering approach, such as concurrency and memory management. Besides, when working with object oriented programs, it is hard to understand the behaviour and even which objects are instantiated through a static analysis, in which case a dynamic approach may be useful [8].

Most of the dynamic approaches presented in this Section generate a FSM model. However, such models lack data like

the navigation map (the set of windows that it is possible to open and the actions needed to open such windows), information about whether or not a window is modal and dependencies among GUI elements.

GUITAR generates a GUI Forest, an EFG and an integration tree. GUI forest represents all the windows of the application. However, it does not describe the interaction steps required to open such windows. In the EFG, two events $e1$ and $e2$ are connected when $e2$ can occur after $e1$. However, this graph does not describe when an event is initially disabled and when an event makes another event possible. So, the behaviour that test cases generated from such models may check during test execution is somehow limited.

III. REGUI

The problem at hand in this research work is to diminish the effort of producing visual and formal models of the GUI of a software application for testing purposes. The approach followed by this work is to extract the necessary information from the application while it is running, i.e., dynamic reverse engineering the AUA.

A. Architecture

Figure 1 depicts the architectural organisation of all the components used by the ReGUI tool.

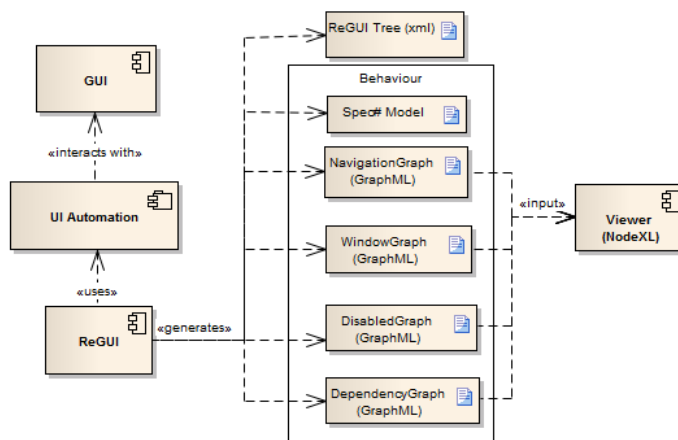


Fig. 1. Architecture of ReGUI

ReGUI uses UI Automation in order to interact with the GUI. UI Automation [21] is the accessibility framework for Microsoft Windows, available on all operating systems that support Windows Presentation Foundation. This framework represents all the applications opened in a computer as a tree (a *Tree Walker*), whose root is the Desktop and whose nodes are the applications opened at a certain moment. The GUI elements are represented as nodes, which are children of the application to which they belong. In the UI Automation framework each of these elements is an Automation Element.

At the end of the execution, the ReGUI tool generates six documents: one to represent the structure of the GUI (*ReGUITree.xml*) and five others to represent its behaviour.

Four of these files are GraphML [22] files: *NavigationGraph.xml*, which represents the navigation map of the GUI, *WindowGraph.xml*, which represents the window graph, *DisabledGraph.xml*, which represents the disabled graph, and *DependencyGraph.xml*, which represents the dependency graph. These files are used as inputs for NodeXL [23], which is a template for Microsoft Excel, that enables the visualisation of the graphs. There is another specification file, written in Spec#, which is input of the test case generation within the AMBER iTest project, and is a specification model read by a MBT tool. These files are described in more detail in Section III-D.

B. Front-End

The ReGUI front-end is shown in Figure 2. This tool is based on the development of a previous one presented in [24]. ReGUI Tool v2.0 is fully automatic and uses a different approach from the previous version of the tool so the results achieved, such as dependencies and graphs produced, are different.

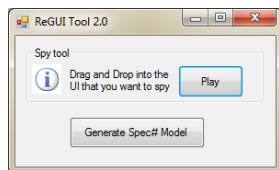


Fig. 2. ReGUI front-end

In order to start the extraction process, it is necessary to identify the GUI to be analysed. In order to do so, it is necessary to drag the *Spy Tool* symbol and drop it on top of the GUI. Following, the user must press the button *Play*, which will start the exploration process. The name of this button changes to *Playing* during the execution and to *Again* at the end. Finally, the user may press the *Generate Spec# Model* button in order to generate the Spec# model. If, for some reason, the user intends to run the ReGUI on the same AUA once more, pressing the button *Again* (button *Play* at the end of the execution) will restart the process.

C. Exploration Process

The exploration process is divided in two phases. The first one navigates through every menu option in order to verify which GUI elements are enabled and which are disabled in the beginning of the execution, i.e., the initial state of the GUI. The second phase also navigates through all the menus but this time the ReGUI tool interacts with all the menus that are enabled at that point. After interacting with each menu item, the ReGUI tool verifies if any window opened, closing it afterwards. Following, ReGUI opens all the menus again in order to verify if any state changed, i.e., if an element previously enabled became disabled or vice-versa.

During the development of the ReGUI tool, it was necessary to face some challenges that are described next:

1) *Identification of GUI elements*: GUI elements may have dynamic properties, i.e., properties which vary along the execution, such as the *automationId* and the *RuntimeIdProcess*. During the exploration process, the identification of an element is performed through an heuristic based on different properties of the element. This heuristic assigns a percentage of similarity to the different elements according to the resemblance between their properties. The properties to be compared may be configurable at the beginning of the execution. Nevertheless, there are properties that allow to differentiate two GUI elements. For instance, when two GUI elements have a different *ControlType* value, they are undoubtedly different. These properties are very useful for the identification.

2) *Exploration order*: In general, the extracted information depends on the order by which the GUI is explored. Currently, ReGUI follows a depth-first algorithm, i.e., all the options of a menu are explored before exploring the next menu and the exploration of each node's children follows the order in which they appear on the GUI. However, if the exploration followed a different order, the dependencies extracted could be different. An example of such may be found in Microsoft Notepad v6.1. The menu item *Select All* requires the presence of text in the main window in order to produce any results. Since, in the beginning, there is no text in the main window, interacting with this menu item does not have any effect. After interacting with the *Time/Date* menu item, that writes the time and date in the main window, the *Select All* menu item would produce visible results (selecting the text and enabling the menu items *Cut*, *Copy* and *Delete* and disabling the menu item *Select All* itself).

3) *Synchronisation*: To automatically interact with a GUI, it is necessary to wait for the interface to respond after each action. One way to solve this problem is to add a waiting time long enough to ensure the GUI is able to respond. However, this would make the exploration process too slow. In order to surpass this problem, ReGUI checks (with event handlers) when any changes occurred in the UI Automation tree (which reflects the state of the screen in each moment) and continues after that. It is yet possible to assign a waiting time to any action, in order to check if the correspondent result could eventually take more time to occur. However, this approach does not allow the detection of a sequence of timely spaced events that are the result of the same action.

4) *Closing a Window*: During the execution it is necessary to close windows that are eventually opened, in order to continue with the exploration process. However, there is no standard way of closing them. Windows usually have a top right button for closing purposes but when this is not available it is necessary to interact with another button, which would close the window. The selection of such button is done according to configurable guidelines.

D. Outputs

A tree (ReGUI tree) and four graphs are used internally to store and represent the extracted information. Every node of

these four graphs corresponds to a node in the ReGUI tree. The information stored in these structures is used to generate the formal model in Spec#. The outputs are:

1) *ReGUI Tree*: The ReGUI tree merges all the UI Automation trees produced during the exploration process. Initially, the ReGUI tree has only the elements visible at the beginning of the exploration and, at the end, it has every element which has become visible at some point of the exploration, such as the content of the windows opened along the process and sub-menu options.

2) *Window Graph*: The window graph shows which windows may be opened in the application. Figure 5 is a visual representation of this graph, in which each node is a window. A window may be modal or modeless, being modal if it does not allow interaction with other windows of the same application while opened and modeless otherwise. An edge between two nodes $w1$ and $w2$ means that it is possible to open $w2$ by interacting with elements of $w1$.

3) *Navigation Graph*: The navigation graph represents the nodes which are relevant to the navigation, i.e., this graph stores information about which user actions must be performed in order to open the different windows of the application. The visual representation of this graph is depicted in Figure 6. A solid edge between a window $w1$ (represented by a square) and a GUI element $e1$ (represented by a circle or a triangle) means $e1$ is inside of $w1$ whilst a dashed edge between two GUI elements $e1$ and $e2$ means it will be possible to interact with $e2$ after interacting with $e1$.

4) *Disabled Graph*: The disabled graph's purpose is to show which nodes are accessible but disabled in the the first phase of exploration process described in Section III-C, i.e., which nodes are disabled (represented by a filled triangle) at the beginning of the exploration. An example of this graph is depicted in Figure 7.

5) *Dependency Graph*: A dependency between two elements A and B means that interacting with A modifies the value of a property of B . After interacting with an element ReGUI looks for any changes in the properties of the different elements (dependencies), as described in section III-C.

Figure 8 is the visual representation of the dependency graph obtained during the exploration process. A solid edge between a window $w1$ and a node $n1$ means $n1$ is inside $w1$ and a dashed edge between two nodes $n1$ and $n2$ means there is a dependency between $n1$ and $n2$.

6) *Spec# file*: The Spec# model is obtained by applying the rules of Figure 3 to the navigation graph. Each window generates a namespace and each edge generates a method annotated with [Action]. Action methods in Spec# are methods that will be used as steps within the following generated test cases. Methods without annotations are only used internally. An example of such model is shown in Figure 9.

IV. CASE STUDY

In this Section, the results of running the ReGUI tool on Microsoft Notepad v6.1 are presented. For this execution, the properties taken into consideration to compare the elements

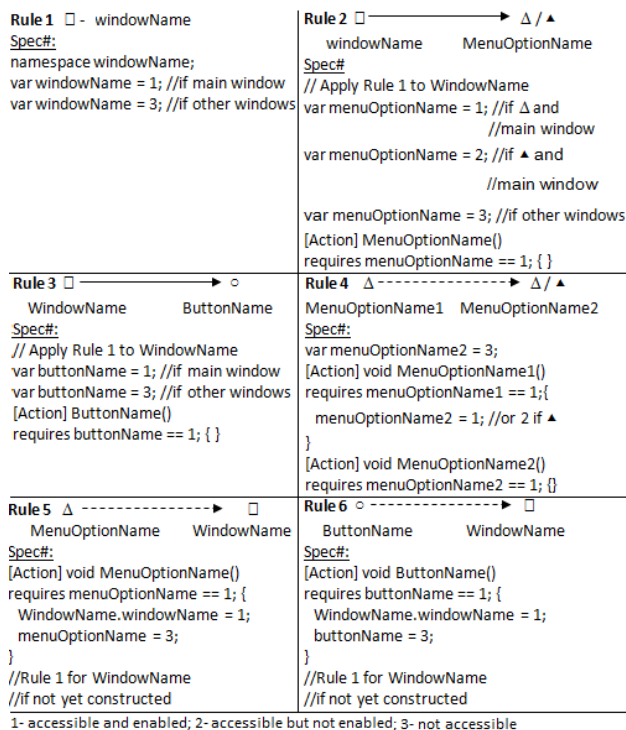


Fig. 3. Rules for the Spec# generation

were the *ControlType*, the *Name*, the *AcceleratorKey*, the *AccessKey*, the *HelpText*, the *ProcessID* and the position. In order to successfully close the windows, ReGUI looked, in this order, for buttons whose name was *Cancel*, *No*, *Close*, *Ok*, *Continue* or *X*.

Figure 4 is a simplified representation of the ReGUI tree after exploring the first menu, the menu *File*.

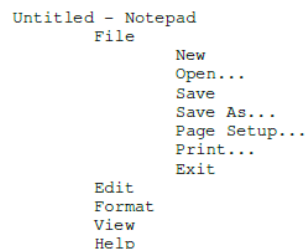


Fig. 4. Part of the ReGUI tree when exploring the menu item *File*

The visual representation of the window graph is represented in Figure 5. In this case, it is possible to conclude that the window *Open*, which is modal, and the window *Windows Help and Support*, which is modeless, may both be opened from the main window of the AUA.

Figure 6 shows the visual representation of the navigation graph. In this example, it is possible to depict that to open the *Save As* window, it is necessary to interact with the menu item *File* and then interact with the menu item *Save* or with the menu item *Save As*. Clicking on the button *Close*, belonging to the window *Save As*, this window is closed and the main

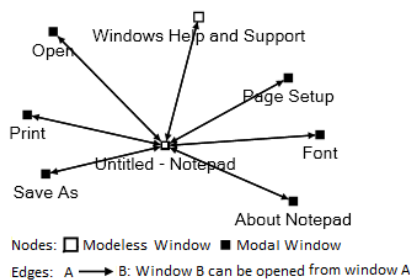


Fig. 5. Visual representation of the window graph

window gets the focus again.

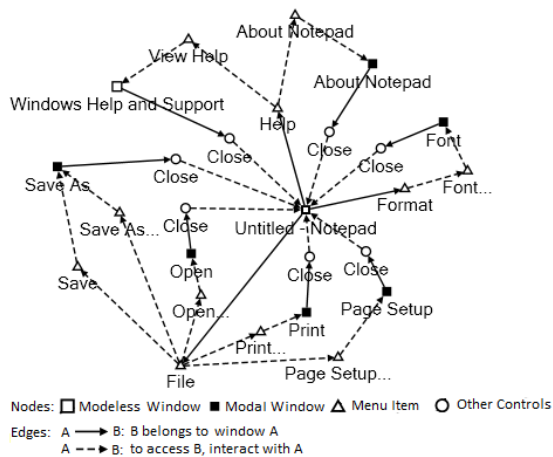


Fig. 6. Visual representation of the navigation graph

Analysing Figure 6 it is possible to verify the window *Find* is not opened during the exploration process as it requires previous insertion of text in the main window.

Figure 7 is the visual representation of the disabled graph, obtained during the first step of the exploration process. In this Figure, the set of menu items *Paste*, *Undo*, *Cut*, *Delete*, *Find Next*, *Find...* and *Copy* are initially disabled. The menu item *Edit* is represented only because it is the father of these menu items.

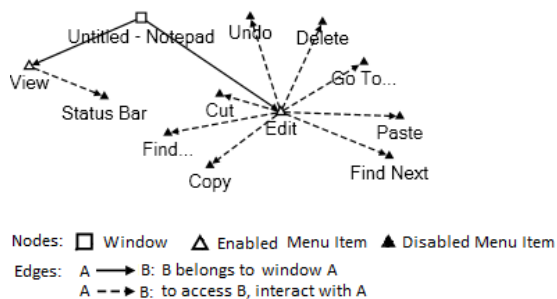


Fig. 7. Visual representation of the disabled graph

Figure 8 shows the visual representation of the dependency graph. When interacting with the menu item *Time/Date*, the

menu item *Undo*, which was initially disabled, as depicted in Figure 7, becomes enabled. Thus, there is a dashed arrow from *Time/Date* to *Undo* in the graph.

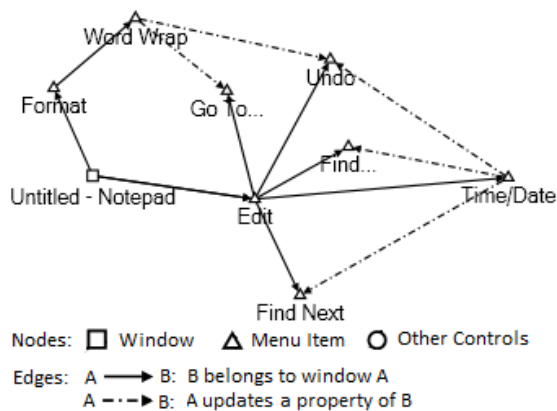


Fig. 8. Visual representation of the dependency graph

Finally, Figure 9 depicts a small sample of the generated Spec# model. The rules applied to generate this Spec# model are in comments. The first namespace corresponds to the main window of the Notepad software application. The two methods within this namespace describe the behaviour when interacting with the menu item *File* and with the menu item *Save*. The second namespace corresponds to the window *Save As* and its method describes the interaction with the button *Close* inside that window.

```

namespace Untitled__Notepad; //Rule 1
var untitle__Notepad = 1;
var menu_itemFile = 1; //Rule 2
var menu_itemSave = 3; //Rule 4
[Action] void Menu_itemFile () //Rule2
requires menu_itemSave == 1;{
    menu_itemSave = 1; //Rule 4}
[Action] void Menu_itemSave ()
requires menu_itemSave == 1; {
    menu_itemSave = 3; //Rule 5
    WindowSave_As.windowSave_As = 1;}

namespace WindowSave_As; // Rule 1
var windowSave_As = 3;
var buttonClose = 3; // Rule 3
[Action] ButtonClose()
requires buttonClose == 1;{
    buttonClose = 3; //Rule 6
    Untitled__Notepad.untitle__Notepad = 1;
}
    
```

Fig. 9. Sample of the Spec# formal model generated

For this sample of the Spec# model, no modifications should be necessary upon the manual verification.

V. CONCLUSIONS

The ReGUI tool is capable of extracting important information about the behaviour of the AUA, such as navigational information and which GUI elements become enabled or disabled after interacting with another element. The exploration

process is fully automatic. The user just has to point out the AUA.

ReGUI generates graphs, which are useful to quickly visualise the structure and behaviour of the AUA in order to understand its functioning. Also, an important part of the Spec# model is already generated by the tool.

When comparing with the *GUITAR* framework described in Section II-B, it is possible to verify that there is a similarity between the information stored in its GUI Forest and the information stored in both the Window graph and the ReGUI tree as the GUI Forest has information about which window may be opened from another window, along with the structure of each of those windows. The main advantage of the approach described in this paper is that it collects important behavioural information, such as dependencies, and the actions needed to open the several windows of the AUA.

ReGUI has still some limitations. For instance, currently, it only supports interaction through the *invoke pattern* [21] but it may evolve to interact through other *patterns*. In addition, it just tries to open windows from the main window and there are still other dependencies that may be explored. Nevertheless, these limitations could be overcome in a following version of the tool. It is yet objective of the authors to analyse the tools response to more complex systems in order to accurately evaluate the quality of the extracted dependency model.

One of the main difficulties faced during the development of ReGUI was the lack of GUI standards. For example, generally, an opened window is, in the UI Automation tree, child of the main application. However, there are some which are siblings of the application. Furthermore, although each window should have an element called *system menu bar*, which corresponds to the top bar where you can usually find the *minimise*, *maximise* and *close* buttons, some windows do not have that element.

This research work was developed on the context of a project with testing purposes, the AMBER iTest. However, once the model is generated and verified it is possible to use it for other purposes. For instance, to use this model to generate code in languages different from the original one, such as transforming a C# application into a Java application or the other way around.

REFERENCES

- [1] S. E. Group, "Amber itest - an automated model-based user interface testing environment," October 2008, http://paginas.fe.up.pt/softeng/wiki/doku.php?id=projects:amber_itest:start, last access on December 2010.
- [2] M. Barnett, K. R. M. Leino, and W. Schulte, "The spec# programming system: An overview," in *CASSIS International Workshop*, March 2004.
- [3] M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, and L. Nachmanson, "Formal methods and testing," R. M. Hierons, J. P. Bowen, and M. Harman, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Model-based testing of object-oriented reactive systems with spec explorer, pp. 39–76.
- [4] A. C. R. Paiva, J. C. P. Faria, N. Tillmann, and R. F. A. M. Vidal, "A model-to-implementation mapping tool for automated model-based gui testing," in *7th International Conference on Formal Engineering Methods*, November 2005.
- [5] E. J. Chikofsky and J. H. Cross II, "Reverse engineering and design recovery: A taxonomy," *IEEE Softw.*, vol. 7, pp. 13–17, January 1990.
- [6] T. Systä, "Dynamic reverse engineering of java software," in *Proceedings of the Workshop on Object-Oriented Technology*. London, UK: Springer-Verlag, 1999, pp. 174–175.
- [7] M. J. Pacione, M. Roper, and M. Wood, "A comparative evaluation of dynamic visualisation tools," in *Proceedings of the 10th Working Conference on Reverse Engineering*, ser. WCRE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 80–.
- [8] T. Systä, "Static and dynamic reverse engineering techniques for java software systems," Ph.D. dissertation, Faculty of Economics and Administration of the University of Tampere, Kalevantie 4, FI-33014 University of Tampere, Finland, 2010.
- [9] L. Bouillon, Q. Limbourg, J. Vanderdonck, and B. Michotte, "Reverse engineering of web pages based on derivations and transformations," in *Proc. of 3rd Latin American Web Congress LA-Web2005 (Buenos Aires, October 31-November 2, 2005)*, IEEE Computer Society Press, Los Alamitos, 2005, 2005, pp. 3–13.
- [10] J. Vanderdonck, L. Bouillon, and N. Souchon, "Flexible reverse engineering of web pages with vaquista," in *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*, ser. WCRE '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 241–248.
- [11] J. C. C. J. C. Silva and J. A. Saraiva, "Gui inspection from source code analysis," *Electronic Communications of the EASST*, 2010, to appear.
- [12] Y. fan R. Chen, G. S. Fowler, E. Koutsofios, and R. S. Wallach, "Ciao: A graphical navigator for software and document repositories," in *In International Conference on Software Maintenance*. IEEE Computer Society, 1995, pp. 66–75.
- [13] M. P. Chase, S. M. Christey, D. R. Harris, and A. S. Yeh, "Managing recovered function and structure of legacy software components," in *Proceedings of the Working Conference on Reverse Engineering (WCRE'98)*, ser. WCRE '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 79–.
- [14] Q. Limbourg, J. Vanderdonck, B. Michotte, L. Bouillon, and V. Lpez-Jaquero, "Usixml: A language supporting multi-path development of user interfaces," in *EHCI/DS-VIS*, ser. Lecture Notes in Computer Science, R. Bastide, P. A. Palanque, and J. Roth, Eds., vol. 3425. Springer, 2004, pp. 200–220.
- [15] R. K. Shehady and D. P. Siewiorek, "A method to automate user interface testing using variable finite state machines," in *Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*, ser. FTCS '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 80–.
- [16] S. Fujiwara, G. von Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Trans. Softw. Eng.*, vol. 17, pp. 591–603, June 1991.
- [17] J. Chen and S. Subramaniam, "A gui environment to manipulate fsm's for testing gui-based applications in java," in *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 9 - Volume 9*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 9061–.
- [18] L. Williams, "Testing overview and black-box testing techniques," 2006.
- [19] D. R. Hackner and A. M. Memon, "Test case generator for guitar," in *Companion of the 30th international conference on Software engineering*, ser. ICSE Companion '08. New York, NY, USA: ACM, 2008, pp. 959–960.
- [20] A. M. Memon, I. Banerjee, and A. Nagarajan, "GUI ripping: Reverse engineering of graphical user interfaces for testing," in *Proceedings of The 10th Working Conference on Reverse Engineering*, Nov. 2003.
- [21] R. Haverty, "New accessibility model for microsoft windows and cross platform development," *SIGACCESS Access. Comput.*, pp. 11–17, June 2005.
- [22] U. Brandes, M. Eiglsperger, and J. Lerner, "Graphml primer," April 2007, <http://graphml.graphdrawing.org/primer/graphml-primer.html>, last access on July 2011.
- [23] Microsoft, "Nodexl: Network overview, discovery and exploration for excel," March 2011, <http://nodexl.codeplex.com/>, last access on May 2011.
- [24] A. M. P. Grilo, A. C. R. Paiva, and J. P. Faria, "Reverse engineering of gui models for testing," in *5a Conferencia Ibrica de Sistemas y Tecnologias de la Informacin*, July 2009.

Towards Design Method Based on Formalisms of Petri Nets, DEVS, and UML

Radek Kočí and Vladimír Janoušek

Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
{koci,janousek}@fit.vutbr.cz

Abstract—Software system development uses specific development techniques and processes to reach desired goals, whereas different kinds of systems usually need to use different approaches. Obviously, there are used different techniques, tools, and formalisms in each development process and the designed models should be automatically or manually transformed to the next development step. The paper is aimed at such development processes, which work with formalisms allowing to design architecture and functionality, analysis of design, testing and system run with no need to change this formalism. Nevertheless, there can be useful to combine more different formalisms and model languages because of developers are used to use these formalisms or there are already created models using these formalisms. The paper deals with UML, Petri Nets, and DEVS application in the systems design and sketches a method how to use the formalisms for modeling a system architecture and its behavior. Its combination decreases a number of transformations of models and makes the architectural description well-arranged.

Keywords—Simulation-Based Design, Object-Oriented Petri Nets, DEVS, UML.

I. INTRODUCTION

The key activities in the system development are specification, testing, validation, and analysis (e.g., of performance, throughput, etc.). Most of the methodologies use models for system specification, i.e., for defining the structure and behavior of developed system. There are different kinds of models, from models of low-level formal basis to pure formal models. Each kind has its advantages and disadvantages. The most popular modeling language in software engineering is UML [1]. It serves as a standard for analytics, designers and programmers. But, own phraseology of UML does not have enough power allowing to realize some fundamental relationships and, in particular, rules, that are branch of every modeled system. For example, how can we define a condition that at least one item has to be at a stack when the operation *pop* is called? The pure UML language does not offer suitable tools. Although the UML language can be completed by OCL (Object Constraint Language), stereotypes, etc., which makes the system description more precise, it makes the checking of system correctness or validity by means of testing or formal methods very complicated.

Therefore, the new methodologies and approaches are investigated and developed for many years. They are com-

monly known as Model-Driven Software Development or Model-Based Design (MBD) [2], [3], [4]. An important feature of these methods is the fact that they use executable models, e.g., Model Driven Architecture (MDA) and Executable UML [5], allowing to simulate models, i.e., to provide simulation testing. The pure formal models (e.g., Petri Nets, calculus, etc.) allow to use formal or simulation approaches to complete the testing, verification, and analysis activities. There is no need of model generation or transformation due to simulation purposes. We only add simulated inputs and expected results and can change any model element for its simulated version [6].

The development methods such as MDA [7] allow for semi-automatic translation of designed models to implementation language (i.e., the code generation). Nevertheless, the result has to be finalized manually, so it entails a possibility of semantic mistakes or imprecision between models and transformed code. In comparison with semi-formal models, formal models bring the clear and understandable modeling and the possibility to check correctness with no need for model transformation. The design technique, which is taken into account in this paper [8] derives benefit from formalisms of Object Oriented Petri Nets (OOPN) [9], [10] and DEVS [11]. These formalisms can be directly interpreted and, consequently, integrated into the target system [12].

The paper is organized as follows. First, we briefly introduce used formalisms of OOPN, DEVS, and UML and the design technique. The next chapter deals with architectural description using different formalisms and the fourth section deals with description of behavior.

II. MODELING FORMALISMS

A. UML

The UML modeling uses a notion *view*. A view of a system is a projection of the system on one of its relevant aspects. Such a projection focuses on certain aspects and ignores others. Therefore it is useful to have different views of a system. UML uses multiple notations (tools) for exhibiting different *views* of the system. We can distinguish four main views: The *structural view* describes layout between objects and classes, their associations and their possible communication channels. The *behavioral view* describes,

how the system components interact, and characterizes the response to external system operations. The *data view* describes the state of the system units (objects) as well as their relationships. The *interface view* focuses on the encapsulation of system parts, and the possible usage from outside.

UML currently has as many as eleven different notations, which constitute different views of UML designs. In the following, we briefly discuss several notations and their usage to describe certain aspects. *Use case diagrams* display the relationship among actors and use cases. A use case is a set of scenarios describing an interaction between a user and a system. The two main components of a use case diagram are use cases and actors. Use case diagrams deal with an interface and behavioral view at the border of the system. *Sequence diagrams* can be used to demonstrate the interaction of objects in a use case. They generally show the sequence of events that occur. Sequence diagrams therefore clearly define behavioral aspects but are based on structural and interface views. They do not describe an internal behavioral of objects. *Class diagrams* are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. They are the central notion for structural aspects. *State Diagrams* are used to describe the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system. *Activity Diagrams* are defined as a special case of a state diagrams. They could be useful for describing internal processing of operations or use cases. Activity diagram models a dynamic flow controlled by internal stimuli.

B. OOPN

An OOPN is a set of classes specified by high-level Petri nets. An *object-oriented Petri net* is a triple (Σ, c_0, oid_0) where Σ is a system of classes, c_0 an initial class, and oid_0 the name of an initial object from c_0 . A *class* is mainly specified by an object net and a set of method nets. Object nets describe possible autonomous activities of objects, while method nets describe reactions of objects to messages sent to them from the outside.

Object nets consist of places and transitions. Every place has its initial marking. Every transition has conditions (i.e., inscribed testing arcs), preconditions (i.e., inscribed input arcs), a guard, an action, and postconditions (i.e., inscribed output arcs). *Method nets* are similar to object nets but, in addition, each of them has a set of parameter places and a return place. Method nets can access places of the appropriate object nets in order to allow running methods to modify states of objects, which they are running in. *Synchronous ports* are special (virtual) transitions, which cannot fire alone but only dynamically fused to some other

transitions, which activate them from their guards via message sending. Every synchronous port embodies a set of conditions, preconditions, and postconditions over places of the appropriate object net, and further a guard, and a set of parameters. Parameters of an activated port s can be bound to constants or unified with variables defined on the level of the transition or port that activated the port s . *Negative predicates* are special variants of synchronous ports. Its semantics is inverted—the calling transition is fireable if the negative predicate is not fireable.

The OOPN dynamics is based on high-level Petri net dynamics, but the semantics of a transition is little bit modified. A transition is *fireable* for some binding of variables, which are present in the arc expressions of its input arcs and in its guard expression, if there are enough tokens in the input places with respect to the values of input arc expressions and if the guard expression for the given binding evaluates to true. A state of the running OOPN model has the form of a marking of a system of net instances. Each net marking consists of places and transitions marking.

C. DEVS

DEVS is a formalism, which can represent any system whose input/output behavior can be described as sequence of events. DEVS is specified as a structure

$$M = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta)$$

where X is the set of input event values, S is the set of state values, Y is the set of output event values, δ_{int} is the internal transition function, δ_{ext} is the external transition function, λ is the output function, and ta is the time advance function. At any time, the system is in some state $s \in S$. If no external event occurs, the system is staying in state s for $ta(s)$ time. If elapsed time e reaches $ta(s)$, then the value of $\lambda(s)$ is propagated to the output and the system state changes to $\delta_{int}(s)$. If an external event $x \in X$ occurs on the input in time $e \leq ta(s)$, then the system changes its state to $\delta_{ext}(s, e, x)$.

This way we can describe atomic models. Atomic models can be coupled together to form a coupled model. The later model can itself be employed as a component of larger model. This way the DEVS formalism brings a hierarchical component architecture.

D. DEVS and OOPN Wrapping

The DEVS formalism, especially its composite model concept, is suitable as a component platform for multi-paradigm modeling and simulation where atomic models are specified by other formalisms. On the other hand, OOPN is a powerful language allowing a high-level description of model dynamics. The OOPN formalism can be wrapped in the DEVS formalism. In such a case, the OOPN models are atomic components of a hierarchical DEVS model.

Let $M_{PN} = (M, \Pi)$ be a DEVS component M , which wraps an OOPN model Π , c_0 is an initial class of the model Π , and oid_0 is an initial object of the class c_0 . Then we define a set of places of the object net oid_0 as $P(oid_0)$, a set of input places $P_{inp} \subseteq P(oid_0)$, and a set of output places $P_{out} \subseteq P(oid_0)$, where $P_{inp} \cap P_{out} = \emptyset$.

Sets X, Y from DEVS formalism are to be specified as structured sets. It allows to use multiple variables for specification of state and we can use input (V_X) and output ports (V_Y) for input and output events specification. Then we can define a mapping of OOPN places into DEVS ports as bijections $map_{inp} : P_{inp} \rightarrow V_X$ and $map_{out} : P_{out} \rightarrow V_Y$.

Informally, if an OOPN model is defined as a DEVS component, then an object net of initial class defines input and output places, this class is instantiated immediately the component is created, and the defined places serve as input or output ports of the component.

E. Design Techniques

As in every design techniques the most problem at its usage is by estimation to abstraction level. Primarily there have to be found essential objects of a modeled system and their relationships. There we can successfully employ resources of UML such us *Use Case, Sequence, Class* diagrams. As UML, the development processes based on OOPN use the concept of *view*. The basic view is the *data view*, the structure encapsulating data and basic behavior on them. The data view can have different roles in the system, whereas each role is described by another view, which encapsulates the original view. Therefore, views create a hierarchical structure where the higher view encapsulates the view on the lower level. The communication and synchronization between views are provided by means of synchronous ports and negative predicates (more details will be demonstrated in the chapter IV-B).

III. SPECIFICATION OF SYSTEM ARCHITECTURE

The system design should be structured into units, whereas each unit is responsible to serve relatively independent activities. The unit can be an object, a package, a component, etc. These units communicate each other by means of specified interfaces. The interface is usually formed by the protocol offered by the object, by the class of component, etc. The units communicate by message passing whereas this communication is usually synchronous.

A. UML-based Specification

Let us have the units U_1 and U_2 , the unit U_1 has an interface which is specified by the class C_1^{U1} . If the object from unit U_2 wants to use the unit U_1 , it sends a message M_1 to an object O_1^{C1} , which is derived from the class C_1^{U1} . It means that there has to be instances of classes, which constitute the unit interface. If there is a special request to interface (just one class instance, asynchronous

communication, etc.), it should be handled in a special way. The system architecture is obviously specified by class diagrams and package diagrams in UML language. The example of this situation is shown in the Figure 1.

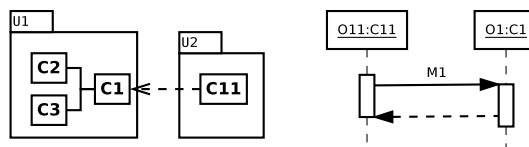


Figure 1. The architectural design using UML.

B. OOPN and DEVS-based Specification

Now, let us have the similar situation, but we use the formalisms DEVS and OOPN for specification of the system architecture. This approach considers the unit as a component based on the DEVS formalism. It means that the interface is established by ports, components are connected via their input and output ports. Let us have the unit U_1 having one input port P_{I1}^{U1} and the unit U_2 having one output port P_{O1}^{U2} . If the component U_2 wants to communicate with the component U_1 , it puts the data into the port P_{O1}^{U2} . The data are then transferred to the port P_{I1}^{U1} and the component can react. This communication is asynchronous (the sender does not wait for the answer). This situation is shown in the Figure 2a), which depicts the connection between two units U_1 and U_2 and the Figure 2b), which depicts implementation in the initial object of the unit U_2 . There is the place $PO1$ representing the output port P_{O1}^{U2} . After the data is put in this port, the next operation of the component U_2 represented by the transition t_2 can execute.

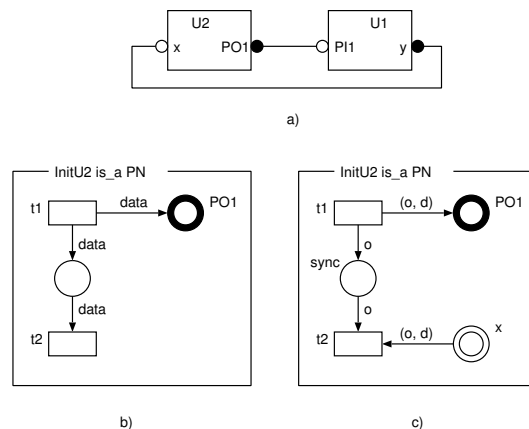


Figure 2. The architectural design using DEVS and OOPN.

Of course, the different requests to interface (e.g., synchronous communication) should be handled, but it can be modeled by a simple way as it is shown in the Figure 2c). The specific identification (object, symbol, etc.) is joined

to transferred data and the next operation represented by the transition t_2 will be executable after the respond is accepted, i.e., the data joined with the identification is put in the input port x (place x).

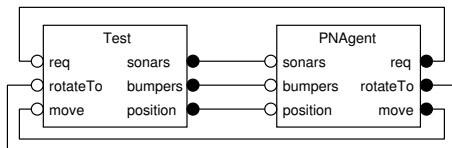


Figure 3. The architectural design using DEVS and OOPN: real example.

The Figure 3 shows a real example of using DEVS and OOPN. There are two units (components) representing one agent (the component PNAgent) and its simulated environment (the component Test). PNAgent receives input from the environment (actual data from sensors) and react by putting commands into its output ports.

IV. SPECIFICATION OF SYSTEM BEHAVIOR

There are a number of techniques in UML to model dynamic aspects of a system. If these techniques are used, the designer statically describes a behavior of a system in a design phase and he cannot make certain of his partial ideas about the system behavior. Next, there can be a problem with understanding to models (diagrams), which arise from graphical whatness of these notions. We could describe this situation by following words: *The nice thing about graphical description techniques is that everyone understands them, the bad thing is that everyone understands them in a different way.* Therefore, the given notions lack a formal foundation, which would make possible understand the model in a only way and also make possible analysis and verifications of the system.

For demonstration purposes, we have chosen a part of the PNTalk system. PNTalk is the tool intended to model a simulate systems using OOPN. We will model a PNTalk processor, which is a central part of the PNTalk system and execute an OOPN model. We will keep an experimental implementation from 2008 [13] in view. First, we use UML to model a chosen part. Second, we adapt these models into OOPN-based models to show how they can be used together.

A. UML-based Modeling

The PNTalk model specifies an OOPN. The OOPN is a set of classes of objects; objects are instances of classes. These classes are translated into *an internal representation* of the PNTalk processor. An internal representation consists of objects corresponding to classes defined by a model. These objects are derived from special classes PNClass and PNSuperClass, as we can see in the Figure 4.

There we can see that both classes and objects of OOPN are represented by instances of PNTalk classes PNClass

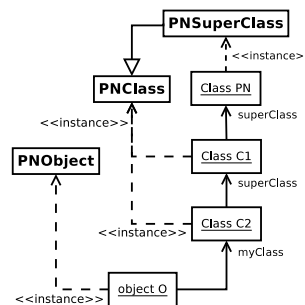


Figure 4. Class diagram of the PNTalk core.

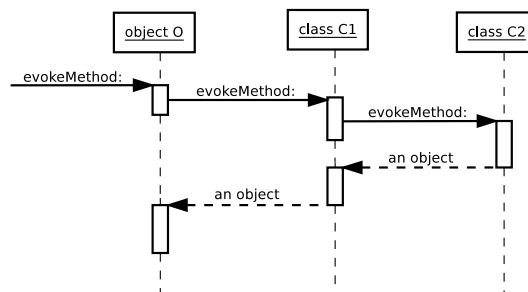


Figure 5. Sequence diagram of the method invocation.

(or PNSuperClass) and PNOBJECT. Objects whose name starts with *Class* represent classes of an OOPN model. An object named *object O* represents an instance of OOPN class C2 during a simulation of OOPN model. An OOPN class *Class PN*, derived from class PNSuperClass, is a special object, which always stays at the top of an inheritance hierarchy of OOPN classes. The inheritance of OOPN classes are represented as an association between appropriate objects, instances of the class PNClass (respectively PNSuperClass). The object of the OOPN super class plays the role of *superClass*. The relationship between OOPN object and its OOPN class is represented by an association between appropriate object. The object of OOPN class plays the role of *myClass*. Summary, the Figure 4 shows one object (it is derived from a class PNClass), two OOPN classes (they are derived from a class PNSuperClass) and one OOPN class, which is derived from class PNSuperClass.

We depict functionality by a part of the PNTalk processor—calling methods of OOPN objects. Because classes of the OOPN model are translated into internal-representation objects in implementation environment, there cannot be applied natural inheritance of given environment there. On that account there must be implemented own inheritance hierarchy direction. Let us specify a letter O as an OOPN object, which is derived from a OOPN class C. Remember these two elements are objects in the PNTalk processor internal-representation. Now, if we attempt to call

an OOPN method M of an object O , the object O devolve this requirement upon a class C . The class C looks for an object, which represents a called method M . If this object has been found, the class C gives it back; if does not, the class C generates a fault. This have been a simple variant without an inheritance. As soon as we add an inheritance, the scenario of an unsuccessful search is changed. The class attempt devolve a requirement upon a superior class (if exists), which process this requirement in the same way like there have been already said. Only if a top class in inheritance hierarchy does not find a called method, there is generated a fault.

If we take a look at the Figure 5, we can see a sequence diagram of an object behavior based on external events. A filled arrow-head stands for an invocated event, the dashed one stand for a returned result of event processing.

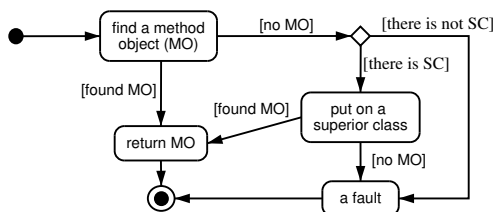


Figure 6. Activity diagram of the method invocation.

Now we focus on an internal behavior of an object representing a PNTalk class close to a method invocation. This is depicted at the Figure 6 subscribing with a description, which has been already depicted there. The letters SC at the picture means a shortcut of a *super class*.

B. OOPN-based Modeling

We have dealt with UML modeling of our chosen part of a system so far. Now we pay attention to OOPN-based modeling. We create the OOPN model according to previous diagrams and designed methodology [8].

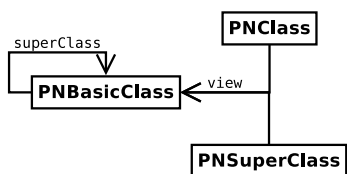


Figure 7. Class diagram of the PNTalk core in the SBD methodology.

The model consists of three classes (see the Figure 7)—PNBasicClass representing a data view, PNCClass representing a view of the OOPN class, and PNSuperClass representing the first class in the OOPN inheritance hierarchy. The first class has a bit different behavior, hence it is modeled as a special case. The views modeled by classes PNCClass and PNSuperClass represent roles what the basic subject can play in the system. We can see, that the

inheritance in design is replaced by object composition, each new view creates a new composition of objects.

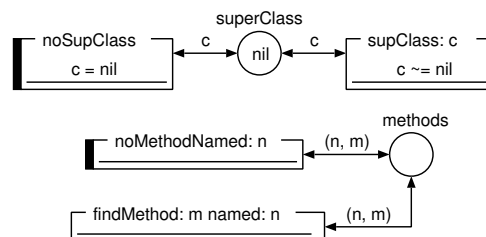


Figure 8. OOPN model representing the class PNBasicClass.

The Figure 8 shows an object net of the class PNBasicClass representing a view of data. There are modeled two places, which store information about the super class (the place *superClass*) and defined methods (the place *methods*). The super class is always represented by some of defined view—in this examples it can be view derived from the class PNCClass or PNSuperClass. There are defined a pair of synchronous port and negative predicate for each place. Synchronous ports (*supClass:* and *findMethod: named:*) allows for finding and getting appropriate data, i.e., the super class or method with given name. If such a method exists (is stored in the place *methods*), the object representing such a method is bound to the variable m and the calling transition is able to work with it (see the Figure 9). Similarly, if the super class is not equal to *nil* and the synchronous port is called, the super class is bound to the variable c . The negative predicates (*noSupClass* and *noMethodNamed:*) are true if there is no super class or no method with given name. Of course, there should be methods for setting the super class and for adding methods, but they are not shown due to simplicity.

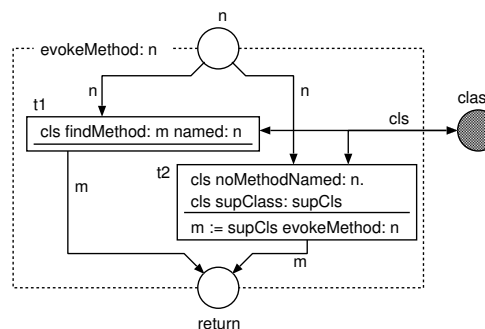


Figure 9. Method evokeMethod: of the view (class) PNCClass.

The Figure 9 shows the method *evokeMethod:* of the view PNCClass, which encapsulates the class PNBasicClass (the encapsulated object is stored in the place *class*). The method has one parameter (a name of the method, which is to evoked). This method process a

requirement to a method invocation of OOPN object (and its relevant OOPN class).

If any OOPN method is invoked, the method *evokeMethod:* of the relevant view *PNClass* is called. The method is processed as follows. The transitions t_1 and t_2 are tested. The system of pairs of synchronous port and negative predicate ensures that just one of these transition can be fired. If the encapsulated object of the class *PNBasicClass* contains a method matching given name n , the synchronous port *findMethod: named:*, together with the transition t_1 , is fired and the found method (the object representing the method, respectively) is bound to the variable m and returned as a result of the method.

If there is no such a method, the negative predicate *noMethodNamed:* called from the transition t_2 is true. After firing this transition, the synchronous port *supClass:*, placed as a second condition in the guard, binds the super class to the variable *cls*. Then the method *evokeMethod:* is called on the super class and its result is returned.

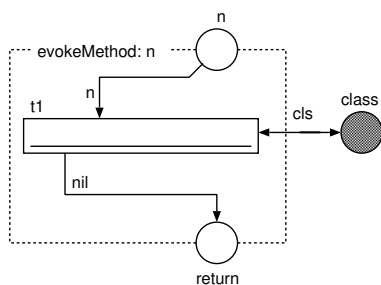


Figure 10. Method *evokeMethod:* of the view (class) *PNSuperClass*.

If the view on the super class is derived from the class *PNSuperClass*, the search of a method object is unsuccessful and its method *evokeMethod:* give back an object *nil* (see the Figure 10). There should be only one view derived from this class representing the first class *PN* in the OOPN inheritance hierarchy.

V. CONCLUSION

The paper dealt with formalisms of OOPN, DEVS, and UML used in the system design. In comparison with UML, using formalisms of DEVS and OOPN for the architectural specification decreases a number of communication points and makes the architectural specification well-arranged. Moreover, selected models in UML can be transformed to DEVS component described by OOPN formalisms in a simple way. The presented approach is a part of the development methodology, which allows to use formal models in all phases of system development including as basic design, analysis and also programming means with a vision to allow to combine simulated and real components and to deploy models as the target system with no code generation. In the

future, we plan to formalize outlined approach and to present complex case study.

Acknowledgment: This work was partially supported by the BUT FIT grant FIT-S-11-1 and the Ministry of Education, Youth and Sports under the contract MSM 0021630528.

REFERENCES

- [1] J. Arlow and I. Neustadt, *UML and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison-Wesley Professional, 2001.
- [2] S. Beydeda, M. Book, and V. Gruhn, *Model-Driven Software Development*. Springer-Verlag, 2005.
- [3] J. Greenfield, K. Short, S. Cook, S. Kent, and J. Crupi, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, 2004.
- [4] M. Broy, J. Gruenbauer, D. Harel, and T. Hoare, Eds., *Engineering Theories of Software Intensive Systems: Proceedings of the NATO Advanced Study Institute*. Kluwer Academic Publishers, 2005.
- [5] C. Raistrick, P. Francis, J. Wright, C. Carter, and I. Wilkie, *Model Driven Architecture with Executable UML*. Cambridge University Press, 2004.
- [6] R. Kočí and V. Janoušek, "OOPN and DEVS Formalisms for System Specification and Analysis," in *The Fifth International Conference on Software Engineering Advances*. IEEE Computer Society, 2010, pp. 305–310.
- [7] D. S. Frankel, *Model Driven Architecture: Applying Mda to Enterprise Computing*, ser. 17 (MS-17). John Wiley & Sons, 2003.
- [8] R. Kočí and V. Janoušek, "System Design with Object Oriented Petri Nets Formalism," in *The Third International Conference on Software Engineering Advances Proceedings ICSEA 2008*. IEEE Computer Society, 2008, pp. 421–426.
- [9] M. Češka, V. Janoušek, and T. Vojnar, *PNtalk — a Computerized Tool for Object Oriented Petri Nets Modelling*, ser. Lecture Notes in Computer Science. Springer Verlag, 1997, vol. 1333, pp. 591–610.
- [10] V. Janoušek and R. Kočí, "PNtalk Project: Current Research Direction," in *Simulation Almanac 2005*. FEL ČVUT, Praha, CZ, 2005.
- [11] B. Zeigler, T. Kim, and H. Praehofer, *Theory of Modeling and Simulation*. Academic Press, Inc., London, 2000.
- [12] R. Kočí and V. Janoušek, *Simulation Based Design of Control Systems Using DEVS and Petri Nets*, ser. Lecture Notes in Computer Science. Springer Verlag, 2009, vol. 5717, pp. 849–856.
- [13] V. Janoušek and R. Kočí, "Embedding Object-Oriented Petri Nets into a DEVS-based Simulation Framework," in *Proceedings of the 16th International Conference on System Science*, ser. volume 1, 2007, pp. 386–395.

Invariant Preservation by Component Composition Using Semantical Interface Automata

Sebti Mouelhi, Samir Chouali, Hassan Mountassir

Computer Science Laboratory (LIFC),

University of Franche-Comté, Besançon, FRANCE

Email: {sebti.mouelhi, samir.chouali, hassan.mountassir}@lifc.univ-fcomte.fr

Abstract—Component assembly is based on the verification of the compatibility between the component interface specifications. In general, these specifications do not combine the three levels of the compatibility check: behavioral protocols, signatures, and semantics of operations. In this paper, we enrich the formalism of interface automata, used to specify component protocols, by the signatures and semantics of operations. We propose a new formalism, called “semantical interface automata” (SIAs), endowed with a stronger compositional semantics than interface automata. The semantics of operations is specified by pre and post-conditions stated over their parameters and a set of variables reflecting the behavioral conduct of components interoperability. First, we show how the component compatibility is checked at the signature, semantic, and protocol levels. Second, we establish a formal methodology to check the preservation of invariants by composition of SIAs.

Keywords—software components; interface automata; action semantics; formal correctness; invariants.

I. INTRODUCTION

An individual component is a software unit of a third-party composition and deployment that encapsulates a set of implemented offered services and asks for a set of required ones [1]. The component interfaces depict its access points and it must be associated to a contractual specification that specify the necessary sufficient of its functional behavior at the levels of the signatures and the semantics of operations and the behavioral protocol, etc. [2], [3].

In this paper, we focus on assembling components whose behaviors are described by interface automata [4] enriched by the semantics of actions. The new formalism combines the protocol and the semantic levels of interface specifications, hence the name *semantical interface automata*. The actions of a semantical interface automaton are annotated by pre and post-conditions stated over the parameters of their correspondent operations and a set of *interface variables* shared by the automaton and its environment. The compatibility check of SIAs takes into account the action constraints specified by these conditions. Furthermore, we found a formal methodology to check correctness properties thanks to the rich interface description of SIAs. Correctness properties are typically *invariants* written in terms of the interface variables. The invariance properties are assessed at

all the states of a labeled transition system representation of a SIA. In particular, we study the invariant preservation by component composition.

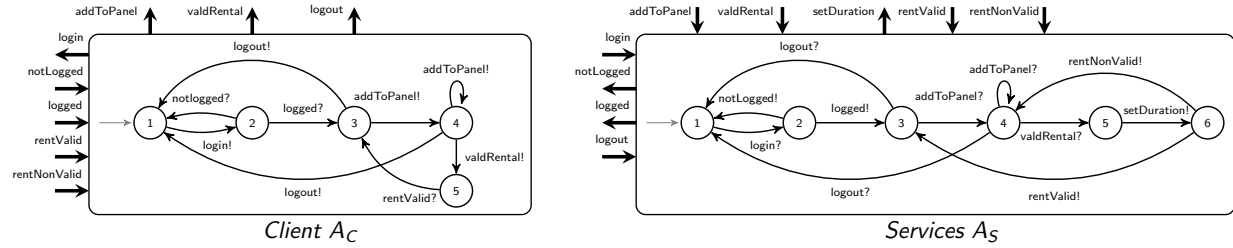
The paper is organized as follows. In Section II, we present the SIAs formalism features and how we check their composability and compatibility. In Section III, we explain how LTS representations are extracted from SIAs and how we check the preservation of invariants by composition of SIAs. Related works are presented in Section IV. The conclusion and future works are presented in Section V.

II. SEMANTICAL INTERFACE AUTOMATA

Interface automata (IAs) [4], [5] have been introduced to model both the output behavior and the environment assumptions of software components. These models are non-input-enabled I/O automata [6], which means that at every state some input actions may be non-enabled. Every component interface is described by one interface automaton where input actions are used to model methods that can be called, the end of receiving messages, and the return values from such calls, as well as exception treatment. Output actions are used to model method calls, message transmissions, exceptions, and sending return values. Hidden actions represents local operations. The alphabet of an interface automaton is built of its action names annotated by “?” for input actions, by “!” for output actions, and by “;” for hidden actions.

Before defining semantical interface automata, we start by giving some preliminaries. The signature of an action a is the signature of its correspondent operation implemented or solicited by the component that provokes the action. It has the form $a(i_1, \dots, i_n) \rightarrow (o)$ where $n \geq 0$. The set $P_a^i = \{i_1, \dots, i_n\}$ represents the set of input parameters of a . The set P_a^o is defined by the singleton $\{o\}$. The absence of input or output parameters is denoted by $()$. We suppose that an action has no signature if it corresponds to a return value.

Given a set of components \mathcal{C} , a SIA A_c of a component c in \mathcal{C} is defined in relation to the other components in $\mathcal{C} \setminus \{c\}$. This relation is based on a set of variables $V_{\mathcal{C}}$ shared between them. We denote, by D_w , the domain of a variable or a parameter w . Given a set of variables V ,


 Figure 1. The semantical interface automata of the components *Client* and *Services*

$Preds(V)$ represents the set of first order predicates whereof free variables belong to V . We denote by p' the predicate obtained by replacing v by v' in $p \in Preds(V)$. The set $Preds'(V)$ is equal to $\{p' \mid p \in Preds(V)\}$.

Definition 1: Given a component $c \in \mathcal{C}$, a semantical interface automaton $A_c = \langle S_{A_c}, i_{A_c}, \Sigma_{A_c}^I, \Sigma_{A_c}^O, \Sigma_{A_c}^H, \delta_{A_c}, L_{A_c}, V_{A_c}, Init_{A_c}, \Psi_{A_c} \rangle$ of c consists of

- a finite set S_{A_c} of states containing an initial state i_{A_c} . A is called “empty” if $S_A = \emptyset$;
- three disjoint sets $\Sigma_{A_c}^I, \Sigma_{A_c}^O$ and $\Sigma_{A_c}^H$ of inputs, output, and hidden actions;
- a set $\delta_{A_c} \subseteq S_{A_c} \times \Sigma_{A_c} \times S_{A_c}$ of transitions;
- a set L_{A_c} of local variables and a set $V_{A_c} \subseteq V_C$ of shared variables. The set $LV_{A_c} = V_{A_c} \cup L_{A_c}$ represents the set of all variables of A_c ;
- Ψ_{A_c} is a function that associates for each action $a \in \Sigma_{A_c}$ a tuple $\langle Pre_{\Psi_{A_c}(a)}, Post_{\Psi_{A_c}(a)} \rangle$ such that $Pre_{\Psi_{A_c}(a)} \in Preds(V_{A_c} \cup P_a^i)$ and $Post_{\Psi_{A_c}(a)} \in Preds(V_{A_c} \cup P_a^o \cup P_a^h)$;

According to Definition 1, the pre and post-conditions are defined only in terms of parameters and shared variables. They are used to verify the compatibility of two semantical interface automata, then they should be defined only on what is shared between them. The presence of local variables can be problematic because the local variables of one automaton are unknown to the others.

Example 1: As an example, we will consider a simple distributed multi-tier application that allows object leasing between users. The component *Services* is a server side component that plays the role of the mediator between the *Client* and the database. It provides the operations *login* that returns a reference to a persistent component *User* (the output action *logged!*) that represents the authenticated user or provokes an exception (*notLogged!*). It provides also, for the registered users, the operations *addToPanel*, *valdRental*, and *logout*. The first method makes an object to lend in the member panel, the second one validates its request to rent the objects saved in his panel if the operation *addToPanel* is called at least once, and the third one allows the member logout. The component *Services* requires the

operation *setDuration* that affects a default rental duration for the chosen resource and validates totally the rental (*rentValid!*). An exception *rentNonValid* is detected if the rental validation cannot be made. In Figure 1, we show the SIAs A_C and A_S of the two components *Client* and *Services*.

The signatures of *Services*'s provided operations are *login*($id, pass$) \rightarrow ($user$), *logout*() \rightarrow (\emptyset), *addToPanel*(res) \rightarrow (\emptyset), *validateRental*(ren) \rightarrow (\emptyset), and *setDuration*(beg, end) \rightarrow (\emptyset). The parameters id , beg , and end are integers. The parameter $pass$ is a string. The parameters $user$, res , and ren , which are references to the persistent components, are records.

According to A_S , *Client* can make at most two connection attempts. Elsewhere, the client connections fail. The set \mathcal{C} of components is defined by $\{Client, Services\}$. The set of shared variables V_C is defined by $\{sess, panel\}$. The variable $sess$ indicates the status of the client session, and the variable $panel$ indicates the panel status. We assume that $V_{A_c} = V_{A_s} = V_C$, $L_{A_c} = \emptyset$, and $L_{A_s} = \{satt\}$. The local variable $satt$ represents the number of connection attempts accorded to clients by the component *Services*. We assume that $D_{satt} = \mathbb{N}$, $D_{sess} = \{active, inactive\}$, and $D_{panel} = \{empty, nonempty\}$.

The semantics $\Psi_{A_C}(login)$ and $\Psi_{A_S}(login)$ of the action *login* are given in Table I as an example of the action semantics. ■

A. Composability

Before defining the composability conditions of two SIAs, we introduce the notion of action effects, which are essential to define the concepts of full and partial control of variables by a component. Mainly, effects are used later in Section III to define the LTS representation of SIAs, but we need to introduce them at this stage to define the criteria by which the composability of two SIAs can be decided.

An effect of an action intervenes its pre and post-conditions and changes the values of shared and local variables because they are needed to check the correctness properties. We define by $Chg_{A_c} : \Sigma_{A_c} \rightarrow 2^{LV_{A_c}}$ the function that associates for each $a \in \Sigma_{A_c}$, the set of variables $Chg_{A_c}(a) \subseteq LV_{A_c}$ modifiable by a .

Definition 2: An effect of an action a is defined by

$$e_{A_c}(a) = \bigvee_{k \geq 1} (grd_k \wedge cmd_k \wedge Unchg_k)$$

Table I
THE SEMANTICS OF THE SHARED ACTION *login*

Client A_C	Services A_S
$Pre_{\Psi_{A_C}(\text{login})} \equiv id > 0 \wedge 8 \leq \text{pass.length}() \leq 10$ $\wedge \text{sess} = \text{inactive}$	$Pre_{\Psi_{A_S}(\text{login})} \equiv id \geq 1 \wedge 6 \leq \text{pass.length}() \leq 10$ $\wedge \text{sess} = \text{inactive}$
$Post_{\Psi_{A_C}(\text{login})} \equiv \text{user.getId}() = id$	$Post_{\Psi_{A_S}(\text{login})} \equiv \text{user.getId}() = id$

Table II
THE EFFECTS OF ACTIONS IN A_C AND A_S

Action	e_{A_C}	e_{A_S}
<i>login</i>	$\text{sess} = \text{inactive} \wedge \text{Unchanged}(LV_{A_C})$	$\text{sess} = \text{inactive} \wedge ((\text{satt} \geq 0 \wedge \text{satt}' = \text{satt} - 1 \wedge \text{sess}' = \text{sess} \wedge \text{panel}' = \text{panel}) \vee \text{Unchanged}(LV_{A_S}))$
<i>notLogged</i>	$\text{sess} = \text{inactive} \wedge \text{Unchanged}(LV_{A_C})$	$\text{sess} = \text{inactive} \wedge \text{Unchanged}(LV_{A_S})$
<i>logged</i>	$\text{sess} = \text{inactive} \wedge \text{sess}' = \text{active}$ $\wedge \text{Unchanged}(\{\text{panel}\})$	$\text{sess} = \text{inactive} \wedge 0 \leq \text{satt} < 2 \wedge \text{sess}' = \text{active}$ $\wedge \text{Unchanged}(\{\text{satt}, \text{panel}\})$
<i>addToPanel</i>	$\text{sess} = \text{active} \wedge ((\text{panel} = \text{vide}$ $\wedge \text{panel}' = \text{nonvide}$ $\wedge \text{sess}' = \text{sess}) \vee \text{Unchanged}(LV_{A_C}))$	$\text{sess} = \text{active} \wedge ((\text{panel} = \text{vide}$ $\wedge \text{panel}' = \text{nonvide} \wedge \text{satt}' = \text{satt}$ $\wedge \text{sess}' = \text{sess}) \vee \text{Unchanged}(LV_{A_S}))$
<i>valdRental</i>	$\text{sess} = \text{active} \wedge \text{Unchanged}(LV_{A_C})$	$\text{sess} = \text{active} \wedge \text{Unchanged}(LV_{A_S})$
<i>setDuration</i>	not defined	$\text{sess} = \text{active} \wedge \text{panel} = \text{nonvide}$ $\wedge \text{panel}' = \text{vide} \wedge \text{satt}' = \text{satt}$ $\wedge \text{sess}' = \text{sess}$
<i>rentNonValid</i>	not defined	$\text{sess} = \text{active} \wedge \text{Unchanged}(LV_{A_S})$
<i>rentValid</i>	$\text{sess} = \text{active} \wedge \text{Unchanged}(LV_{A_C})$	$\text{sess} = \text{active} \wedge \text{Unchanged}(LV_{A_S})$
<i>logout</i>	$\text{sess} = \text{active} \wedge \text{sess}' = \text{inactive}$ $\wedge \text{Unchanged}(LV_{A_C} \setminus \{\text{sess}\})$	$\text{sess} = \text{active} \wedge \text{sess}' = \text{inactive} \wedge \text{satt}' = 2$ $\text{Unchanged}(LV_{A_S} \setminus \{\text{sess}, \text{satt}\})$

such that

- the predicate $\text{grd}_k \in \text{Preds}(LV_{A_c})$, for $k \geq 1$, is a one of the guards of a ;
- $\text{cmd}_k \in \text{Preds}'(V_k)$, where $V_k \subseteq \text{Chg}_{A_c}(a)$, is a command predicate defined in terms of primed variables v' that represents the variables $v \in V_k$ after the execution of a , if grd_k is satisfied;
- $\text{Unchg}_k = \text{Unchanged}(LV_{A_c} \setminus V_k)$ is a predicate defined in terms of variables in $LVA_c \setminus V_k$ that still unchanged after the execution of a . The predicate $\text{Unchanged}(V)$, for a set of variables V , is

$$\bigwedge_{v \in V} v' = v.$$

The pre and post-conditions of an action a are not sufficient to define the full semantics of an action because they ignore local variables. The effect $e_{A_c}(a)$ of a imposes guards grd_k on the local and shared variables and, for each guard, defines a modification cmd_k on variables LVA_c .

Example 2: Consider the previous example, the effects of actions in A_C et A_S are defined in Table II. The reader can easily deduce Chg_{A_C} and Chg_{A_S} . ■

The *fully-controlled* variables by A_c are the shared variables in V_{A_c} whereof A_c is conscious of all the environment actions that can modify them. This requirement states that a specification must include all the actions that can modify its shared fully-controlled variables. This condition is crucial in

component-based assembly, it ensures that the composite of two SIAs is consistent with both of them.

The set of *partially-controlled* variables by A_c are the shared variables that can be modified by the environment actions unknown to A_c . We denote by $\Sigma_A^{\text{ext}} = \Sigma_A^I \cup \Sigma_A^O$ the external actions of A .

Definition 3: The set $V_{A_c}^{\text{fc}}$ of fully-controlled variables by A_c is defined by $\{v \in V_{A_c} \mid (\forall c' \in \mathcal{C} \setminus \{c\}, a \in \Sigma_{A_{c'}} \mid v \in \text{Chg}_{A_{c'}}(a) \Rightarrow a \in \Sigma_{A_c}^{\text{ext}})\}$. The set of partially-controlled variables is $V_{A_c}^{\text{pc}} = V_{A_c} \setminus V_{A_c}^{\text{fc}}$.

Example 3: The variable *sess* is fully-controlled by A_C and A_S . Contrariwise, the variable *panel* is fully-controlled by A_S and partially-controlled by A_C because *setDuration* $\notin \Sigma_{A_C}^{\text{ext}}$ and it changes the variable (*panel* $\in \text{Chg}_{A_S}(\text{setDuration})$). ■

Given two SIAs A_{c_1} and A_{c_2} , $\text{Shared}(A_{c_1}, A_{c_2}) = (\Sigma_{A_{c_1}}^I \cap \Sigma_{A_{c_2}}^O) \cup (\Sigma_{A_{c_2}}^I \cap \Sigma_{A_{c_1}}^O)$ is the set of shared input and output actions of A_{c_1} and A_{c_2} . The external shared actions should have the same signatures in both A_{c_1} and A_{c_2} .

Definition 4: Two semantical interface automata A_{c_1} and A_{c_2} of two components c_1 and c_2 in \mathcal{C} are composable iff

- $\Sigma_{A_{c_1}}^I \cap \Sigma_{A_{c_2}}^I = \Sigma_{A_{c_1}}^O \cap \Sigma_{A_{c_2}}^O = \Sigma_{A_{c_1}}^H \cap \Sigma_{A_{c_2}}^H = \Sigma_{A_{c_1}}^O \cap \Sigma_{A_{c_2}}^H = \emptyset$;
- $L_{A_{c_1}} \cap L_{A_{c_2}} = \emptyset$;
- $\forall a \in \text{Shared}(A_{c_1}, A_{c_2}), \phi \in \mathcal{I}_{V_{A_{c_1}} \cap V_{A_{c_2}}} \mid e_{A_{c_1}}(a) \wedge e_{A_{c_2}}(a)$ is satisfiable;

- for all $a \in \text{Shared}(A_{c_1}, A_{c_2})$ whereof the signature is given by $a(i_1, \dots, i_n) \rightarrow (o)$ in A_{c_1} and by $a(i'_1, \dots, i'_n) \rightarrow (o')$ in A_{c_2} for all $n \in \mathbb{N}$
 - if $a \in \Sigma_{A_{c_1}}^O$, then $D_{i_k} \subseteq D_{i'_k}$ for $1 \leq k \leq n$ and $D_o \subseteq D_{o'}$;
 - if $a \in \Sigma_{A_{c_1}}^I$, then $D_{i_k} \supseteq D_{i'_k}$ for $1 \leq k \leq n$ and $D_o \supseteq D_{o'}$.

The composition of two semantical interface automata A_{c_1} and A_{c_2} may take effect if (i) their actions are disjoint except shared input, output, hidden ones, (ii) their shared input and output actions have the same effect on variables in $V_{A_{c_1}} \cap V_{A_{c_2}}$ ($e_{A_{c_1}}(a) \wedge e_{A_{c_2}}(a)$ is satisfiable), (iii) the parameter sub-typing [7] property of actions in $\text{Shared}(A_{c_1}, A_{c_2})$ is satisfied, and (iv) their local variables are disjoint.

Example 4: We have $\text{Shared}(A_C, A_S) = \Sigma_{A_C}$. According to Definition 4 and the indications given in the previous examples, A_C and A_S are composable. ■

B. Compatibility

The semantical compatibility [8] of external shared actions and the synchronized product of two composable SIAs is defined as follows.

Definition 5: Given an action $a \in \text{Shared}(A_{c_1}, A_{c_2})$, if one of the following conditions is satisfied then the action a in A_{c_1} is semantically compatible with the action a in A_{c_2} , denoted by $SComp_a(A_{c_1}, A_{c_2}) \equiv \text{true}$ (false otherwise):

- if $a \in \Sigma_{A_{c_1}}^O$, then (1) $Pre_{\Psi_{A_{c_1}}}(a) \Rightarrow Pre_{\Psi_{A_{c_2}}}(a)$, and (2) $Post_{\Psi_{A_{c_1}}}(a) \Leftarrow Post_{\Psi_{A_{c_2}}}(a)$,
- if $a \in \Sigma_{A_{c_1}}^I$, then (1) $Pre_{\Psi_{A_{c_1}}}(a) \Leftarrow Pre_{\Psi_{A_{c_2}}}(a)$, and (2) $Post_{\Psi_{A_{c_1}}}(a) \Rightarrow Post_{\Psi_{A_{c_2}}}(a)$.

We assume that the name of parameters are the same in the semantics of actions in A_{c_1} and A_{c_2} .

The definition of the synchronized product is defined as follows.

Definition 6: Given two components $c_1, c_2 \in \mathcal{C}$ whose the semantical interface automata A_{c_1} and A_{c_2} are composable, their product $A_{c_1} \otimes A_{c_2}$ is defined by

- $S_{A_{c_1} \otimes A_{c_2}} = S_{A_{c_1}} \times S_{A_{c_2}}$; $i_{A_{c_1} \otimes A_{c_2}} = (i_{A_{c_1}}, i_{A_{c_2}})$;
- $\Sigma_{A_{c_1} \otimes A_{c_2}}^I = (\Sigma_{A_{c_1}}^I \cup \Sigma_{A_{c_2}}^I) \setminus \text{Shared}(A_{c_1}, A_{c_2})$;
- $\Sigma_{A_{c_1} \otimes A_{c_2}}^O = (\Sigma_{A_{c_1}}^O \cup \Sigma_{A_{c_2}}^O) \setminus \text{Shared}(A_{c_1}, A_{c_2})$;
- $\Sigma_{A_{c_1} \otimes A_{c_2}}^H = \Sigma_{A_{c_1}}^H \cup \Sigma_{A_{c_2}}^H \cup \{a \in \text{Shared}(A_{c_1}, A_{c_2}) \mid SComp_a(A_{c_1}, A_{c_2}) \equiv \text{true}\}$;
- $L_{A_{c_1} \otimes A_{c_2}} = L_{A_{c_1}} \cup L_{A_{c_2}}$; $V_{A_{c_1} \otimes A_{c_2}} = V_{A_{c_1}} \cup V_{A_{c_2}}$;
- $((s_1, s_2), a, (s'_1, s'_2)) \in \delta_{A_{c_1} \otimes A_{c_2}}$ iff
 - $a \notin \text{Shared}(A_{c_1}, A_{c_2}) \wedge (s_1, a, s'_1) \in \delta_{A_{c_1}} \wedge s_2 = s'_2$,
 - $a \notin \text{Shared}(A_{c_1}, A_{c_2}) \wedge (s_2, a, s'_2) \in \delta_{A_{c_2}} \wedge s_1 = s'_1$,
 - $a \in \text{Shared}(A_{c_1}, A_{c_2}) \wedge (s_1, a, s'_1) \in \delta_{A_{c_1}} \wedge (s_2, a, s'_2) \in \delta_{A_{c_2}} \wedge SComp_a(A_{c_1}, A_{c_2}) \equiv \text{true}$;

- $\Psi_{A_{c_1} \otimes A_{c_2}}$ is defined by:
 - $\Psi_{A_{c_i}}$ for $a \in \Sigma_{A_{c_i}} \setminus \text{Shared}(A_{c_1}, A_{c_2})$ for $i \in \{1, 2\}$;
 - $\langle Pre_{\Psi_{A_{c_1}}}(a), Post_{\Psi_{A_{c_2}}}(a) \rangle$ for $a \in \text{Shared}(A_{c_1}, A_{c_2}) \cap \Sigma_{A_{c_1}}^O$ such that $SComp_a(A_{c_1}, A_{c_2}) \equiv \text{true}$;
 - $\langle Pre_{\Psi_{A_{c_2}}}(a), Post_{\Psi_{A_{c_1}}}(a) \rangle$ for $a \in \text{Shared}(A_{c_1}, A_{c_2}) \cap \Sigma_{A_{c_1}}^I$ such that $SComp_a(A_{c_1}, A_{c_2}) \equiv \text{true}$.

We assume that $e_{A_{c_1} \otimes A_{c_2}}$ is defined by $e_{A_{c_1}}(a) \wedge e_{A_{c_2}}(a)$ for all $a \in \text{Shared}(A_{c_1}, A_{c_2})$, by $e_{A_{c_1}}(a)$ for all $a \in \Sigma_{A_{c_1}} \setminus \text{Shared}(A_{c_1}, A_{c_2})$, and by $e_{A_{c_2}}(a)$ for all $a \in \Sigma_{A_{c_2}} \setminus \text{Shared}(A_{c_1}, A_{c_2})$.

The incompatibility between A_{c_1} and A_{c_2} is due to (i) the existence of some *illegal states* (s_1, s_2) in the set of transitions $\delta_{A_{c_1} \otimes A_{c_2}}$ where one of the two SIAs A_{c_1} and A_{c_2} outputs a shared action a from s_1 , which is not accepted as input from s_2 or vice versa, or (ii) from that states they synchronize on the action a but $SComp_a(A_{c_1}, A_{c_2}) \equiv \text{false}$.

Definition 7: The set of illegal states $Illegal(A_{c_1}, A_{c_2}) \subseteq S_{A_{c_1}} \times S_{A_{c_2}}$ is defined by $\{(s_1, s_2) \in S_{A_{c_1} \otimes A_{c_2}} \mid (\exists a \in \text{Shared}(A_{c_1}, A_{c_2}) \mid (C_1 \vee C_2 \text{ holds}))\}$

$$C_1 = \left(\begin{array}{l} (a \in \Sigma_{A_{c_1}}^O(s_1) \wedge a \notin \Sigma_{A_{c_2}}^I(s_2)) \vee (a \in \Sigma_{A_{c_1}}^O(s_1) \wedge \\ a \in \Sigma_{A_{c_2}}^I(s_2) \wedge SComp_a(A_{c_1}, A_{c_2}) \equiv \text{false}) \end{array} \right)$$

$$C_2 = \left(\begin{array}{l} (a \in \Sigma_{A_{c_2}}^O(s_2) \wedge a \notin \Sigma_{A_{c_1}}^I(s_1)) \vee (a \in \Sigma_{A_{c_2}}^O(s_2) \wedge \\ a \in \Sigma_{A_{c_1}}^I(s_1) \wedge SComp_a(A_{c_1}, A_{c_2}) \equiv \text{false}) \end{array} \right)$$

The reachability of states in $Illegal(A_{c_1}, A_{c_2})$ do not implies that A_{c_1} and A_{c_2} are not compatible. The existence of an environment E (a semantical interface automaton) that produces appropriate inputs for $A_{c_1} \otimes A_{c_2}$ ensures that illegal states are not reached. The compatible states, denoted by $Comp(A_{c_1}, A_{c_2})$, are states in which $A_{c_1} \otimes A_{c_2}$ avoids reaching illegal states by enabling output or internal actions (optimistic approach) [4].

Definition 8: A_{c_1} and A_{c_2} are compatible iff the initial state of $A_{c_1} \otimes A_{c_2}$ is compatible.

The verification steps [4] of the compatibility between A_{c_1} and A_{c_2} without considering the semantics of actions are listed below.

Algorithm

Input : Two SIAs A_{c_1} and A_{c_2} .

Output : $A_{c_1} \parallel A_{c_2}$.

Algorithm steps :

- 1) compute the product $A_{c_1} \otimes A_{c_2}$,
- 2) compute $Illegal(A_{c_1}, A_{c_2})$,
- 3) compute the set of incompatible states in $A_{c_1} \otimes A_{c_2}$: the states from which the illegal states are reachable by enabling only internal and output actions,
- 4) compute the composition $A_{c_1} \parallel A_{c_2}$ by eliminating from the automaton $A_{c_1} \otimes A_{c_2}$, the illegal state, the

incompatible states, and the unreachable states from the initial state,

- 5) if $A_{c_1} \parallel A_{c_2}$ is empty then A_{c_1} and A_{c_2} are not compatible, therefore c_1 and c_2 can not be assembled correctly in any environment. Otherwise, A_{c_1} and A_{c_2} are compatible.

Our approach increases the complexity¹ of the previous proposed one by taking into account the semantic compatibility check of actions in $Shared(A_1, A_2)$, whereof the complexity is determined by the logic and the context theories within the formulas are defined.

Example 5: According to Table I and II and the previous definitions and examples, A_C and A_S are compatible if $SComp_a(A_C, A_S) \equiv true$ for all $a \in Shared(A_C, A_S)$. ■

Theorem 1: The composition \parallel between SIAs is a commutative and associative operation.

Proof: The proof is based on that presented in [5] by considering the action semantics. ■

III. CHECKING INVARIANCE PROPERTIES

In this section, we found formal methodology to design and check the correctness properties of semantical interface automata thanks to their rich semantics based on the use of variables. The correctness properties, as said in the beginning of the paper, are invariants. Commonly, specifiers have to model a system by a transition system to check invariants and other types of system temporal properties. The states of a such transition system are associated to a set of atomic propositions stated on a set of modifiable variables. Our contribution follows a similar procedure. A SIA A is translated to a labeled transition system $LTS(A)$ whose states are the variable valuations. Starting from an initial valuation, the effects of actions update the variables and $LTS(A)$ is generated.

A. LTS representation

Before defining the LTS representations of a SIA, we start by defining some preliminaries. A *valuation* of a set of variables V is defined by

$$\phi : V \rightarrow \bigcup_{v_i \in V} D_{v_i}$$

that associates to each $v_i \in V$ a value in D_{v_i} . We denote by $\phi \langle V' \rangle$ the restriction of ϕ to the set $V' \subseteq V$. The set \mathcal{I}_V is the set of all possible valuations ϕ in V . Given an action $a \in \Sigma_A$, we denote by $\mathcal{E}(\phi, e_A(a)) \in \mathcal{I}_{LV_A}$ the valuation of variables LV_A after the execution of a for a valuation ϕ of LV_A .

The LTS representation of a SIA A transforms its set of transitions to a labeled transition system whereof the states

¹The complexity of checking the compatibility between two interface automata A_1 and A_2 is in time linear on $|A_1|$ and $|A_2|$ [4].

belong to \mathcal{I}_{LV_A} and the labels are the actions in Σ_A . The LTS representations of SIAs allows the separation between the task of checking interoperability and that of the correctness properties check. It's clear that a SIA has fewer states than its LTS representation, which allow to reduce the complexity of the interoperability checking. The LTS representations are devoted to check correctness properties.

Definition 9: The LTS representation $LTS(A) = \langle S_{LTS(A)}, I_{LTS(A)}, \Sigma_{LTS(A)}, \delta_{LTS(A)} \rangle$ of a semantical interface automata A is a labeled transition system defined by

- $S_{LTS(A)} \subseteq \mathcal{I}_{LV_A}$;
- $I_{LTS(A)} = Init_A$ where $Init_A$ is the initial valuation of LV_A ;
- $\Sigma_{LTS(A)} = \Sigma_A$;
- $(\phi_1, a, \phi_2) \in \delta_{LTS(A)}$ iff $\phi_1 \in \mathcal{I}_{LV_A}$, for $s \in S_A$, $a \in \Sigma_A(s)$, and $\phi_2 = \mathcal{E}(\phi_1, e_A(a))$.

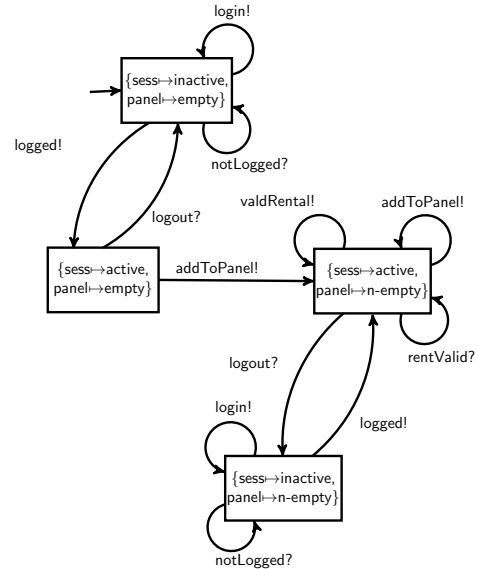


Figure 2. The LTS representation $LTS(A_C)$ of A_C

Example 6: The LTS representation $LTS(A_C)$ of the semantical interface automaton A_C of the component *Client* is shown in Figure 2 where $Init_{A_C} = \{sess \mapsto inactive, panel \mapsto empty\}$. ■

Given two SIAs A_1 and A_2 , the following property establishes that for each state $\phi \in S_{LTS(A_1 \parallel A_2)}$, only the valuations of fully-controlled variables of both A_1 and A_2 are the same in A_1 , A_2 et $A_1 \parallel A_2$. These variables are modifiable exclusively by actions in $Shared(A_1, A_2)$. We denote by LV_A^{fc} , the set $V_A^{fc} \cup L_A$ of a SIA A .

Property 1: Given $W = LV_{A_1 \parallel A_2}$, $V = LV_{A_1}^{fc} \cap LV_{A_2}^{fc}$, $V' = LV_{A_1}^{fc} \cup LV_{A_2}^{fc}$, $V_1 = LV_{A_1}^{fc} \setminus LV_{A_2}^{fc}$, and $V_2 = LV_{A_2}^{fc} \setminus LV_{A_1}^{fc}$ where A_1 and A_2 are two compatible SIAs

and $Init_{A_1}\langle V_{A_1} \cap V_{A_2} \rangle = Init_{A_2}\langle V_{A_1} \cap V_{A_2} \rangle$, for all $\phi \in S_{LTS(A_1 \parallel A_2)}$, there exists $\phi_1 \in S_{LTS(A_1)}$ and $\phi_2 \in S_{LTS(A_2)}$ such that

1) $\phi\langle V \rangle = \phi_1\langle V \rangle = \phi_2\langle V \rangle$ knowing that $(V = LV_{A_1}^{fc} \cap LV_{A_2}^{fc} = V_{A_1}^{fc} \cap V_{A_2}^{fc})$;

2) $\phi\langle V' \rangle$ is defined as follows :

$$\begin{cases} \phi\langle V \rangle & \text{for all } v \in V; \\ \phi_1\langle V_1 \rangle & \text{for all } v \in V_1; \\ \phi_2\langle V_2 \rangle & \text{for all } v \in V_2. \end{cases}$$

B. Invariant specification

Given an LTS representation $LTS(A)$ of a SIA A , we denote by $\varphi[V] \in Preds(V)$, a first order formula whose free variables belong to $V \subseteq LV_A$. A formula $\varphi[V]$ is satisfied at the state $\phi \in S_{LTS(A)}$ iff ϕ satisfies $\varphi[V]$, i.e., the following condition is satisfied:

$$\left(\bigwedge_{v \in LV_A} v = \phi(v) \right) \Rightarrow \varphi[V].$$

Definition 10: Given an LTS representation $LTS(A)$ of a SIA A and a set $V \subseteq LV_A$, an invariant $\varphi[V]$ of $LTS(A)$ (written $LTS(A) \models \varphi[V]$) is a first order formula such that for all $\phi \in S_{LTS(A)}$, $\phi\langle V \rangle$ satisfies $\varphi[V]$ ($\phi\langle V \rangle \models \varphi[V]$) and ϕ satisfies $\varphi[V]$ ($\phi \models \varphi[V]$).

Example 7: The predicate $\varphi[LV_{A_S}] = \neg(satt = 2 \wedge sess = active)$ is an invariant of $LTS(A_S)$ shown in Figure 2. ■

C. Invariant preservation by composition

The following theorem establishes that only invariants stated on local and fully-controlled variables can be preserved by the LTS representations of the composition $A_1 \parallel A_2$ of two compatible SIAs A_1 and A_2 because they are aware of all the environment actions that can modify these variables. An invariant stated on partially-controlled variables of A_1 (resp. A_2) cannot be preserved by composition because it is possible that A_2 (resp. A_1) modifies the values by actions unknown to A_1 (resp. A_2) in such way the invariant is violated. Corollary 2 can easily be deduced from that theorem.

Theorem 2 (Invariant Preservation by SIA Composition): Given two $LTS(A_1)$ and $LTS(A_2)$ respectively of two compatibles SIAs A_1 and A_2 and $Init_{A_1}\langle V_{A_1} \cap V_{A_2} \rangle = Init_{A_2}\langle V_{A_1} \cap V_{A_2} \rangle$, for all $\varphi_1[LV_{A_1}^{fc}]$ and $\varphi_2[LV_{A_2}^{fc}]$, if $LTS(A_1) \models \varphi_1[LV_{A_1}^{fc}]$ and $LTS(A_2) \models \varphi_2[LV_{A_2}^{fc}]$, then $LTS(A_1 \parallel A_2) \models (\varphi_1[LV_{A_1}^{fc}] \wedge \varphi_2[LV_{A_2}^{fc}])$.

Proof: We have the following assumptions:

- 1) $\forall \phi_1 \in S_{LTS(A_1)} \mid \phi_1\langle LV_{A_1}^{fc} \rangle \models \varphi_1[LV_{A_1}^{fc}]$ and $\phi_1 \models \varphi_1[LV_{A_1}^{fc}]$;
- 2) $\forall \phi_2 \in S_{LTS(A_2)} \mid \phi_2\langle LV_{A_2}^{fc} \rangle \models \varphi_2[LV_{A_2}^{fc}]$ and $\phi_2 \models \varphi_2[LV_{A_2}^{fc}]$;

We have to prove that, for all $\phi \in S_{LTS(A_1 \parallel A_2)}$, $\phi\langle LV_{A_1}^{fc} \cup LV_{A_2}^{fc} \rangle \models (\varphi_1[LV_{A_1}^{fc}] \wedge \varphi_2[LV_{A_2}^{fc}])$ and $\phi \models (\varphi_1[LV_{A_1}^{fc}] \wedge$

$\varphi_2[LV_{A_2}^{fc}])$? According to the property 1(2), we have for all $\phi \in S_{LTS(A_1 \parallel A_2)}$, there exists $\phi_1 \in S_{LTS(A_1)}$ and $\phi_2 \in S_{LTS(A_2)}$ such that $\phi\langle LV_{A_1}^{fc} \cup LV_{A_2}^{fc} \rangle$ is equal to

$$\begin{cases} \phi\langle LV_{A_1}^{fc} \cap LV_{A_2}^{fc} \rangle & \forall v \in LV_{A_1}^{fc} \cap LV_{A_2}^{fc}; \\ \phi_1\langle LV_{A_1}^{fc} \setminus LV_{A_2}^{fc} \rangle & \forall v \in LV_{A_1}^{fc} \setminus LV_{A_2}^{fc}; \\ \phi_2\langle LV_{A_2}^{fc} \setminus LV_{A_1}^{fc} \rangle & \forall v \in LV_{A_2}^{fc} \setminus LV_{A_1}^{fc}. \end{cases}$$

We can deduce, according to the property 1(1), that

$$\phi\langle LV_{A_1}^{fc} \cup LV_{A_2}^{fc} \rangle = \begin{cases} \phi_1\langle LV_{A_1}^{fc} \rangle & \text{for all } v \in LV_{A_1}^{fc}; \\ \phi_2\langle LV_{A_2}^{fc} \rangle & \text{for all } v \in LV_{A_2}^{fc}. \end{cases}$$

We have $\phi_1\langle LV_{A_1}^{fc} \rangle \models \varphi_1[LV_{A_1}^{fc}]$ (assumption 1) and $\phi_2\langle LV_{A_2}^{fc} \rangle \models \varphi_2[LV_{A_2}^{fc}]$ (assumption 2), then we can deduce that $\phi\langle LV_{A_1}^{fc} \cup LV_{A_2}^{fc} \rangle \models \varphi_1[LV_{A_1}^{fc}]$ and $\phi\langle LV_{A_1}^{fc} \cup LV_{A_2}^{fc} \rangle \models \varphi_2[LV_{A_2}^{fc}]$. Consequently, $\phi\langle LV_{A_1}^{fc} \cup LV_{A_2}^{fc} \rangle \models (\varphi_1[LV_{A_1}^{fc}] \wedge \varphi_2[LV_{A_2}^{fc}])$ and $\phi \models (\varphi_1[LV_{A_1}^{fc}] \wedge \varphi_2[LV_{A_2}^{fc}])$. ■

Corollary 1: Given two $LTS(A_1)$ and $LTS(A_2)$ of two compatibles SIAs A_1 and A_2 such that the assumptions of Theorem 2 are satisfied, for all $\varphi[V_{A_1}^{fc} \cap V_{A_2}^{fc}]$, if $LTS(A_1) \models \varphi[V_{A_1}^{fc} \cap V_{A_2}^{fc}]$ and $LTS(A_2) \models \varphi[V_{A_1}^{fc} \cap V_{A_2}^{fc}]$, then $LTS(A_1 \parallel A_2) \models \varphi[V_{A_1}^{fc} \cap V_{A_2}^{fc}]$.

Example 8: The predicate $\varphi[LV_{A_S}^{fc}] = \neg(satt = 2 \wedge sess = active)$ is an invariant of $LTS(A_S)$. We can deduce that $LTS(A_C \parallel A_S) \models \neg(satt = 2 \wedge sess = active)$. ■

IV. RELATED WORKS

Luca de Alfaro and al. [9] has proposed ‘‘sociable’’ interface modules SIMs to specify component interfaces. The formalism communicates via both actions and shared variables and the synchronization between actions is based on two main principles: (i) the first principle is that the same actions can label both input and output transitions, and (ii) the second is that global variables can be updated by multiple interfaces. The authors show that the compatibility and the refinement check of SIMs can be made thanks to efficient symbolic algorithms implemented in the tool TICC [10] (Tool for Interface Compatibility).

The main differences between SIMs and our proposed approach can be identified in the following points. First, in SIMs, internal actions are not considered. They concertize the local behaviors of components and the synchronization of shared input and output actions. Internal actions are necessary to develop closed systems composed of a prefixed set of components. They can be also useful when a component instance is associated with a specific client. For example, the EJB *stateful* session beans cannot be composed with many clients. Shared input and output actions disappear (become internal) in the composition between a stateful the session bean instance and its client in such way other clients cannot be connected to the bean using the same shared actions. Thus, SIAs can be applied to specify both closed and open

component-based systems by cons, SIMs are rather relevant to specify only open systems.

Second, in SIMs, a component could require and offer the same service (an action can label both input and output transitions). In our approach, this type of components is not considered. Furthermore, in our approach, we explicitly define to what input and output actions correspond (operation calls, receiving return values, exceptions, etc) which is not the case in SIMs.

Third, we demonstrate that only invariants stated on fully-controlled variables (history variables in SIMs) can be preserved by composition. The authors in [9] do not take into account this issue. Finally, SIAs unifies the use of operation parameters and variables to describe the compositional semantics of components. In addition, we show that we can separate protocols, used commonly to verify component compatibility and composition from models used to check correctness. SIAs (simple states, actions, and semantics) are used to check the component compatibility and their LTS representations are used to check correctness properties preservation by composition.

Ivana Černá and al. [11] have founded “Component-interaction automata” (CoIN) to reason about the behavioral aspects of component-based systems by respecting a given architecture. They also proposed methods to check component assembly correctness by verifying properties, like consequences of operation calls and fairness without using variables. These properties are expressed in an extended version of the linear temporal logic called CI-LTL and verified using model-checking techniques [12].

The approach proposed in [13] is a formal methodology for describing behavioral protocols of interacting, concurrent components with data states. The authors describes component protocols by means of labeled transition systems, which specify the scheduling of operation calls and the data states updates by using of pre and post-conditions. Furthermore, they endow protocols with a model-theoretic semantics describing the class of all correct implementations (refinement) of an abstract protocol.

In [14], the authors have proposed an approach endowing Sun’s Enterprise JavaBeans (EJB) component by behavioral protocols. The proposed framework provides a set of mechanisms allowing the automated extraction of protocols from EJB components and the verification of coherence between these protocols. Protocols are represented by particular labeled transition systems.

V. CONCLUSION AND FUTURE WORKS

In this paper, we propose a formalism based on interface automata enriched by the use of the action semantics to describe behavioral protocols of components and to check their compatibility and safety. In particular, We study the problem of invariant preservation by composition. In the future, we intent to adapt the alternating simulation approach

used to refine interface automata to support the treatment of the action semantics and to ensure the requirement of invariant preservation also by refinement.

REFERENCES

- [1] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [2] D. Konstantas, *Interoperation of object-oriented applications*. Hertfordshire, UK: Prentice Hall International Ltd., 1995, pp. 69–95.
- [3] P. Wegner, “Interoperability,” *ACM Comput. Surv.*, vol. 28, pp. 285–287, March 1996.
- [4] L. de Alfaro and T. A. Henzinger, “Interface automata,” *SIGSOFT Softw. Eng. Notes*, vol. 26, no. 5, pp. 109–120, 2001.
- [5] L. d. Alfaro and T. A. Henzinger, “Interface-based design,” in *Engineering Theories of Software-intensive Systems*. Springer, 2005, pp. 83–104.
- [6] N. A. Lynch and M. R. Tuttle, “Hierarchical correctness proofs for distributed algorithms,” in *PODC ’87: Proc. of the 6th ACM Symp. on principles of distributed computing*. New York, USA: ACM, 1987, pp. 137–151.
- [7] B. H. Liskov and J. M. Wing, “A behavioral notion of subtyping,” *ACM Trans. Program. Lang. Syst.*, vol. 16, pp. 1811–1841, November 1994.
- [8] S. Heiler, “Semantic interoperability,” *ACM Comput. Surv.*, vol. 27, pp. 271–273, June 1995.
- [9] L. d. Alfaro, L. D. d. Silva, M. Faella, A. Legay, P. Roy, and M. Sorea, “Sociable interfaces,” in *The Proc. 5th Int. WS. on Frontiers of Combining Systems, LNAI 3717*. Springer-Verlag, 2005, pp. 81–105.
- [10] L. d. Alfaro, L. D. d. Silva, M. Faella, A. Legay, V. Raman, and P. Roy, “Ticc: A tool for interface compatibility and composition,” in *The Proc. 18th Int. Conf. on Computer Aided Verification (CAV), volume 4144 of LNCS*. Springer, 2006, pp. 59–62”.
- [11] B. Zimmerova, P. Vařeková, N. Beneš, I. Černá, L. Brim, and J. Sochor, “The common component modeling example.” Berlin, Heidelberg: Springer-Verlag, 2008, ch. Component-Interaction Automata Approach (CoIn), pp. 146–176.
- [12] J. Barnat, L. Brim, I. Černá, P. Moravec, P. Ročkal, and P. Šimeček, “Divine - a tool for distributed verification,” in *Computer Aided Verification*, ser. LNCS. Springer Berlin / Heidelberg, 2006, vol. 4144, pp. 278–281.
- [13] S. S. Bauer, R. Hennicker, and S. Janisch, “Behaviour protocols for interacting stateful components,” *Electron. Notes Theor. Comput. Sci.*, vol. 263, pp. 47–66, June 2010.
- [14] A. Farías and M. Südholt, “On components with explicit protocols satisfying a notion of correctness by construction,” in *Confederated Int. Conf. DOA, CoopIS and ODBASE 2002*. London, UK: Springer-Verlag, 2002, pp. 995–1012.

Method for CMMI-DEV Implementation in Distributed Teams

Tiago da Cunha Oliveira

Departamento de Engenharia Informática
 Instituto Superior Técnico, Universidade Técnica de Lisboa
 Lisboa, Portugal
 tiagocoliveira@ist.utl.pt

Miguel Mira da Silva

Departamento de Engenharia Informática
 Instituto Superior Técnico, Universidade Técnica de Lisboa
 Lisboa, Portugal
 mms@ist.utl.pt

Abstract—Organizations tend to perform their work in off-shore sites to become more competitive. But managing these teams is not an easy task because it is needed a great level of coordination. So, some organizations adopt maturity models as CMMI-DEV to normalize and coordinate the tasks across the different sites. But it faces difficulties due to the different work practices and cultures in the distributed teams, which can imply a great resistance to change. Thus, when an organization wants to put their development process in compliance with CMMI-DEV, we propose that a first assessment should be done by an understanding of the development processes in each location, making it possible to normalize/standardize the work processes with smaller changes, reducing the cost and resistance to change. This proposal was evaluated by applying these methods in a distributed organization with two development branches. One branch has ISO 9001:2008 certification and works in two countries, and the other branch in three countries. The data below supports the objectives of our proposal, pointing to a careful analysis of the different teams, and therefore easier to adopt models such as CMMI-DEV.

Keywords- *Development Process, CMMI-DEV, Geographically Distributed Teams, Organizational Change.*

I. INTRODUCTION

Software projects management has been and remains as one of the crucial problems of computing. Although there have been several efforts to make project management more effective and efficient, it still has several problems such as [1] [2]: objectives of the project unrealistic or disorganized; inaccurate estimations of the resources necessary to implement the project; requirements of the system ill-defined; weak monitoring of the status and progress of the project; risks poorly or inadequately managed; lack of communication between stakeholders (customers, users and developers); immature use of technology; lack of capacity to deal with the complexity of the project; careless and poorly formalized development practices; poor project management; politics of stakeholders; commercial pressures; inadequate quality control; ineffective control of change.

In addition to these problems outlined above, organizations must be much more effective and efficient due to the high competitive environment in the market where they perform their work [16].

So, resulting from that fact, a solution that has grown and tended to become more popular, is the outsourcing of information technology services in offshore sites [16]. (According to an IDC market research report [23], the estimated market size of IT offshoring reached US\$29.4 billion by 2010). It holds, as the key benefits, the product launch to market sooner, with lower development costs through access to skilled manpower and specialized resources [3] [4].

Over the past ten years, emerged a series of facts that must be taken into consideration when selecting suppliers of computer services such as [4]:

- Globalization - opening the borders to the international market;
- Business environment - growing interest of countries in developing the economy, creating mechanisms for attracting foreign investment through tax incentives, reduction of bureaucracy and building technology parks;
- Decrease the cost of telecommunications;
- Standardization of methodologies and tools in software development.

To these facts, join two more important ones as the large difference in wages in different locations around the globe to perform the same function and the increasing standardization of the culture of companies that are increasingly multinational.

As managers have to make commitments, most often based on price/quality of service rather than patriotic or emotional factors, the choice of suppliers began to be increasingly made outside the country of origin of the company - offshoring [4].

When the managers of organizations opt for the offshoring choice, they rely on two key factors [5]:

1. Reduction of development costs
2. Rapid increase of skilled labor

Despite all the benefits resulting from offshoring, managing distributed teams is not an easy task, because these teams frequently suffer crises of trust and coordination problems [12].

So, many organizations choose to do the alignment between operations and processes based on maturity models, which suggest the best practices in the industry, giving the organization a competitive advantage [6] [7].

Traditional maturity models as Carnegie Mellon's Capability Maturity Model Integration (CMMI) [21] help organizations find their state of optimization, providing a structure that allows alignment between process areas suggested by the model and process management. CMMI-DEV enables organizations to achieve a high level of process optimization, following the goals suggested by the model, considered as best practices within the development branch. With the adoption of the model, the organization takes deep knowledge of their processes and patterns of behavior that should be established [8].

In this paper, we present geographically distributed teams and organizational change in distributed team's related work. Thereon, we approach the problem of CMMI-DEV implementation in distributed teams, and its resolution proposal. Additionally, we will be present and evaluate the preliminary results in an organization that has its workforce distributed, by the application of the proposal.

II. RELATED WORK

Aiming to achieve a better control, coordination and monitoring, a first analysis of the distributed teams and the organizational changes is in order. To point out the main details, issues and implications became our main goal in Section II.

A. Geographically Distributed Teams

These teams are located in countries where manpower is cheaper, usually with additional time zone with the country of origin of the organizations in order to take advantage of 24 hours of daily work [9].

The distributed teams were the result of globalization, but this has no implication as the standardization of cultures, as they continue to be diversified with different values and beliefs that result in different behaviors [11]. Since organizations are dependent on people, all these differences in the context of distributed teams become of utmost importance when one wants to maximize and make use of distributed teams to a competitive advantage [11].

Organizations must cope with challenges such as conflicts arising from their employment relationship between their teams. Conflicts at the completion of tasks [13] are due to differing views and opinions regarding the tasks of the team. These resolutions are more complicated due to lack of physical meetings, which means that the agreement between different views is complicated due to limitations in terms of trust arising from the singularities of distributed teams [10] Conflicts also arise in terms of processes [13], i.e., teams use different ways of working to accomplish the same result.

Hence, understanding the differences urges as an extreme need in the help in improving relations of trust between teams [11] this understanding should be done by [12]:

- Sharing identity, so that the effects of geographical dispersion are reduced
- Sharing context, i.e., the team members can access the same information, using the same tools
- Possibility of spontaneous communication through access to tools [10] that allow informal, unplanned interactions between members, thereby strengthening the relations of trust [14].

Understand and respect the particularities of the distributed teams is essential to get the maximum benefits and reduce its complications and shortcomings. The problems that most commonly affect these teams are not technical problems as they become salient faster than the non-technical [15].

B. Organizational Change in Distributed Teams

The CMMI-DEV can have implications as a process of change, since there may have to be redefinitions of procedures performed, in order to conform to the model.

This process of change reaches further complexity in the context of offshoring, i.e., put all organizational units under the objectives suggested by the model. Once alone, the offshoring originates process of change [17].

The framework of change processes must be studied very carefully as they can be influenced by three contexts that influence change in organizations [17]:

- External Context: Factors of legislation, commercial and social
- Internal Context: social aspects of the organization, technical infrastructure, management style
- Individual Background: actors who develop their roles in the organization with their views of working methods.

So it has to be taken into account in the phenomena of change, the surrounding environment, processes and people in order to avoid negative aspects for the organization and for business. Understanding this framework of change processes is important for this to an end, thus reducing the resistance to change that tends to increase with the number of different organizational units. (Figure 1)

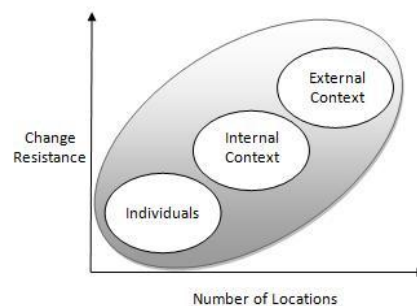


Figure 1 - Relation between the change resistance impact with the number of different locations

During the process of change there are other factors (hard factors) also important to be taken into consideration [18]:

- Duration: The time it takes for the changing process to be implemented
- Integrity of the performance of the development team to complete a task on time. This factor depends on the skills of employees in respect of a project component
- Commitment among top managers and employees affected by the change
- Efforts caused by the change process

All these factors must be taken into account, since the intention is that the change generated by the adoption of CMMI-DEV is less abrupt as possible so that implementation is done with greater adherence by all participants in the process, in order to reduce resistance to change.

III. PROBLEM

To take advantage from offshoring, organizations must have an effective coordination in the different locations, which have great influence on productivity and performance [24].

There are different opinions related with the coordination between these teams. One relates with standardization of processes to reduce the conflict and differences between the sites, and another, which refers that its normalization can generate suspicion and resentment at offshore sites embedded in different cultures and ways to execute their work.

So, there are organizations, which adopt CMMI-DEV to be enabled to benefit from greater control, coordination and monitoring, resulting in improvements in the development process [5]. However, this model still suffers from significant shortcomings regarding the best practices that should be followed to organizations whose work focus is based on distributed teams [6].

When an organization decides to adopt CMMI-DEV has to be very careful, as its adoption usually involves a standardization/normalization of processes. However, the maturity models are poorly adapted to the reality of offshoring [6], and it might not make sense that distributed teams run processes in the same way as they have different frameworks. From these frameworks emerge some of the limitations of offshoring as [6] [16] [22]:

- Difficulty for clarification of requirements for lack of physical meetings;
- Failure of coordination due often to failures of communication;
- Large differences in experience among staff, with implications on the performance of the project;
- Time zone Difference, which can influence the time to solve problems;
- Different infrastructures, such as unstable Internet connections or electricity;
- Cultural differences.

Some of the problems posed by distributed teams still do not have the best response from the CMMI-DEV, such as the lack of communications in person, redundant information, lack of motivation, conflict resolution. This model is still based on traditional working practices and does not take into account the growing trends of global organizations and distributed teams [6]. The adoption of CMMI-DEV can imply the existence of organizational change processes, difficult to manage.

In the different locations, there are various formal and informal rules that have predominance in the interaction of the workspace, since there may be differences in organizational politics, in government legislation for human resources, stability and efficiency in economic and political environment. These factors are often not taken into account, since what usually happens is the definition of new processes without giving sufficient attention to its implementation, hoping that the new procedures and technologies make by themselves the change of processes. This situation means that there is misalignment, since the teams change their practices but not its definition. The resultant misalignments of these facts make it difficult to share the best practices across the organization.

The problem arises since the adoption of CMMI-DEV already tends to standardize the business processes, which in a distributed organization with various implementations of the same process, face great resistance to change (Figure 2).



Figure 2 - Applying CMMI-DV in different locations

IV. PROPOSAL

We believe that only through a detailed analysis of the particularities of each of the distributed teams, understanding their differences, taking advantage of its strengths and identifying its limitations, we are able to first know their work processes, to boost and improve them afterwards.

Understand the current processes of the teams also allows to assess the level of adherence that their methods of work already have with the CMMI-DEV model, thereby finding the areas in, which no change is necessary, reducing the difficulties inherent in the processes of change. So our objective is to archive a great level of coordination between the different sites, based in the model CMMI-DEV, but respecting the local work processes and cultures.

Thus, in order to meet the objectives outlined above this paper proposes a method to implement CMMI-DEV trying to keep the various implementations of processes in the distributed organization, reducing risks and costs of implementation. It is

also expected a decrease in resistance to change by stakeholders in the processes.

To evaluate this proposal, it will be use in an organization that works with teams distributed in Canada, Guatemala, Portugal and India and wants to implement CMMI-DEV.

V. PRELIMINARY RESULTS

Under this project, some work was already done in order to better understand the processes undertaken in an organization that uses distributed teams. This organization intends to evaluate and put their processes in accordance with CMMI-DEV, so they can improve their work processes and have a better coordination and control between the different sites. The choice stood by this model because, in their opinion, it is the most famous in development area and with better known results.

In this company there are two distinct branches of development. One (branch A), which makes maintenance and minor improvements to an old product, and is certified with ISO 9001:2008 headquartered in Canada, with team members also in Portugal. The software development is done between Portugal and Canada, the definition of requirements and quality analysis is performed in Canada. There is another (branch B) to develop a product, which is not yet in production with the software development done in Guatemala, Portugal and India, with the respective definition of requirements and quality analysis performed in Guatemala.

Thus, in an early stage of this work, there was an incorporation in the team of internal auditors of ISO 9001:2008 of the organization, This analysis led to better understand the processes carried out and based on existing work in the area done by Mutafelija and Stromberg [19] [20] it was possible to perform a mapping between the ISO 9001:2008 and CMMI-DEV 1.3. This mapping aims the notion of taking advantage of the resources and synergies between the two models, having no influence in the SCAMPI of the CMMI-DEV.

There was thus a first survey of the faults to cover so that the branch (A) can converge with CMMI-DEV model. An example of the mapping between ISO 9001:2008 and CMMI-DEV is in Table 1, with the process area project planning.

TABLE 1- GAP IDENTIFICATION BASED ON ISO/CMMI-DEV MAPPING

Project Planning	
	Required Improvement based on Typical work products suggested by CMMI documentation.
SG 1 – Establish Estimates	
SP 1.1 - Estimate the Scope of the Project	Task descriptions; Work package descriptions; WBS;
SP 1.2 Establish Estimates of Work Product and Task Attributes	Technical approach; Size and complexity of tasks and work products; Estimating models; Attribute estimates
SG 2 – Develop a Project Plan	

SP 2.2 Identify Project Risks	Identified risks; Risk impacts and probability of occurrence; Risk priorities;
SP 2.3 Plan for Data Management	Data management plan; Master list of managed data; Data content and format description; Data requirements list for acquirers and for suppliers; Privacy requirements; Security requirements; Security procedures; Mechanism for data retrieval, reproduction, and distribution; Schedule for collection of project data; Listing of project data to be collected;
SP 2.5 - Plan for Needed Knowledge and Skills	Inventory of skill needs; Staffing and new hire plans; Databases (e.g., skills and training);
SP 2.6 - Plan Stakeholder Involvement	Stakeholder involvement plan
SG 3 – Obtain Commitment to the Plan	
SP 3.1 Review Plans That Affect the Project	Record of the reviews of plans that affect the project
SP 3.2 Reconcile Work and Resource Levels	Revised methods and corresponding estimating parameters (e.g., better tools and use of off-the-shelf components) Renegotiated budgets Revised schedules Revised requirements list Renegotiated stakeholder agreements
SP 3.3 Obtain Plan Commitment	Documented requests for commitments Documented commitments

In branch B, for each process area there has been made a first survey of the practices, which are followed, against the specific practices of CMMI-DEV in order to have a first iteration of the flaws to cover. This work was already made to all process areas of CMMI level 2. Please note that this survey of flaws is based on interviews with top managers.

An example of this more detailed survey for Project Planning is in Table 2.

TABLE 2- GAP IDENTIFICATION BASED ON INTERVIEWS WITH TOP MANAGERS OF THE ORGANIZATION

Project Planning	
	Required Improvements
SG 1 – Establish Estimates	
SP 1.1 – Estimate the Scope of the project	Stakeholder Form; Milestones Form; Meetings Form
SP 1.2 – Establish Estimates of Work Product and Task Attributes	Metrics Spreadsheet
SP 1.3 – Define Project Lifecycle Phases	In conformity
SP 1.4 – Estimate Effort and Cost	In conformity
SG 2 – Develop a Project Plan	

SP 2.1 – Establish the Budget and Schedule	Schedule and Project Cost Form
SP 2.2 – Identify Project Risks	Risks Form
SP 2.3 – Plan Data Management	In conformity
SP 2.4 – Plan the Project's Resources	Needed Ressources Form
SP 2.5 – Plan Needed Knowledge and Skills	Employees Skills Form; Relation Skills/Needed Resources Form
SP 2.6 – Plan Stakeholder Involvement	Involvement Plan
SP 2.7 – Establish the Project Plan	In conformity
SG 3 – Obtain Commitment to the Plan	
SP 3.1 – Review Plans That Affect the Project	Revision Plans Definition
SP 3.2 – Reconcile Work and Resource Levels	In conformity
SP 3.3 – Obtain Plan Commitment	Establish commitments

work already done, the mapping between ISO-CMMI (Figure 4).

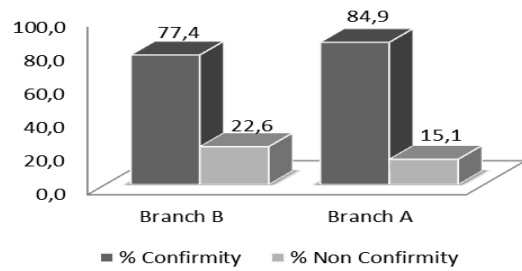


Figure 4- Total percentage of conformity with CMMI-DEV level 2

It was also possible to denote that although a branch being certified ISO 9001:2008, is not, in some cases, closer to the objectives of the model CMMI-DEV (Figure 5). We can, for example, remove that stance from the tables of preliminary results section, noting that the SP 2.3, which has no ISO 9001:2008 certification in the branch, already is consistent with the model, while the other branch certified resulting from the analysis made of the mapping done with CMMI-DEV, has some flaws that should be covered.

Based on mapping already done by Mutafelija and Stromberg, for each section of ISO there were a percentage of conformity related with the specific practices of CMMI-DEV (0%, 30%, 60% and 100%). So, our analysis of the branch A, based on this mapping, applies for the branch B, since the assessment was supported with the same percentages, resulting the following graphics.

The graphic below (Figure 3) is an example of the conformity analysis that the two branches have with each of the process areas, based on the assessment done, so, it was possible to know how far away each branch is to have their processes in compliance with CMMI-DEV.

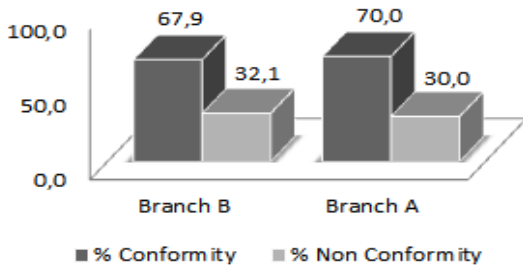


Figure 3- Percentage of conformity with the process area project planning

VI. EVALUATION

In this first analysis of a case of practical application of our proposal, it was possible to make a first evaluation.

Although the two branches of the same organization make development and want to adopt the same CMMI-DEV model across the organization, both branches and the teams are very different and work in dissimilar way.

This fact is a result not only from the particularities of the distributed teams, which have been discussed in this report but also from other factors more related to their work processes. Therefore, the initial factor, with a branch certified ISO 9001:2008 and the other not, raises great differences with regard to working methods. The adoption of CMMI-DEV, although not directly, allows it to become easier through the

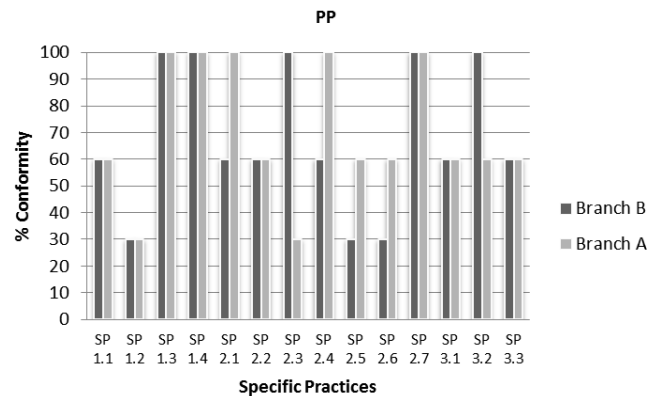


Figure 5- Percentage of conformity for each specific practice in process area project planning

Another factor that makes completely different the work practices is related to the development of the products of these two branches, which use different processes and tools. Consequently, its development cycle has also wide disparities in both the type and the manner of tasks to perform.

VII. CONCLUSION

This study proposed a method to evaluate the compliance that an organization has with CMMI-DEV, to enable their easier implementation in the nearest future. We propose that the differences between sites should be taken into account, respecting the distributed practices and culture, reducing the conflict between the various processes performed by geographically distributed teams. These allow a better knowledge of organizational processes, often unknown by top managers. This knowledge is essential to an organization in order to define the adjustments to apply to their processes.

So, in our work we study an organization with distributed teams. First it was done an assessment to know the compliance or each branch with CMMI-DEV, and after that there were proposed improvements to the processes, in order for them to be in accordance with the model.

Thus, it is expected that the proposed method contributes to solving the problem of organizational adaptability to CMMI-DEV. Expected to reduce the impact of the change processes through a deep understanding of current processes of the organization, allowing us to find the flaws in order to develop an action plan. It is believed that this would make the compliance with CMMI-DEV easier to implement even in work performed with resource to distributed teams.

REFERENCES

- [1] Charette, R.N., 2005, Why Software Fails. Journal IEEE Spectrum, Volume 42, Number 9
- [2] Jones, C., 2004, Software Project Management Practices: Failure Versus Success, Crosstalk, Citeseer Volume 17, Number 10
- [3] Rao, N.M., 2009, Challenges in Execution of Outsourcing Contracts, Proceedings of the 2nd India Software Engineering Conference, ACM, 75-79
- [4] Blinder, A.S., 2006, Offshoring: The Next Industrial Revolution?, Foreign affairs, Volume 82, Number 2, 113-128
- [5] Carmel, E. and Agarwal, R., 2002 Tactical Approaches for Alleviating Distance in Global Software Development, Journal IEEE, Volume 18, Number 2, 22-29
- [6] Azderka, M. and Grechenig, T., 2009, Project Management Maturity Models: Towards Best Practices for Virtual Teams, Engineering Management Conference, 2007 IEEE International, 84-89
- [7] Jugdev, K. and Thomas, J., 2002, Project Management Maturity Models: The silver Bullets of Competitive Advantage, Project Management Journal, Volume 33, Number 4, Proquest ABI/INFORM, 4-14
- [8] Wang, Q. and Li, M., 2006, Software Process Management: Practices in China, Unifying the Software Process Spectrum, Springer, 317-331
- [9] Edwards, H.K. and Sridhar, V., 2003, Analysis of the Effectiveness of Global Virtual Teams in Software Engineering Projects, IEEE Computer Society
- [10] Al-Ani, B. and Redmiles, D., 2009, Supporting Trust in Distributed Teams Through Continuous Coordination, IEEE Software, Volume 99, Number 1, 35-40
- [11] Siakas, K., Maoutsidis, D. and Siakas, E., 2006, Trust facilitating good software outsourcing relationships, Springer, 171-182
- [12] Hinds, P.J. and Mortensen, M., 2005, Understanding conflict in geographically distributed teams: The moderating effects of shared identity, shared context, and spontaneous communication, Organization Science, Volume 16, Number 3, Institute for Operations Research and the Management Sciences, 290-307
- [13] Huang, H. and Ocker, R., 2006, Preliminary Insights Into the In-Group/Out-Group Effect in Partially Distributed Teams: An Analysis of Participant Reflections, Proceedings of the 2006 ACM SIGMIS CPR Conference on Computer Personnel Research: Forty Four years of computer personnel research: achievements, challenges in the future, ACM, 264-272
- [14] Suchan, J. and Hayzak, G., 2002, The Communication Characteristics of Virtual Teams: A case study, vol 44, Number 3, IEEE, 174-186
- [15] Akbar, R. and Hassan, M.F., 2010, Limitations and Measures in Outsourcing Projects to Geographically Distributed Offshore Teams, 2010 International Symposium in Information Technology (ITSim), Volume 3, IEEE, 1581-1585
- [16] Chatfield, A.T. and Wanninayaka, P., 2008, IT Offshoring Risks and Governance Capabilities, IEEE Computer Society
- [17] Ramanathan, TR., 2009, The Role of Change management in Implementing the Offshore Outsourcing Business Model: A Processual View
- [18] Sirkin, H.L., Keenan, P., and Jackson, A., 2005, The Hard Side of Change Management, Harvard Business Review, Volume 83, Number 10
- [19] Yoo, C., Yoon, J., Lee, B., Lee, C., Lee, J., Hyun, S. and Wu, C., 2006, A Unified Model for the Implementation of Both ISO 9001: 2000 and CMMI by ISO-Certified Organizations, Journal of Systems and Software, Volume 7, num 7, Elsevier, 954-951
- [20] Mutafelija B. and Stromberg H., 2009 Mappings of ISO 9001:2000 and CMMI Version 1.2
- [21] CMMI Produt Team., 2010, CMMI for Development , Version 1.3 (available online at <http://www.sei.cmu.edu/reports/10tr033.pdf> , accessed in May 2011)
- [22] Guzman, G., Ramos, S., Seco, A. and Esteban, S., 2010, How to Get Mature Global Virtual Teams: A Framework to Improve Team Process Management in Distributed Software Teams Volume 18, Number 4, Software Quality Journal, Springer, 409-435}
- [23] IDC. IDC Executive Market Watch Market Research, 2006 available at http://cdn.idc.com/uk/downloads/events/idc_executive_market_watch.pdf , accessed in May 2011).
- [24] Sidhu, J.S. and Volberda, H.W. 2011, Coordination of globally distributed teams: A Co-Evolution Perspective on Offshoring, International Business Review, Elsevier

Advanced Object Oriented Metrics for Process Measurement

Shreya Gupta

Indian Institute of Information Technology
Deoghat, Jhalwa, Allahabad, India
gupta.shreya29@gmail.com

Ratna Sanyal

Indian Institute of Information Technology
Deoghat, Jhalwa, Allahabad, India
rsanyal@iitaa.ac.in

Abstract— Process improvement requires measurement of specific attributes of process. Measurement of a process gives us a clear insight into the system. It provides effective ways of estimation and evaluation. Then, it is essential to develop a set metrics covering the attributes. Computed measures are used as indicators for process improvement areas. These indications if incorporated into the software development, will lead to development of an effective and reliable system. Mood metrics has defined some indicators for inheritance like Attribute Inheritance Factor (AIF), Method Inheritance Factor (MIF), and for hiding are Attribute Hiding Factor (AHF), Method Hiding Factor (MHF). We are proposing extensions to these metrics. These extensions are more specific and give a better hint towards inheritance and hiding properties.

Keywords—Mood Metrics; Attribute Inheritance Factor; Method Inheritance Factor; Attribute Hiding Factor; Method Hiding Factor.

I. INTRODUCTION

Object orientation aims to model a system [1]. They reflect a natural view and understanding of the system. Using object modeling, a system is represented as number of objects and their interaction. Objects are categorized into classes along with their respective behavioral properties [2]. Inheritance provides the facility for classes to inherit the behavioral properties of other classes. Encapsulation packages functions and data in a class. Representing essential features with exclusion of background explanations is called abstraction [3].

Object Oriented Software Paradigm gives the way for effective reuse of program components. The process of reuse expedites the software development and thereby resulting in high quality work in minimum time. They are easy to understand, adapt and scale because of modular structure, relatively low coupling and high cohesion. Merely applying object oriented programming will not reap great results. It is the combination of object oriented domain analysis, requirement analysis, object oriented design, database systems and computer aided software engineering that will lead to best results.

If software is developed without any proper measurement activities, the resulting product could be unreliable, inefficient and non-maintainable. We need to realize the ideology that software needs to be engineered. For this, standard engineering principles and guidelines are

to be established. Software metrics come into play as quantify the attributes in the development. Errors undetected in a development phase are passed in the next phase. Relative cost of fixing it increases many times. Therefore, tracing errors early in lifecycle and fixing them are essential.

Second section describes the prior work in the field of software metrics particularly C.K. Metrics and Mood metrics. Since the research paper is proposing an extension to the AIF, MIF, AHF and MHF, a detailed explanation of these metrics has been provided with reference to published research papers. Third section of the paper proposes the extension to AIF, MIF, AHF and MHF computation along with the extended formulas for the same. Fourth section is Result and Analysis section for AIF, MIF, AHF and MHF, considering a system and showing the variation in values obtained by the original formulas and the extended ones. Furthermore, a case study has been taken to validate the results of these metrics.

II. PRIOR WORK

Six software metrics were proposed for object oriented design, known as C. K. Metrics [4]. These metrics are Response of a Class (RFC), coupling between the objects (CBO), Weighted Methods per Class (WMC), Number of Children (NOC), Lack of Cohesion Methods (LCOM) and Depth of Inheritance Tree (DIT). The empirical evidence specifies how object oriented metrics determine software defects is described [5].

Mood Metrics (Metrics for Object Oriented Design) were proposed by Abreu as described [6]. These metrics aim to evaluate object oriented principles in the software code. It considers inheritance factor which computes attribute inheritance factor and method inheritance factor, encapsulation factor which computes attribute hiding factor and method hiding factor. All of these metrics result in the probability value between 0 and 1.

In the following subsections, we have explained and mentioned the formulas of the existing parts of MOOD metrics.

A. Attribute Inheritance Factor (AIF)

Attribute Inheritance Factor (AIF) is the ratio of two measurements. Numerator represents the sum of number of inherited attributes of all classes in system and denominator represents sum of number of available attributes which may

be local or inherited for all classes in system. It expresses the level of reuse in the system. A threshold is maintained for AIF measure that is roughly around 50%. Higher values of AIF indicate high inheritance level thereby leading to greater coupling and reducing the possibility of reuse. MOOD Metrics propose the computation of AIF [6] as given below:

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)} \quad (1)$$

where $A_a(C_i) = A_d(C_i) + A_i(C_i)$

$A_i(C_i)$ is the count of attributes that are inherited.

$A_d(C_i)$ is the count of defined attributes. These attributes can be of any access modifier.

$A_a(C_i)$ is the count of attributes that can be referenced by class C_i

TC - total count of classes in system/ package.

B. Method Inheritance Factor (MIF)

Method Inheritance Factor (MIF) is the ratio of two measurements. Numerator represents the sum of number of inherited methods of all classes in system and denominator represents sum of number of available methods which may be local or inherited for all classes in system. Method Inheritance Acceptable range is 20% to 80%. It highly depends on the design pattern that we follow. High values of MIF indicate superfluous inheritance and low values indicate heavy use of overrides or lack of inheritance. MOOD Metrics propose the computation of MIF [6] as given below:

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)} \quad (2)$$

where $M_a(C_i) = M_d(C_i) + M_i(C_i)$

$M_i(C_i)$ is the count of methods that are inherited. These methods should not be overridden.

$M_d(C_i)$ is the count of defined non-abstract methods. These methods can be of any access modifier.

$M_a(C_i)$ is the count of methods that can be called by class C_i .

TC - total count of classes in system/ package.

C. Attribute Hiding Factor (AHF)

AHF measures the extent of encapsulation of attributes in a system. Firstly, it will calculate the visibility of each attribute with respect to each class. Visibility function assigns 1 for each class, if the attribute is visible from those classes and 0 if not visible. Visibility measure for class in which attribute is present itself is considered to be 0. It sums up the visibility for a particular attribute and then divides by the (total no. of classes minus 1). Likewise, the visibility of each attribute is calculated and then values are substituted in AHF formula. Thus, AHF represents the average amount of hiding of attributes among all classes in system. Visibility of private attributes is always zero. Protected attributes act as a public attribute in the package to which the attribute belongs, and are visible only in the subclasses in other

packages. Public attributes are visible to all classes in the system. If all the attributes are private, then AHF=100% and if all the attributes are public, AHF is 0%. MOOD Metrics propose the computation of AHF [6] as given below:

$$AHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{Ad(C_i)} (1-V(A_{mi}))}{\sum_{i=1}^{TC} Ad(C_i)} \quad (3)$$

where

$$V(A_{mi}) = \sum_{i=1}^{TC} is_visible(A_{mi}, C_j) / (TC - 1)$$

and

$$is_visible(A_{mi}, C_j) = \begin{cases} 1 & \text{iff } j \neq i \text{ and } C_j \text{ may reference } A_{mi} \\ 0 & \text{otherwise} \end{cases}$$

$A_d(C_i)$ is the count of defined attributes. These attributes can be of any access modifier. They should not be inherited ones.

D. Method Hiding Factor (MHF)

It measures the extent of encapsulation of methods in a system. Firstly, it will calculate the visibility of methods with respect to each class. Visibility function assigns 1 for each class, if the method is visible from those classes and 0 if not visible. Visibility measure for the class in which method is present itself is considered to be 0. It sums up the visibility for a particular method and then divides by the (total no. of classes minus 1). Likewise the visibility of each method is calculated and then values are substituted in MHF formula. Thus, MHF represents the average amount of hiding of methods among all classes in system. Visibility of private methods is always zero. Protected methods act as a public method in the package to which the method belongs, and are visible only in the subclasses in other packages. Public methods are visible to all classes in the system. If all the methods are private, then MHF=100% and if all the methods are public MHF is 0%. MOOD Metrics propose the computation of MHF [6] as given below:

$$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{Md(C_i)} (1-V(M_{mi}))}{\sum_{i=1}^{TC} Md(C_i)} \quad (4)$$

where

$$V(M_{mi}) = \sum_{i=1}^{TC} is_visible(M_{mi}, C_j) / (TC - 1)$$

and

$$is_visible(M_{mi}, C_j) = \begin{cases} 1 & \text{iff } j \neq i \text{ and } C_j \text{ may reference } M_{mi} \\ 0 & \text{otherwise} \end{cases}$$

$M_d(C_i)$ is the count of methods and constructors. These methods can be of any access modifier. They should not be abstract or inherited.

III. PROPOSED EXTENSION

A. Extension in AIF and MIF

Problem with the AIF/MIF formula is that it considers the count of members a class can reference in a system or a package. But, when we calculate AIF/MIF for each class, members outside the class (except for the members that are inherited) are not to be considered. Justification is that denominator of the formula of AIF and MIF states that “Total no. of members that a class C_i can reference”, all the members that are public can be referenced by a class, no matter whether it is in its same package or outside the package. Even protected members act as public members in their own package. Thus, while calculating AIF, MIF the count of uncoupled members in the denominator should not be considered, because access to public, protected members does not reflect the measure of inheritance factor.

Thus, an extension to the empirical formula is proposed by us. For denominator, consider the members of ancestor classes of class C_i and the members defined inside class C_i only. If a class “x” is present in same package as that of class C_i and has public members, but has no interaction with the class C_i , then members of class “x” are not considered.

When a class inherits considerable number of members from the ancestor classes, it will contribute to a high measure of AIF, MIF. When a class redefines the ancestor members and adds the new members will always contribute to a low measure of AIF, MIF. The extended equation for AIF is given below:

$$\text{AIF extended} = \sum_{i=1}^{TC} A_i(C_i) / \sum_{i=1}^{TC} A_{ex}(C_i) \quad (5)$$

where $A_{ex}(C_i) = A_d(C_i) + A_i(C_i)$

$A_i(C_i)$ is the count of attributes that are inherited.
 $A_d(C_i)$ is the count of defined attributes. These attributes can be of any access modifier.
 $A_{ex}(C_i)$ is the extended variable. It is the count of attributes that can be referenced by class C_i considering the attributes of ancestor classes of class C_i and the attributes defined inside class C_i only.

The extended equation for MIF is given below:

$$\text{MIF extended} = \sum_{i=1}^{TC} M_i(C_i) / \sum_{i=1}^{TC} M_{ex}(C_i) \quad (6)$$

where $M_{ex}(C_i) = M_d(C_i) + M_i(C_i)$

$M_i(C_i)$ is the count of methods that are inherited. These methods should not be overridden.
 $M_d(C_i)$ is the count of defined non-abstract methods. These methods can be of any access modifier.

$M_{ex}(C_i)$ is the extended variable. It is the count of methods that can be called by class C_i considering the methods of ancestor classes of class C_i and the methods defined inside class C_i only.

Thus, AIF extended and MIF extended give an accurate idea about the actual level of inheritance that exists in the code. If the level of inheritance is high, then it is a hindrance to the reusability, maintainability and understandability of system. It will be difficult to reuse the modules of code into some other system because of its dependency on other modules.

B. Extension in AHF and MHF

Original AHF equation consists of visibility function that checks that if class may reference the attribute in consideration. But, in the extension that I have proposed, it checks whether actually the class has referenced the attribute or not. This extension in AHF is more specific in nature and gives a clear hint of the hiding factor. It also checks for a good design characteristic that attributes of a class should be accessed by methods of the class only. If they are directly accessed by the objects of some other class, then design is not stable. The extended equation for AHF is given below:

$$\text{AHF extended} = \sum_{i=1}^{TC} \sum_{m=1}^{Ad(C_i)} (1 - V(A_{ex})) / \sum_{i=1}^{TC} A_d(C_i) \quad (7)$$

where

$$V(A_{ex}) = \sum_{i=1}^{TC} is_visible(A_{ex}, C_j) / (TC - 1)$$

and

$$is_visible(A_{ex}, C_j) = \begin{cases} 1 & \text{iff } j \neq i \text{ and } C_j \text{ did reference } A_{mi} \\ 0 & \text{otherwise} \end{cases}$$

$A_d(C_i)$ is the count of defined attributes. These attributes can be of any access modifier. They should not be inherited attributes.

Original MHF equation consists of visibility function that checks that if class may reference the method in consideration. Same extension goes with MHF. We check whether actually the class has referenced the method or not. The extended equation for MHF is given below:

$$\text{MHF extended} = \sum_{i=1}^{TC} \sum_{m=1}^{Md(C_i)} (1 - V(M_{ex})) / \sum_{i=1}^{TC} M_d(C_i) \quad (8)$$

where

$$V(M_{ex}) = \sum_{i=1}^{TC} is_visible(M_{ex}, C_j) / (TC - 1)$$

and

$$is_visible(M_{ex}, C_j) = \begin{cases} 1 & \text{iff } j \neq i \text{ and } C_j \text{ did reference } M_{mi} \\ 0 & \text{otherwise} \end{cases}$$

$M_d(C_i)$ is the count of methods and constructors. These methods can be of any access modifier. They should not be abstract or inherited.

TC - total count of classes in system/ package.

Thus, AHF extended and MHF extended propose a change in the visibility function of their respective calculations. This visibility function ensures that whether the members of a class have been actually referenced by outside members or not. This helps us in understanding the amount of abstraction in the system thereby giving clarity in estimation of actual hiding factors.

IV. RESULTS ANALYSIS

To demonstrate the variation in AIF and AIF extended values, MIF and MIF extended values, we have used a small example considering a design for university database.

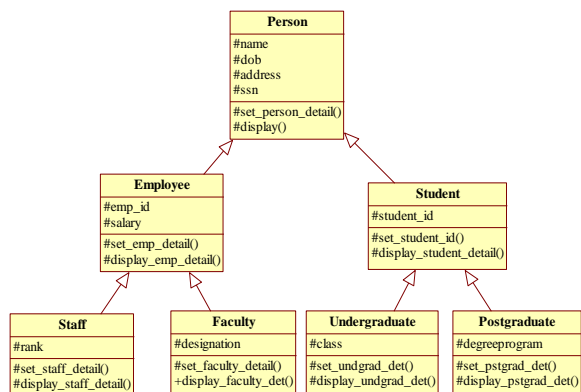


Figure 1. University Database

We have calculated the AIF values [6] and proposed extended AIF for each class as well as MIF values [6] and proposed extended MIF. The class diagram of the example system along with tabulated results and graph of AIF and AIF extended, MIF and MIF extended is given below.

A. AIF Analysis

A threshold value of 0.5 is maintained in order to determine whether level of inheritance is acceptable or not. For AIF values greater than 0.5, extent of inheritance is high. Classes employee, student, undergraduate, postgraduate have “AIF extended” values greater than 0.5 and “AIF” values less than 0.5. Class diagram in Fig. 1 shows us effectively that these classes inherit large number of attributes from ancestor classes than the attributes they actually contain, thereby depicting unacceptable level of inheritance.

TABLE I. ANALYSIS OF AIF

Classes	AIF for each class (Farooq,2005)	AIF Extended for each class (Proposed)
Person	0.00	0.00
Employee	0.36	0.67
Staff	0.54	0.86
Faculty	0.54	0.86
Student	0.36	0.67
Undergraduate	0.45	0.83
Postgraduate	0.45	0.83
Main	0.00	0.00

Total AIF of the system can be calculated using (1):

$$AIF = (0+4+6+6+4+5+5) / (11+11+11+11+11+11+11) = 0.39$$

Here, the numerator is the number of attributes inherited from ancestor classes for each class. As “person” is the base class, it does not inherit any attribute, “employee” class inherits four attributes, “staff” class inherits six attributes in total from person class and employee class, “faculty” class six, and so on for rest of the classes. Sequence of classes used in the formula is same as the sequence given in the table I. Denominator is number of attributes that can be referenced by each class. Attributes in the class diagram are protected in nature, but we know that protected members are public in their own package. Therefore, each class can reference all the public and protected members in the system. Denominator is eleven for each class.

We compute AIF extended using (5). The numerator remains the same as that of AIF but denominator changes as we consider the attributes of ancestor classes of class C_i and the attributes defined inside class C_i only. For example, “person” class is base class; we consider only the four attributes defined inside it. “Employee” class is inheriting from person class, so the attributes in consideration are six, out of which four are from person class and two from employee class. Similarly, denominators are determined for rest of the classes.

$$AIF \text{ extended} = (0+4+6+6+4+5+5) / (4+6+7+7+5+6+6) = 0.73$$

Therefore, AIF extended is giving a clear idea that level of inheritance in the system is not acceptable as it is greater than 0.5.

B. MIF Analysis

Same extension is followed in MIF extended but with respect to the methods. Classes staff, faculty, undergraduate,

postgraduate have “MIF extended” values greater than 0.5 and “MIF” values less than 0.5.

TABLE II. ANALYSIS OF MIF

Classes	MIF for each class (Farooq,2005)	MIF Extended for each class (Proposed)
Person	0.00	0.00
Employee	0.14	0.5
Staff	0.29	0.67
Faculty	0.29	0.67
Student	0.14	0.5
Undergraduate	0.29	0.67
Postgraduate	0.29	0.67
Main	0.00	0.00

Total MIF of system is calculated using (2):

$$MIF = (0+2+4+4+2+4+4+0) / (14+14+14+14+14+14+14+14) = 0.18$$

Sequence of the classes in the formula remains same as given in table II. Numerator is number of methods inherited by each class from ancestor classes. Denominator is number of methods that can be referenced by each class. As mentioned earlier that protected members are public in their own package, each class can reference all public and protected methods in system. Denominator is fourteen for each class. Now, we calculate MIF extended of system using (6).

$$MIF \text{ extended} = (0+2+4+4+2+4+4+0) / (2+4+6+6+4+6+6+1) = 0.57$$

Numerator remains the same as that of MIF. Denominator changes according to the proposed work. We need to consider methods of ancestor classes of a class C_i and the methods defined inside class C_i only. “Person” is a base class and has two methods of its own. “Employee” class is inheriting two methods from person class and has two methods of its own, therefore a count of four. Likewise, we do the calculation. Therefore, MIF extended is giving a clear idea that level of method inheritance in system is not acceptable as it is greater than 0.5.

C. AHF Analysis

We have considered a code to demonstrate the hiding factor. The sample code is as follows:

```
class Account_bank
{
    public double balance_amt;
    public double acc_no;
    public void initialize_data( double amt, double no )
    {
        balance_amt = amt;
        acc_no=no;
    }
    public void deposit_bank ( double amt )
    {
        balance_amt += amt;
    }
    public double withdraw_bank ( double amt )
    {
        if (balance_amt >= amt)
        {
            balance_amt -= amt;
            return amt;
        }
        else
            return 0.0;
    }
}
```

```
class Interest_Account_bank extends Account_bank
{
    private static double interest_default = 7.95;
    private double rate_int;
    public void Initialize_interest()
    {
        balance_amt = 0.0;
        rate_int = interest_default;
    }
    public void add_interest_monthly()
    {
        balance_amt = balance_amt + (balance_amt * rate_int / 100) / 12;
    }
    private double get_balance()
    {
        return balance_amt;
    }
}
```

```
class Account
{
    public static void main(String a[])
    {
        Interest_Account_bank my_account = new Interest_Account_bank();
        my_account.Initialize_interest();
        my_account.initialize_data(8325,2520);
        my_account.deposit_bank(300.00);
        System.out.println(my_account.acc_no);
        System.out.println("Net Balance " +my_account.balance_amt);
        my_account.withdraw_bank(50.00);
        System.out.println("Net balance " +my_account.balance_amt);
        my_account.add_interest_monthly();
    }
}
```

On the basis of above code, we check the current references made to attributes and methods. First we calculate AHF. Consider the attributes of Account_bank class, they are balance_amt, acc_no. Attribute balance_amt may be referenced by rest of the two classes, i.e. Interest_Account_bank and Account as it is a public variable. Therefore, using (3), visibility (bank_amt) is 2/(3-1), that is 1. Thus, value of (1-V(bank_amt)) is 0. Similarly, for the attribute acc_no, (1-V(acc_no)) is 0. Now, we consider the attributes of class Interest_Account_bank, they are interest_default, rate-int.

TABLE III. ANALYSIS OF AHF

Classes	Attributes	(1-V(A _{mi})) in AHF	(1-V(A _{ex})) in AHF ext.
Account_bank	balance_amt	0	0.5
Account_bank	acc_no	0	0.5
Interest_Account_bank	interest_default	1	1
Interest_Account_bank	rate-int	1	1

Both the attributes are private; therefore none of the classes can access them. Visibility is 0 for both the attributes, (1-V(A_{mi})) is 1. Now, we apply the formula for AHF from (3).

$$AHF = (0+0+1+1) / (4) = 0.5$$

Now, we calculate AHF extended. For the attributes balance_amt and acc_no of Account_bank class, they are actually referenced by the object of Interest_Account_bank. Thus, only one class has made an access to these attributes. Visibility for these attributes is 1 / (3-1), i.e. 0.5. Therefore,

(1-V(A_{ex})) is (1-0.5) i.e. 0.5. Similarly, for interest_default and rate-int attributes, none of the classes has accessed them, therefore visibility is 0 and (1-V(A_{ex})) is 1. Now, we apply the formula for AHF extended from (7):

$$AHF \text{ extended} = (0.5+0.5+1+1) / (4) = 0.75$$

Higher value of AHF extended indicates that attributes are not actually referenced, thereby imparting a private attribute behavior to them. Visibility of attributes is not properly used by the design of the system.

D. MHF Analysis

First, we calculate MHF. Consider the methods of Account_bank class. This class has three public methods, namely initialize_data, deposit_bank and withdraw_bank. All three methods are public, and can be accessed by rest of the two classes. Therefore, using (4) visibility of all four

methods is 2/(3-1), that is 1. Thus, value of (1-V(M_{mi})) is 0 for all three methods. Class Interest_Account_bank has three methods, namely initialize_interest, add_interest_monthly, and get_balance. Getbalance method is a private method that cannot be referenced by outside classes. Therefore, its visibility is 0 and (1-V(M_{mi})) is 1.

TABLE IV. ANALYSIS OF MHF

Classes	Methods	(1-V(M _{mi})) in MHF	(1-V(M _{ex})) in MHF ext.
Account_bank	Initialize_data	0	0.5
Account_bank	withdraw_bank	0	0.5
Account_bank	deposit_bank	0	0.5
Interest_Account_bank	Initialize_interest	0	1
Interest_Account_bank	add_interest_monthly	0	1
Interest_Account_bank	get_balance	1	1

Now, we apply the formula for MHF from (4):
MHF= (0+0+0+0+0+1) / (6) = 0.17

Now, we calculate MHF extended. For methods initialize_data, deposit_bank and withdraw_bank of Account_bank class have been actually referenced by the object of Interest_Account_bank class. By using (8), visibility of these methods is 1/(3-1) i.e. 0.5. Therefore, (1-V(M_{ex})) is (1-0.5) i.e. 0.5. Methods of Interest_Account_bank have not been referenced by any other class, therefore, their visibility is 0 and (1-V(M_{ex})) is 1. Now, we apply the formula for MHF extended from (8):

$$MHF \text{ Extended} = (0.5+0.5+0.5+1+1+1) / (6) = 0.75$$

Such a high value of MHF extended indicates that most of methods are not being actually referenced by the outside classes.

E. Case Study Analysis

Library Management system for a college is used as a case study. It has separate java files for books, catalogue, members, librarian etc. Books may be reference book or issuable book. Members may be student or a faculty member. All the four metric i.e. AIF, MIF, AHF and MHF were applied on the case study. Also, the proposed extensions to these metrics were applied.

TABLE V. CASE STUDY RESULT

Metric	(Farooq,2005)	Extended Versions
AIF	0.25	0.52

MIF	0.18	0.59
AHF	0.52	0.98
MHF	0.94	0.11

There was variation in results, confirming the extensions were specific and gave a hint about design of the system. Metric values are capable to comment on stability of design and actual hiding factors. In all the cases, extended metrics resulted in values higher than the original metrics. Extended AIF and extended MIF gave values higher than threshold indicating the system has higher inheritance. Classes are highly coupled in system. Extended AHF and extended MHF also result in higher values than original metric showing greater hiding factor.

CONCLUSION AND FUTURE WORKS

The extensions in AIF and MIF are more accurate than previous definitions as they give a better idea about usage of inheritance property in the code. Results are accompanied with analysis part showing the variation in the values. Clearly, classes that have AIF, MIF values greater than threshold value needs some modification in their design. Extensions in AHF and MHF check whether a member (data or method) has been actually referenced or not. This gives clarity in estimation of actual hiding factors. Therefore, proposed extensions give accurate estimation of inheritance

and hiding factor. Regarding future works, developed tool must have a provision for suggesting corrections to user, based on result of metrics. Developed tool analyses java source files and class files. Thus, tool can give results only after coding phase. An approach may be developed to apply metrics in earlier phases of development.

ACKNOWLEDGMENT

The authors highly acknowledge the immense support of Indian Institute of Information Technology, Allahabad, for providing the adequate resources to carry out this research work.

REFERENCES

- [1] Jacobson I., Christerson M., Jonsson P., and Overgaard G. Object-Oriented Software Engineering, Pearson Education, Singapore, Ninth Indian Reprint, 2004.
- [2] Pressman R. S., Software Engineering: A Practitioner's Approach, McGraw Hill Publication, Singapore, Sixth edition, 2005.
- [3] Balagurusamy E., Programming with Java: A Primer, McGraw Hill Publication, New Delhi, Thirtieth reprint, 2006.
- [4] Shyam R. C. and Chris F. K. "A Metrics Suite for Object Oriented Design" IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994.
- [5] Subramanyam R. and Krishnan M.S. "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects" IEEE Transactions on Software Engineering, Vol. 29, No. 4, April 2003.
- [6] Farooq A., Braungarten R., and Dumke R.R. "An Empirical Analysis of Object-Oriented Metrics for Java Technologies" 9th International Multitopic Conference, pp. 1-6, IEEE INMIC 2005.

Quality Issues in Global Software Development

Sanjay Misra

Dept. of Computer Engineering, Atilim University
Ankara, Turkey
smisra@atilim.edu.tr

Luis Fernández-Sanz

Dept. of Computer Sciences, Universidad de Alcalá
Alcalá de Henares, Spain
luis.fernandezs@uah.es

Abstract— The most advantageous features of Global Software Development (GSD) are its cost saving benefits and the easily availability of resources. Also the technological advancement especially in Information and Communication Technology (ICT) makes GSD a common practice in software industry. But GSD is also facing a lot of challenges. Maintaining quality in software development processes and products in GSD environments is one of the major challenges. This paper presents a survey on the challenges and factors which impact on the quality of the products in GSD environments. This report identifies that most of the factors which affect the quality of software product appear as part of two major challenges: requirements and coordination. We further demonstrate that how these two challenges are affected by several factors. Finally, we present the possible solution to reduce the complexity of those various factors.

Keywords-Global Software Development; process quality; product quality; requirement and coordination challenges.

I. INTRODUCTION

The last decade is the evidence for the changing trends from the traditional software developments process, which was mainly confined on in-house software development, to the Global Software Development (GSD), where whole development process is distributed at different locations all around the world. In fact, advancement in tools and techniques in software development process has allowed geographically and culturally diverse groups to come together in global software development teams [1]. Further, the technological improvement in Information and Communication Technology (ICT) helps GSD to become a common practice in software industry. More specifically, Internet changed the whole society set up and; also make GSD as one of the most popular trends amongst software community.

There are several benefits of GSD. The most important ones which attract to the companies are cost and easy access to human resources. Although these are major advantages, several challenges also exist in GSD which makes difficult maximizing the benefits. One of the major challenges is how can one control the quality, while assigning the task at different locations where the actual work on project or sub projects are going to be performed. In fact, a unique feature of software product is that different parts cannot be developed as isolated activities. This is because there are complex dependencies between several tasks/parts of project; therefore development team members communicate with each other to fulfill their tasks during the whole

development process [2]. In case of GSD, tasks are carried out at different locations distributed all around world, so communication suffers [3], and, as a result, the quality of product is also affected.

One can find several papers which address the different types of challenges in distributed software development such as contextual, cultural, organizational, geographical, temporal, and political [4]. However, the particular work on quality issues involved in GSD has not been researched yet to the appropriate extent. There are only few papers which addresses quality issues specifically. Ivček et al. [4] have presented a paper ‘Aspects of Quality Assurance in Global Software Development Organization’ in which the authors reported a case study of a project developed in two offshore countries. In their work they explained the way for applying quality assurance techniques in GSD environments. In another paper [5], the authors have explored whether working with others in GSD environment really matters for the quality of the software that is produced in global settings.

The lack of proper works for quality issues in GSD environments motivate us to work in this area. In this paper, we focus more particularly into quality issues: especially how quality of the final product is affected by the different factors involved in GSD. The observations reflect that, in general, the most of the challenges and issues of GSD seriously affect the quality of product. This paper presents a systematic study on how these challenges affect the quality and the ways to cope with these challenges. This paper is primarily a survey report which tries to identify the quality issues and their possible solutions in a GSD environment.

The paper is organized as follows. In the following sections, we start our investigation of quality challenges in software process and product in a GSD environment. In the same section, we also discuss and identify the several issues which affect process and product qualities. The solutions for the common problems which are responsible for quality are given in Section 3. The discussion on the work and the conclusion are given in Section 4 and Section 5, respectively.

II. QUALITY OF SOFTWARE PRODUCT IN A GSD ENVIRONMENT

IEEE defines the software quality as the degree to which a system, components or processes meet the specified requirements [6]. A modified form of this definition is: the degree to which a system, component or process meets customer or user needs or expectation [7]. In other words, quality means conformance to fulfilling the customers’ requirements [8]. Juran [9] defined quality as “Quality

consists of those product features which meet the needs of customers” and thereby provide product specification. Further, Pressman [10] defines the software quality as conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected of all professional developed software. If we combine and analyze all these definitions, we can observe that quality is mainly concerned with conformance to requirements. In general, the quality in requirements is achieved by using proper and effective elicitation techniques, which can provide the correct requirements.

The quality in product in a GSD environment is also affected by the type of software process as well as the existence of proper control of different activities within the software development process. However these, above activities i.e., collecting requirements and proper control of software process activities are not easy tasks in GSD environments. In globally distributed environment, several factors such as intercultural factors language barriers (i.e., the communication problems) reduce the chances of collecting the proper and sufficient requirements and make more difficult the coordination amongst the distributed team. As a result, it becomes difficult to control the quality in the processes at different locations of the world. Bartelt et al. [2] also support our claim that the key challenge of global software engineering is to establish appropriate communication and coordination habits in a global project environment [11]. However, in our opinion, communication is the main for proper coordination.

If we analyze how to achieve the quality objectives in global software distributed environment, firstly we have to evaluate how we can achieve the quality in eliciting the correct requirements. This, which is really not an easy job in distributed environment where customers and developers might be physically far away to each others. Secondly, we have to control quality in the software development process.

In summary, we can conclude that:

1. Quality in software product in GSD environment is challenged by the two major factors (Table 1).
 - a. The lack of sufficient and correct requirements (due to several factors of GSD) and
 - b. The lack of proper coordination in software development process.

TABLE I. QUALITY AND CHALLENGES IN GSD

Quality in Product in GSD Environment		
	Requirement Quality	Processes Quality
Identified Challenges	Requirement [12] [13]	Coordination [11] [2]

The conclusion of above discussion is that the collection of proper requirements and coordination are main challenges in achieving quality in software development process in GSD environment. There are several factors which are responsible for these challenges. We will discuss these challenges and factors in more detail in the next section.

III. CHALLENGES FOR QUALITY IN GSD: THE REQUIREMENT AND COORDINATION AND THE FACTORS EFFECTING THEM

In the previous section, we have concluded that quality in GSD environment is challenged by requirement-related issues and problems in coordination. There are other factors, which impact the quality of the process and product in GSD environment. We treat them as factors of these two main challenges.

Gathering requirements from customers is always a challenge for software engineers. The quality of the product is highly affected by the absence of clear requirements. Guzman et al. [14] stated that normally there is a lack of common understanding of goals and requirements amongst development team members in GSD environments. Further, the growing number and the volatility of requirements also create additional challenges. Lormans et al. [12] stated that the evolution of requirements is usual: no matter how thorough the requirements specification has been set up, the requirements for any non-trivial system will change not only after the system has been built but also during the process of implementing the system. Further, the paper highlighted that the evolution of requirements appears due to a great variety of reasons [13]. In fact, requirements change and evolve mainly due to changing the requirement of business needs and also due to change in technology. Requirements are also modified in the process of designing, implementing, and writing test cases for requirements. These all examples show that there is always a need of modification of the initial set of requirements throughout the lifecycle of the project: and, if customers are far away from the development site, it becomes a challenge.

The agile software development approach appears in practice as a way to overcome the above problem. However, the principles of GSD are contradictory with the philosophy of agile development: in GSD, teams are distributed all around the world and customers are not necessarily be at the same site of development while agile promotes development, where the software is developed through iterative process with close involvement of the customers representatives. Agile approach is a widely accepted practice in software development and it claims getting more successful process in comparison to other practices. One of the reasons for this success rate is that customers and the development team members’ work together more affectively. This technique reduces the understanding of customer’s requirements up to maximum level. In case of any ambiguity in requirements they can solve the problems immediately and therefore reduce the time to take decisions [15]. In an opposite situation, the customers and development team members in GSD work in distant places from each other. This distance creates lots of problems in development, e.g., communication, language, time zone, etc. In the case of changes, modifications, or clarification of any requirement, it is not very easy to have an agile and effective action. Therefore, one cannot pretend the goal of agility in GSD except if the agile development processes are used separately at the nodes of the software development team.

International Standard Organization (ISO) also correlates software quality with the requirements and defines the software quality assurance activity [16] as a set of activities designed to evaluate the processes by which the software products are developed in order to satisfy stated or implicit requirements. IEEE [17] also focuses in satisfying requirements for Software Quality Assurance (SQA). It provides uniform and minimum acceptable requirements for preparation and contents of SQA plans [18]. These standards emphasize that satisfying requirements is the prime objective in quality assurance. On the other hand, the general problems involved in GSD (e.g., languages, cultural and time zone differences and lack of proper knowledge management) are responsible of arising difficulties in requirement engineering tasks. Further, the physical distance between stakeholders and group members limit their face-to-face interaction and thus create special challenges in the communication. As a consequence, this directly reduces the quality of requirements and therefore the quality of the product [19].

Coordination and control of the software development sites, distributed all around the world is also a major challenge in GSD. Quality of the development process is highly affected by the lack of proper coordination amongst the team members. There are several issues, which make this job as one of the most challenging in GSD. For example, ambiguities in understanding the organizational processes, the management practices, the requirements and/or the design, may arise in some of the distributed teams [20]. If these ambiguities are not solved in time it may cause long delays, leaving teams idle and frustrated, and reducing final quality of the project. Furthermore threats, threat of opportunism, security [21], trust, cultural issues and languages barrier are other factors which make the coordination management more difficult and challenging. The communication issues (e.g., distance, time zone difference, infrastructure support, distinct backgrounds, and lack of informal communication) may cause the loss of control over the teams located far away. If the company appoints a manager at the remote site and manager is not from their own vicinity, the manager might face a lot of problems. The employees behave as a "loose cannon" or shows excessive defensiveness or negativity [22]. The employees also feel these managers as a micromanager or 'put on the spot'. In fact these reflections of employees towards the manager may occur due to differences in intercultural factors [23].

One major issue, which is responsible for controlling the quality of product, is the criteria for the distribution of the tasks amongst several companies or teams involved in the development of single project. So deciding which task is given to which company and what should be the criteria for selection of those companies distributed around the world is a difficult problem to be solved. However, the answer of the question might be very simple. Based on the expertise of the company or the developers, one manager can decide. But, is this sufficient? For example, if one outsourcing company is in India then should the testing be performed there or should be at customers site? [24] These are examples of questions that, if not properly solved, would directly affect the low

quality of the product. Further, less obvious issues in task assignments also influence the quality of product. For example, an assignment decision may fail due to a high staff turnover rate. If people leave the company every year, then you get no knowledge and no return on investment.

Cross cultural issues [23], [25], [26] impact heavily in achieving quality objectives. MacGregor et al. [24] identified that intercultural factors [27] manifest themselves on a variety of levels in technical professions, all of which potentially impact the success of a project in GSD. The authors [24] gave examples of several researchers' works and argued that the intercultural factors impact very seriously in GSD projects. Neglecting these factors may cause low quality of process/development and, therefore may cause the failure of the project. Several intercultural factors were identified, e.g., need for cross-cultural sensitivity [28], communication challenges in mediated communication [29], difficulty with planning and management of global innovation [30], differences in work-style [31], and power, hierarchy and agency [32] These all factors directly or indirectly impact on the quality of process and projects. In a case study of a GSD project distributed in three continents: North Europe, Asia and South America. Gibbs [33] observed that the global team was loosely coupled due to team members' multiple cultural identifications, geographical dispersion, time differences and electronic rather than face-to face communication [33]. The loose coupling may be advantageous up to some extent but it seriously affects on the quality of process and product.

If we organized all the above factors, which are responsible for quality issues in development process and final product, we can conclude that the quality is affected by two major challenges: Requirement and Coordination. The factors affecting these two challenges can be summarized as follows:

Requirements challenges:

1. Communication: affected by several sub factors:
 - 1.1 Distance between stockholders and team members.
 - 1.2 Languages ([34]).
 - 1.3 Culture and time zone differences [34].
2. Ambiguity in understanding the requirements.

Coordination Challenges:

Coordination challenges are affected by the following factors:

1. Communication [35][36].
2. Lack of Trust [37].
3. Intercultural issues [38].
4. Work allocation assignments [24].
5. Ambiguity in understanding:
 - a. Organizational process,
 - b. Management practices
 - c. Requirement and design
6. Project planning and follow up [30]
7. Loss of control

IV. POSSIBLE SOLUTIONS

In the previous sections, we have shown the factors which are responsible for increasing the complexity in achieving the quality in both process and product in GSD. In this section, we are suggesting possible solutions, which are extracted from various sources. In fact, the different solutions available in literature for the problems of GSD are not normally concentrated on the quality aspect. In this paper, we have concentrated only on those ones which impact on quality aspects of GSD. We have identified them and then unified and presented as possible solutions for quality issues in GSD. Further, it is observed that the quality of a software product is impacted by most of the problems of GSD. However, we do not find any paper in which all those issues are unified for the evaluation of both the quality of process and product. The following list is the results of our work:

1. QA strategies should be applied: To control the quality in software process and product in GSD environment, these QA strategies should include a quality management network, change control, active risk management, quality audits, inspection strategy, delivery strategy and reporting and measurement [4]. The authors have applied these practices in GSD environment and tested on a real project [4] and found that these are very effective in achieving quality in both process and product.

2. Well defined process. The type of development processes should be clearly defined. Requirements should be presented in such a way that it can be understood easily. One of the ways for clear understanding of requirement is modeling. Furthermore, the architecture design and its dependencies should also be elaborated with full care. Components and their interfaces should be well and precisely defined. Assignment of the task should be clear-cut to independent teams. There should not be any ambiguity in the task assignment. At remote sites, some particular person should be accountable for any query and conversations from outside.

The coordination and requirement challenges can be overcome adopting the following practices:

1. Proper planning and scheduling of all activities as a way to overcome the coordination and requirement problems [39][40].

2. Near shoring. One of the major problems in cross cultural environment is the behavior of people which varies from place to place and also increases the perception of lack of trust. Development should be done as near to the customers as possible i.e., it should be in close proximity of the parent company. It will reduce the cultural and language related problems. In this way, we can also solve the problems related to requirement challenges. Due to common/similar languages and close time zone, the development team can interact with customer(s) not only more frequently but also understanding the customers' requirements and problems in more fruitful way. Some of the typical examples for near shoring are: South Korea, Eastern European countries, Middle and South American countries as recommendable places for outsourcing for China, West European countries,

and US respectively [41][42]. The sites in the same time zone are better options [34]. Of course, this may also be solved by extending the working hours at the different sites [43].

3. Strategic choice. Only those components which are not cultural sensitive should be outsourced reducing the risk of cross-cultural problems [25]; e.g., middleware or a component to be embedded in an Operating System (OS). In fact, the software for the middleware and for OS are not strongly affected by the surrounding environment and possible cultural issues because generally independent software and the specifications are clear from the beginning. Another suggestion was to outsource only those projects whose benefits are expected to outweigh the risks [23].

4. Transfer of knowledge and people. In [25], it is suggested that acceptance of the project in distributed environments should be made on the basis of some transfer of knowledge in exchange for the cultural risks. The quality suffers, if the staff at remote sites is not knowledgeable or expert. The authors also suggested that effective in-depth working relationships should be achieved amongst the development team throughout the project. Another suggestion for reducing the negative impact of intercultural factors is to create a "negotiated culture" [3]. Furthermore, a small number of the staff members should also be trained in other sites language [34]. Some staffs should also be exchanged for short periods of time. This practice will dramatically improve the working environment [11].

5. Synchronization in organization, processes, or technology in all the locations of the project is an effective technique for improving collaboration [44].

V. DISCUSSION

The acceptance of outsourcing in software development is increasing very rapidly. Only in India, Outsourcing to India has been increased 10 times in 10 years (1998-2007) and it is expected to increase on the annual rate of 28 percent for IT projects [45]. Additionally, not only outsourcing but global in-sourcing (i.e., development with own subsidiaries in foreign countries [24]) is also growing rapidly. For example, IBM has increased its staff in India from 53,000 to 73,000 in just one year in 2007 [46]. Naturally, if the rate of outsourcing is increasing to fulfill the demand of market, several new companies are also entering to the software market. However, it is not easy to control the quality at remote sites. Besides several other reasons, evaluating the capability and the talents at remote sites [47] are challenging job [47]. Additionally, it is not unusual to observe the lack of experts at the remote site: even they exist, it is usual that the number is not sufficient to fulfill what it is required. It is observed through surveys and studies, it is observed that these are the reasons why; the work assigned at distributed sites normally takes significantly longer time than what it is usual in other environments [35]. Again, the challenges for the requirements management and the project coordination are the causes. Further, several factors which contribute to these challenges (e.g., communication [35] [36], lack of trust [27], intercultural factors, lack of qualified and experienced professionals etc.) reduce the quality of the process and

product. These all factors increase the costs and reduce the probability of success.

Many works have been carried out by different researchers (e.g., [48], [49], [50]) to identify the problems, issues or challenges and solutions for GSD. If we analyze the majority of the available literature, we can easily find out that, only few of them have given emphasis on the quality aspect of GSD. By keeping all these issues, we conclude that there is still a need of investing more thinking and work to establishing procedures and methods for achieving the quality product in distributed environments.

VI. CONCLUSIONS

In this paper, we have focused more particularly into quality aspect of the product in global distributed environments. The quality of a software product developed in GSD environment is heavily impacted by problems of this type of environments. We have identified two major challenges responsible of causing most of the problems: requirements management and project coordination. Several issues and several factors which are responsible for these two challenges have been also suggested. We have shown how these challenges finally affect the quality of the product. Since all mentioned issues can become serious obstacles to quality of software products and they are not properly researched yet in point of view for quality aspects in GSD, our work may create attract the attention of the academic community towards this important issue. In this sense we consider this paper as a valuable contribution to the software engineering community.

REFERENCES

- [1] D.W. Karolak, *Global Software Development: Managing Virtual Teams and Environments*, New York: Wiley-IEEE Computer Society Press, 1998.
- [2] C. Bartelt, M. Broy, C., Herrmann, E., Knauss, M., Kuhrmann, A., Rausch, B., Rumpe, and K. Schneider, "Orchestration of Global Software Engineering Projects - Position Paper" Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering (ICGSE '09). IEEE Computer Society, Washington, DC, USA, 2009, pp. 332-337.
- [3] G.M. Olson and J.S. Olson "Distance Matters" *Human-Computer Interaction*, 15, 2000, pp. 139-178.
- [4] M. Ivček and T. Galinac, "Aspects of Quality Assurance in Global Software Development Organization", Proceedings of the 27th International Conference on Telecommunications and Information of the 31th International Convention MIPRO 2008, 2008, pp. 150-155.
- [5] O., Gotel, V., Kulkarni, M., Say, C. Scharff, and T. Sunetnanta, "Quality Indicators On Global Software Development Projects: Does 'Getting To Know You' Really Matter?" Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering (ICGSE '09), 2009, pp. 3-7.
- [6] IEEE, IEEE Std 610.12-1990-IEEE Standards Glossary of Software Engineering Terminology, IEEE Computer Society, 1991.
- [7] D. Galin, *Software Quality Assurance, From theory to implementation*, Pearson Addison Wesley, 2004.
- [8] P.B. Crosby, *Quality is free*, New York: McGraw-Hill, 1979.
- [9] J.M. Juran, *Juran's Handbook Quality control*, New York, McGraw Hill, 1988.
- [10] R.S. Pressman, *Software Engineering*, NewYork: McGraw-Hill, 2008.
- [11] J.D. Herbsleb, "Global Software Engineering: The Future of Socio-technical Coordination". Proceedings of the 2007 Future of Software Engineering (FOSE '07), 2007, pp. 188-198.
- [12] M., Lormans, H., van Dijk, A., van Deursen, E. Nöcker, and A. de Zeeuw, "Managing Evolving Requirements In An Outsourcing Context: An Industrial Experience Report". Proceedings of the Int. Workshop on Principles of Software Evolution (IWPS'E04), 2004 , pp. 149-158.
- [13] M. M. Lehman, "Software's future: Managing evolution" *IEEE Software*, vol. 15, 1998, pp. 40-44.
- [14] J., García Guzmán, J., Saldaña Ramos, A., Amescua Seco and A. Sanz Esteban, "Success Factors for the Management of Global Virtual Teams for Software Development", *International Journal of Human Capital and Information Technology Professionals*, vol. 2, 2011, pp. 48-59.
- [15] A. Cockburn, and J., Highsmith, "Agile Software Development: The People Factor", *Computer*, vol. 34, November 2001, pp. 1-3
- [16] ISO, ISO/IEC 9126-1Software Engineering - Product quality - Part 1: Quality model, Geneva: International Standards Organization, 1997.
- [17] IEEE, IEEE std. 730-2002 Standard for Software Quality Assurance Plans, Washington: IEEE Computer Society, 2002.
- [18] A. Abram and J. W. Moore, *Guide to Software Engineering*, Body of Knowledge, Washington: IEEE Computer Society, 2004..
- [19] R. Ahmad, *Analysing Requirements Issues in Global Software Development*, Dissertation, Honours Programme of the School of Computer Science and Software Engineering, The University of Western Australia, 2010.
- [20] R., Sangwan, M., Bass, N., Mullick, D.J.Paulish, and J. Kazmeier, *Global Software Development Handbook*, Boston: Auerbach Publications, 2007.
- [21] P. Mohagheghi, *Global Software Development, Inssue , Solution And Challenges*, Trial lecture, Dept. Computer and Information Science (IDI), University of Science and Technology (NTNU),Trondheim, Norway, 2004.
- [22] L. Laroche, *Managing Cultural Diversity In Technical Professions*. Burlington: Butterworth-Heinemann, 2002..
- [23] G. Hofstede, *Culture's consequences*. 2nd Ed., Thousand Oaks: Sage, 2001.
- [24] A. Lamersdorf, *Towards a Global Software Development Distribution Model: Empirically-based Model Building for Distributed Software Development*, Master thesis, University of Kaiserslautern, 2008.
- [25] S, Krishna, S. Sahay, and G. Walsham, "Managing cross-cultural issues in global software outsourcing", *Communications of the ACM*, vol. 47, 2004, pp. 62-66.
- [26] G. Borchers, "The software engineering impacts of cultural factors onmulti-cultural software development teams", *Proc. of 25th Intl.Conference on SW Engineering,, 2003*, pp. 540-545.
- [27] P. Kruchten, "Analyzing intercultural factors affecting global software development – A position paper", *Proceedings of GSD2004 3rd International Workshop on Global Software Development*, 2004, pp. 59-62.
- [28] M. L. Maznevski and K. M. Chudoba, "Bridging Space Over Time:Global Virtual Team Dynamics and Effectiveness", *Organization Science*, vol. 11, 2000, pp. 473-492.
- [29] S. Whittaker, "Theories and Methods In Mediated Communication" In *The Handbook Of Discourse Processes*, Graesser A.C., Gernsbacher M.A. and Goldman S.R. (eds), LEA, Mahwah, NJ. 2003, pp. 243-286.
- [30] P. Banerjee, "Narration, Discourse and Dialogue: Issues in theManagement of Intercultural Innovation", *AI & Society*, vol. 17, 2003, pp. 207-224.
- [31] G. Walsham, *Globalization and ICTs: Working across cultures*, WP 8/2001 Research Papers In Management Studies, University of Cambridge, 2001.

- [32] B.Nicholson, and S. Sahay, Some Political And Cultural Issues In The Globalisation Of Software Development: Case Experience From Britain and India, *Information and Organization*, vol. 11, 2001, pp. 25-43.
- [33] J. L. Gibbs, Loose coupling in global teams: tracing the contours of cultural complexity, Ph. D. dissertation, University of Southern California, Los Angeles, 2002.
- [34] E. Carmel, and R. Agarwal, "Tactical Approaches for Alleviating Distance in Global Software Development.", *IEEE Software*, vol. 18, 2001, pp. 22-29.
- [35] J.D., Herbsleb, A., Mockus, T.A. Finholt, and R.E. Grinter, "An Empirical Study Of Global Software Development: Distance and Speed" Proceedings, International Conference on Software Engineering, Toronto, Canada, 2001, pp. 81-90.
- [36] J.D. Herbsleb and A. Mockus, "An Empirical Study of Speed and Communication in Globally-Distributed Software Development". *IEEE Transactions on Software Engineering*, vol. 29, 2003, pp. 481-494.
- [37] N. B. Moe and D. Smite, "Understanding a Lack of Trust in Global Software Teams: A Multiple-Case Study", *Software Process*, vol. 13, 2008, pp. 217-231.
- [38] G. Hofstede, *Culture and organization – software of the mind*. 3rd Ed., New York: McGraw-Hill, 2010.
- [39] H. Kerzner, *Project Management: A System approach to Planning, Scheduling and Controlling*, New York: John Wiley & Sons, 2003.
- [40] PMI, *PMBOOKA Guide to the Project Management Body of Knowledge (PMBOK Guide)*, Pennsylvania: Project Management Institute, 2004.
- [41] D. Dubie, "Outsourcing Moves Closer to Home", *CIO Today*, December 18th, 2007.
- [42] C. Zarley, "7 Reasons the Bloom is Off Asian Outsourcing". *The Channel Wire*, December 12th, 2007.
- [43] G. Lee, W. DeLone, and J. A. Espinosa, "Ambidextrous Coping Strategies In Globally Distributed Software Development Projects", *Communications of the ACM*, vol. 49, 2006, pp. 35-40.
- [44] R., Heeks, S., Krishna, B. Nicholson, and S. Sahay, "Synching or Sinking: Global Software Outsourcing Relationships". *IEEE Software*, vol. 18, 2001, pp. 54-60.
- [45] M. Kobayashi-Hillary, "India faces battle for outsourcing" *BBC news*, August 17th, 2007, <http://news.bbc.co.uk/2/hi/business/6944583.stm>, Last accessed August 2011.
- [46] J. Ribeiro, "IBM expects \$1 billion in India revenue this year", *InfoWorld*, December 17th, 2007.
- [47] W. Kobitzsch, H.D. Rombach, and R.L. Feldmann, "Outsourcing in India", *IEEE Software*, vol. 18, 2001, pp. 78-86.
- [48] J.D. Herbsleb and D. Moitra, "Global software development", *IEEE Software*, vol.18, 2001, pp.16-20.
- [49] J. D. Herbsleb, D. J. Paulish, and M. Bass, "Global software development at siemens: experience from nine projects", *Proceedings of the 27th international conference on Software engineering (ICSE '05)*. 2005, pp. 524-533.
- [50] R. Prikladnicki, J.L. Nicolas Audy and, R. Evaristo, "Global software development in practice: lessons learned". *Software Process: Improvement and Practice*, vol. 8, 2003, pp. 267-281.

A Systematic Review of Self-adaptation in Service-oriented Architectures

M. Pilar Romay

Dept. Inf. & Com. Syst. Engineering (DISIT)
St. Paul-CEU University
Boadilla del Monte, Madrid, Spain
pilar.romayrodriguez@ceu.es

Luis Fernández-Sanz, Daniel Rodríguez

Dept. of Computer Science
University of Alcalá (UAH)
Alcalá de Henares, Madrid, Spain
luis.fernandezs@uah.es, daniel.rodriguezg@uah.es

Abstract—The study of adaptivity, i.e., the capability to react to changes in the environment, is becoming ever more important in many fields of study, and in the development of software in particular. This paper presents a systematic review in which both the extension and complexity of this notion are examined. After studying the influence from external fields, this review checks the hypothesis of using the scope of service-oriented architecture as a comparable model for the whole field. As part of the systematic review, the influence of the most relevant bibliography is considered, and the terminology is clarified.

Keywords – *adaptivity; self-adaptation; service architecture; autonomic systems; SOA; systematic review.*

I. INTRODUCTION

The growing complexity, along with continuous operation, of software systems –not only conventional ones, but also the next complexity level, so-called *large-scale software systems* [1] – has greatly increased the interest of a series of techniques for self-managing system features. These techniques make possible for them to guarantee a wide range of properties, all by themselves. Traditionally, these properties had been dealt with manually, or had to be developed from specific requirements. Instead of that, the new approach considers them as *intrinsic* system properties, and thus they should be dealt with automatically, and considered as just another issue in conventional software systems development.

Systems conceived in such a way are generically known as *adaptive systems* or, more specifically, as *self-adaptive systems* [2].

Therefore, we have systems able to deal with faults and critical situations (*self-healing*), able to control their own behavior (*self-managing*) [3], able to observe and evaluate their own performance (*self-monitoring*), able to modify their own configuration to react to changes in their environment (*self-configuring*), or even to automatically guarantee certain system-level properties, such as protection, fault tolerance, etc. (*autonomic systems*) [4] [5], among many others.

The wide range of systems which could make use of these *adaptive properties* causes a great variability; therefore many different approaches could be conceived. For this reason, it is reasonable to focus our efforts on a specific area of study: in our case, *software services*. This area has been chosen because it still covers a wide range of systems and shows a great variability itself, and therefore it can be considered as a representative, even a lower-scale analog, for

the whole of the field of adaptive systems. The goal of this work is, therefore, to study *self-adaptive software services*.

Software services define, due to their own properties, an area of a great potential to describe and use adaptive (or adaptation-related) features. Moreover, *service* and *service-oriented architectures* are among the systems where the need for these features is clearer, and more compelling: the nature of services is inherently dynamic, and this implies the need for adaptation; and also the structure of service architectures requires the flexibility that self-adaptation provides. In short, this makes our specific goal (consider adaptation in services, rather than in general systems) even more pragmatic. Finally, considering the growing, relevance and broad dissemination of *service ecosystems*, this is also the environment in which this approach is currently pertinent and more interesting.

Adaptivity is often described at different levels, namely at *service* level or the wider *system* level [6], but this will not be the main interest of our study. Instead of that, we will focus on exploring and analyzing adaptivity and all its related properties, a set which is often generically known as *self-**.

Moreover, this paper will also consider the impact of *self-organization* (considered as a related notion, rather than as an adaptive feature) within the specific area of service-oriented architecture. Our main goal is to determine which properties are implied in adaptive systems, with a special focus on service architectures – i.e., to be able to *evaluate adaptivity* in services.

For this purpose, this paper presents an initial study of the field, which will be used to delimit the boundaries of the area and to check the reliability of the hypothesis about the service-oriented approach and its applicability to evaluation. The core of this study is structured as a systematic review: after defining a set of goals and the corresponding research questions, and discussing the background on the field, the review makes an extensive bibliography review, which is carefully examined and analyzed in order to achieve the corresponding conclusions.

The paper is structured as follows: first, we present the context of our study, including the definition of four primary goals and the method of our systematic review. Then, we provide some background justifying the interest of this study, as well as the implicit connections between its areas. After that, we characterize the revised information, and outline the method we have used to locate and classify this information, describing the performed searches and their results. We end by summarizing the conclusions from several perspectives.

II. CONTEXT OF THE STUDY

Though it might seem a secondary issue, the relevance of adaptivity is such that even well-known authors as Kramer & Magee have claimed [7] that “a significant advance in the techniques which are required for the effective development of adaptive systems would imply an advance of an order of magnitude in every fundamental aspect of Software Engineering”.

Having this relevance in mind, the main goal which has driven the conception of this study is focused in finding a *model* which makes possible, by using a set of attributes, to define and assess *adaptivity* in the context of *services*. This model could alternatively take the form of a framework, or even a methodology.

Then, this paper intends to provide a characterization of the field of adaptivity. For this purpose, it lays out a set of specific *goals* to drive the study, which should make possible to measure and digest the breadth of the field, and to confirm the need of the study itself. Moreover, it also intends to value the most important contributions in the process.

From these goals, the paper follows a methodological approach based on [8] with the purpose to achieve a greater soundness than a traditional narrative description. The more important limitations of such a study are also considered: *publication limitations* (publishing bias), and *selection limitations* (selection bias). The first one refers to the relative impact of *negative* studies –i.e. which have not significant differences with previous proposals–, when these have not been published, or are only rarely referenced in the literature. Also, there could be interesting studies written in another language, or even duplicate references which could later influence the *metanalysis*. The second one is related to the definition of inclusion and exclusion criteria: the purpose is to avoid to neglect the inclusion of relevant work, and also to include misleading papers, which hinder dealing with the relevant topics in an objective way.

The systematic review will be developed in the next sections. First, we will describe the main goals of the study, and outline our methodological approach. Next, we will briefly explore the background on service-oriented architectures and their relationship to adaptivity, focusing on the need for evaluation and the influence of dynamic service composition models. Then the systematic review itself is unfolded: after presenting the main data sources, the search strategies and selection criteria are described – to later present the results of the review and discuss the conclusions.

A. Goals of the Study

Therefore, the goals of our systematic study are: (1) To confirm the breadth and applicability range of adaptivity. (2) To verify the novelty of this field of study within the context of Software Engineering. (3a) Related to the previous one, to evaluate adaptivity in service-oriented architectures. (3b) To assess interesting contributions which could be applied to the study’s primary goal (i.e., to determine the properties which characterize adaptive systems). (4) By exploring the previous four points, to identify the used terminology.

B. Methodological Approach in the Study

This (systematic) study begins by planning the review, then conducting the review, and finally reporting the review. The first activity of this process is a bibliographic search. Based on a set of *research questions* related to context definition, modelling, and management, we defined a list of *keywords* and search strings used for our investigation.

The defined searches will be oriented to cover the goals of the study, as proposed in section II.A. In this part of the process, the selected keywords and their synonyms are of a great relevance: the obtained results strongly depend on a good selection of these terms.

For this reason, we also designed and realized an specific search, focusing on articles and papers which tried to provide a wider vision of the field, such as (other) research reviews, overviews, state-of-the-art articles, etc.

The initial terminology search should just be considered as an approximation, and it will be later tuned and adjusted, to be refined by means of the obtained results during all the process. To some extent, the process itself serves as the main *control* in this initial search phase, and it could cause an additional iteration within the systematic review process – it just depends on the actual extension and variability of the terminology in the field.

After the search, we proceed to select and evaluate the obtained information. For this purpose, as already noted, the study defines acceptance and rejection criteria related to its specific goals, and in particular to the main goal – which was the reason to do the study, in the first place.

To finish, the obtained results will be analyzed and interpreted. In this phase, the process will make possible to synthesize the results with regard to the proposed goals.

III. BACKGROUND: ADAPTIVITY IN SERVICE-ORIENTED ARCHITECTURES

Nowadays, the notions of service orientation (or *service-oriented computing*, SOC) [9] and service-oriented architectures (SOA) have been totally integrated in the current conception of software. This is the reason why they define a perfect workbench to assess adaptive properties in generic software systems.

Therefore, this section reviews the context of work in both fields, focusing in the assessment of adaptivity, and service-oriented architectures.

A. Adaptivity Assessment & Evaluation

Adaptive systems can be defined as “systems able to react to automatically adapt themselves to changes in their environment”. This reaction can be specifically programmed, or could rise from an *emergent* behavior. This category of systems has been globally designated with the name of *self-* systems* [10], which explicitly refers to the variability of the concrete aspect to consider. However, in recent times most authors prefer to designate them with the generic name of *adaptive* (or *self-adaptive*) *systems*, like this paper did also in the Introduction.

Adaptivity is a complex field of study. First, because the term has explicitly been conceived to be generic, so we must first decide which specific feature (“attribute”) are we going to consider every time. And second, because too often we lack a clear reference model which could serve as the basis to compare to the system’s degree of adaptivity. Therefore, it is necessary to have some kind of *model* to make possible to assess those capabilities, either quantitatively (in the ideal case) or at least roughly, by approximation.

In general, the development of adaptive systems, as well as the more concrete development of *adaptive services*, lacks a clear set of methods and metrics able to assess the actual capabilities of a specific implementation. That is, it is really difficult to even decide if a given system is “adaptive” or not. In fact, this particular distinction is almost intuitive; however the increasing importance of this features, and the difficulties in their implementation, highlight the relevance of achieving the definition of a quantitative approach, able to deal with concrete values. For this reason, one of the main goals of this study focuses in checking existing references, which enable or guide the process to obtain either the model or relevant metrics, to be able to assess the level of adaptivity, even in a qualitative way.

B. Service-Oriented Architecture: Service Composition Mechanisms

A well-known definition of service-oriented architecture (SOA), as given by Michael Papazoglou [11] states that it is “a *meta-architectural style*, based in loosely coupled services, which provides flexibility to business processes in an interoperable way, and independently from the technology”. Therefore, its main goal is *interoperability*, which is itself a consequence of loose coupling.

However, a standard definition of SOA is still debated, in spite of the popularity of the term – probably because it has been used with different meanings in different contexts, and referring to different technological aspects. Beyond those details which distinguish the many variants of the concept of *service* (web services, RESTful services, grid systems, etc.), there are still several intrinsic features in its definition. These features imply that service-oriented architectures are *a priori* more dynamic and flexible than many “traditional” ones, in particular component-based architectures – and this can be considered inherent to its own nature.

From this point of view, it is interesting to note at least two of these features, which suggest this kind of architecture as a good evaluation workbench for adaptivity:

1) *External Composition Mechanisms*. First, services are always part of a modular system – they are conceived to be used as part of a larger structure. However, there is a subtle difference to more traditional approaches: service systems are designed to be composed at runtime.

Services cannot assume anything about the rest of the elements in the composition. First, their interface is separated from the rest of the service, and therefore services never interact *directly* to the rest of the system. Second, they are not designed as part of a concrete compound: instead of that, once they are implemented and deployed, they are included in some composite system, which was *later* conceived.

These are the reasons why the well-known composition models for services (choreography and orchestration) must be conceived as external compositions. Thus a service does not even need to know if it is contained in a composite: the business logic (the “intelligence” of the system) belongs in the structure itself, not in its individual components. Within an *orchestration*, it is in the orchestrator; but *choreographies* are even more complex, as the composition schema is distributed along the composite – i.e. it is *decentralized*. Every individual service receives just a “local” subset of instructions, without a perspective of the global plan. Even *service mashups*, a promising approach, are again an external composition model – in fact, essentially an orchestration. Another consequence is *crosscutting*. Unlike traditional composition, service models do not preclude that *the same* service is simultaneously a part of more than *one* composite. This implies that every service composition is *orthogonal* to any other which is performed later [6].

2) *Intrinsically Open Architecture*. Of course, many existing systems, and distributed systems in particular, have claimed to define an open architecture. In practice, an *open system* is every system which, by defining or using an standard interface, is able to compose any external element defined as a client of that interface. However, if constraints imposed by this interface are too strict, the limits they define hinder the capture of information about the different clients – i.e. it would present an homogeneous architecture, which is exactly the opposite of our goal.

Services use a different approach: the interface is defined at the beginning, to offer a concrete functionality (a service), and to guarantee a certain quality level (i.e. QoS). But apart from that, services are conceived, even at the technical level, to be composed to *any* other element able to interact to them. Therefore, they are presented as the ultimate *open* system: in the specific case of RESTful web services, for instance, the only actual constraint is the use of the HTTP protocol, which was conceived using the REST architectural style itself - and this is not an actual constraint, nowadays.

Also, we have to consider that the current evolution of service systems has a clear trend towards a significant rise of the *scale*. The original “XML web services” were in general small modules, of a scale comparable to that of objects, or even smaller. Currently, the concept is clearly shifting to be equivalent to so-called *Software as a Service* (SaaS) – where the scale of a service is similar to that of a complete application. In fact, the approach itself is evolving from the potential provided by a concrete technology which focused on interoperability, to the design of a new, generic software distribution model (shifting from “product” to “service”).

In any case, current *service-oriented architectures*, when this term is understood in the wider sense [12], present the same features of flexible and open composition we have already noted – and this makes them adequate as a workbench for *adaptivity evaluation* in software systems.

IV. REVISED INFORMATION & METHODOLOGY

This study is based on information obtained from several digital bibliography search engines. Specifically, we have

used search engines from the best known and most widely recognized publishers in the fields of Computer Science & Information Technology, as well as Google Scholar.

To have a preliminary structuring of the area, we first considered the results provided by Google Scholar. The goal was to assess the research activity on *adaptive systems* in the period 2000-2011, including every potential environment, and comparing these results to those in the specific subarea of service-oriented architectures in section A .

The remaining searches followed a more systematic approach, guided by specific goals (in the form of questions), specifically those which were proposed in section II.A.

Throughout all this search process, we have considered the possibility of evaluating the used terminology, with the purpose of extending the search to a wider scope – but still within the parameters of the study. This evaluation has made possible to change and evolve the initial searches, to the final form we will describe in the following.

After performing those search processes, our inclusion and exclusion criteria were used to select the most relevant articles. Then we also examined the references cited in these papers, with the purpose to select other relevant papers, which were not located previously due to their publication stage, or which have been published by some additional publisher. This way, the publishing bias we mentioned in section II.B.

A. Search Strategy and Selection of Areas

The search strategy has been guided by our goals, by answering to a set of questions.

The questions were bound to specific terminology. The variety of meanings of some of the terms used in our search made necessary to apply an iterative, evolutionary approach, in which those search terms were finely tuned. At the end of the process, our study has made possible to obtain a specific terminology summary, which covers goal (4). This specific terminology, obtained from multiple sources in the revised information, has been represented using a pyramidal mesh, which will be detailed in section IV.C. Therefore the most significant terms and notions related to our field of study have been collected, also emphasizing their similarities and differences, something which is not always completely clear. This way, in our iterative process we have refined concepts such as *autonomic* vs. *autonomous*, *adaptation* vs. *self-adaptation*, *adaptive*, *self-organization*, *self-monitoring*, etc.

The definition of these terms, as part of the results for our goal (4), is briefly explained in section IV.C, where it also explains the aforementioned pyramidal structure.

Within these terms, we should emphasize those which were considered for our search, namely:

- Adaptation, adaptive, adaptivity, self-*
- “Software service”, service-oriented, SOA
- Evaluation, “quality model”

In order to fulfill our first and second goals, we performed a series of searches on Google Scholar, as well as other databases. In the final search on Scholar, the questions related to these goals were the following:

- Assessment of the number of articles dealing with adaptivity, against the number of those doing the same in the service-oriented architecture area.
- Which disciplines (research areas) are dealing with and applying adaptation?

The first search, which intends to identify the different fields of study related to adaptivity, is driven by the following queries, referring to the compared subsets:

- Query #1: (“autonomic” OR adaptive OR adaptation OR autonomous OR adaptivity OR self) AND (evaluation OR quality)
- Query #2: (“autonomic computing” OR adaptive OR adaptation OR autonomous OR adaptivity OR self) AND (evaluation OR quality) AND (“software service”) OR (“service-oriented Architecture”) OR (“Service Oriented Architecture”))

These queries, on the Google Scholar engine, resulted in about 7.806.600 references for query #1 and a total of 17.565 for query #2. The refined search provides roughly about 1500 results every year, from 2000 to 2011. The scope of the study is very wide, covering almost any scientific area – which is not surprising and confirms our intuition.

2011	2010	2009	2008	2007	2006	2005	2004	2003	2002	2001	2000
770	2980	3180	3050	2630	1980	1410	668	379	243	189	86

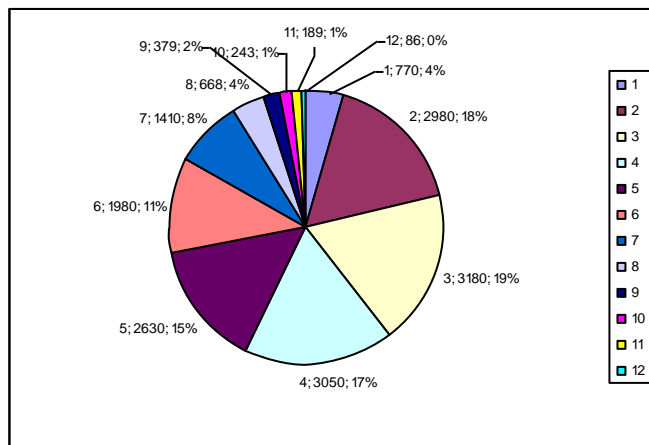


Figure 1. References for Query #2 from 2000 to 2011 (1-12)

The most representative areas for query #1 are: Life Science, Engineering, Social Science and Law, Mathematics and Statistics, Medicine and Computer Science (e.g. Ubiquitous Computing, Grid Environments [13], mobile systems and services [14] [15], Domotics [16], etc.)

Goals numbered as (3) are essential in the context of this study – i.e. the evaluation of adaptivity in service-oriented architectures. Related searches have been more specific, and they have already been performed in bibliography databases from the publishers themselves. The purpose was to obtain a more accurate list of articles, trying to reach all the relevant information – without any accidental loss. We also have used

references from the selected articles, and in relevant cases we have also searched for the corresponding citations.

For instance, a representative query could be:

- Query #3: (((“autonomic computing” or self) and (adaptive or adaptation or adaptivity)) and (evaluation or quality)) and (“software service”) or (“service-oriented Architecture”) or (“Service Oriented Architecture”))

B. Inclusion and Exclusion Criteria.

1) *Related to goals (1) and (2):* Neither inclusion nor exclusion criteria were defined – this search was delimited just by query clauses themselves, i.e. queries #1 and #2. This could seem less “systematic” than the remainder of the study. However, we did not intend to do a detailed classification of areas and fields of study, but to assess if our suggestion (to focus on service architectures) was reasonable. This goal alone could be used to justify a specific systematic study, which would be even more complex than the one presented here. The reason to include this goal is to perform a shallow examination of some of the areas suggested by many search engines, with the purpose of perceiving the actual extension of the field, as well as its growing rate. A systematic study on this specific aspect would be of great interest to detect methods or tools (from other fields) which could be applied in the context of adaptive software.

2) *Related to goals numbered as (3).* In this search process, queries are quite more specific, and they mainly focus in evaluating adaptivity by means of self-properties. Therefore it considers papers including models, frameworks, metrics and evaluations on the topic. This study excluded papers not dealing with self-properties, and those which did not focus on assessing adaptivity/autonomic features.

3) *Related to goal (4):* The resulting terminology has been extracted from papers selected in the previous phase. Therefore their inclusion and exclusion criteria are the same. However, some additional selection criteria are also added; specifically, articles which define or clarify terminological aspects, or which perform reviews in which terminological features are also clarified.

C. Results

1) *Related to goals (1) and (2):* The range and scope of the many fields of study which apply adaptivity is too wide to be considered in this paper – in fact, it would require a specific study itself. Therefore, for this purpose we refer to the results outlined in section IV.A, and to the conclusions summarized in sections V.B and V.C, which expose a global vision for this part of our study.

2) *Related to goal (3):* This goal, together with results about terminology from goal (4), provides a characterization of adaptivity. The following table summarizes briefly this part of the study. It describes representative categories of existing work, indicating for each one of them references, goals, projects, metrics and their organization.

TABLE I. EVALUATION OF ADAPTIVITY

Ref	Evaluation of Adaptivity	
	Goal/ Project/ Context	Metrics/ Organization
[28] [29]	Metrics to evaluate Self-* systems criteria / -- / Web-based C/S, E-learning (AHA!), Videoconference, Multiagent Systems	The many metrics for each Proprieties (reuse, genericity...)/methodological, architectural, intrinsic characteristic and runtime
[30]	Metrics for restarting strategies in WS Reliable Messaging (WSRM) / -- / WSRM	Effective Transmission Time (ETT _i), Unnecessary Resource Consumption (URC _i), Savings (SAV _i) / Adaptation parameters (structures, payoff, environments, time)
[34]	Quality Model for the software architecture of self-healing applications (based on ISO 9126) / Attribute-based architectural styles (ABAS) / --	Traditional quality attributes (Maintainability –Modifiability, Extensibility-, Reliability –Fault tolerance, Robustness-) Specific Autonomic Quality attributes (Support for detecting anomalous system behavior, Failure Diagnosis, Simulation of expected behavior, Differencing between expected and actual behavior, Testing of correct behavior). Autonomic Metrics: Detection ratio, Detection time, Fault Model Observability, Awareness, Coupling / Traditional and Autonomic attributes
[35]	User-level Quality of Service (QoS) (Context awareness) / PLASTIC, model PFM / Pervasive Networking Environment	Performance evaluation
[36]	Quality model to evaluate Self-* attributes (adopts 6 features of ISO 9126: Reliability, Efficiency, Maintainability, Usability, Functionality, Portability) / -- / --	The autonomic maturity of each level in complex software (Complexity of development, business domain and management) / Three-level Autonomic Evaluation Model (Software Complexity, Relative Quality Factor, Autonomic features). Fuzzy comprehensive evaluation (qualitative factors)

3) *Related to goal (4):* These results are summarized in Fig. 2, which shows the wide spectrum of so-called self-properties, ranging from very generic properties which can be applied in many systems (such as context-awareness) to specific attributes which are only found in some approaches (like emergence). Apart from these, there are several other, less frequent, properties – also, many of them are referred to using different names and variants (self-managing vs. self-management). All these issues have been considered in the study, and they are implicitly included in this paper.

Fig. 2 represents *three* pyramids rather than one – they are conceptually related, but they must be studied separately. Pyramid #1 represents *environmental adaptation*, i.e. the

capability of a system to perceive its own environment and integrate in it. Pyramid #2 represents *behavioral adaptation*, i.e. the capability of a system to modify its behavior to adapt to different conditions, ranging from pure observation to full self-management. And pyramid #3 depicts *self-adaptation*, i.e. the capability of the system to manage its own adaptivity, possibly including its own emergent behavior. Together, this triple representation describes the full range of *adaptation*.



Figure 2. The spectrum of self-properties: a pyramidal representation

This (triple) pyramid represents a *gradient*, rather than strict layers – i.e. each level is more complex than the one below itself (at least inside its own pyramid), but it is not necessarily using its services, though it is probably supported by some of the layers below. The same applies to the three pyramids – their separation depicts a gradient, but they can be considered independently. For example, an autonomic system is in the cusp of pyramid #2 – this means it is more complex than a self-healing system, but not necessarily that there is an emergent behavior (from pyramid #3) above it.

Therefore the pyramidal representation must not be understood literally – its purpose is to give an idea of their relative conceptual scope and size. As noted, some of these properties are built on top of the previous step (for instance, self-management should always rely on self-healing), but this is not always true (for instance, self-organization is not necessarily based on context adaptation).

This representation also helps to outline the distinction between similar but different terms: for instance, *adaptation* (i.e. the full range in the triple pyramid) vs. *self-adaptation* (i.e. just the range in pyramid #3). A similar conflict appears to differentiate *autonomous* (i.e. the capability of a system to act independently) from *autonomic* (understood here as the combination of several self-properties [4][22]). Indeed, there is an intimate relationship between adaptation and autonomy; though they describe different features, to fully achieve each one of them, the other is also required, at least partially.

V. CONCLUSIONS

We present our conclusions in the following, structured as the next four sections.

A. Adaptivity and self-properties

Regarding adaptation, there are significant differences in the way in which these autonomous changes in the system must be performed. This is mainly related to the way they are managed [17]. The range covers from the *ad hoc* way, in which adaptation (or the adaptors) needs the intercession of some stakeholder [18], to the *automatic* way, in which adaptation (and the adaptors) is fully generated by tools [19].

Self-organization can also be studied within the context of adaptivity [20], as we have already done in the previous section (Fig. 2). It should be considered nevertheless as an independent property, with the same level or complexity and interest than adaptation itself – of course, the same applies to the evaluation process [21]. This feature can also be considered in relation to several self-properties (such as self-adaptation or self-assembly, in particular), though it is more basic (and at the same time, can be more complex) than the majority of the properties listed in Figure 2. This reflection also requires a discussion of the terminology.

In many cases, the evaluation of adaptivity needs to have into account the specific context to deal with – some systems require to be adaptive even when their flexibility is minimal. This relative scale must also be considered.

B. Adaptivity in different areas

The wide scope of the field suggests that there could be methods and techniques designed for the evaluation of adaptivity [2] [4] [5] [22] which could be applied at the software architecture level. Several techniques have also been inspired in other fields, such as the Control Loop Model [2], and some others can still be transferred – much of them in the context of natural systems, in particular in the context of self-organization.

The growing relevance of this field is even more apparent in the context of “new” kinds of applications which are appearing right now and in the near future. An obvious example is adaptation in the context of *mobile systems*, where context-awareness, which includes a wide range of techniques, has been an active line of research.

C. Adaptivity in Software Engineering

An immediate conclusion, with respect to the field of software engineering, is that the evaluation and assessment of adaptivity is still a relatively new area. A review of the existing literature shows that there are still several aspects to define, such as languages or methods [23][24][25], etc. Once this is done, the quality of service (QoS) could be influenced by adaptivity, just like it is now by interoperability – this would be used as the criteria to select and use certain systems [17]; in summary, this could provide soundness to autonomous systems. There is already some amount of work in this direction, but these are still proposals under discussion, the first contributions which must be refined.

Adaptive systems also begin to be considered within the specific subfield of Requirements Engineering, for instance [26]. But, besides deciding when to adapt (adaptation time), we are also interested in the nature of adaptive capabilities, and how to define generic models which could determine our adaptive systems.

D. Evaluation of Adaptivity

In summary, we can conclude that currently there is not any effective method able to evaluate the adaptivity of a software system [27][28] [29][30][31][32][33][34][35][36] – not even when we refer to this property not in the wider sense, but focusing on a concrete feature.

Also, as deduced from section IV.B, the scope of service-oriented architecture is comparatively much smaller than the general scope of adaptivity. But while the size of the field has maintained constant, the importance of services has increased – therefore, we can conclude that our hypothesis is reasonable, and then, that adaptive services can be used as a model for generic adaptivity.

REFERENCES

- [1] L. Northrop, *Ultra Large-Scale Systems: The Software Challenge of the Future*, SEI Books, Software Engineering Institute, 2006.
- [2] B.H.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, *Software Engineering for Self-Adaptive Systems*, Lecture Notes in Computer Science 5525, Springer, 2009.
- [3] B.H.C. Cheng, *et al*, “Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2010),” 32nd International Conf. on Software Engineering (ICSE 2010), ACM Press, 2010, pp. 447-448.
- [4] J.O. Kephart and D.M. Chess, “The Vision of Autonomic Computing,” *IEEE Computer*, vol. 36, 2003, pp. 41-50.
- [5] M.C. Huebscher and J.A. McCann, “A Survey of Autonomic Computing -- Degrees, Models and Applications,” *ACM Computing Surveys*, vol. 40, 2008.
- [6] C.E. Cuesta and M.P. Romay, “Elements of Self-Adaptive Systems - A Decentralized Architectural Perspective,” *Self-Organizing Architectures*, Lecture Notes in Computer Science, vol. 6090: Springer, 2010, pp. 1-20.
- [7] J. Kramer and J. Magee, “Self-Managed Systems: an Architectural Challenge,” *Future of Software Engineering (FoSE 2007)*, IEEE Computer Society, 2007, pp. 259-268.
- [8] B. Kitchenham, “Procedures for performing systematic reviews,” *Joint Technical Report*, Keele University and National ICT, 2004.
- [9] M.P. Papazoglou and D. Georgakopoulos, “Introduction to the Special Issue on Service-Oriented Computing,” *Communications of the ACM*, vol. 46, 2003, pp. 24-28.
- [10] O. Babaoglu, M. Jelasity, *et al*, *Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations*, Lecture Notes in Computer Science 3460, Springer, 2005, pp. 1-20.
- [11] M.P. Papazoglou, *Web Services: Principles and Technology*, Prentice-Hall, 2007.
- [12] NEXOF Reference Architecture Specification, Version 1.0, 2010. <http://www.nexof-ra.eu/?q=node/695>. Last access: 07/15/2011.
- [13] D. Ardagna, S. Lucchini, R. Mirandola, and B. Pernici, “Web Services Composition in Autonomic Grid Environments,” *Business Process Management Workshop*, Springer, 2006, pp. 375-386.
- [14] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S.O. Hallstensen, J. Lorenzo, A. Mamelli, and U. Scholz, “MUSIC: Middleware support for self-adaptation in ubiquitous and service-oriented environments,” *Software Engineering for Self-Adaptive Systems*, Springer, Lecture Notes in Computer Science 5525, 2009, pp. 164-182.
- [15] G. Wrzesinska, J. Maassen, and H.E. Bal, “Self-adaptive applications on the grid,” *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, New York, NY, USA: ACM, 2007, pp. 121-129.
- [16] J. Ferreira, J. Leitão, and L. Rodrigues, “A-OSGi: A Framework to Support the Construction of Autonomic OSGi-Based Applications,” *Autonomic Computation and Communication Systems: Autonomics 2009*, Lecture Notes of the ICST 23, Springer, 2010, pp. 1-16.
- [17] C. Canal, J.M. Murillo, and P. Poizat, “Software Adaptation,” *L’Objet: logiciel, bases de données, réseaux*, vol. 12, 2006, pp. 9-31.
- [18] C. Peper and D. Schneider, “On runtime service quality models in adaptive ad-hoc systems,” 2009 ESEC/FSE workshop on Software Integration and Evolution @ runtime, ACM, 2009, pp. 11-18.
- [19] A. Bottaro and R. Hall, “Dynamic Contextual Service Ranking,” *Software Composition*, Springer 2007, pp. 129-143.
- [20] M. Randles, A. Taleb-Bendiab, and D. Lamb, “Cross Layer Dynamics in Self-Organising Service Oriented Architectures,” *Self-Organizing Systems*, Springer, 2008, pp. 293-298.
- [21] L. Liu, S. Thanheiser, and H. Schmeck, “A Reference Architecture for Self-organizing Service-Oriented Computing,” *Architecture of Computing Systems (ARCS 2008)*, U. Brinkschulte, T. Ungerer, C. Hochberger, and R. Spallek, eds., Springer, 2008, pp. 205-219.
- [22] P. Lin, A. MacArthur, and J. Leaney, “Defining Autonomic Computing: a Software Engineering Perspective,” *Proceedings Australian Conference on Software Engineering (ASWEC 2005)*, IEEE Computer Society Press, 2005, pp. 88-97.
- [23] M. Wolski, C. Mazurek, P. Sychala, and A. Sumowski, “The architecture of distributed systems driven by autonomic patterns,” *Software Engineering Techniques: Design for Quality*, K. Sacha, ed., Springer Boston, 2007, pp. 49-60.
- [24] Y. Liu, M. Tan, I. Gorton, and A. Clayphan, “An Autonomic Middleware Solution for Coordinating Multiple QoS Controls,” *Service-Oriented Computing (ICSOC 2008)*, A. Bouguettaya, I. Krueger, and T. Margaria, eds., Springer, 2008, pp. 225-240.
- [25] D. Menasce, H. Gomaa, S. Malek, and J. Sousa, “SASSY: A Framework for Self-Architecting Service-Oriented Systems,” *IEEE Software*, Early Access article, IEEE, in press.
- [26] K. Welsh and P. Sawyer, “When to Adapt? Identification of Problem Domains for Adaptive Systems,” *Proceedings of the 14th international conference on Requirements Engineering: Foundation for Software Quality*, Springer-Verlag, 2008, pp. 198-203.
- [27] J.A. McCann and M.C. Huebscher “Evaluation issues in autonomic computing”. *Proceedings of Grid and Cooperative Computing Workshops (GCC)*, IEEE CS Press, 2004, pp. 597-608.
- [28] L. Masciadri and C. Raibulet, “Frameworks for the Development of Adaptive Systems: Evaluation of Their Adaptability Feature Through Software Metrics,” 4th International Conference on Software Engineering Advances (ICSEA 2009), 2009, pp. 309-312.
- [29] C. Raibulet and L. Masciadri, “Evaluation of Dynamic Adaptivity through Metrics: an Achievable Target?,” *Joint Working IEEE/IFIP Conference and European Conference on Software Architecture (WICSA/ECSA 2009)*, IEEE CS Press, 2009, pp. 341-344.
- [30] P. Reinecke, K. Wolter, and A. van Moorsel, “Evaluating the adaptivity of computing systems,” *Performance Evaluation*, vol. 67, 2010, pp. 676-693.
- [31] D. Robinson and G. Kotonya, “A Self-Managing Brokerage Model for Quality Assurance in Service-Oriented Systems,” *High-Assurance Systems Engineering*, IEEE, 2008, pp. 424-433.
- [32] G. Feuerlicht, “Simple Metric for Assessing Quality of Service Design,” *Service-Oriented Computing*, Springer, 2011, pp. 133-143.
- [33] Luqi and G. Jacoby, “Testing Adaptive Probabilistic Software Components in Cyber Systems,” *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems (16th Monterey Workshop)*, LNCS 6662, Springer, 2011, pp. 228-238.
- [34] S. Neti and H.A. Muller, “Quality Criteria and an Analysis Framework for Self-Healing Systems,” *Proc. Software Engineering for Adaptive and Self-Managing Systems (SEAMS’07, ICSE)*, IEEE Computer Society, IEEE Digital Library, 2007, p. 6.
- [35] M. Autili, P. Inverardi, and M. Tivoli, “Run Time Models in Adaptive Service Infrastructure,” *Run-time models for Self-managing Systems and Applications*, Springer, 2010, pp. 125-152.
- [36] H. Zhang, H. Whang, and R. Zheng, “An Autonomic Evaluation Model of Complex Software,” *International Conference on Internet Computing in Science and Engineering*, 2008, pp. 343-348.

A Formal Specification of G-DTD: A Conceptual Model to Describe XML Documents

Zurinahni Zainol^{*,^}, Bing Wang^{*}

^{*}Department of Computer Science
University of Hull
UK

[^]School of Computer Sciences
Universiti Sains Malaysia
Penang, Malaysia

z.zainol@2007.hull.ac.uk, b.wang@hull.ac.uk

Abstract – This paper provides a formal specification in Z of a conceptual model for an XML document called Graph-Document Type Definition (G-DTD). This model has been used for describing XML documents at the schema level and also assists the user to arrange the content of XML documents. More importantly G-DTD can be used as a tool to simplify the XML document design in a simple and precise way. The specification presented here provides a formal account of the state and operation of this model and a sound basis for instantiations of the model to be built.

Keywords – XML model and design; graphical notation; DTD; formal methods

I. INTRODUCTION

It is well known that XML documents can be regarded as a new type of database, and such data are particularly good for information exchange on the internet. Like relational databases, poorly designed documents may contain too many unnecessary redundancies and these redundancies may contain update anomalies [2, 7, 14, 15]. Data redundancies and anomalies can occur in XML documents if the schema that is DTD (Document Type Definition) [11] or XML Schema [13] is not well defined. In order to avoid these problems, it is very important to have a well defined schema for XML documents. To achieve this aim, a conceptual model Graph Document Type Definition (G-DTD) [16] is proposed to describe XML documents at the schema level. G-DTD has richer syntax and structure which incorporates attribute entity, simple data types, complex element data types, relationship types, hierarchical structure, cardinality, sequence and disjunctions between elements or attributes. The benefit of the G-DTD data model is that, it can be used to capture the syntax and semantics of XML documents in a simple but precise way. Having G-DTD as a tool helps the user to arrange

the content of XML documents in order to give a better understanding of DTD structures, improves XML design and assists the normalization process as well. The conceptual model G-DTD is a first layer of an XML document design system which we have formally constructed.

The benefits of having such a formal specification are firstly, to make a precise description of the complete G-DTD model at the conceptual level in order to remove ambiguity that may arise from its graphical representation. Secondly, to make G-DTD itself a modelling notation so that it can be used as the basis for a rigorous tool for XML design and finally, to eliminate inconsistencies in XML design at a schema level. This formal specification is used to describe a fundamental framework of *what* the system can do and also as an abstraction of a full complete system which can serve as a reliable blueprint for those who want to implement the program later. This formal specification is important before the implementation of the real system is developed, as it allows a designer to understand the big picture of the system and helps to discover error early in the development process.

There is a related work by Anutariya et al. [1], which has proposed a formal data model for an XML database using XML Declarative Description (XDD) theory. However, the most related work using a formal method to present formally a data model for semistructured data called Object Relational Attribute for Semistructured (ORA-SS) is done by Lee et al. [8,9]. They used different types of formal method languages to present the syntax and semantics of the model. For instance, Lee et al [8] used Z formal language to validate the syntax and semantics of the ORA-SS model. They also validated the model to check the correctness of ORA-SS at both schema and instance levels. Similar to this work, the formalization of ORA-SS using OWL was presented to

improve verification performance. Recently, Lee et al [9] have used a different approach to define a formal specification for ORA-SS using Prototype Verification System (PVS) language. However, to the best of our knowledge, no formal specification has been developed to define an XML document design system. This paper describes the first layer of the system.

The rest of the paper is organized as follows: Section II provides background knowledge on G-DTD notations, structure and operations. Section III presents the Z formal specification of G-DTD. In Section IV, we demonstrate the formal specification of G-DTD operations defined in Section II. We conclude the paper with our future work in Section V.

II. BACKGROUND

DTD is commonly represented as textual representation. In practice, it often causes difficulties when designing even a simple XML document. More importantly, in DTD, the semantic constraints and relationship between the elements in the XML document cannot be represented precisely and clearly. For instance, as shown in Figure 1, the relation between *course* and *student* is not defined explicitly. The semantic relation between the elements presents only one-to-many relationships, while other relationships such as many-to-many or many-to-one relationships cannot be defined. However, G-DTD overcomes the above problems by using a graphical notation to visually represent an XML document structure at the schema level. This notations are shown clearly in the example provided in Figure 2. In this way, we believe the user can have a better understanding of XML document structure. Indeed, Mok and Embley [10] make the argument that “*the graphical conceptual modelling languages offer one of the best human-oriented ways of describing an application*”

Representation of G-DTD is slightly different from the DTD. Firstly, we distinguish explicitly the difference between complex elements, simple element and attribute. We emphasise that a simple element is an element with no child elements, while an attribute is a key or candidate key of a complex element. The reason for this is to make the normalization process easier. Secondly, we present the G-DTD structure as a hierarchical structure of elements which is similar to XML document structure, to provide an accurate picture of the XML document. The advantages of G-DTD over DTD are: it allows users to define explicitly the structure of attribute nodes, simple element nodes and complex element nodes in a hierarchical way and also allows the user to determine the relationship dependency between the nodes.

A. Syntax and Semantics of G-DTD

Some of the notations of G-DTD have been adopted and improved upon from the current data model ORA-SS

[5] notations and conventional ER model [4]. G-DTD [15] consists of six basic components:

(1) *Complex element node*. A complex element node is used to represent an ‘*ELEMENT*’ in DTD. The complex element node is illustrated as a labelled rectangular box. This notation is adopted from the ER model [4] which is similar to entity. The label is written in the rectangle as a tuple $\langle name, level \rangle$, where *name* represents the name of the node and *level* represents the depth of the node in G-DTD.

(2) *Simple element node*. A simple element node is used to represent an ‘*ELEMENT*’ associated with #PCDATA or #CDATA. It is illustrated as a labelled rounded rectangular box with the form $\langle name, level, type \rangle$ where *name* is the name of the simple element, *level* is the depth of the node in the G-DTD and *type* represents PCDATA or CDATA or string ‘S’. All simple element nodes are assumed to be mandatory and single valued, unless the node contains the symbol ‘?’ which signifies it is single valued and optional, or + which signifies that it is multi-valued and required, or an * which shows that it is optional and multi-valued. This notation is similar to ORA-SS [6]. The symbol is written in front of the tuple $\langle name, level, type \rangle$ to differentiate among them accordingly.

(3) *Attribute node*. An attribute node is used to represent an *attribute* defined in *ATTLIST*. The attribute node is an identifier for a complex element node. It is represented as an ID which is unique and mandatory among the instances of complex elements. Attributes can be classified as single attributes and composite attributes. A single identifier attribute has an atomic value and composite attributes have more than one identifier attributes. A single identifier attribute is represented as an oval and a composite attribute as a double oval.

(4) *Set relationship type*. Three types of relationship are used in G-DTD: *Hierarchical link*, *part_of link* and *has_a link*. The *Hierarchical link* is a relationship between complex element nodes. This link shows the relationship between parent node to child node or ancestor node to descendant node. For *Hierarchical link*, a relationship dependency, which is indicated by the *connectivity* between complex element occurrences, is important. Basic constructs for connectivity are: *one-to-one* (unary or binary relationship), *one-to-many* (unary or binary relationship), *many-to-one* and *many-to-many* (unary or binary relationship). All these types of relationship are indicated by directional arrows. The notation is presented as $(name, d, cp, cc)$ where *name* represents the name of the relationship, *d* is the degree of relationship, *cp* and *cc* are cardinality constraints for parent and child respectively. This notation is similar to ORA-SS [6]. The degree can be two, three or n-ary. The cardinality of *cp* and *cc* in a relationship is represented as 2 tuple (min: max). The constraint $(0:N)$, $(0:1)$ and $(1:N)$ is represented as the

operators *, ? and + respectively, except the cardinality constraint (1:1) is presented as 1. For instance, the diagram in Figure 2 illustrates a binary *Hierarchical link* between complex element *student* and complex element *courses*, where a student can take zero or many courses while many *courses* can be taken by zero or many students. *Part_of link* is a relationship between a complex element node and an attribute node. It is illustrated as a bold double arrow. *Has_a link* is a relationship between complex element node and a simple element node. It is illustrated as a single double arrow.

(5) *Semantic constraint between set relationships.* There are two types of set relationships: First, sequence between a set of child element nodes. We emphasize in our notation that the attribute node(s) must be located in the first position in the sequence. To express such ordering in a G-DTD, we draw a directed upwardly curving arrow labelled with {sequence} across all the set of relationships involved. Second, is disjunction between the set of sibling nodes. To illustrate this, we draw a line labelled with {XOR} across all the set of relationships involved.

(6) *Root node.* A root node is used to represent *DOCTYPE*. Its notation is similar to complex element notation, as it is a special case of a complex element node and its level is always zero.

Figure 2 shows a G-DTD describing the structure of an XML document corresponding to the DTD in Figure 1. The root node *Department* has a binary hierarchical link with the complex element node *course*. The semantic relationship between them reveals that the *Department* can have one-to-many *courses* at one time. The complex element *course* has a sequence of attribute *cno*, simple element node *title* and complex element node *student*.

```

<!DOCTYPE department[
  <!ELEMENT department(course*)>
  <!ELEMENT course(title, student*)>
    <!-- ATTLIST course cno ID #REQUIRED -->
  <!ELEMENT title (#PCDATA)>
  <!-- ATTLIST student(fname|lname?, lecturer) -->
    <!-- ATTLIST student Sno ID #REQUIRED -->
  <!ELEMENT fname(#PCDATA) >
  <!ELEMENT lname(#PCDATA) >
  <!ELEMENT lecturer (tname)>
    <!-- ATTLIST lecturer tno ID #REQUIRED -->
  <!ELEMENT tname (#PCDATA)>
]
    
```

Figure 1. A DTD for the university database

The part-of link attribute is a mandatory relationship where the attribute node *cno* is required and unique for every course in the XML document. The simple element node *title* is part-of the complex element *courses*. One *course* can be taken by many *students* while the complex element *student* consists of a sequence of attribute node *sno*, simple elements *fname*, *lname* and complex element *lecturer*. Attribute node *sno* is required for the complex element *student*. Complex element node *student* requires only one of its subelements, either *fname* or *lname*, to appear in the XML document while the simple element *lname* is optional. The semantic relationship between course, student and lecturer is indicated as a ternary relationship since each student is assigned to a lecturer who is teaching the course.

As shown in Figure 2, the semantic relationships between the complex element nodes have been added at the hierarchical link to present more semantics at the schema level. The reason we add this type of semantics is to make the relationship between the nodes more explicit, which will help during the normalization process.

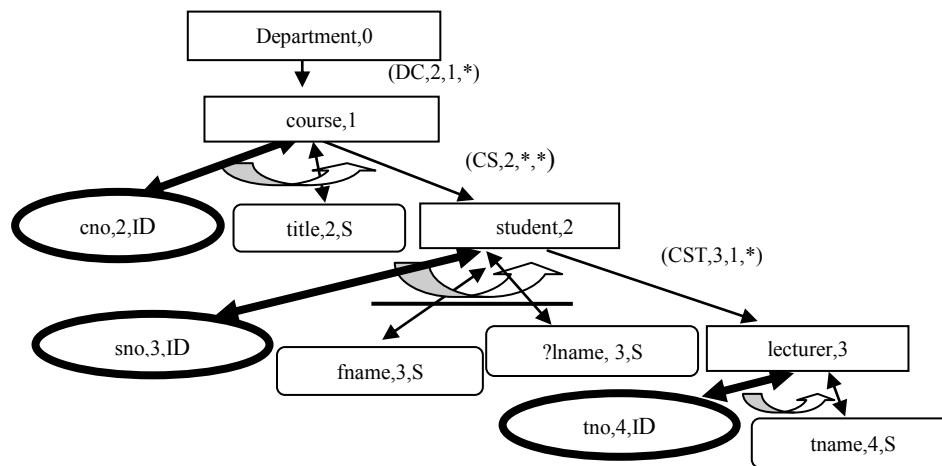


Figure 2. G-DTD

B. G-DTD Operations

The operations of the G-DTD model describe the dynamic properties of the model. G-DTD model operations are classified into five main parts. Query Operations, Insert Operations, Delete Operations, Searching Operations, and Update operations. An operation to determine the root and leaves of the G-DTD is also required. Later, these operations will be used in normalizing the G-DTD into normal forms. In the following description, we will conceptually discuss the semantic connection of these operations according to this classification.

(1) Query Operations

Query operations allow the user to query the node types and information, related nodes and links information defined in G-DTD.

(a) Query a Node Type and Information

The operations of querying node types allow the user to query different types of node stored in G-DTD such as complex element, simple element or attribute nodes. The user can also query information of a particular node, such as name, level and node type. If the queried node does not exist, an error message is given.

(b) Query a Related Node

Since the structure of G-DTD is like a tree structure, the query operations allow the user to query the related node that links to a particular node using a path through an existing link such as a Hierarchical, Part_of or Has_A link. For instance, the user can detect the parent of a complex element node by using the hierarchical link between two complex element nodes. Another example, the simple element for a particular complex element node can be determined through the has-a link.

(c) Query a Hierarchical Link

Hierarchical links are the most important links in G-DTD. This operation allows the user to query the instance of a hierarchical link, such as name of link, degree of relations and parent and child constraint.

(2) Insert Operations

Insert operations allow the user to add new nodes to the G-DTD. When a new node is being inserted in the G-DTD model, the following situations are possible:

- A new node of type complex element node, simple element node or attribute node is created
- A new hierarchical link is built between the complex element node and created complex element node
- A new has-a link is built between the created complex element node and a simple element node

- A part-of link is built between the created complex element node and an attribute node

To ensure the new node is not redundant with any node in the given G-DTD, it must be checked whether the node already exists. Then the proper location of the new node needs to be determined before it can be inserted into the G-DTD. More importantly, it must satisfy the data integrity constraint of the given G-DTD.

(a) Inserting a Node

In this case a new node is inserted into the G-DTD. Whether the new node is a complex element, simple element or attribute node, the properties of the inserted node such as ID, level and types are inserted and stored together in the G-DTD. The operation implies that when the node is inserted, related nodes such as parent node or child node should be reported to the user since the structure of the G-DTD is changed. If the newly inserted node is a complex element node, the position of the new complex element node is based on the rules provided in the normalization procedure [17]. In such a situation, a hierarchical link is created with its parent node. In this case, the parent node may be a root node or another complex element node based on the normalization rules provided. However if the created node is a simple element or an attribute node, a Part_of link or Has_A link is built between it and the parent node, which is a complex element node.

(b) Inserting an Instance of a Hierarchical Link

Inserting an instance of a hierarchical link means that the semantic relation between two complex element nodes has to be created. The user needs to know the semantic relationships before he/she can insert them to the G-DTD. The user can make links and insert the corresponding link information such as name, degree, parent constraint and child constraint. In contrast, for a Part_of link or Has-A link, the user is not required to put any instance for the links.

(3) Delete Operations

Delete operations result in the corresponding data being removed from the G-DTD. Since the structure defined in the G-DTD is a tree structure, deleting will affect the location of the existing nodes in the G-DTD, especially the parent node and child node. The delete operation in G-DTD must satisfy the conditions and constraints given in the normalization rules [17]. In the following, we will discuss the different situations of delete operations in the G-DTD.

(a) Deleting a Complex Element Node

Deleting a complex element node is a complex deletion process in G-DTD. This is because every complex element node is related to its parent node and child node. Before the deletion process of a complex

element node is started, it is important for the user to find its related nodes such as its parent node and child nodes. Eventually, by deleting a complex element node, its attribute and simple element nodes with the relevant, Part_of and Has_A links are automatically deleted as well. Then, new links are built up with its new parent node and child node.

(b) *Deleting a Hierarchical Link type and its Instance*

According to the hierarchical link type definition, each instance of a hierarchical link type represents a semantic relationship between two complex element nodes. When such an instance is deleted, the specific relationship between the two nodes has no further semantic link between them.

(4) *Update Operations*

Update operations change the location of the current node. A complex element node or simple element node can be moved around from one location to another. In the process of moving a node, all the related nodes including complex element nodes and simple element nodes should be notified if the moving node has a relationship with them. The only case we consider here is moving a complex element node. It may be necessary to move a complex element node up to another level when there exists dependency between an attribute node and simple element node of a complex element node. In this situation, it is not necessary to create a new element node but rather to restructure the G-DTD by moving up the complex element node at level n (n_n) to level n-1 (n_{n-1}) along with its corresponding children.

(5) *Determine the root node and last node*

This operation will determine the root node and last node (last level) in the G-DTD. The last node may be a simple element node or attribute node. These operations are very important because in order to avoid duplication, we need to move the corresponding node to a position as close as possible to the root node.

III. THE SPECIFICATION OF G-DTD

In this paper we provide a formal specification of the G-DTD which represents a formal, concise and readable definition of the G-DTD and its operations. The specification can be used as the basis for implementation, as well as a framework for further XML document design. We choose the language Z [11] to formalise our model for a number of reasons. First, the language is based upon primitive mathematical notation such as set theory and first order predicate logic, making it accessible to researchers from variety of different backgrounds. Second, it is expressive enough to allow consistent, formal and unified representation of a system and its associated operations. Third, it is model oriented [3]. A

model-oriented specification language seems more appropriate to specify an XML design model and it is easier to understand. Finally, in particular, we have found that Z is an established language, widely accepted and appropriate for building formal frameworks [9]. A specification written in Z is a mixture of formal mathematical statements and informal explanatory text. Both have their importance: the formal part gives a precise definition of the system being specified, while the informal text makes the specification more comprehensive and readable, linking the abstract definition of the system to the real world. In this paper we present only some basic components and operations, due mainly to space limitations; other results will be published in a forthcoming paper.

A. *Basic types*

We use the basic types [*ID*, *Element_Name*, *Attribute_Name*, *Relation_Name*] as a given set which will be used in the later schema definition. *ID* represents each nodes identifier, which is unique; both *Element_Name* and *Attribute_Name* are used to represent the set of all possible XML element nodes and attribute nodes respectively. *Relation_Name* is a set for relationship names.

B. *The Data Structure of G-DTD*

As described in Section II(A), we captured the characteristics of each type of node such as simple element, complex element and attribute nodes using the following schema type. There is no constraint we need to add in each of the declarations

(1) *Simple Element Node*

The type definition for a simple element is defined as follows:

```
Simple_Element_Type ::= singlevalue | multivalued | op_singlevalue |
op_multivalued
SimpleElementNode
identity: ID
name: Element_Name
level: N
elementType: Simple_Element_Type
```

(2) *Attribute Node*

The *AttributeNode* schema captures the properties of an attribute node as follows:

```
Attribute_Type ::= composite | required | reference
AttributeNode
identity: ID
name: Attribute_Name
level: N
AttType: Attribute_Type
```

(3) *Complex Element Node*

The *ComplexElementNode* schema represents the properties of a complex element node with its identity, name and level.

$\begin{array}{l} \text{ComplexElementNode} \\ \text{identity:ID} \\ \text{name: Element_Name} \\ \text{level:N} \end{array}$
--

(4) Parent for Complex Element Node, Simple Element Node and Attribute Node

Because the structure of the G-DTD is a tree structure, it is important to define a parent for each complex element node, simple element node and attribute node to describe precisely the relationship between them. The functions *parent_ce*, *parent_se* and *parent_att* are defined using the axiomatic function as a total function because every complex element node, simple element node and attribute node must have its own parent node and no node can have more than one parent.

$\begin{array}{l} \text{parent_ce: ComplexElementNode} \rightarrow \text{ComplexElementNode} \\ \text{parent_se: SimpleElementNode} \rightarrow \text{ComplexElementNode} \\ \text{parent_att: AttributeNode} \rightarrow \text{ComplexElementNode} \end{array}$
$\forall ce1, ce2: \text{ComplexElementNode} \bullet$ $ce1 \mapsto ce2 \in \text{parent_ce} \Leftrightarrow (ce1 \neq ce2 \wedge$ $ce2.level < ce1.level \wedge$ $ce2.level - ce1.level = 1) \vee$ $(\forall se: \text{SimpleElementNode}; ce: \text{ComplexElementNode} \bullet$ $se \mapsto ce \in \text{parent_se} \Leftrightarrow (ce.level < se.level \wedge$ $se.level - ce.level = 1)) \vee$ $(\forall att: \text{AttributeNode}; ce: \text{ComplexElementNode} \bullet$ $att \mapsto ce \in \text{parent_att} \Leftrightarrow (ce.level < att.level \wedge$ $att.level - ce.level = 1))$

In the state invariant, it is stated that complex element $ce1 \mapsto ce2 \in \text{parent_ce}$ means that *ce2* is the parent of *ce1* if and only if *ce1* is not the same as *ce2* and the level position of *ce2* must always be less than the level position of *ce1* by one level difference only. The same meaning is applied for the second and third predicates associated with the parent for a simple element node and parent for an attribute node, respectively.

(5) Relationship

We define three types of relationship which are Hierarchical_Link, Part_of_Link, and HasA_Link using the following schemas.

(a) Hierarchical_Link

The *Hierarchical_Link* schema consists of a relation *hierarchical_link* which is used to define a homogeneous relation between complex element nodes. The first and second predicates of the schema state that an ordered pair of complex element nodes $ce1 \mapsto ce2$ is an element of *hierarchical_link* if and only if *ce2* is an immediate parent of *ce1* or *ce2* is a hierarchical parent of *ce1*, $ce1 \mapsto ce2 \in \text{hierarchical_link}^+$, that to say, it is a transitive closure relation. The third predicate of the schema defines that the child complex element should not be the same set as the parent complex element node and finally the relation must be cycle free, which means no complex element node is mapped to itself. This is defined using transitive closure to capture the idea of some complex element nodes (homogeneous binary relation) can be directly reached in the same link. The relation *hierarchical_link* is known as a homogeneous relation [4] since the complex elements are from the same set. One of the benefit of this relation

is that it can be composed among such links themselves. Thus, we can form the relation *hierarchical_link;hierarchical_link*. This can also be written as *hierarchical_link*². The *hierarchical_link* can be repeated as many times as desired. The constraint relationship on the *hierarchical_link* must be a positive number. The properties of the schema also consist of name, degree of relationship, parent cardinality and child cardinality constraints.

$\begin{array}{l} \text{Hierarchical_Link} \\ \text{hierarchical_link: ComplexElementNode} \leftrightarrow \text{ComplexElementNode} \\ \text{degree: N}_i \\ \text{parentconstraint: N..N}_i \\ \text{childconstraint: N..N}_i \\ \text{name: Relation_Name} \end{array}$
$(\forall ce1: \text{ComplexElementNode}; ce2: \text{ComplexElementNode}$ $\bullet ce1 \mapsto ce2 \in \text{hierarchical_link}$ $\Leftrightarrow \text{parent_ce}(ce1) = ce2$ $\wedge ce1 \mapsto ce2 \in \text{hierarchical_link}^+$ $\wedge ce1 \neq ce2$ $\wedge (\exists ce: \text{ComplexElementNode} \bullet$ $ce \mapsto ce \notin \text{hierarchical_link}^+)$ $\wedge (\forall n1, n2: \text{name} \bullet n1 \neq n2)$ $\wedge (\forall d: \text{degree} \bullet \neq d \geq 2)$ $\wedge (\forall \text{card}: \text{N..N}_i \bullet \text{second}(\text{card}) \geq \text{first}(\text{card}))$

(b) Part_of_Link

The *Part_of* link is a binary relationship rather than n-ary relationship. It consists of *Attribute_key* function and *Composite_key* relation. The *Attribute_key* function is a total and injective type because each complex element node has a unique attribute node. The *Composite_key* relation is a relation between a complex element and attributes. In the first predicate, $ce \mapsto att \in \text{Attribute_key}$ if and only if the attribute type is *required*. The second predicate states that, $ce \mapsto attcom \in \text{Composite_key}$ if and only if the attribute type is *composite*. The last predicate indicates that the domain for the *Attribute_key* function and *Composite_key* relation is a member of a complex element node.

$\begin{array}{l} \text{Part_of} \\ \text{Attribute_key: ComplexElementNode} \rightarrow \text{AttributeNode} \\ \text{Composite_key: ComplexElementNode} \leftrightarrow \text{AttributeNode} \end{array}$
$\forall ce: \text{ComplexElementNode}; att: \text{AttributeNode} \bullet$ $(ce \mapsto att) \in \text{Attribute_key} \Leftrightarrow att.attType = \text{required} \wedge \text{parent_att}(att)$ $= ce$ $\forall ce: \text{ComplexElementNode}; attcom: \text{AttributeNode} \bullet$ $(ce \mapsto attcom) \in \text{Composite_key} \Leftrightarrow attcom.attType = \text{composite}$ $\wedge \text{parent_att}(attcom) = ce$ $\text{dom Attribute_key} \cup \text{dom Composite_key} \in \text{ComplexElementNode}$

(c) Has_A Link

The schema *Has_A* consists of a *has_a* relation which describes that a complex element node has a relation with a simple element node where a simple element can be a single value, multivalued, optional single value or optional multivalued and must have a complex element node as a parent.

$\begin{array}{l} \text{Has_A} \\ \text{has_a: ComplexElementNode} \leftrightarrow \text{SimpleElementNode} \end{array}$
$\forall ce: \text{ComplexElementNode}; se: \text{SimpleElementNode} \bullet$ $(ce \mapsto se) \in \text{has_a} \Leftrightarrow se.seType = \text{singlevalue} \vee se.seType =$ $\text{multivalued} \vee se.seType = \text{op_singlevalue} \vee se.seType = \text{op_multivalued}$ $\wedge \text{parent_se}(se) = ce$

C. The State Space of Schema G-DTD

To finally organize the structure of the G-DTD, all the above-defined node types and relationship types are used in the *schemaGDTD* definition.

The *SchemaGDTD* consists of seven variables which include a root node type, set of *ComplexElementNode*, set of *SimpleElementNode*, set of *AttributeNode* and set of relation *Hierarchical_Link*, *Has_A* and *Part_of* types. The first predicate of the *SchemaGDTD* states that there must exist one root node. The second, third and fourth predicates indicate that at any point in time, each complex element node, simple element node and attribute node must have a unique name. The last four predicates ensure that all types of nodes and relationships defined exist in *SchemaGDTD*.

<i>SchemaGDTD</i>
$root: ComplexElementNode$ $Cnodes: \mathbb{P}ComplexElementNode$ $Snodes: \mathbb{P}SimpleElementNode$ $Attmodes: \mathbb{P}AttributeNode$ $HierarchicalLink: \mathbb{P}Hierarchical_Link$ $HasA: \mathbb{P}Has_A$ $Partof: \mathbb{P}Part_of$
$\exists root: ComplexElementNode \bullet root.level = 0$ $\forall ce1, ce2: Cnodes \mid ce1 \neq ce2 \bullet ce1.name \neq ce2.name$ $\forall se1, se2: Snodes \mid se1 \neq se2 \bullet se1.name \neq se2.name$ $\forall att1, att2: Attmodes \mid att1 \neq att2 \bullet att1.name \neq att2.name$ $\forall partlink: Partof \bullet partlink.AttributeKey \neq \emptyset$ $\forall hl: HierarchicalLink \mid haslink: HasA \mid partlink: Partof \bullet$ $dom\ partlink.Attribute_key = dom\ partlink.Composite_key$ $\wedge ran\ haslink.hasA = Snodes \wedge ran\ partlink.Attribute_key = Attmodes$

D. Initial State of Schema G-DTD

Before any operation can be performed on the model, we must define the initial state of the G-DTD. In our case, the initial state of the G-DTD refers to the situation in which there are no elements existing in the schema. This schema describes the *InitialG-DTD* in which the sets of simple element nodes, complex element nodes and attribute nodes are empty: in consequence, the *HierarchicalLink*, *HasA* and *Partof* relations are empty too. This is characterized by the following schema definition:

<i>InitialG-DTD</i>
$\Delta SchemaGDTD$ $Snodes = \emptyset$ $Cnodes = \emptyset$ $Attmodes = \emptyset$ $Partof = \emptyset$ $HasA = \emptyset$ $HierarchicalLink = \emptyset$

IV. OPERATIONS SPECIFICATION IN G-DTD

The operations defined in schema G-DTD describe the behaviour or state change of the G-DTD during editing and manipulating nodes. We present some of the operations which are query operations, create, insert and delete operations. However, before we present these operations we must first define the following functions.

(1) Create Complex Element Node

<i>Create_NewComplexElementNode</i> : $(ID \times Element_Name \times \mathbb{N}) \rightarrow ComplexElementNode$
$\forall newid: ID; newname: Element_Name; l: \mathbb{N}_1; schema:$ $SchemaGDTD \bullet (\exists ce, newnode: ComplexElementNode;$ $schema'. SchemaGDTD)$ $newnode = ce \bullet$ $(ce.identity = newid \wedge ce.name = newname \wedge ce.level = l) \wedge$ $newnode \notin schema.Cnodes \wedge$ $schema'. Cnodes = schema.Cnodes \cup \{newnode\}$ $\Rightarrow Create_NewComplexElementNode$ $(newid, newname, l) = newnode$

The first predicate of the function assigns an instance of a new complex element node. The second predicate gives a *pre-condition* for the success of the operation. The new complex element to be added must not already be one of the members of complex element nodes in G-DTD. This is because only one unique complex element is allowed in the G-DTD schema. If this condition is satisfied, the new complex element node is added to the set of complex element nodes.

(2) Create Attribute Node

The description of the *Create_AttributeNode* function is similar to the *Create_ComplexElementNode* function

<i>Create_AttributeNode</i> : $(ID \times Attribute_Name \times \mathbb{N}_1 \times Attribute_Type) \rightarrow AttributeNode$
$\forall newid: ID; newname: Attribute_Name; l: \mathbb{N}_1; type: Attribute_Type;$ $schema: SchemaGDTD \bullet$ $(\exists att, newnode: AttributeNode; schema'. SchemaGDTD)$ $newnode = att \bullet$ $(att.identity = newid \wedge att.name = newname \wedge$ $att.level = l \wedge att.type = type) \wedge$ $newnode \notin schema.Attmodes \wedge$ $schema'. Attmodes = schema.Attmodes \cup \{newnode\}$ $\Rightarrow Create_AttributeNode(newid, newname, l, type) = newnode$

(3) Create Has_a link

Create_Has_a_Link is a function to create a new *HasA* link between a complex element node and a simple element node. The first predicate of the function maps both of the given complex element node and simple element node and assigns between them a new *has* link. Then the new *has* link is added to the set of new *has* links in *SchemaGDTD*.

<i>create_Has_a_Link</i> : $(ComplexElementNode \times SimpleElementNode) \rightarrow HasA$
$\forall ce: ComplexElementNode; se: SimpleElementNode;$ $schema: SchemaGDTD \bullet$ $(\exists new_Haslink, newlink: HasA; schema'. SchemaGDTD)$ $new_Haslink = newlink \bullet$ $ce \rightarrow se \in newlink.has_a$ $\wedge schema'. HasA = schema.HasA \cup \{new_Haslink\}$ $\Rightarrow create_Has_a_Link(ce, se) = new_Haslink$

Create_Hierarchical_Link is a function to create a new *Hierarchical_Link* between two complex element nodes. The first predicate of the function maps both of given complex element node and complex element node and assigns between them a new *Hierarchical_Link* if and only if it is satisfied that the relation of these complex element nodes is not a cyclic one. The remaining predicate is used to assign a new relation name, new level, parent constraint and child constraint to the new *Hierarchical_Link*. The last predicate ensures that the new *has* link is added to the set of new *Hierarchical_Link* in *SchemaGDTD*.

(4) Create Hierarchical link

$\text{Create_Hierarchical_Link: } (\text{ComplexElementNode} \times \text{ComplexElementNode}) \rightarrow \text{HierarchicalLink}$
$\forall ce1, ce2: \text{ComplexElementNode}; \text{schema: SchemaGDTD} \bullet$ $(\exists \text{new_HierarchicalLink, newlink: HierarchicalLink; level: } \mathbb{N}_1;$ $pc, cc: \mathbb{N} \times \mathbb{N}_1;$ $\text{newname: Relation_Name; schema': SchemaGDTD})$ $\text{new_HierarchicalLink} = \text{newlink} \bullet$ $ce1 \mapsto ce2 \in \text{newlink.hierarchical_link} \Leftrightarrow$ $(ce1 \mapsto ce2 \notin \text{newlink.hierarchical_link}^+)$ $\wedge ce2 = \text{parent_ce}(ce1)$ $\wedge \text{name}(\text{newlink}) = \text{newname}$ $\wedge \text{degree}(\text{newlink}) = \text{level}$ $\wedge \text{parentconstraint}(\text{newlink.hierarchical_link}) = pc$ $\wedge \text{childconstraint}(\text{newlink.hierarchical_link}) = cc)$ $\wedge \text{schema'.Hierarchical_Link} =$ $\text{schema.Hierarchical_Link} \cup$ $\{ \text{new_HierarchicalLink} \}$ $\Rightarrow \text{Create_Hierarchical_Link}(ce1, ce2) =$ $\text{new_HierarchicalLink}$

(5) Create Partof link

The function *Create_partof_Link* is used to create part-of links between complex element nodes and attribute nodes. The argument of this function is a relation between a complex element node and attribute node and return a partof link. The new part_of link can be either Attributekey or Compositekey and the parent of the attribute node must be a complex element node. Finally, a new partof link is added to the set of partof links in *SchemaGDTD*.

$\text{create_Partof_Link: } (\text{ComplexElementNode} \times \text{AttributeNode})$ $\rightarrow \text{partof}$
$\forall ce: \text{ComplexElementNode}; \text{att: AttributeNode}; \text{new_partoflink},$ $\text{partoflink: partof}; \text{schema: SchemaGDTD} \bullet$ $\exists \text{schema': SchemaGDTD} \{$ $\text{new_partoflink} = \text{partoflink} \bullet$ $ce \mapsto \text{att} \in \text{partoflink.AttributeKey} \Leftrightarrow$ $\text{att.attType} = \text{required} \wedge \text{parent_att}(\text{att}) = ce$ $\vee ce \mapsto \text{att} \in \text{partoflink} \text{.CompositeKey} \Leftrightarrow$ $\text{att.attType} = \text{composite} \wedge \text{parent_att}(\text{att}) = ce$ $\wedge \text{schema'.Partof} = \text{schema.Partof} \cup \{ \text{new_partoflink} \}$ $\Rightarrow \text{create_Partof_Link}(ce, \text{att}) = \text{new_partoflink}$

A. Query Operations

Before manipulating the structure of any complex element node in the G-DTD, we should be aware of its related nodes. Since the structure of the G-DTD is like a tree structure, a child or descendants and parent or ancestor of a given complex element node needs to be queried in some cases. The status of a queried node is defined using a set of messages. It is defined by enumeration type

Report ::= Existence | Nonexistence | Inserted | Created

Based on this set, we define the following schema *Success* to output a confirmatory message that the operation being performed has been successfully completed.

Success report! Report $\text{report! = Existence}$

The following *Get_AttributeKey* shows how to get an attribute key of complex element node using the part_of link

Get_AttributeKey $\exists \text{SchemaGDTD}$ $ce?: \text{ComplexElementNode}$ $\text{attkey! AttributeNode}$
$\forall \text{part_of: Partof} \bullet$ $\text{attkey!} = \text{part_of.AttributeKey}(ce?)$

Get_SimpleElement schema captures how to get a simple element node by using has_a link

$\text{Get_SimpleElementNode}$ $\exists \text{SchemaGDTD}$ $ce?: \text{ComplexElementNode}$ $se!: \mathbb{P} \text{SimpleElementNode}$
$\forall \text{has_link: HasA} \bullet$ $se! = \text{has_link.hasA}(\{ce?\})$

Each operation can only go wrong if the complex element *ce?* is not in *SchemaGDTD*. This case is captured by means of the schema *UnknownNode*.

UnknownNode $\exists \text{SchemaGDTD}$ $ce?: \text{ComplexElementNode}$ report! Report
$ce? \notin \text{dom has_link.hasA} \vee$ $ce? \notin \text{dom HierarchicalLink.hierarchical_link}$ $\text{report! = Nonexistence}$

Based on the schema definition above, we can finally define the following schemas, which describe the state in which a simple element node or attribute node has been successfully queried.

$\text{Do_Query_AttributeKey} \triangleq \text{Get_AttributeKey} \wedge \text{Success} \vee$
 UnknownNode
 $\text{Do_Query_SimpleElementNode} \triangleq \text{Get_SimpleElementNode} \wedge \text{Success} \vee$
 UnknownNode

The following schema is used to capture the query operation for a complex element node. This schema means that the existing complex element node whose name is equal to the input name is found.

$\text{Get_ComplexElementNode}$ $\exists \text{SchemaGDTD}$ $ce_name?: \text{Element_Name}$ $ce!: \text{ComplexElementNode}$ $\text{found_ce: Element_Name} \Rightarrow \text{ComplexElementNode}$
$\exists ce: \text{ComplexElementNode} \bullet$ $ce.name = ce_name? \Rightarrow \text{found_ce } ce_name? = ce!$

$\text{Do_Query_ComplexElementNode} \triangleq \text{Get_ComplexElementNode} \wedge$
 $\text{Success} \vee \text{UnknownNode}$

A query about the ancestor or descendants of complex element node can be made by using a Hierarchical_Link. We achieve this by forming the transitive closure of Hierarchical_Link

Ancestors $\exists \text{SchemaGDTD}$ $ce?: \text{ComplexElementNode}$ $\text{ancestor_ce!} \mathbb{P} \text{ComplexElementNode}$
$\forall \text{hl: HierarchicalLink} \bullet$ $\text{ancestor_ce!} = (\text{hl.hierarchical_link}^+) \setminus \{ce?\}$

$\text{Do_Query_AncestorNode} \triangleq \text{Ancestors} \wedge \text{Success}$

Descendants $\exists \text{SchemaGDTD}$ $ce?: \text{ComplexElementNode}$ $\text{descendant_ce!} \mathbb{P} \text{ComplexElementNode}$
$\forall \text{hl: HierarchicalLink} \bullet$ $\text{descendant_ce!} = (\text{hl.hierarchical_link}^+) \setminus \{ce?\}$

$\text{Do_Query_Descendants} \triangleq \text{Descendants} \wedge \text{Success}$

B. Insert Operation

Insert_NewComplexElement_Node schema is used to insert a new complex element node into G-DTD. In the

signature of the schema, the declaration $\Delta SchemaGDTD$ alerts the user to the fact that the schema is describing a state change. The functions *Create_New_ComplexElement* and *Create_Hierarchical_Link* are used to create a new node and create a new link respectively. Before the node can be inserted, a *pre-condition* is given to check whether it exists already. The new complex element to be inserted must not already be one in the G-DTD. This is because only one unique complex element is allowed in the G-DTD schema. If this condition is satisfied, the new complex element node is inserted and a *hierarchical link* is created between the new node and its parent node. When the operation is successful, the output will take a value *inserted*.

<i>Insert_NewComplexElement_Node</i>
$\Delta SchemaGDTD$
$level?: \mathbb{N}$
$newname?: Element_Name$
$newid?: ID$
$\forall newnode : ComplexElementNode ; newlink : HierarchicalLink \bullet$
$newnode =$
$Create_New_ComplexElement(newid?, newname?, level?) \wedge$
$newlink =$
$Create_Hierarchical_Link(newnode, parent_ce(newnode))$

The schema *success* just outputs a confirmatory message that the operation being performed has been successfully completed.

<i>Success</i>
$rep! : Report$
$rep! = Inserted$

To capture the condition where the simple element node is already a member of G-DTD, the following schema is used:

<i>AlreadyExisted</i>
$\exists SchemaGDTD$
$se_name?: Element_Name$
$se! : SimpleElementNode$
$found_se : Element_Name \Rightarrow SimpleElementNode$
$report! = Report$
$\exists se : SimpleElementNode \bullet se.name = se_name?$
$\Rightarrow found_se\ se_name? = se! \wedge report! = Existed$

To perform *Do_Insert_NewComplexElementNode* operation the following is used.

$$Do_InsertNewComplexElementNode \triangleq Insert_NewComplexElementNode \wedge Success \vee AlreadyExisted$$

C. Delete Operation

The operation to delete a simple element node from the G-DTD is specified by the following schema:

<i>Delete_SimpleElements_Node</i>
$\Delta SchemaGDTD$
$Get_SimpleElementNode$
$se?: \mathbb{P}SimpleElementNode$
$se? \in Snodes$
$\exists parent : complexElementNode ; link : hasa $
$parent_se = link^{-1}\{se?\} \bullet$
$delete_partoflink(parent_se, link, schema)$
$Snodes' = Snodes \setminus \{se?\}$

Before the node can be deleted, it must be checked that the given node is a member of simple element nodes in the G-DTD and the parent of the simple element node needs to be determined. The node can be deleted from the G-DTD if the input node is present in the G-DTD. If this

pre-condition is not satisfied, then this will be captured by the following schema:

<i>UnknownNode</i>
$\Delta SchemaGDTD$
$se?: \mathbb{P}SimpleElementNode$
$report! : Report$
$se? \notin Snodes$
$report! = Nonexistence$

The complete specification of the operation to delete a simple element node from *SchemaGDTD* is given by the schema:

$$Do_DeleteSimpleElementNode \triangleq Delete_SimpleElement \wedge success \vee UnknownNode$$

V. CONCLUSION

We have presented a formal specification of a G-DTD model using Z notation style which gives precise, mathematical meaning to basic conceptual structures. The formalization of the G-DTD model is required for a deeper understanding of modelled syntax, structure, and semantics of model properties. The use of formal specification techniques contributes to the clarity and conciseness of the model, and enables formal derivation of model properties to be performed easily. Obviously, this paper has reported only the beginning of formal development of an XML document design model, since it includes just a description of the G-DTD model structure and its basic operation. Currently we have constructed a complete formal specification for an XML document design model using G-DTD by applying those functions and schemas (defined in Sections III and IV). This specification includes finding of various functional dependencies, checking the G-DTD normal forms and normalization procedure operation. However, these results will be the subject of another paper.

REFERENCES

- [1] Anutariya, C., Wuwongse, V., Nantajeewarawat, E., and Akama, K., "Towards a Foundation for XML Document Database, Electronic Commerce and Web Technologies", LNCS, Springer, Vol. 1875, pp. 324-333 (2000).
- [2] Arenas, M. and Libkin, L., "A Normal Form for XML Documents", ACM Transaction on Database System, Vol. 29(1), pp. 195-232 (2004).
- [3] Bottaci, L., and Jones, J. "Formal Specification using Z". London: International Thomson Publishing Inc(1995).
- [4] Chen, P. P., "The entity-relational model: Towards a unified view of data", ACM transaction on Database System, 14 (1976).
- [5] Diller, A. Z., "An Introduction to Formal Methods", England, John Willey (2001).
- [6] Dobbie, G., Xiaoying, W., Ling, T.W. and Lee, M.L., "ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data". Technical Report, Department of Computer Science, National University of Singapore (2000).
- [7] Kolahi, S., "Dependency-preserving normalization of relational and XML data", Journal of Computer and System Sciences, pp. 636-647 (2007).

- [8] Lee, S.J., Sun, J., Dobbie, G., and Groves, L., "Formal Verification of Semistructured Data in PVS", *Journal of Universal Computer Science*, Vol. 15(1), pp. 241-272 (2009).
- [9] Lee, S.J., Sun, J., Dobbie, G. and Li, Y.F. "A Z Approach in Validating ORA-SS Data Models", *Electronic Notes in Theoretical Computer Science*, Elsevier, Vol. 157, pp. 95-109 (2006).
- [10] Mok, W.Y., Ng Y., Embley, D.. A Normal Form for Precisely Characterizing Redundancy in Nested Relations. *ACM Transaction on Database System*, 21(1), pp. 77-106(1996)
- [11] Powell, G., "Beginning XML databases", Indianapolis, Indiana, Willey Publishing (2007).
- [12] Spivey, J., "Understanding Z". Cambridge: University Press, Cambridge (1988).
- [13] Tompson, H. S., Beech, D., Moloney, and Meldensohn, Noah, "XML Schema W3C Recommendation". Retrieved on January 7, 2011 Accessed <http://www.w3.org/TR/xmlschema-1> (2011).
- [14] Wang, J. and Topor, R., "Removing XML data redundancies using functional and equality-generating dependencies", 16th Australasian Database Conference, pp. 65-74 (2005).
- [15] Yu, C. and Jagadish, J.H., "XML schema refinement through redundancy detection and normalization", *The VLDB Journal*, pp. 203-22 (2008).
- [16] Zainol, Z. and Wang, B., "GN-DTD: Graphical Notation for Describing XML Documents", In *Preceeding of 2nd International Conference on Advances in Databases, Knowledge, and Data Applications, DBKDA, IEEE*, pp. 214-221 (2010).
- [17] Zainol, Z. and Wang, B., "XML Document Design via GN-DTD", *European Journal of Scientific Research*, Vol. 44(2), pp. 314-336 (2010).

Formal Specification of Software Design Metrics

Meryem Lamrani

Laboratoire Conception et Systèmes
University Mohammed V Agdal
Department of Computer Science
BP 1014 RP Rabat, Morocco
lamrani@fsr.ac.ma

Younès El Amrani

Laboratoire Conception et Systèmes
University Mohammed V Agdal
Department of Computer Science
BP 1014 RP Rabat, Morocco
elamrani@fsr.ac.ma

Aziz Ettouhami

Laboratoire Conception et Systèmes
University Mohammed V Agdal
Department of Computer Science
BP 1014 RP Rabat, Morocco
touhami@fsr.ac.ma

Abstract—Given the significant interest in applying formal methods to object oriented paradigms, this paper presents a formal approach to define software design quality metrics upon a formal specification of the UML metamodel using the Z language. This multi-level formalization benefits greatly to design metrics as it allows a non ambiguous interpretation and a more rigorous definition, which, in turn, can assist the implementation of tools to measure the software design quality for industrial application. Our achievement gives precise meaning to software design metrics definitions in order to facilitate verification and validation. We, especially, applied our approach to one of the most well known set of metrics: the CK metrics.

Keywords-formalization; UML metamodel; Z; CK metrics;

I. INTRODUCTION

“*Door meten tot weten*” [24] is a famous saying of the Dutch physicist and Nobel laureate Kamerlingh Onnes (1853 - 1926) literally translated as “Through measurement to knowledge”. It attests that the quantifying process leads to a better insight and understanding over the measured element. The software engineering area is no exception. It has been widely recognized that the use of software metrics, for being considered as quality indicators, can accurately help improve the final results and keep time and cost estimation under control while assuring quality according to the desired properties.

At first, code metrics such as cyclomatic complexity measure or lines of code measure were defined and applied to track faultiness during software development but have soon shown a weak side for being measured till the implementation phase, which is already a very late phase considering the whole software life cycle. Since then, many software metrics concerned with the design phase were defined and commonly known as design metrics. A combination of both code and design metrics has also been explored with positive results [25].

Several authors have proposed various design metrics such as the MOOD and MOOD2 (Metrics for Object-Oriented Design) [28], MOOSE (Metrics for Object-

Oriented Software Engineering) also known as the CK metrics [5], EMOOSE (Extended MOOSE) [29] and QMOOD (Quality Model for Object-Oriented Design) [30]. Most of them are lacking rigor and formalism in their definition.

This paper addresses the problematic lying in software measurement area due to the lack of formalization. Therefore, we present an approach to define formally software design metrics using the Z language [1, 2] over our proposed formal specification of the UML metamodel [3] based on the Laurent Henocque [4] transformation of UML class structures concept. This approach is intended to provide precise and complete formalized definition of software design metrics.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 presents a brief overview of the Z language. Section 4 illustrates the Z formalization of the UML metamodel. Section 5 introduces an approach to formalize software design metrics definition and finally, conclusions are drawn in Section 6.

II. RELATED WORK

Measurement has always been a fundamental step to understandability and control. When it comes to quality, measurement is obviously more difficult to obtain due to its subjectivity, however, some of its aspects can be measured and verified and thus be considered as objective. Software engineering, for being a very recent field and especially a more human-intensive discipline [26], suffers from a lack of measurement which, undeniably, leads to an out of control in delivery and cost estimation of the software production.

With a massive research concerns, measurement has reached an early stage of the software life cycle. Therefore, the software design metrics were defined according to the commonly approved properties considered as quality indicators.

Many software metrics exist nowadays [5-7] however their practical use remains unpopular in the software

industry mostly because of their ambiguity and non reliability [8]. Knowing that measurements have to be standard to mean the same thing to everyone, metrics should enforce their definitions using formal methods to become more useful, convenient and trust worthy.

Among authors who attempt to give a formal definition of software metrics, Baroni et al. [10], which proposed a Formal Library for Aiding Metrics Extraction (FLAME) [9] that uses OCL [11] as a metric definition language. El-Wakil et al. [12] built metric definitions using XQuery [13] language. McQuillan et al. [14] based their work on Baroni's approach and extended the UML metamodel 2.0 to offer a framework for metric definitions. Harmer and Wilkie [15] expressed metric definitions as SQL queries over a relational schema. Goulao et al. [16] also used the Baroni's approach for defining component based metrics and used the UML 2.0 metamodel as a basis for their definitions. In all related approaches, the UML metamodel is described in a subset of UML itself, supplemented by a set of well-formedness rules provided in OCL and natural language (English). Unfortunately, these approaches neither offer the possibility to check certain system properties nor they exclude the ambiguous use of UML itself to express the UML metamodel. Whereas in this article, there are two main contributions: the first contribution is to express UML metamodel in a formal language without any reflexive reference to UML, it results in more clarity. The second contribution is to express the CK metrics in a rigorous definition that enables to check certain system properties involving metrics. This could not be achieved with previous definitions using OCL.

In this paper, a Z formal model of UML metamodel is described. The model is enough general to express any set of metrics defined upon the UML metamodel 2.3. Then the authors provide a formal definition of the CK metrics. Expressing, for the first time, the CK metrics in a state-based formal method.

III. Z OVERVIEW

Z [1, 2] is a formal specification language originally created by J.-R. Abrial and then developed by the Programming Research Group at Oxford. Its notation is based upon set theory and mathematical logic, which consists in a first-order predicate calculus.

One aspect of the Z notation is the schemas. The notion of schema in Z is closely related to a class structure in Object-oriented concept. It combines two parts: a declaration part and a predicate part. Another particularity of Z is the use of types. Types in Z can be either basic or composite.

We used Z notation to build our formalization because of its maturity and the ability to check consistency of the design using proof theorems unlike the Object-Z [17] language, which was specifically developed to gain

facilities with object oriented specification aspects to the detriment of formalization advantages mentioned earlier for Z language.

Some authors proposed a formalization of UML class constructs using PVS specification language (PVS-SL) [31], a language based on higher-order logic, where relationships and other constituents of UML diagrams are represented as PVS theories. Other approaches suggested the use of Description Logics (DLs) [32-33] where Object-oriented concepts are modeled in means of *concepts* (unary relations) and *relations* (n-ary relations). However, most attempts were done using Z. Among them, there are Hall [18-19] and Hammond [20], which, in their approaches, supported class, association and inheritance. Malcolm Shroff and Robert B. France [21-22] based their approach on the Hall and Hammond's Z formalization approach of the class structures with the particularity of introducing inheritance relationship as an attribute in the inheriting class. We discarded Hall's original approach because it predates UML definition and it does not consider aggregation which is used in the core backbone of the UML metamodel. We also discarded France's modeling because it uses a global system approach, he models properties of objects as functions from identities to property values. This approach is less appealing than the intuitive encapsulation of each object's state which is more natural to object-oriented thinking.

After investigating these different methods, we choose the Laurent Henocque approach [4], which was elaborated to give a formal specification to Object Oriented Constraint Programs. This choice is mostly justified by the approach to represent inheritance and aggregation relationships and also its responds to our need for a formalization of the object system as part of the specification.

Since the objective of this paper is to present a formalization of design metrics, we settled for providing a description of the Henocque approach [4], gradually through our formalization of the UML metamodel.

IV. Z FORMALIZATION OF THE UML METAMODEL

The UML metamodel is the result of many years of effort to standardize software engineering practices. Itself defined in UML, it is considered as the standard model to represent object models using UML. The following transformation concerns the core backbone of the UML metamodel, captured and reconstituted from the UML metamodel 2.3.

A. Different Level of Abstractions of the Metrics

Definition of each metric considered in the formalization is done upon the UML metamodel at different levels of abstraction:

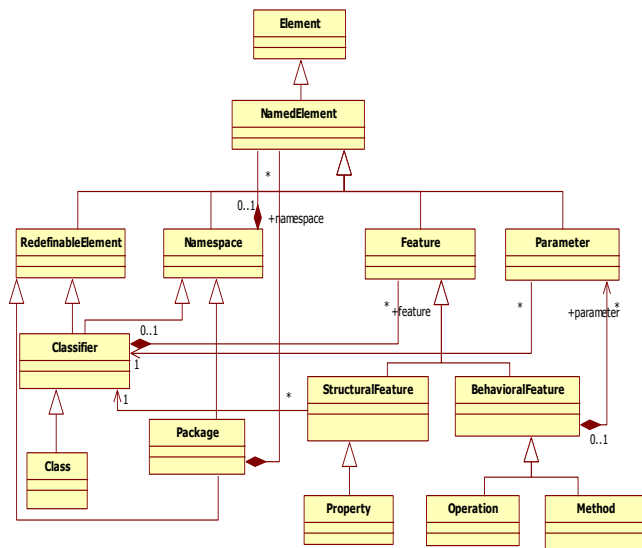


Figure 1. A fragment of the core backbone of the UML metamodel

B. Z Transformation

The following formalization is analyzed and validated using Z/EVES tool [23].

At the beginning, Laurent Henocque [4] defines an uninterpreted dataType [*ObjectReference*] considered as a set of object references and [*ReferenceSet*] as a finite set of object references later used to model object types.

$$ReferenceSet == F ObjectReference$$

For practical reasons, a global class names is defined using free type declaration syntax:
 $CLASSNAME ::= ClassElement | ClassNamedElement | \dots$

A function instances describes the mapping between class names and the set of instances of that class

$$instances: CLASSNAME \to ReferenceSet$$

And then, he defines *ObjectDef* as a predefined super class for all future classes. This class will be used to bijectively map each object to a unique individual from the set ObjectReference.

An instance of each class presented is identified by its respective object identifier *ident* which is of type declared as a basic type.

$$\begin{array}{|l} \hline \mathbf{ObjectDef} \\ \hline ref: ObjectReference \\ class: CLASSNAME \\ \hline \end{array}$$

For our metrics transformation, we extend the ObjectDef with a NIL object to represent a undefined object.

$$NIL: ObjectDef$$

According to Henocque [4], each class is implemented via two constructs:

- **A class definition:** a schema in which we find, in its invariant part, both the class attributes and the inheritance relationships and in its predicate part, specification of class invariants.

$$\begin{array}{|l} \hline \mathbf{ClassDefElement} \\ \hline name: seq CHAR \\ \hline \end{array}$$

with [CHAR] being a given set containing all characters. The attribute name was introduced in this transformation because the Z/EVES tool [23] does not allow the construction of an empty class. In the following, even though the UML metamodel class constructs contains attributes and predicates, we will only focus on the relationship between classes in order to simplify readability of our metrics transformation.

- **A class specification:** a combination of a class definition extended with the ObjectDef and class references.

$$\begin{array}{|l} \hline \mathbf{ClassSpecElement} \cong ClassDefElement \wedge [ObjectDef \\ | class = ClassElement] \\ \hline \end{array}$$

The \cong symbol offers a different way to define a schema and the logical operator \wedge allows the extension.

As stated in the first part of the class constructs, inheritance relationship is defined in the class definition:

$$\begin{array}{|l} \hline \mathbf{ClassDefNamedElement} \\ \hline ClassDefElement \\ \hline \end{array} \quad \begin{array}{|l} \hline \mathbf{ClassDefClassifier} \\ \hline ClassDefNamespace \\ ClassDefRedefinableElement \\ \hline \end{array}$$

In both cases, simple inheritance or multiple inheritance, the inheritance relationship is built simply by importing the schema definition of inherited superclasses into the class that inherit from them.

Beside the inheritance relationship, we are also concerned with the aggregation and relations with multiplicities. General relations are free of constraints, which mean that every tuple can be accepted. The multiplicity is naturally stated in the predicate part as the cardinal of related target objects for each source object.

$$\begin{array}{|l} \hline pc: Parameter \leftrightarrow Classifier \\ \hline \forall c: Classifier \cdot \#(pc (\{c\})) < 1 \\ \hline \end{array}$$

The aggregate relation is more constrained than a general one, thus we have to change the type of relation to make a distinction between both. In the different aggregate relations given in our UML metamodel fragment, the multiplicity is of 0..1 which means that each component occurs in at most one composite. Consequently, its relational inverse is an injective partial function.

```

hasNamedElement: Namespace ↔ NamedElement
-----
hasNamedElement ~ ∈ NamedElement ↗→ Namespace
    
```

The ↗→ symbol represents the partial function and the ~ stands for the relational inverse.

And finally, we define class types for a better understanding of what the types really represent. They are defined using an axiomatic definition:

```

Element, NamedElement, Namespace, ... : ReferenceSet
-----
Element = instances ClassElement ∪ NamedElement
NamedElement
= instances ClassNamedElement ∪ Namespace ∪
RedefinableElement ∪ Feature
...
instances ClassElement
= { o: ClassSpecElement | o.class = ClassElement • o.ref }
instances ClassNamedElement
= { o: ClassSpecNamedElement | o.class = ClassNamedElement
• o.ref }
...
∀ i: instances ClassElement • ∃ x: ClassSpecElement • x.ref = i
∀ i: instances ClassNamedElement • ∃ x: ClassSpecNamedElement
• x.ref = i
...
    
```

The type sets defined in the declaration part correspond to the existing classes of our given model. Each type is defined as a finite set of object references. The predicate part describes the properties of these sets. First, we have a type equal to the union of the corresponding class instances and the type of all its subclasses. And then, that each object reference is used at most once for an object which means that no two distinct object bindings share the same object reference.

V. AN APPROACH TO FORMALIZE DESIGN QUALITY METRICS DEFINITIONS

Among existing metrics, we will discuss the CK Metrics [5] proposed by Chidamber and Kemerer, one of the most well known suites of Object-oriented metrics. These metrics help measuring different aspects of an Object-Oriented design including complexity, coupling and cohesion. Several studies [26-27] have confirmed their usefulness as quality indicators.

An OCL formalization of the CK metrics was proposed by the authors Baroni et al. [10], defined using functions formalized in FLAME [9]. Although, OCL is based on mathematical logic, it still does not provide a formally defined semantics, furthermore, its syntax is given by a grammar description and no metamodel is available unlike the metamodel of UML which means that it suffers from an absence of well-formedness rules.

Considering that most metrics formalization efforts are made in OCL but yet still unpopular in the software industry, we argue that a more rigorous method of formalization should be explored in order to overcome OCL limitations.

As a simple example, the expression *iterate*, used in the OCL formalization of the DIT metrics, is known to be potentially non-deterministic since there is no precision on order evaluation leading to different possible results[34].

Classifier:: DIT(): Integer

```

= if self.isRoot( ) then 0
else if PARN( ) = 1 then
1 + self.parents( ) -> iterate( elem:
GeneralizableElement; acc: Integer = 0
| acc + elem.oclAsType( Class ).DIT( ) )
else
self.parents( ) -> iterate( elem: GeneralizableElement;
acc: Integer = 0
| acc + elem.oclAsType( Class ).DIT( ) )
endif
endif
    
```

Also, in each metrics defined with OCL, we could find many OCL keywords (*self*, *asSet*...) and predefined functions (*OclAsType*, *OclIsKindOf*...) that are not precise enough semantically. Therefore, we propose a formal definition for those frequently used predefined functions in order to obtain a complete and precise definition of the CK metrics.

A. Formalizing OCL Predefined Functions

OclIsTypeOf and *OclIsKindOf* have the same signature. They are both applied to an object, take a type as parameter and return a Boolean as a result. The only difference is that the first one deals with the direct type of the object when the second one determines whether the type given in parameter is either the direct type or one of the supertypes of the object.

When it is certain that the actual type of the object is the subtype, the object can be re-typed using the *OclAsType* operation. Otherwise, the expression is undefined.

We propose a Z-formalization of these predefined operations using the Henocque approach [4].

$oclIsTypeOf: ObjectDef \times ReferenceSet \rightarrow Boolean$

$\forall o: ObjectDef; t: ReferenceSet \mid instances\ o.class = t \cdot oclIsTypeOf(o, t) = TRUE$

$\forall o: ObjectDef; t: ReferenceSet \mid instances\ o.class \neq t \cdot oclIsTypeOf(o, t) = FALSE$

The formalization is given as an axiomatic function. It takes the ObjectDef and a ReferenceSet as parameter and it returns a Boolean. When instances of o.class refering to the object's type is equal to the type given in parameter the expression of OclIsTypeOf is true. When both types are not the same, the operation return false.

$oclIsKindOf: ObjectDef \times ReferenceSet \rightarrow Boolean$

$\forall o: ObjectDef; t: ReferenceSet \mid instances\ o.class \subseteq t \cdot oclIsKindOf(o, t) = TRUE$

$\forall o: ObjectDef; t: ReferenceSet \mid \neg instances\ o.class \subseteq t \cdot oclIsKindOf(o, t) = FALSE$

When the type of the object given in parameter (expressed as instances o.class) is part of the ReferenceSet given in parameter, the expression oclIsKindOf returns true. Otherwise, it returns false.

$oclAsType: ObjectDef \times ReferenceSet \rightarrow ObjectDef$

$\forall o: ObjectDef; t: ReferenceSet \mid instances\ o.class = t \cdot oclAsType(o, t) = o$

$\forall o: ObjectDef; t: ReferenceSet \mid \neg instances\ o.class \subseteq t \cdot oclAsType(o, t) = NIL$

$\forall o: ObjectDef; t: ReferenceSet \mid instances\ o.class \subseteq t$
 $\cdot \exists r: ObjectDef \mid r.ref = o.ref \wedge instances\ r.class = t \cdot oclAsType(o, t) = r$

With oclAsType operation we distinguish between three cases:

The first one is when the type given in parameter corresponds to the object's type, which means the result of applying oclAsType is the object itself.

The second one is when the object's type is not the same nor is it a part of the ReferenceSet given in parameter, which means that the expression is undefined and in that case we return the NIL value defined earlier as an extension to ObjectDef.

Finally, the third one is when the object's type is part of the ReferenceSet given in parameter. In that case, the expression OclAsType returns an object which has the same reference as the object in entry (that means it is the same object) but having as type the ReferenceSet in parameter.

B. Formalizing the CK metrics

Each of the above metrics refers to an individual class and not to the whole system.

- **Weighted Methods Complexity:** the sum of the complexity of all methods for a class. If all method complexities are considered to be unique, WMC is equal to the number of methods.

$WMC: ObjectDef \times Classifier \rightarrow \mathbb{N}$

$\forall o: ObjectDef; c: Classifier; S: \mathbb{P} Operation \mid S = allOperations(o, c)$
 $\cdot WMC(o, c) = \# S$

- **Number of Children:** counts the number of children classes that inherit directly from the current class.

$NOC: ObjectDef \times Classifier \rightarrow \mathbb{N}$

$\forall o: ObjectDef; c: Classifier; n: \mathbb{N} \mid n = CHIN(o, c) \cdot NOC(o, c) = n$

- **Depth of Inheritance Tree:** measures the length of the inheritance chain from the current class to the root.

$DIT: ObjectDef \times RedefinableElement \rightarrow \mathbb{N}$

$\forall o: ObjectDef; r: RedefinableElement \mid isRoot(o, r) = TRUE \cdot DIT(o, r) = 0$

$\forall o: ObjectDef; r: RedefinableElement; R: \mathbb{P} RedefinableElement; n: \mathbb{N}; S: \mathbb{P} \mathbb{N}$

$\mid PARN(o, r) \geq 1$
 $\wedge R = parents(o, r)$
 $\wedge S = \{ depth: \mathbb{N} \mid \forall r': R \cdot depth = DIT(o, r') \}$
 $\wedge n = max\ S \cdot DIT(o, r) = n$

- **Coupling Between Classes:** the number of coupling with other classes.

$CBO: ObjectDef \times Classifier \rightarrow \mathbb{N}$

$\forall o: ObjectDef; c: Classifier; C: \mathbb{P} Classifier \mid C = coupledClasses(o, c)$

$\cdot CBO(o, c) = \# C$

- **Response for Class:** the number of methods in the current class that might respond to a message received by its object, including methods both inside and outside of this class. It can be defined as $\mid RS \mid$ where RS is the response set for the class expressed as:

$$RS = \{ M \} \cup all\ i \{ R\ i \}$$

with:

- $\{ R\ i \}$ = set of methods called by method i
- $\{ M \}$ = set of all methods in the class.

$RFC: ObjectDef \times Classifier \rightarrow \mathbb{N}$

$\forall o: ObjectDef; c: Classifier; m, mc: \mathbb{P} Operation$
 $\mid m = allOperations(o, c) \wedge mc = allClientOperations(o, c)$
 $\bullet RFC(o, c) = \#m + \#mc$

- **Lack of Cohesion of Methods:** The degree of similarity of methods in the current class. This metric was first improved by Chidamber and Kemerer themselves, calling it LCOM2, then by Henderson-Sellers by proposing the following expression:

$$LCOM3 = (m - \text{sum}(mA) / a) / (m - 1)$$

with:

- **m:** number of methods in a class.
- **a:** number of attributes in a class.
- **mA:** number of methods that access the attribute a.
- **sum(mA):** sum of all mA over all the attributes in the class.

$LCOM3: ObjectDef \times Classifier \rightarrow \mathbb{N}$

$\forall o: ObjectDef; c: Classifier; m: \mathbb{P} Operation; a: \mathbb{P} Property; n: \mathbb{N}$
 $\mid m = allOperations(o, c)$
 $\wedge a = allAttributes(o, c)$
 $\wedge (\forall A: a \cdot n = n + \text{sum} \langle mA(A, m) \rangle)$
 $\bullet LCOM(o, c) = (\#m - n \text{ div} \#a) \text{ div} \#a - 1$

VI. CONCLUSION AND FUTURE WORK

In this work, we were mainly concerned about the formal definition of the CK metrics as a restricted application of our formalization approach, which consists on expressing formally the UML metamodel and then giving a formal definition of software design quality metrics for the sake of validation and verification.

As future work, we plan to extend our contribution to MOOD and MOOD2 - Metrics for Object-Oriented Design [28], EMOOSE- Extended MOOSE [29] and QMOOD Quality Model for Object-Oriented Design [30]. We, also, plan to build a support tool that will, first, automate the formal Z representation of design models according to our UML metamodel formalization and then, implement already formalized metrics expressions to automate their calculation and compare results.

REFERENCES

- [1] M. Spivey, "The Z Notation," Prentice-Hall, 1992.
- [2] J. Woodcock and J. Davies, "Using Z: Specification, Proof and Refinement," Prentice Hall International Series in Computer Science, 1996.
- [3] The Object Management Group, UML 2.3 superstructure specification, 2010 <http://www.omg.org/spec/uml/2.3/>
- [4] Laurent Henocque, "Z specification of Object Oriented Constraint Programs," RACSAM, 2004.
- [5] Shyam R. Chidamber and Chris F. Kemerer, "A metric suite for Object Oriented Design," Journal IEEE Transactions on Software Engineering Volume 20 Issue 6, 1994, pp. 476 – 493.
- [6] Lorenz M. and Kidd J., "Object-Oriented Software Metrics," Prentice Hall Object-Oriented Series, 1994.
- [7] Norman E. Fenton and Lawrence Peeger S., "Software Metrics: A Rigorous and Practical Approach," International Thompson Computer Press, 1996.
- [8] L. Briand, J. Daly and J.Wüst, "A unified framework for coupling measurement in object-oriented systems," IEEE Transactions on Software Engineering 25, 1999.
- [9] Aline L. Baroni and F. Brito e Abreu, "An OCL-Based Formalization of the MOOSE Metric Suite," In Proceedings of the 7th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QUAOOSE'2003), Darmstadt, Germany, 2003.
- [10] Aline L. Baroni and F. Brito e Abreu, "A Formal Library for Aiding Metrics Extraction," International Workshop on Object-Oriented Re-Engineering at ECOOP, 2003.
- [11] The Object Management Group, Object Constraint Language 2.2, 2010 <http://www.omg.org/spec/OCL/2.2/>
- [12] Mohamed M. El-Wakil, A. El-Bastawisi, Mokhtar B. Riad, and A. Fahmy., "A novel approach to formalize Object-Oriented Design," 9th International Conference on Empirical Assessment in Software Engineering (EASE 2005), April 2005.
- [13] XQuery 1.0 Standard by W3C XML Query Working Group. <http://www.w3.org/TR/2010/REC-xquery-20101214/>
- [14] Jacqueline A. McQuillan and James F. Power, "Towards re-usable metric definitions at the meta-level," In *PhD Workshop of the 20th European Conference on Object-Oriented Programming (ECOOP 2006)*, Nantes, France, 3-7, July 2006.
- [15] F. Wilkie And T. Harmer, "Tool support for measuring complexity in heterogeneous object-oriented software," In Proceedings of IEEE International Conference on Software Maintenance, Montreal, Canada, 2002.
- [16] M. Goulao and F. Brito e Abreu, "Formalizing metrics for COTS," In Proceedings of the ICSE Workshop on Models and Processes for the Evaluation of COTS Components, Edinburgh, Scotland, 2004.
- [17] D. Duke, P. King, G.A. Rose, and G. Smith, 1991. The Object-Z Specification Language, version 1, Technical Report 91-1, Department of Computing Science, University of Queensland, Australia.
- [18] J. A. Hall, "Specifying and Interpreting Class Hierarchies in Z," In Bowen and Hall, pp. 120-138.
- [19] J.P. Bowen and J.A. Hall, editors, Z User Workshop, Cambridge 1994, Workshops in Computing. Springer-Verlag, New York, 1994.
- [20] J. A. R. Hammond, "Producing Z specifications from Object-Oriented Analysis," In Bowen and Hall, pp. 316-336.
- [21] Robert B. France, J.-M. Bruel, M. M. Larrondo-Petrie, and M. Shroff, "Exploring the Semantics of UML Type Structures with Z", In: Proceedings of the Formal Methods for Open Object-based Distributed Systems (FMOODS'97), Springer, pp. 247-257.
- [22] M. Shroff and Robert B. France, "Towards a Formalization of UML Class Structures in Z", In Proceedings of the 21st Computer Software and Application Conference (COMP-SAC'97), IEEE Press, 646-651.

- [23] I. Meisels and M. Saaltink, The Z/EVES 2.0 Reference Manual. Technical Report TR-99-5493-03e, ORA Canada, October 1999.
- [24] 'The Significance of Quantitative Research in Physics', Inaugural Address at the University of Leiden (1882). In Hendrik Casimir, Haphazard Reality: Half a Century of Science (1983), 160-1.
- [25] Y. Jiang, B. Cukic, T. Menzies, and N. Bartlow: "Comparing Design and Code Metrics for Software quality Prediction": PROMISE (2008).
- [26] Sandro Morasca: Software Measurement (2007).
- [27] Victor R. Basili, Fellow, IEEE, Lionel C. Briand, and Walcelio L. Melo: "A Validation of Object-Oriented Design Metrics as Quality Indicators": In IEEE Transactions on Software Engineering, vol. 22, NO. 10, October 1996, pp. 751 – 761,
- [28] F. Brito e Abreu and R. Carapuça, "Object-Oriented Software Engineering: Measuring and Controlling the Development Process," 4th Int. Conf. on Software Quality, McLean, VA, USA, 3-5 October 1994.
- [29] W. Li, S. Henry, D. Kafura and R. Schulman, "Measuring object-oriented design," *Journal of Object-Oriented programming*, vol. 8, NO. 4, pp. 48-55. July/August 1995.
- [30] J. Bansiya and C. Davids: "Automated metrics and object-oriented development," *Dr. Dobbs Journal*, pp. 42–48, December 1997.
- [31] Demissie B. Aredo, I. Traore, and K. Stølen: "Towards a formalization of UML Class Structure in PVS," Research Report no. 272, Department of Informatics, University of Oslo, August 1999.
- [32] A. Cali, D. Calvanese, G. De Giacomo, and M. Lenzerini: "Reasoning on UML Class Diagrams in Description Logics," In Proceedings of IJCAR Workshop on Precise Modelling and Deduction for Object-oriented Software Development (PMD 2001). 2001.
- [33] L. Efrizoni, W.M.N Wan-Kadir and, R. Mohamad: "Formalization of UML Class using Description Logics," In the International Symposium in Information Technology (ITSim), 2010.
- [34] M. Richters and M. Gogolla: "On Formalizing the UML Object Constraint Language," In Proceedings of the 17th International Conference on Conceptual Modeling. Springer-Verlag, London, UK, 1998.

Appendix:

The whole specification, of 426 lines in Latex, was entirely written and verified using the Z/EVES tool but for space reason, this appendix does not contain all of it.

The following is a description of the previously declared functions in the metrics formalization chapter.

%Subset of Properties (from one set of Features) belonging to the current Classifier.

feature2AttributeSet: ObjectDef × P Feature → P Property

$$\begin{aligned} & \forall o: \text{ObjectDef}; S: \mathbb{P} \text{Feature} \\ & \quad | \text{instances } o.\text{class} = \text{Feature} \\ & \quad \wedge S = \{ f: \text{Feature} \mid \text{oclIsKindOf}(o, \text{Property}) = \text{TRUE} \} \\ & \quad \bullet \text{feature2AttributeSet}(o, S) = \{ f: S \mid \text{oclAsType}(o, \text{Property}) = o \} \end{aligned}$$

%Subset of Operations (from one set of Features) belonging to the current Classifier.

feature2OperationSet: ObjectDef × P Feature → P Operation

$$\begin{aligned} & \forall o: \text{ObjectDef}; S: \mathbb{P} \text{Feature} \\ & \quad | \text{instances } o.\text{class} = \text{Feature} \\ & \quad \wedge S = \{ f: \text{Feature} \mid \text{oclIsKindOf}(o, \text{Operation}) = \text{TRUE} \} \\ & \quad \bullet \text{feature2OperationSet}(o, S) = \{ f: S \mid \text{oclAsType}(o, \text{Operation}) = o \} \end{aligned}$$

%Set of Features declared in the Classifier, including overridden Operations.

definedFeatures: ObjectDef × Classifier → P Feature

$$\begin{aligned} & \forall o: \text{ObjectDef}; c: \text{Classifier}; p: \mathbb{P} \text{Feature} \\ & \quad | \text{instances } o.\text{class} = \text{Feature} \wedge p = \{ f: \text{Feature} \mid f \in \text{Classifier} \} \\ & \quad \bullet \text{definedFeatures}(o, c) = p \end{aligned}$$

%Set of Classes from which the current GeneralizableElement derives directly.

parents: ObjectDef × RedefinableElement → P RedefinableElement

$$\begin{aligned} & \forall o: \text{ObjectDef}; r: \text{instances } \text{ClassRedefinableElement} \\ & \quad | \text{instances } o.\text{class} = \text{RedefinableElement} \\ & \quad \bullet \text{parents}(o, r) \\ & \quad \quad = \{ r': \text{RedefinableElement} \\ & \quad \quad \quad | \text{instances } \text{ClassRedefinableElement} \subset \text{instances } o.\text{class} \} \end{aligned}$$

%Set of directly derived Classes of the current GeneralizableElement.

children: ObjectDef × RedefinableElement → P RedefinableElement

$$\begin{aligned} & \forall o: \text{ObjectDef}; r: \text{RedefinableElement} \mid \text{instances } o.\text{class} = \\ & \text{RedefinableElement} \\ & \quad \bullet \text{children}(o, r) \\ & \quad \quad = \{ r': \text{RedefinableElement} \\ & \quad \quad \quad | \text{instances } o.\text{class} \subset \text{instances } \text{ClassRedefinableElement} \} \end{aligned}$$

%Number of directly derived Classes.

CHIN: ObjectDef × RedefinableElement → N

$$\begin{aligned} & \forall o: \text{ObjectDef}; r: \text{RedefinableElement}; S: \mathbb{P} \text{RedefinableElement} \\ & \quad | S = \text{children}(o, r) \bullet \text{CHIN}(o, r) = \# S \end{aligned}$$

%Number of Classes from which the current RedefinableElement derives directly.

$PARN: ObjectDef \times RedefinableElement \rightarrow \mathbb{N}$

$\forall o: ObjectDef; r: RedefinableElement; S: \mathbb{P} RedefinableElement$
 $| S = parents(o, r) \cdot PARN(o, r) = \# S$

%Indicates whether the RedefinableElement has ascendants or not.

$isRoot: ObjectDef \times RedefinableElement \rightarrow Boolean$

$\forall o: ObjectDef; r: RedefinableElement | PARN(o, r) = 0 \cdot isRoot(o, r) = TRUE$

$\forall o: ObjectDef; r: RedefinableElement | PARN(o, r) \neq 0$
 $\cdot isRoot(o, r) = FALSE$

%Set containing all Features of the Classifier itself and all its inherited Features.

$allFeatures: ObjectDef \times Classifier \rightarrow \mathbb{P} Feature$

$\forall o: ObjectDef; c: Classifier; r: RedefinableElement$
 $\cdot allFeatures(o, c) = \cup \{allFeatures(oclAsType(o, Classifier), c)\}$

% Set containing all Properties of the Classifier and all its inherited Attributes (directly and indirectly).

$allAttributes: ObjectDef \times Classifier \rightarrow \mathbb{P} Property$

$\forall o: ObjectDef; c: Classifier; S: \mathbb{P} Property$
 $| S = feature2AttributeSet(o, (allFeatures(o, c)))$
 $\cdot allAttributes(o, c) = S$

% Set containing all Operations of the Classifier itself and all its inherited Operations.

$allOperations: ObjectDef \times Classifier \rightarrow \mathbb{P} Operation$

$\forall o: ObjectDef; c: Classifier; S: \mathbb{P} Operation$
 $| S = feature2OperationSet(o, (allFeatures(o, c)))$
 $\cdot allOperations(o, c) = S$

% Types (Classifiers) of all attributes that are accessible within the current Classifier.

$typesOfAllAccessibleAttributes: Classifier \rightarrow \mathbb{P} Classifier$

$\forall o: ObjectDef; c: Classifier; S: \mathbb{P} Property; F: \mathbb{P} Feature; T: \mathbb{P} Classifier$
 $| S = allAttributes(o, c) \wedge feature2AttributeSet(o, F) = S \wedge F \subseteq T$
 $\cdot typesOfAllAccessibleAttributes c = T$

% True if the first Classifier has an accessible attribute of type given as second Classifier.

$hasAttribute: Classifier \times Classifier \rightarrow Boolean$

$\forall c, c': Classifier | c' \in typesOfAllAccessibleAttributes c$
 $\cdot hasAttribute(c, c') = TRUE$
 $\forall c, c': Classifier | c' \notin typesOfAllAccessibleAttributes c$
 $\cdot hasAttribute(c, c') = FALSE$

% Set of Classifiers to which the current Classifier is coupled (excluding inheritance).

$coupledClasses: Classifier \rightarrow \mathbb{P} Classifier$

$\forall c: Classifier; S: \mathbb{P} Classifier$
 $| S = \{ c': Classifier | hasAttribute(c, c') = TRUE \}$
 $\cdot coupledClasses c = S$

% Set of Operations that might respond to a message received by its object.

$allClientOperations: ObjectDef \times Classifier \rightarrow \mathbb{P} Operation$

$\forall o: ObjectDef; c: Classifier; C: \mathbb{P} Classifier; M: \mathbb{P} Operation$
 $| coupledClasses c = C \wedge M = \cup \{ c': C \cdot (allOperations(o, c')) \}$
 $\cdot allClientOperations(o, c) = M$

E-FOTO: Development of an Open-Source Educational Digital Photogrammetric Workstation

Jorge Luís Nunes e Silva Brito, Rafael Alves de Aguiar, Marcelo Teixeira Silveira, Luiz Carlos Teixeira Coelho Filho, Irving da Silva Badolato, Paulo André Batista Pupim, Patrícia Farias Reolon, João Araújo Ribeiro, Jonas Ribeiro da Silva, Orlando Bernardo Filho, Guilherme Lucio Abelha Mota

School of Engineering
Rio de Janeiro State University
Rio de Janeiro, Brazil

jsilvabr@gmail.com, rafael.kamui@gmail.com, marts@ele.puc-rio.br, luizcoelho@luizcoelho.com,
irvingbadolato@gmail.com, pandreengineer@gmail.com, patricia.reolon@gmail.com, joao.araujo@gmail.com,
jonas.xiko.ribeiro@gmail.com, orlandobernardof@yahoo.com.br, guimota@gmail.com

Abstract—The E-FOTO project aims to develop an educational, digital photogrammetric workstation, under the General Public License (GNU/GPL). E-FOTO software does not intend to overcome commercial software solutions in its field. Its main purpose is to provide a software solution that could be used by anyone who is interested in learning digital photogrammetry. E-FOTO users are offered access to practical software applications of theoretical issues and concepts in the field of Digital Photogrammetry. The project also has an Internet home page from which aerial photographs, camera calibration certificates and other technical data can be downloaded, thus allowing users to fully experience the software. The webpage also provides access to different articles and publications derived from the project development. This paper will report the experiences and results of the development of E-FOTO, which is, to the best of our knowledge, the only academic, GNU/GPL software development initiative for Digital Photogrammetry in the world.

Keywords—*Digital Photogrammetry; Digital Photogrammetric Workstation; Education in Photogrammetry; Open-source code for geospatial applications; Extreme Programming; Agile Methods.*

I. INTRODUCTION

Photogrammetry is the science that studies methods for reconstructing 3D objects, e.g., Earth's surface, from a set of 2D stereoscopic images. Remote Sensing techniques are typically used for the acquisition of photogrammetric imagery. Currently, it is possible to find a reasonable number of imaging satellites orbiting around the Earth. These devices have achieved spatial resolutions as low as 0.50m per pixel. Airborne remote sensors which use optical devices coupled with Global Navigation Satellite Systems (GNSS) receivers and Inertial Measurement Units (IMU) also play a very important role in topographic mapping activities. Those activities include the production of digital surface models (DSM), the production of ortho-images, and the generation of 3D spatial databases. These products are typically named "cartographic products".

A digital photogrammetric workstation is a set of hardware and software that is able to generate cartographic

products. Hardware components include a digital computer and other devices such as topomouses, digitizing tablets and stereoscopic glasses. Some examples of computer software developed for a digital photogrammetric workstation are photogrammetric algorithms for automatic DSM extraction and digital ortho-image rectification.

The cost of a digital photogrammetric workstation ranges from US\$ 25,000.00 for educational versions up to US\$ 100,000.00 for professional solutions. These prices are too expensive for educational institutions in developing countries.

As a response to such high prices, Coelho Filho [1], proposed the development of a digital photogrammetric softcopy kit for educational purposes. His solution was designed and implemented under the General Public License (GNU/GPL) of the Free-Software Foundation. The kit was named "E-FOTO", which stands for "Educational Digital Photogrammetric Workstation" in Portuguese.

There were many free software solutions for geospatial applications. One could mention POSTGRES, MAPSERVER, GvSIG and GRASS GIS, among others. However a digital photogrammetric workstation with a free GNU/GPL license did not exist yet. In addition to providing users free access to photogrammetric software, a free softcopy kit would allow developers to examine and improve its source code and produce derived projects. Consequently, users would be able to understand photogrammetric algorithms and techniques, while also using their code for various purposes.

It is worth mentioning that the E-FOTO Project was not designed for competition with its commercial counterparts. E-FOTO's main objective is to provide access to Digital Photogrammetry to anyone who is interested in learning about that subject. As a result, E-FOTO users will better understand and practice the theoretical concepts taught in formal classrooms.

A paperback book on Digital Photogrammetry [2] was published parallel to E-FOTO's project development. This book, which is available only in Portuguese, covers most theoretical concepts used for the development of the E-FOTO software. It is currently in its first edition, published

in 2007, even though a provisional e-book version of it had been available since 2002, when the project started.

The development of Photogrammetric software is a complex task. In fact, it involves a multidisciplinary team of professionals with different backgrounds and skills, mostly dealing with Geomatics and Computer Science. In 2004, the project was granted sponsorship by the National Council for Scientific and Technological Development of Brazil (in Portuguese, CNPq), which allowed it to be reallocated to the School of Engineering of The Rio de Janeiro State University. With proper funding, its software development was carried out in a more consistent way. It was supported by both Masters Dissertations and Undergraduate Final projects. In 2008, after having completed all of its modules, the E-FOTO development team started their integration. Finally, in November 2010, the first fully integrated version of E-FOTO was published on the project's website (www.efoto.eng.uerj.br).

This article is organized according to the following structure: Section II presents experiences of the E-FOTO team during the development of the aforementioned software; Section III reports the project's benefits; Section IV envisions future work to be done and Section V concludes this paper.

II. THE E-FOTO DEVELOPMENT EXPERIENCE

E-FOTO's team has dealt with many issues which arose from the development of such a complex system. This section reports E-FOTO's development approaches and experiences.

A. Academic Work and the development of E-FOTO's Photogrammetric Modules

E-FOTO was originally designed in 2002 as a system composed of eight modules: (a) image rectification, (b) interior orientation, (c) exterior orientation, (d) bundle block adjustment, (e) image normalization, (f) photogrammetric stereoplotting, (g) extraction of digital elevation models and (h) ortho-rectification.

In accordance with free software principles, a free digital photogrammetric workstation should not contain any part of its code based upon proprietary solutions. Thus, it was decided that the project would be developed using free environments such as GCC/G++ (C++) and the Qt graphic user interface for easy multiplatform porting. At first, an interior orientation module was implemented as a proof of concept in 2002.

After E-FOTO was granted sponsorship in 2004, an official schedule was proposed, which aimed to complete the proposed modules in two years, with the possibility of a one year extension. Development effectively began at its new venue, The Rio de Janeiro State University.

Initially, photogrammetric modules were developed as part of undergraduate and graduate students' academic work, often counting on proprietary software, such as Mathcad and Matlab. This step allowed the team to test if algorithms were fully functional and correctly produced desired results. The digital image rectification [3][15][20], exterior orientation [4][22] and bundle block modules [5][22] were developed in

the Mathcad environment. An image ortho-rectification module [8][22] was developed with Matlab.

Some other modules were implemented from the beginning in C++. For example, a digital airborne imagery stereo-visualization and measurement module [7][14][24] was developed and used for stereoplotting and digital elevation model extraction. An image normalization module [6][20] was also developed. Figure 1 shows photogrammetric stereoplotting using E-FOTO.



Figure 1. The photogrammetric stereoplotting module.

Modules initially developed with mathematical software were gradually reprogrammed by team members. The first rewritten module was the image rectification module. Next, the stereoplotting module was integrated with an ortho-rectification solution. With the release of these first modules, it was possible to assign students to test the software and write down their impressions about its ease of use. This helped find bugs and adapt E-FOTO's interface so it would be more user-friendly. New modules were built based on these impressions. So, programs for exterior orientation, bundle block adjustment and normalization were developed according to this new philosophy and look and feel while earlier models were adapted to conform to an easier graphic interface. Figure 2 shows the exterior orientation module after the proposed modifications.

In early 2007, the deadline for the creation of the modules was successfully met by the team. However, their integration was not yet desirable. Thus, the need for integration among these modules using techniques of systems analysis became evident. Additionally, another problem arose: in early 2006, Trolltech (Qt's former developer) migrated it from version 3 to version 4. The latter introduced several changes which turned it incompatible with earlier Qt versions. The project's development team decided that porting all modules from Qt3 to Qt4 would be counterproductive at that moment. So, it was decided that all modules would continue to be implemented in Qt3, and then ported as a whole to Qt4.

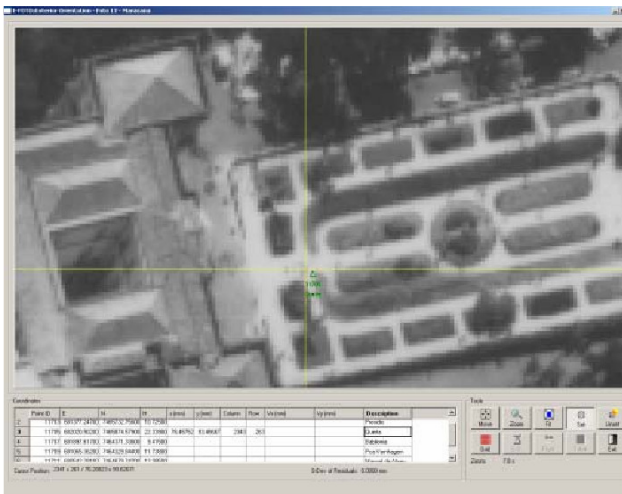


Figure 2. The exterior orientation module.

B. Migration from Qt3 to Qt4

After Trolltech stopped providing support for its Qt3 framework, in July 2007, E-FOTO developers were forced to start porting its code to Qt4, to continue benefiting from future Qt updates. Initially, it was expected that the conversion could be done without major problems, by using automated migration tools provided by Trolltech. However, due to certain incompatibilities between the two versions, this migration has been very difficult. The main problems experienced throughout the porting process were related to interface and graphic content conversions, since the main resources used for these developments in the previous version, such as bit block transfer (bit-blt), were discontinued. Thus, modules that used these resources, such as the photogrammetric stereoplotting module, had to be more intensely reworked. As a consequence of this migration process, interfaces for currently finished modules had their appearance and functionality slightly changed. Drastic changes were minimized in order to reduce their impact to the end user [12].

One of the most unexpected changes the Qt4 graphics engine brought was the adoption of the Scalable Vector Graphics (SVG) format as the basis for its main graphic elements display system. Even though this format has been widely used in the development of graphics applications, especially as the basis of the widely-used Flash format, it has a notable disadvantage: it needs some support from the operating system to deliver expected results. This does not pose a problem when using a Windows OS, but when a Linux distribution is used, serious performance issues can be experienced, especially when working with images displayed outside of their standard resolution, i.e. with a different scale factor than 1:1.

Since one of the main goals of the E-FOTO project is to provide an easily portable software, both in terms of operating system and hardware resources, the development team sought alternatives to Qt4's native graphics display system. OpenGL was finally picked as the ultimate solution, both because of its performance, since it was created

specifically for the direct use of the resources present in commercially available graphics cards, and because of its easy integration with most frameworks for developing graphical user interfaces (GUIs). Qt provides support for displaying processed graphics through the OpenGL standard, so the team chose to keep it for other graphical elements (windows, menus, buttons, etc.), thus avoiding the extra effort of adapting the code to redraw those elements. Also, it helped keep the look and feel of the programs similar to their earlier versions [13].

E-FOTO's 1.0 version already had a couple models fully converted to Qt4: interior and exterior orientation modules. Those two were picked not only because they are central to implementing the first stages of the photogrammetric process, but also because they were the ones in which users were not commonly able to work with their own set of data, being limited to examples already provided by the software. The remnant modules are currently being ported from Qt3 to Qt4. Figure 3 shows the results of the interior orientation module migration from the Qt3 to Qt4.

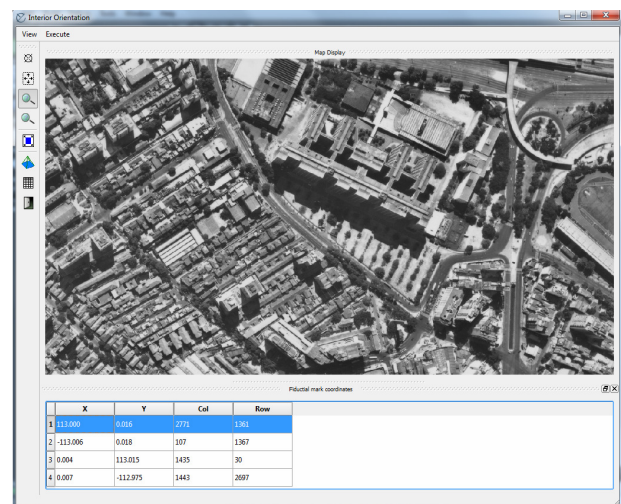


Figure 3. Interior orientation module after having been ported to Qt4.

C. The Integration of E-FOTO Modules

An important characteristic of the photogrammetric process is its division into smaller, sequential steps. This makes software organization in modules fairly intuitive. Therefore, the implementation of initial versions of E-FOTO followed that modular architecture. But, in these versions, the sequential characteristic of the process was overlooked, and so each module was developed independently of the others, without any concern about data exchange among them. Since it was known that the software would get a major overhaul due to the porting from Qt3 to Qt4, the development team took this opportunity to restructure the entire system architecture, and to offer a solution according to which all modules would be able to communicate through a common language.

This common language was implemented through a specification of the XML (eXtensible Markup Language)

standard created by the project's development team, called E-FOTO Photogrammetric Project, and given the ".app" file extension [9][10][23]. The choice for this implementation was based upon the extensibility offered by the language and the possibility to view and audit the project without the need to develop complex solutions, since most Internet browsers can understand XML files. To interpret these files, the development team implemented a solution based on the DOM (Document Object Model) approach, and incorporated it in all re-factored modules so that they are able to obtain the information required for their part of the process from a common data source [12][19].

Concerning the general structure of the software, the whole process required one change in relation to the older versions: the addition of another module to the system. Until then, the software contemplated all the stages of the mathematical process, but lacked a module to represent the stage of structuring the photogrammetric project. To solve this problem, a photogrammetric project management module was created, with a set of registration forms [11] and the duties of giving the user an overview of the current state of software and allowing the selection of the next step to be executed. Figure 4 shows the main window of the photogrammetric project management module.

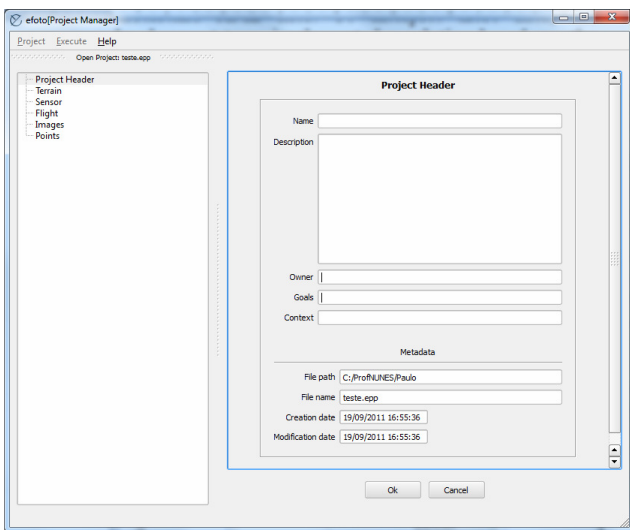


Figure 4. The E-FOTO project management module.

To finish the basic architecture of the system, a "central" manager to the whole system was developed, responsible for memory management and the main control flow. The existence of this manager is what actually characterizes the software's integration, since it is this manager which ensures that the various modules are accessing the same data set, thus keeping the integrity of the project throughout the execution of the software.

III. LESSONS LEARNED

Due to some of E-FOTO's characteristics, such as the constant presence of a client-like figure in the development team (in this case, the project's lead manager) and the

experimental research scenario in which it is included, the developers decided to try out a different approach to the development process during this new phase of the project's development cycle. After some research, it seemed reasonable to adopt a development process based on the agile methods [17], since their foundations apparently fit very well into the project's context. Out of all the agile methods, the developers chose to adopt the basis of agile modeling [18] and eXtreme Programming [19], mainly due to the fact that some of the developers already had some experience with them.

At first, everything seemed to run smoothly as expected. However, as time passed, the project's own development environment proved itself a big problem to be solved. Since E-FOTO is developed inside a University, with many of its working hands being teachers and students, it became increasingly harder to keep up with XP's so-called "good practices". Everyone had their own schedules, making whole-team meetings very hard; this affected the whole communication effectiveness very badly, since more meetings were needed, costing much more of the developer's time, and not everyone was always to pairing with the latest decisions. This even managed to get worse during exam periods - the teachers needed to design and grade exams, and the students needed to study, effectively slowing down the development to almost a halt. After these periods, the production rate slowly started to rise again, eventually reaching the ideal point, just to be ruined again by another battery of exams. This cycle went on until the first concrete results were achieved - much later than the developers thought they would.

As such, one of the project's greatest lessons learned in the last few years is that developing software in an academic context is very different from the scenario usually found outside the university walls. Sure, enterprise workers also have other things to do, and some of them could even be students or teachers. However, inside the academy these problems present themselves in much higher levels, possibly reaching the point where some of today's more well-known development processes lose much of their effectiveness. The project's development team still tries to apply some of the Agile concepts into their everyday work, but is still trying to find the point where the academic context will present itself not as a problem, but as one of E-FOTO's major advantages in its development process.

Another big lesson that the project's development team learned came from the troubles experienced in the migration from Qt3 to Qt4. As a defensive move to avoid having similar problems in the future, the project received a major architectural overhaul: not only was it to be developed according to the photogrammetric process' modularity, but also following some architecture that could minimize the effects that another framework change could have. The solution found was to follow a scheme based on MVC (Model-View-Controller), where the whole Model and Controller layer were to be developed without the aid of any external libraries/frameworks, isolating these on the View layer. Even then, the communication between the Controller and View layers should be done according to a pure-C++

interface, so that any "low-level-code" changes in the View implementation would not affect the other layers at all.

So far, this architecture has proven itself very successful at its job, allowing the developers to meddle with many of the software's "high-level" aspects while keeping the core business intact, such as the addition of different image visualization components. As a bonus, the development of new modules has proven to be very simple, since the modules are now connected to the same model, with the addition of a new controller and user interface to interact with it. Therefore, it can be said that this MVC-based structure is the greatest reason for the success of E-FOTO's integration process.

IV. BENEFITS OF THE PROJECT

E-FOTO has been tested mainly in Digital Photogrammetry classes, both in the Masters program in Geomatics, and in the undergraduate program of Cartographic Engineering at the Rio de Janeiro State University. It has also been used in many academic institutions over the world. Anyone can access the E-FOTO software by downloading its versions from the E-FOTO's home page on the Internet [16]. The software runs in both Windows and Linux environments. The following photogrammetric data and training material is also available in the E-FOTO's home page: (a) a set of three digitized-frame, 9"x9" aerial images; (b) a pdf file containing a photogrammetric camera calibration certificate; (c) a set of twelve ground control point coordinates for photogrammetric processing; (d) a set of seven tutorials about the functionalities of the E-FOTO solutions, and (e) the academic works and the publications generated as a consequence of the E-FOTO development. Another point that is worth mentioning is the statistics about the visits to the E-FOTO's home page: those statistics show the daily and monthly accesses to the E-FOTO's Home page. It also has the tracking of the geographical source of the accesses. Figure 5 shows examples of such statistics.

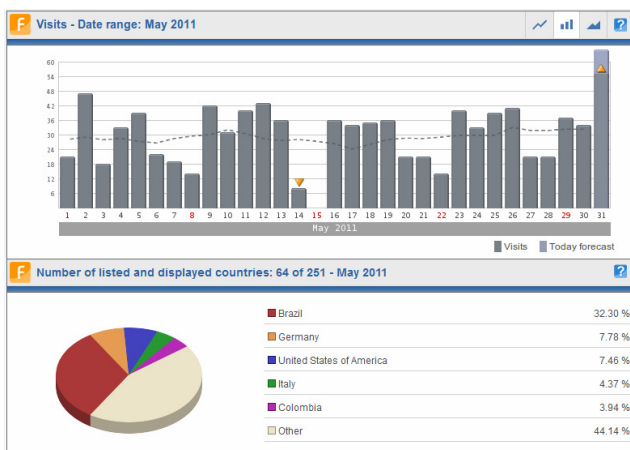


Figure 5. E-FOTO home page accesses statistics. Source: ShinyStat.

The graphic at the top of Figure 5 shows the visits to the E-FOTO home page in May 2011, with an average of 30

visits. The pie chart on the bottom of Figure 5 depicts the percentage of visits to the E-FOTO home page by country in the same period.

V. FUTURE PLANS

E-FOTO's educational software is being currently used in many different countries, which brings to the development team great responsibility. There is still much work to be done. In fact the team is currently preparing a new integrated version of the software with many improvements. These improvements deal with the correction of systematic effects from earth curvature, lens distortions, and atmospheric refraction. Another improvement is the extension of the E-FOTO's source code for dealing with optical satellite imagery, through the use of the Rational Polynomial Functions model. The development of the bundle adjustment phototriangulation algorithm is undergoing. Stereoscopic visualization capabilities with either passive or active shutter glasses is another topic of research. The E-FOTO team is also translating the tutorials about the use of the integrated version of the software into English and Spanish.

VI. CONCLUSIONS

Digital Photogrammetry has manifold applications of potential interest for users of geographical information. In developing countries, such as Brazil, there is a great need of proper formation of new professionals who will work in this field. The motivation and goal of the E-FOTO project is to make available a GNU/GPL free software platform for education in Digital Photogrammetry. E-FOTO does not intend to replace commercial photogrammetric solutions. However, it does intend to provide a software solution that could be used anytime, anywhere by anybody who is interested in learning about photogrammetric principles. Because of that, E-FOTO could be extremely useful for those people and institutions that cannot afford to acquire digital photogrammetry software solutions, even when purchased for educational purposes.

ACKNOWLEDGMENTS

We would like to thank the Brazilian National Council for Scientific and Technological Development (CNPq), which provided financial support to the E-FOTO project. Also, we would like to thank those who have contributed voluntarily to E-FOTO's development, especially Prof. Eddison José Araya Morales, from Costa Rica, who translated E-FOTO tutorials into Spanish.

REFERENCES

- [1] L. C. T. Coelho Filho, "Estação Fotogramétrica Digital Educacional," Undergraduate Technical Report, Instituto Militar de Engenharia, Brazil, 2002.
- [2] L. C. T. Coelho Filho and J. L. N. S. Brito, Fotogrametria Digital, 2nd ed. Brazil: EdUERJ, 2007.
- [3] S. A. Lima, "Estratégias para Retificação de Imagens Digitais," Undergraduate Technical Report, Universidade do Estado do Rio de Janeiro, Brazil, 2003.

- [4] F. J. C. Silveira, "Fototriangulação pelo Método dos Feixes Perspectivos," Undergraduate Technical Report, Universidade do Estado do Rio de Janeiro, Brazil, 2004.
- [5] F. J. C. Silveira, "Fototriangulação Pelo Método dos Feixes Perspectivos com Auto-calibração," Master Thesis, Universidade do Estado do Rio de Janeiro, Brazil, 2007.
- [6] G. J. Sokal, "Normalização de Imagens Fotogramétricas Digitais," unpublished.
- [7] M. T. Silveira, "Visualização e Medição Estereoscópicas de Imagens Fotogramétricas Digitais," Master Thesis, Universidade do Estado do Rio de Janeiro, Brazil, 2005.
- [8] G. J. Oliveira and F. Cardoso, "Ortorretificação de Imagens Digitais," Undergraduate Technical Report, Universidade do Estado do Rio de Janeiro, Brazil, 2004.
- [9] R. J. M. Fonseca, "Estação Fotogramétrica Digital: Uma abordagem Sistêmica sob as Óticas de Aspectos de objetos," Master Thesis, Universidade do Estado do Rio de Janeiro, Brazil, 2008.
- [10] R. P. Silva, "Arquivo XML de Projeto Fotogramétrico e sua Auditoria no Ambiente E-FOTO," Undergraduate Technical Report, Universidade do Estado do Rio de Janeiro, Brazil, 2008.
- [11] M. V. Meffe, "Adaptação da Arquitetura Model-View do Qt para a Apresentação e Edição de Dados Compostos na GUI da EFD E-Foto," Undergraduate Technical Report, Universidade do Estado do Rio de Janeiro, Brazil, 2010.
- [12] I. S. Badolato and R. A. Aguiar, "A Integração do Software E-Foto em um Ambiente de Desenvolvimento XP," Undergraduate Technical Report, Universidade do Estado do Rio de Janeiro, Brazil, 2010.
- [13] S. L. C. Santos, "Reengenharia da Visualização de Imagens de Alta Resolução no Projeto E-Foto," Undergraduate Technical Report, Universidade do Estado do Rio de Janeiro, Brazil, 2010.
- [14] V. Silva and R. M. A. Fonseca, "Módulo de Visualização da Modelos Numérico de Superfície da EFD E-Foto," Undergraduate Technical Report, Universidade do Estado do Rio de Janeiro, Brazil, 2009.
- [15] D. L. Bastos, "Uma Métrica para Dimensionamento de Software Científico Aplicada à Fotogrametria Digital," Master Thesis, Universidade do Estado do Rio de Janeiro, Brazil, 2007.
- [16] E-FOTO, "E-FOTO: A free GNU/GPL educational digital photogrammetric workstation", available at <<http://www.efoto.eng.uerj.br>>. (Retrieved: August 2011).
- [17] K. Beck et al., "Manifesto for Agile Software Development" Available at <<http://agilemanifesto.org>>. (Retrieved: August 2011).
- [18] S. Ambler, Modelagem Ágil: práticas eficazes para a programação eXtrema e o processo unificado, 1st ed. Brazil: Bookman, 2004.
- [19] K. Beck and C. Andres, Extreme Programming Explained: Embrace Change, 2nd ed. USA: Addison-Wesley, 2004.
- [20] T. Schenk, Digital Photogrammetry, 1st ed., vol.1. USA: Terra Science, 1999.
- [21] E. M. Mikhail, J. S. Bethel, and J. C. McGlone, Introduction to Modern Photogrammetry, 1st ed. USA: Willey, 2001.
- [22] K. Kraus, Photogrammetry: Fundamentals and Standard Processes, 1st ed. vol.1. Germany: Dümmlers, 1993.
- [23] J. Blanchette and M. Summerfield, C++ GUI Programming with Qt4, 1st ed. USA: Prentice Hall, 2006.
- [24] T. Lillesand, R. W. Kiefer, and J. Chipman, Remote Sensing and Image Interpretation, 6th ed. USA: Willey, 2007.

Vitalizing Local ICT-industry by Acceleration of FLOSS-based Software Product Development: A Case Study of the ICT-industry in Okinawa

Jun Iio
 Mitsubishi Research Institute, Inc.
 Chiyoda-ku Tokyo, Japan
 iiojun@mri.co.jp

Yasuyuki Minei
 OCC Corporation
 Urasoe Okinawa, Japan
 minei@occ.co.jp

Masato Kubota
 E-Sir, Inc.
 Naha Okinawa, Japan
 kubota@e-sir.co.jp

Kazuhiro Ooki
 NEC Corporation
 Minato-ku Tokyo, Japan
 ooki_kazuhiro@bc.jp.nec.com

Abstract—Occupational condition differentials between major enterprises in areas around central Japan and small-medium-enterprises in rural area are caused by hierarchical structure in Japanese industries on the information communication technologies (ICT). In order to remove this inadequate situation, we have drawn a blueprint to reform the ICT-industry by proposing a reference model to develop Free/Libre/OpenSource-based software products and a system that can support the development. The system has been implemented in an organization to help the local ICT-enterprises in Okinawa and the reference model is also now provided by that organization. This paper reports an outline of the system and that of the reference model. A result of test-run of software products development along with the reference model, using the support system, is also shown in this paper.

Keywords- OSS-based software products; local ICT-industry; development support.

I. INTRODUCTION

The industrial structure of the information and communication technologies (ICT) industry in Japan is similar to that of construction industry. That is, it is organized in a pyramid structure, where a main contractor exists at the top of the hierarchical structure. In the pyramid, not only sub contractors but also sub-sub contractors, and often much deeper degree of contractors can be members of the hierarchical structure. In such an industrial structure, only contractors in upper layers have opportunities to perform creative activities, because they play roles in upper process, such as requirement analysis and/or system conceptual design. Small-medium enterprises (SMEs) located at the bottom of the hierarchical structure are usually associated with lower process of system development. Hence, they mainly take charge of coding or relatively smaller unit-function development, and tend to accept relatively non-challenging work.

The occupational condition differential between major enterprises in the Tokyo area and SMEs in rural areas is caused by these hierarchical structures of Japanese ICT-industry. To eliminate these differentials and to vitalize the local ICT-industry, this industry structure should be modified, or more specifically, it should be corrupted and restructured. In order to break the dependencies in the hierarchical structure, it is

important for bottom members to have their own products. As the first step to rebuild the structure, we have designed a reference model of work process to develop Free/Libre/Open Source Software (FLOSS) based products and a development support system for the SMEs in the rural ICT-industries.

The ICT-industry in Okinawa is also embedded in this typical problematic pyramid structure. In the ICT-industry in Okinawa, many of ICT-enterprises are SMEs and main customers of them are sub contractors and/or sub-sub contractors in a large project. They seldom close a contract with neither user enterprises nor public sectors directory. That is one of the reasons why the ICT-related SMEs in Okinawa have an occupational wage problem. Three years ago, we have drawn a blueprint of an organization for industrial reform to solve this problem. This organization is now realized as an institute, named Ryukyu software business support center.

II. OKINAWA SOFTWARE DEVELOPMENT ACCELERATION PROJECT

Based on the blueprint, “Okinawa software development acceleration project” has been conducted from the fiscal year 2009. The aim of this project was to transform the local ICT-industries in Okinawa from commission-based business model into production-based business model. Because the production-based business can help SMEs to make contract directly with end-users, this transformation was considered to be needed to vitalize the local ICT-industries.

The goal of the project was to establish the standardized software development method based on FLOSS project and to implement the supporting system environment which developers were able to utilize for developing their software products. These functions are consolidated in the Ryukyu software business support center in order to develop their own software product based on FLOSS and its resources.

A. Background and Objective

In the Internet, there are varieties of software database called “FLOSS repository,” where many FLOSS projects are registered by developers from FLOSS communities. Such software can be used with no expense. Furthermore, we

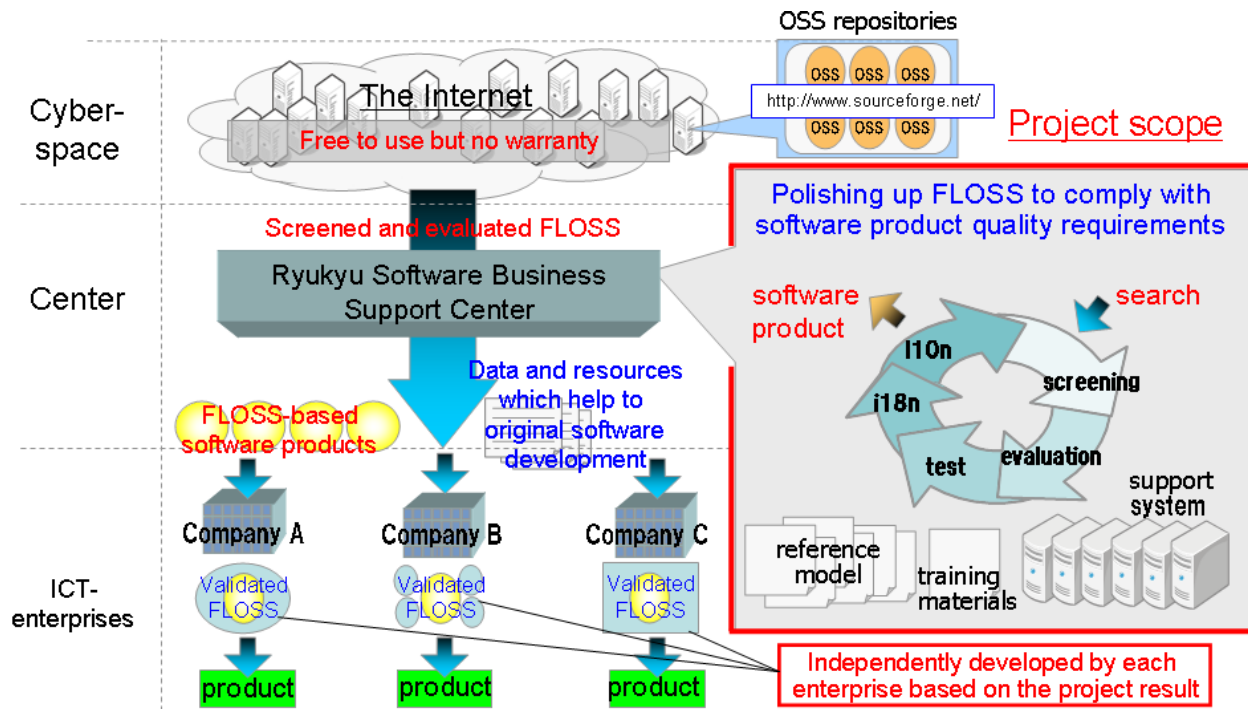


Figure 1. An overview of the Okinawa software development acceleration project and activities around the Ryukyu software business support center located at the center of this figure.

can use, modify, and redistribute freely under the condition of licenses bundled with software. Many of FLOSS are equipped attractive functions for users and they have potential capacity to be commercial software products. However, there is always a possibility that the software quality of FLOSS cannot satisfy appropriate level as commercial product. For example, it can be considered hard to use for novice users due to lack of install-function to install software into different environment, non-Japanese messages shown by non-internationalized software prevent average Japanese users from operating the software easily, and so on.

In our project, we have established the reference model of work process to develop FLOSS-based software products, which has several steps starting with searching FLOSS, screening, evaluating, and applying internationalization including localization, if needed. We designed not only the reference model but also implementation of the system environment that can be used to support the work process of software development. In addition, the framework for operating these systems efficiently was considered. These services can be provided to ICT-enterprises in Okinawa prefecture by Ryukyu software business support center so that the ICT-industry in Okinawa will be vitalized and be activated.

B. Overview of the Project

This framework is expected to help the ICT-enterprises in Okinawa to acquire material as the basis of software

product development, several data, and other resources. ICT-enterprises in Okinawa can put extra effort only to develop additional functions that are considered to be needed in order to suit it as their software product. This framework supports ICT-enterprises to accelerate their software development much rapidly and efficiently, that results in an establishment of effective software business infrastructure in the local ICT-industry.

A basic concept of the project is shown in Figure 1. It also illustrates a series of validation processes, an overview of supporting system, and an image of acceleration of software development by participants from the ICT-industries in Okinawa. The whole validation process contains screening, evaluating, testing, internationalizing, and localizing.

III. SUPPORT SYSTEM AND REFERENCE MODEL OF PRODUCT DEVELOPMENT

The system environment for the FLOSS-based software product development and the reference model are explained in this section.

A. Support System for FLOSS-based Software Product Development

Ryukyu software business support center provides its working guidelines for FLOSS-based software development, and members of the center can use the supporting system for their development activities, which is implemented in the center.

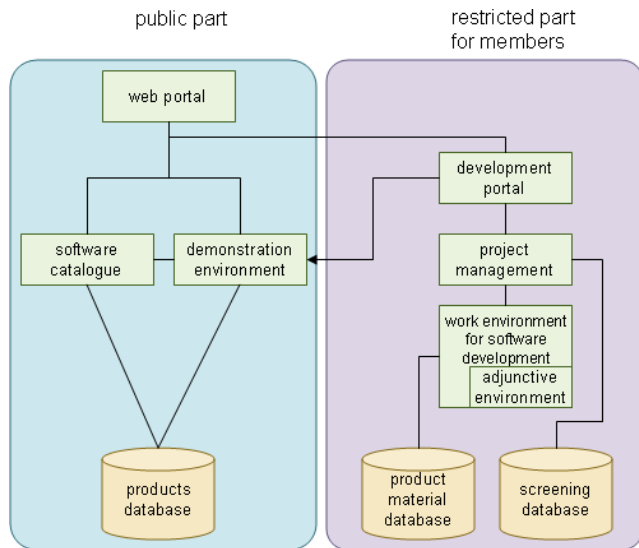


Figure 2. An overview of the software development support system provided by the Ryukyu software business support center.

An overview of the system is shown in Figure 2. The system is roughly divided into two parts. The public part of the system shown in the left half of the Figure 2 can be accessed by everyone via the Internet and the restricted part of the system shown in the right half of the Figure 2 can be accessed by authorized members, which are ICT-enterprises in Okinawa prefecture. In either case, user has to access “the web portal” at first.

Main content of the public part consists of a catalogue and demonstration function for software products which are developed and offered by ICT-enterprises in Okinawa. The information provided only by the software catalogue is considered insufficient to present a whole image of software; thus demonstration system is prepared so that user can figure out usability of the software as a practical manner. Optionally, software can be exhibited by operation movies.

Restricted part of the system helps ICT-enterprises to develop their software products which can be registered into the software catalogue. Firstly users have to log into “development portal” and then carry on process management, member assignment and progress management, using the project management function that is implemented within the restricted part of the system. For the practical work in software development process, developers can use working environment for their software product development. For the evaluation, internationalization, and localization process, users can access adjunctive environment provided as a function of the system. Results of development are registered into product material database. After the registration of product materials, if some ICT-enterprises proceed with additional development by themselves and decide to make a sale of the software as their own products, they can register

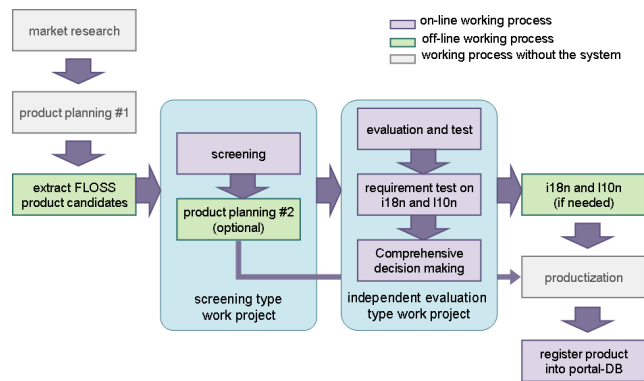


Figure 3. Work steps in the FLOSS-based software product development project, begin with marketing process and end with registering materials into portal database.

their products into the product catalogue and demonstration environment and they will be public to the consumer.

B. Reference Model of Work Process

Participants in the FLOSS-based software product development project execute tasks up to the work steps shown in Figure 3.

Before starting the software product development process, a person in charge conducts market research to make clear that what type of software is required and what type is not required. This market survey is conducted on a regular basis to catch up with market trends. The result of the market research would be used for deciding what area should be focused on and which software should be tailored as a software product by each enterprise. This decision is made in the first product planning step. Next step is to extract FLOSS-based product candidates. To find the candidates, FLOSS repositories such as SourceForge.net will be explored. In this step, at least five or more FLOSS projects should be selected, because these candidates will be filtered at the following steps and will be dropped depending on results of successive evaluations.

The software quality of each candidate is roughly estimated in the screening process, from the standpoint of development stability, the number of current users, activity of its community, and so on. In this step, every evaluation is judged by their information delivered from the website of the candidate projects. Note that FLOSS installation and/or source-code analysis are not required. These practical evaluations are performed in the subsequent evaluation step. The second product planning step that is optional will be conducted. In this step, what problems will be faced with in the process of making FLOSS into software products is discussed. The decision whether this action is taken or not is depending on business strategies of ICT-enterprises. From a practical perspective, this step will be started after the screening process, and it will be being performed simulta-

neously with the other subsequent processes.

The FLOSS candidates filtered by the screening step are subjects of evaluating-and-test step. In this step, software quality and license condition are investigated in details. Especially in desktop applications, localization is an important issue for the end-users. If FLOSS is not internationalized nor localized for Japanese, it is not familiar with Japanese users. Validating whether the software is internationalized and localized for Japanese or not is very important for the Japanese users. This work will be done in testing requirement for the internationalization and localization step.

Finally, development process will be shifted into comprehensive decision making phase. In this step, possibility of the attractive software products with the candidates is examined based on the results of the sequential steps of screening, evaluating, and considering the needs of internationalization and localization. The results are registered into the database if the candidates have some possibilities to make a business contract. In a practical sense, the development requires several developing steps on the internationalization and localization for the Japanese users. After that, considering the second product planning step, developed software products can be published via the web portal in the public part of the system.

IV. EVALUATION OF THE SYSTEM AND OPERATION

For the purpose of validating the efficiency of the system and reference model of operation, preliminary test-run was designed and conducted from the middle of December 2010 to the end of February 2011.

The trial validation was conducted along with following steps:

- 1) Interviews with key persons in higher educational institutes in Okinawa
- 2) Workshop on the overview of the test-run
- 3) Workshop on the practical work in the marketing step
- 4) Workshop on the practical work in the screening step
- 5) Workshop on the practical work in the evaluation-and-test step
- 6) Workshop on the practical work in the internationalization and localization step
- 7) Briefing session on how to operate the Protex IP to check licensing issues

A. Implementation Structure of the Validation

The trial had twenty two participants from eight companies that are the member companies of this project. All participants were divided into five groups in order to evaluate the proposed system and the working model.

There were two types of groups; the one was single-membered group and the other one was hybrid group. "Single-membered" represents that all the members in the group work for the same company and they work in the same office. On the other hand, the members in the hybrid group

Table I
THE SCREENING RESULTS OF THE TEST-RUN.

FLOSS	Score in the screening step	Result
magento	41	NG
prestashop	70	OK
zabbix	59	NG
Simple Groupware	51	NG
Open Atrium	55	NG
Concursive	38	NG
]project-open[60	OK
vtiger CRM	63	OK
ERP 5.0	36	NG

were gathered from different companies. In addition, designers in charge of each step of the reference working model were participated in the training phase of the trial validation, as lecturers teaching the details of each procedure.

The reason why these two types of working groups is because the Ryukyu software business support center is located in the rural area of Okinawa and it is far from Naha, the central region of Okinawa prefecture. Thus many of engineers will use the system remotely, and sometimes they will work in cooperation. The hybrid group can be considered as an emulation of practical use of the functions provided by the support center.

Table I illustrates the results of the screening process during the test-run. Scores were calculated from 0 to 100 in conformity with evaluation criteria in the screening process. At the end of this procedure, the final result was decided to OK if evaluation score is 60 or over. On the other hand, if the final score was under 60, which is the case of NG in the Table I, evaluation process for the FLOSS candidate stopped.

After finishing the test-run, only the "prestashop" shown in the Table I has been reached to the final judgement and the other factors has increased its final score up to 74 points, which results in the conclusion that it has possibility to be a software product sold by the ICT-enterprise in Okinawa. Note that ERP 5.0 got lower score than expected. The reason of the low score is considered that the software has been already famous enough in Japan and there is a company supporting localization and promotion for the software in Japan. That is, ERP 5.0 has already been well-suited for Japanese market and there seems no room for developing in our project. Thus, the score of ERP 5.0 is relatively low.

Participants in this validation process have contributed their comments to improve the reference model and supporting system.

Followings are selected reviews:

- Although there are many difficult points to be not able to make a decision in each process, appropriate advices from instructor help me to solve the questions.
- I did not have opportunities to evaluate the tools pro-

vided in the evaluation-and-test process, because the period of validation process was too short.

- It can be difficult unless the participants in the evaluation process have enough skills.
- The idea and method to narrow the list down to effective products are considered valuable when we have to struggle with FLOSS projects in the future.

V. RELATED WORK

Many FLOSS and its community evaluation methods have been proposed so far, like Open BQR[1], Open Source Maturity Model (OSMM)[2], method for Qualification and Selection of Open Source software (QSOS)[3,4], Open BRR (Business Readiness Rating for Open Source)[4], SQO-OSS Quality Model[5], and QualiPSo[6]. Evaluation criteria used in our screening process are based on a mixture of the criteria defined in those evaluation methods.

In our evaluation-and-test process, check items are arranged in accordance with the international standard, ISO9126[7] quality standards. As described in this standard, the operator works on checking three categories of the software quality: inner quality, outer quality, and quality-in-use. The inner quality and the outer quality are separately defined as six quality features and twenty seven sub-quality features. Also the quality-in-use is defined as four quality features.

The evaluation-and-test process has an additional function to verify the risk of license violation. According to Monden et al.[8], about ten percent of FLOSS has contained reused code in its source code. To avoid GPL violation, it should be confirmed that the software product does not have any GPL contaminated code. Thus, Black duck software's Protex IP has been introduced in order to make sure of compliance with the FLOSS licenses.

VI. CONCLUSION

In this paper, the reference model and the system for developing software product based on prospective FLOSS are described. The model and the system are provided by the Ryukyu software business support center in order to transform the structure of the local ICT-industry in Okinawa.

Implementing the system using well-known FLOSS, such as MySQL, Track, PHP, Zend, and Subversion, has reminded us that the FLOSS products were helpful to construct a specific system quickly and efficiently. With the help of these FLOSS products, we could develop the highly practical system in a short development period. The system has also its web-based user interface and it is provided by the combination of Linux, Apache, MySQL, and PHP; that is the LAMP system. The decision to use LAMP stack resulted in quick development and stable operation at present.

In the reference model using the system, working processes for screening and evaluating FLOSS products are needed to be managed by administrators. However, since

those processes are complicated enough and there has some room to modify in instruction of the working process, those two working processes have been designed as independent on the system.

Practically, the reference model is described with two documents. One is system operation manual and the other is a guidance document on the reference model. Several methods were discussed and refined through the design of the reference model; it is composed of a method to focus on a specific area in the software classification as market survey, a method to extract FLOSS according to the result of the market survey, a method to apply screening process onto FLOSS candidates, a method to derive evaluation items in accordance with the (sub-) quality features that are suitable to validate the candidate software, and so on.

Regular operations of the Ryukyu software business support center have just started after preliminary discussion phases. The ICT-enterprises in Okinawa prefecture should bring forward their activities on developing their own software products with the help of the functions provided by the center. Promotion on the center's operation remains as our future work.

REFERENCES

- [1] D. Taibi, L. Lavazza, and S. Morasca, *OpenBQR: a framework for the assessment of OSS*, Open Source Development, Adoption and Innovation, IFIP International Federation for Information Processing, Vol. 234/2007, pp. 173–186, 2007.
- [2] B. Golden, *Succeeding with Open Source*, Addison-Wesley Professional, 2004.
- [3] A. Origin, *Method for Qualification and Selection of Open Source software (QSOS), version 1.6*, April 2006, <http://www.qsos.org/download/qsos-1.6-en.pdf>
- [4] J.C. Deprez and S. Alexandre, *Comparing Assessment Methodologies for Free/Open Source Software: OpenBRR and QSOS*, Product-Focused Software Process Improvement, Lecture Notes in Computer Science, Vol. 5089/2008, pp. 189–203, 2008.
- [5] I. Samoladas, G. Gousios, D. Spinellis, and I. Stamelos, *The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation*, Open Source Development, Communities and Quality, IFIP International Federation for Information Processing, Vol. 275/2008, 2008.
- [6] V. Bianco, L. Lavazza, S. Morasca, and D. Taibi, *Quality of Open Source Software: the QualiPSO Trustworthiness Model*, Open Source Ecosystems: Diverse Communities Interacting, IFIP Advances in Information and Communication Technology, Vol. 299/2009, pp. 199–212, 2009.
- [7] ISO/IEC 9126-1:2001 *Software Engineering Product Quality-Part 1: Quality model*, June 2001.
- [8] A. Monden, S. Okahara, Y. Manabe, and K. Matsumoto, *Guilty or Not Guilty: Using Clone Metrics to Determine Open Source Licensing Violations*, IEEE Software, Vol.28, No.2, pp. 42–47, Mar./ Apr. 2011.

Empirical Case Study of Measuring Productivity of Programming Language Ruby and Ruby on Rails

Tetsuo NDOA

Information-processing Center
Shimane University
Matsue, Japan
nodat@soc.shimane-u.ac.jp

Chi JIA

Faculty of Law and Literature
Shimane University
Matsue, Japan
jiachi@soc.shimane-u.ac.jp

Abstract— This short paper is intended as a trial balloon of the evaluation of open source software, by measuring the productivity of programming language Ruby and web application framework Ruby on Rails, compared with other open source software.

Keywords-component; Open Source; Ruby; Ruby on Rails; Productivity; Programming Language

I. INTRODUCTION

Ruby is the Object-oriented Script Language released by Mr. Yukihiro Matsumoto, called "Matz" in open source communities, and was opened to the public in 1993. Matz lives in Matsue City in Japan and has been developing Ruby with many open source developers all over the world through the Internet. The number of core committers of Ruby is about seventy in 2011, and the two-third of them are Japanese. So Ruby is one of very few open source projects that Japanese engineers are mainly engaged in developing.

At first, though Ruby commanded attention through geeks, it had not been spread in business uses. But, in 2005, David Heinemeier Hansson – a programmer in Denmark, released Ruby on Rails, web application framework constructed by Ruby. Hence, Ruby came to attract attention and to be used also in enterprise areas. According to TIOBE Programming Community Index, which announces the ranking based on the retrieval by keyword of the search engine, the share of Java is 18.5% in the investigation of in 2010, and PHP is confronted to 7.8%, and Ruby is at level of 1.9% (ranking 10th place) [1]. But the number of Ruby's engineers has been increasing remarkably. It is forecast that the engineer who will use Ruby by 2013 reaches four million people according to the investigation of United States research company Gartner [2].

Then IPA (Information-technologies Processing Agency) [3], the Japanese government agency, started to support the Ruby project. It has been driving forward the standardization of Ruby. Because Ruby is open source, there are many implementations of Ruby. Besides the Ruby 1.8 affiliate (implemented by C language) and this Ruby 1.9 affiliate (implemented by virtual machine YARV), IronRuby (implemented to operate Ruby on .NET Framework), MacRuby(implemented to operate Ruby on Mac OS X), and Rubinius (bytecode interpreter on a virtual machine), etc. Thus there are variety of implementations. But the standard

specifications of Ruby language had not existed. So, IPA started standard specifications making, first domestically based on standard specifications in 2008, and constituted it as JIS (Japanese Industrial Standards) in 2011. Now, Ruby is proposed to ISO (International Standard).

This process will improve the interconnectivity of the portability and external systems by making it in accordance with this standard. Moreover, it will develop the foothold of the specification when the server environments to execute the program written with Ruby. The reason why IPA, the Japanese government agency, is bringing forward this process is to increase the market of Ruby, Japanese-Oriented Programming Language, in enterprise areas.

However, the reason why Ruby and Ruby on Rails have been used recently is the productivity of them. The productivity of Ruby and Ruby on Rails has been said tententiously as such "The productivity of Ruby is ten times higher than that of Java". But we must proof the productivity of them empirically and scientifically, if Ruby and Ruby on Rails are good for enterprise areas. So, we measured the productivity of them compared with other script languages. In this context, the term of "productivity" means software productivity by man-hours, including experience years of using language. In this paper we mention the method and the result of the productivity's comparison, and we also hope this method will be an "active pointer" of measuring software's productivity.

II. PRODUCTIVITY OF RUBY

The script languages like Ruby have to be compiled the source codes to object codes at each execution, so that the processing speed of them tends to slow extremely. The simpler the characteristic of language is, the slower the processing speed of it becomes. However, due to faster grow of the information processing abilities symbolized in the Moore's Law, the improvement of the processing speed has become to be owed to computer hardware, mainly the power of CPU. And as for the service of Web, prompt (agile) and flexible development and release is required.

The amount of the description of codes by Ruby is less than that of other programming languages, and the grammar expresses man's imagination similarly near human language, so that the its productivity of development becomes higher as a result. Therefore, the productivity of Ruby is evaluated in

the development of the Web application from which quick release and a frequent change are required. If the domination of such productivity is actually proven, introducing Ruby into the production site of development will be able to not only raise the productivity of development, but also decrease the stress of engineers. So we measured the productivity of Ruby compared with other programming languages.

We compare the productivity among Ruby, Java, and Perl cooperated with an IT company [4]. We developed the Web applications that have the same functions (Message board systems that have functions of comments contribution, multiple contribution prevention, indispensable check, and the automatic deletion) by Ruby, Java and Perl, daringly without using web application frameworks. TABLE I is the result of the comparison of each programming language's productivity.

TABLE I. COMPARISON OF PRODUCTIVITY (2010)

Languages	Java	Ruby	Perl
Lines	177	46	42
Man-Hour (Coding and Test)	Coding : 8 hours Test: 1 hours	Coding and Test: 2 hours	Coding and Test: 0.75 hours
Require Modules	19	2	4 (uses)
Operating Condition	Servlet	Http Server	Http Server
Operating Checking Server	Tomcat	Apache Anhttp	Apache Anhttp
Experience Years of Using Language	7 years	0 years	5 years
Experience Years of Development	7 years	7 years	7 years

As a result, Ruby exceeded both amounts of the codes and the manufacturing time greatly compared with Java in productivity. Ruby was proven to be as several times productive as Java at the manufacturing time (4.5 times if it simply compare). But it was proved that Ruby is not more productive than Perl. However, in spite of the first manufacturing in Ruby engineers could write the same amount of codes that Perl engineers, who need five years' experiences, write. Though, the speed manufacturing time of Ruby engineers is slower than that of Perl engineers, if they are trained coding, the productivity will be expected much higher.

III. PRODUCTIVITY OF RUBY ON RAILS

As has been previously described, Ruby became to attract attention since the release of Ruby on Rails. So we continuously measured the productivity of Ruby on Rails compared to Java's developing framework cooperated with an IT company [5]. We tried combustion and additional function requirement of 70% of the Working management system by Ruby on Rails. The system had previously (in 2007) developed by Java and JBoss Seam, web application framework.

To compare the amount of source codes, we divided the system into three elements, Model which is the kernel of

processing of the software design, View which rules display and output, and Controller: which receives input and controls View and Model according to the content And we compared each number of steps for these three elements. TABLE II is the result of the comparison.

TABLE II. TABLE TYPE STYLES (2010)

Elements	Ruby on Rails	Java + Jboss Seam	Java + Jboss Seam /0.7
Controller	5.1K	18.4K	26.3K
Model	1.2K	12.6K	38.1K
View	4.2K	4K	5.7K
Totla	10.5K	35K	50K

If we simply compare the numbers of steps, the productivity of Ruby on Rails is three times of that of Java and its web application framework. Moreover, if we consider that the development by Ruby on Rails was 70% of that by Java, the productivity is five times of Java. And, by the function point method (FP/man-hour comparison), which is an unit of measurement to express the amount of business functionality an information system provides to a user. The cost (in dollars or hours) of a single unit is calculated from past projects, the productivity of Ruby on Rails is 1.4 times of that of Java.

IV. ISUUES AND FORESIGHT

Though the productivity of Ruby and Ruby on Rails was measured by these case studies, the number of cases is obviously few. So we must continue to study much more cases. And, it will be difficult to compare productivity under the same developing condition. However, as the method of comparison of productivity was put on a firm footing, continuance of this study will enable us to compare the productivity of software empirically and scientifically.

Moreover, the performance of software must be measured by considering the effective speed in processing. For this reason, IPA is driving forward the standardization of Ruby. At the same time, in this study we compared the productivity in developing. Then we must measure and evaluate the performance of software totally. This process will be conducive to the evaluation of open source software which does not receive baptism of market pricing.

- [1] TIOBE Programming Community Index <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>
- [2] Ruby is Fastest-Growing Web Development Language: Gartner <http://www.pdfzone.com/cp/bio/Darryl-K.-Taft/>
- [3] IPA (Information-technologies Processing Agency) <http://www.ipa.go.jp/index-e.html>
- [4] Central Information Cooperation in Hiroshima <http://www.cis-net.co.jp/outline.html>
- [5] TOSCO Cooperation in Okayama <http://www.tosco.co.jp/>

Querying Source Code Using a Controlled Natural Language

Oleksandr Panchenko Stephan Müller Hasso Plattner Alexander Zeier
Hasso Plattner Institute for Software Systems Engineering
PO Box 900460, 14440 Potsdam, Germany
Email: {panchenko, stephan.mueller, office-plattner, zeier}@hpi.uni-potsdam.de

Abstract—Source code documents are of dual nature: they are in fact texts containing information for developers and they have an explicit structure for compilers and other tools. Several representations for the structured information of source code exist: abstract syntax tree, call graph, data flow graph, and others. Although the questions developers ask about source code seem easy to formulate, the complex code structure requires writing intricate queries. Developers use both, lexical and structured information for queries, though they dislike writing complex queries. Querying source code is an important activity in software development and maintenance. But often it cannot rely on predefined queries alone and requires writing more intricate queries. There is a need for a simple, user-friendly querying interface. This paper discusses an implementation of such a user interface based on a controlled natural language which is an unambiguous subset of the English language. When the developer enters the query, the source code grammar and the actual search results are used to automatically propose possibilities for query refinement and further navigation on the result set. The controlled natural language queries are then transformed to structured queries to retrieve data from a source code repository. The proposed approach provides a better expressiveness compared to simple keyword-based queries and enables consideration of complex structured relations between source code elements.

Keywords-Source code repository; source code query language; development tools; controlled natural language.

I. INTRODUCTION

The availability of efficient software development tools leads to lower development and maintenance costs, better code quality, and better organization of the development process. An important area of activities to be supported by tools is information gathering: searching source code, navigating through code and investigating related entities, performing various code inspections, calculating metrics, and mining code repositories. Source code repositories have been developed to enable code search using structured information contained in it. In general, this representation is based on a graph or a tree, whose vertices represent source code entities and whose edges represent relationships (implicit or explicit) between entities. Depending on the selected granularity level, entities can be single tokens in source code, modules, or entire subsystems. Examples of such structures are abstract syntax trees (ASTs), call graphs, data flow graphs, and module dependency graphs. Since each node contains lexical information, queries against such a

data structure should be able to combine both types of data: keyword-based search and structured queries.

Usually, the repositories require some structured query language: e.g., SQL, XPath, XQuery, Relation Manipulation Language [1] or Datalog [2]. Given a complex grammar of a programming language and complex relations between entities in the repository, even simple developers' questions require writing complex queries. Although developers use both, lexical and structured information for queries, they refuse formulating complex queries. For certain kind of tasks (metrics calculation, code inspections, etc.) a complex notation is acceptable, because the queries are written once and the developer can reuse them. Other scenarios, however, require more interaction and the possibility to enter a query manually with an easily understandable syntax and semantic of the query language.

This paper introduces a user-friendly interface for flexible source code querying that allows for queries up to high complexity using a controlled natural language (CNL). Further, we use XPath as the underlying structural query language. XPath expressions are evaluated on tree representations of ASTs. In our previous work we have shown that AST-based representation of source code can leverage querying of syntactical patterns in source code [3]. While we started with a query language that is similar to the surface programming language, in this project we investigate CNL as a query language.

Instead of using keywords or code snippets, the developer can compose queries using a CNL. Queries in CNL are transformed into a suitable query language to retrieve data from a code repository. This paper exemplifies the proposed approach based on a repository which contains detailed structured information in the form of abstract syntax trees and uses XPath as a structured query language.

The following section relates our approach to existing research. An introduction of controlled natural languages is given in Section 3, and Section 4 introduces a simple use case. Section 5 describes our implementation details. We conclude the paper in Section VI and give an overview of possible directions for future work.

II. RELATED WORK

Traditionally, regular expressions have been used to search source code. Although this simple method is useful for many

tasks, the relations between source code entities are disregarded. Moreover, the search scope is limited to few files because of performance issues. To overcome performance limitations, Marcus et al. proposed to use an information retrieval approach to store lexical information in an inverted index [4]. However, structured information of the source code was not included in the index.

There is a large body of research on source code repositories and query languages. Bull et al. proposed to combine regular expressions with structured queries [5]. The approach works well for simple structured queries, but has limitations because of restricted expressiveness. A set of query languages (e.g., Relation Manipulation Language [1]), which are based on predicate calculus, have been used to query software artifacts. Hajiyeve et al. have proposed to use safe Datalog, a query language based on the use of logic programming. Their tool, CodeQuest, maps safe Datalog queries to a relational database system [2]. JQuery is a tool supporting exploration of source code [6], [7]. The flexibility of the exploration views is achieved by a query-based customization of the content presented in the views. CodeQuest and JQuery aim at supporting high-level navigation and understanding software systems. Since existing source code querying systems store only coarse-grained software artifacts, the complexity of possible queries remains limited. But, as soon as the source code data model gains in complexity, queries become unwieldy.

The idea of providing a natural language interface for developers is not new. Würsch et al. presented a framework based on an OWL ontology to present data extracted by classical software analysis tools [8]. They used knowledge processing technologies from the Semantic Web and a guided-input natural language to answer questions about static source code information. The approach presented in our paper addresses developers' questions of the similar type. However, we focus on smaller syntactic code patterns based on AST representation of code. The major difference to the existing approaches is the capability of automatic generation of refinement proposals.

Further, there is a need for a flexible interface that enables keyword-based search over source code and takes into account its fine-grained and complex structure.

III. CONTROLLED NATURAL LANGUAGE

A controlled natural language is an unambiguous subset of a natural language with a restricted grammar and a domain-specific vocabulary. As a subset of the natural English language, CNL can be read and understood by a human user without any training. Although writing requires some training, it can be efficiently used to express formal statements, lowering the entry barrier to formal languages. Instead of learning a new language, the user is simply trained which subset of the ordinary language to apply. The writing process can be further supported by corresponding

intelligent authoring tools. Among many available CNL implementations, Attempto Controlled English (ACE) stands out due to high research activity and a wide range of available tools [9] and is thus used in this project. The ACE Editor is an example of a menu-based editor that facilitates the construction of ACE sentences with no need to explicitly know the syntactical restrictions [10].

Being effectively a formal language, ACE is unambiguously translated to first-order logic, that is appropriate for reasoning about the expressed contents by the machine. Since many formal languages use first-order logic as a logical foundation, transformations to those languages are possible. In the ACE implementation, sentences are translated to discourse representation structures (DRS) which is a syntactical variant of first-order logic [11].

A CNL is applicable in a variety of areas. It can be used for software specifications, documentation, ontology authoring, rules and policy formulation as well as an interface to other formal languages. The idea of transforming CNL into a query language is not new. The tool LingoLogic is an implementation that translates a CNL to SQL [12]. The contribution of this paper is the use of the structure of the underlying data model and actual search result to generate refinement proposals and offer these to developers.

Opposed to plain keywords, a sentence written in CNL is capable of carrying more semantics. The words relate to one another and enable the construction of complex sentences with higher expressivity.

IV. EXAMPLE USE CASE

Generally speaking, many of the programmers' queries reported so far [13] can be answered by our approach. This paper illustrates the approach with several simple examples. In the test scenario, a developer wants to find all references to a variable *xyz*, where the variable is assigned a value. The structured query to answer this trivial question is quite complex: all references to the variable should be found; it should be ensured, that no other variable with the same name, but from another namespace, pertains to the result set. Finally, the references should be selected, where the variable is placed in the left side of the assignment operator. Since the query is supposed to be used ad hoc, developers expect the tool to provide this functionality at their fingertips. On the other hand, since there are thousands of such questions, it is not possible to prepare a list of queries for developers to select from. A simple, user-friendly query language would allow a flexible, handy interface to a complex information repository.

The proposed interactive approach guides the developer from a very simple keyword-based query to the required result by refining the query step by step selecting one of the proposed alternative queries. The developer starts with a simple keyword query:

xyz (1)

In the background, this query is automatically extended to a correct CNL sentence that conforms to the ACE construction rules [9]:

Which entities are named xyz? (2)

Then, the CNL form is translated into an XPath statement, which is executed by the source code repository. The result set is presented with alternatives for refinement, which are proposed by the query generator:

Which classes are named xyz? (3)

Which entities are named xyz and are methods? (4)

Which variables are named xyz? (5)

Which entities are named xyz and are parameters? (6)

At this point it is important to mention that the query generator checks if the proposed queries may return non-empty result set. In our example queries 3 and 4 will be hidden if there is no class or method called *xyz*.

The developer chooses the query 5 and hereupon the query generator proposes a set of further possibilities:

Which statements define a variable that is named xyz? (7)

Which statements use a value of a variable that is named xyz? (8)

Which statements read a value of a variable that is named xyz? (9)

Which statements change a value of a variable that is named xyz? (10)

By selecting the query 10, the developer gets the intended query and the desired result set. XPath statements created and executed by the source code repository are as follows while query 11 corresponds to 2, 12 to 5, and 13 to 10:

`//*[.='xyz']` (11)

`//*[@IDNT[.='xyz']` (12)

`//*[@COMPUTE/RESULT/IDENT[.='xyz']` (13)

This example demonstrates how a query is interactively created with just a few clicks. Figures 1 and 2 illustrate the transformation of the query 10 into query 13. Figure 1 represents the query tree of the original CNL expression generated by the ACE parser. The phrase structure parsing approach decomposes the query into its elements. The query which is effectively a question consists of a noun phrase (np) and a verb phrase (vp), which are each refined recursively.

While the noun phrase is made up of a question determiner (qt) and a noun (n), the verb phrase consists of a verb (v) and another noun phrase, decomposing it further.

Figure 2 shows the resulting query tree in which the CNL parse tree was transformed. For simplicity reasons, a XPath query tree, which is the main part of the 13, is shown. The description of each vertex in the query tree includes the type of the vertex (in the Figure 2 it is reflected as the name of the corresponding Java class in the upper box) and the local name of the instance of the class, as shown in the bottom box. Each edge corresponds to a movement along an axis and is labeled by the name of the corresponding axis. Some transformation patterns can be recognized: e.g., the vertex *rel_cl* (Figure 1) is transformed into the vertex *PredicateNodeTest* (Figure 2). Nevertheless, the complete list of transformation rules is still to be designed.

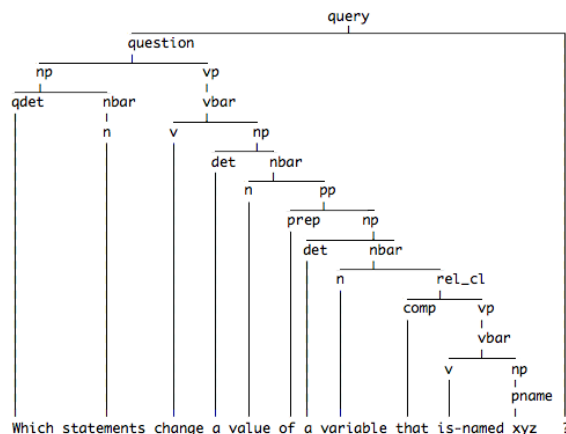


Figure 1. CNL parse tree of query 10

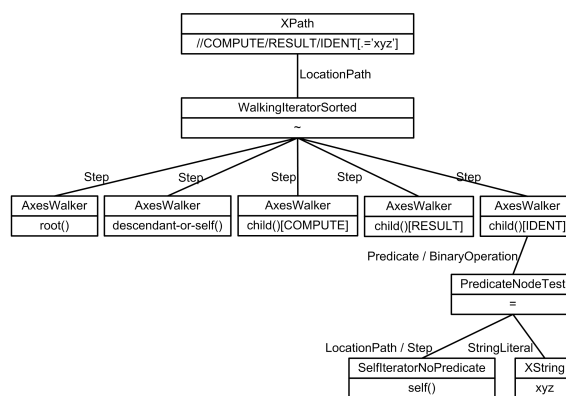


Figure 2. XPath query tree of query 13

V. PROPOSED ARCHITECTURE

The proposed architecture is presented in Figure 3. The client part is implemented as an Eclipse plugin that enables

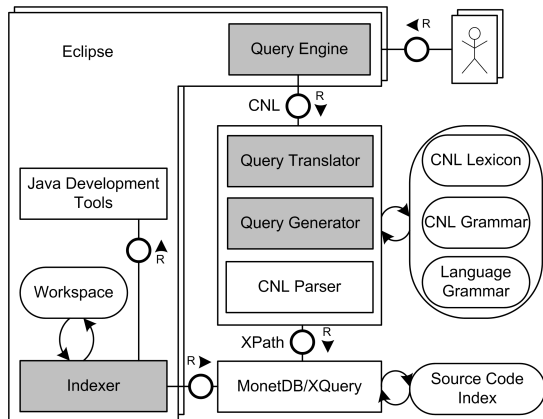


Figure 3. Prototype architecture

indexing of the local workspaces, which are sent to a central repository, and provides a user interface for querying. Eclipse Java development tools are used to parse source code and to construct ASTs. The server side is responsible for receiving developers’ requests formulated in CNL, translating these requests into XPath, generating proposals for the query refinements, and returning the result set. As a database of the source code repository, MonetDB/XQuery is used which natively supports XQuery as a method for accessing data.

The central source code repository stores code in the form of ASTs. The indexer parses code into ASTs, annotates the vertices of the trees with meta data (class name, responsible, last editor, data and time of the last modification, etc.), and stores the trees in the index.

The primary function of the query translator is the transformation of CNL statements as provided by the developer, or selected from the proposed ones, into a logical form. The logical form is the DRS and can be regarded as the actual system language for information integration. The parsing is done by the Attempto parsing engine, though CNL grammar and CNL lexicon containing domain-relevant words and their grammatical categories are prepared in advance to describe a certain programming language. The resulting DRS of the query 10 is depicted below:

```
[A, B, C, D, E, F]
object(A, xyz, named, na, eq, 1)-1
query(B, which)-1
object(B, statements, countable, na, geq, 2)-1
object(C, value, countable, na, eq, 1)-1
object(D, variable, countable, na, eq, 1)-1
predicate(E, named, D, A)-1
relation(C, of, D)-1
predicate(F, change, B, C)-1
```

For querying the index, this logical form is transformed into the XPath query. The implementation of corresponding transformations is subject to the ongoing research. The logical form from the CNL parser can serve domain-

independently as the integrative platform.

In order to represent the result of the query in a user-friendly way, it has to be structured appropriately and to yield functionality for further activities. For this purpose, once the query has been entered by the developer, the source code grammar and the actual search results are used to automatically propose possibilities for query refinement and further navigation on the result set. In this example to generate queries 7-10, source code grammar should reflect that a variable has a definition, variables are used in statements, and statements can read or change the value of a variable. All this information is already available in language specifications and should be made available to the CNL parser. Thus, a set of CNL queries are generated and proposed to the developer. This is facilitated through the ACE verbalization that takes the logical form, which is the DRS, and generates valid English sentences [14].

VI. CONCLUSION

The utilization of user-friendly interfaces for flexible information exploration in complex software environments leverages an indispensable contribution to a top-quality development framework, allowing for precise formulation of information needs. This leads to accurate information access, easy-to-use handling, flexibility and extensibility of interface functionality, high reusability in other domains, and significant lower development costs.

This paper discusses the usage of controlled natural language for querying source code. This approach is complementary to the keyword-based search and is a simple parlance that enables expression of complex relations between source code entities. Due to the fact that CNL is a subset of natural language, it can be read without any training.

The syntactic restrictions that have to be considered in the writing process are handled by a smart query authoring tool. Moreover, in most of the cases developers have only to select the query out of few automatically generated proposals.

The AST is not the only information to be indexed. There is a lot of relevant information available that is gathered in the development or maintenance process: test coverage measurements, code convention checks, change frequency, organizational metadata, customer complain messages, hot fixes, etc. This data can be made available as search criteria in the index.

REFERENCES

- [1] D. Beyer, A. Noack, and C. Lewerentz, “Efficient relational calculation for software analysis,” *IEEE Transactions on Software Engineering*, vol. 31, no. 2, pp. 137–149, 2005.
- [2] E. Hajiyev, M. Verbaere, and O. de Moor, “CodeQuest: Scalable Source Code Queries with Datalog,” in *Proceedings of the European Conference on Object-Oriented Programming*, ser. LNCS, vol. 4067. Springer, 2006, pp. 2–27.

- [3] O. Panchenko, J. Karstens, H. Plattner, and A. Zeier, "Precise and Scalable Querying of Syntactical Source Code Patterns Using Sample Code Snippets and a Database," in *Proceedings of the 19th IEEE International Conference on Program Comprehension*, 2011, pp. 41–50.
- [4] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An Information Retrieval Approach to Concept Location in Source Code," in *Proceedings of the Working Conference on Reverse Eng.*, 2004, pp. 214–223.
- [5] R. I. Bull, A. Trevors, A. J. Malton, and M. W. Godfrey, "Semantic grep: regular expressions + relational abstraction," in *Proceedings of the 9th Working Conference on Reverse Engineering*, 2002, pp. 267–276.
- [6] E. McCormick and K. D. Volder, "jQuery: Finding Your Way through Tangled Code," in *Proceedings of the Conf. on Object-oriented programming systems, languages, and applications*. ACM, 2004, pp. 9–10.
- [7] D. Janzen and K. D. Volder, "Navigating and querying code without getting lost," in *Proceedings of the international conference on Aspect-oriented software development*. ACM, 2003, pp. 178–187.
- [8] M. Würsch, G. Ghezzi, G. Reif, and H. C. Gall, "Supporting Developers with Natural Language Queries," in *Proceedings of the International Conference on Software Engineering*. IEEE Computer Society, 2010.
- [9] N. E. Fuchs, K. Kaljurand, and T. Kuhn, "Attempto Controlled English for Knowledge Representation," in *Reasoning Web, Fourth International Summer School*, ser. LNCS, C. Baroglio, P. A. Bonatti, J. Maluszyński, M. Marchiori, A. Polleres, and S. Schaffert, Eds., no. 5224. Springer, 2008, pp. 104–124.
- [10] T. Kuhn and R. Schwitter, "Writing Support for Controlled Natural Languages," in *Proceedings of the Australasian Language Technology Association Workshop*, 2008, pp. 46–54.
- [11] H. Kamp and U. Reyle, *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Dordrecht: Kluwer, 1993.
- [12] C. W. Thompson, P. Pazandak, and H. R. Tennant, "Talk to Your Semantic Web," *IEEE Internet Computing*, vol. 9, no. 6, pp. 75–78, 2005.
- [13] B. de Alwis and G. C. Murphy, "Answering Conceptual Queries with Ferret," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE. New York, NY, USA: ACM, 2008, pp. 21–30.
- [14] N. E. Fuchs, "Verbalising Formal Languages in Attempto Controlled English," University of Zurich, deliverable I2-D5, 2005.

Towards Complementing User Stories

Christian Kop

Institute of Applied Informatics
Alpen-Adria-Universitaet Klagenfurt
Klagenfurt, Austria
chris@ifit.uni-klu.ac.at

Abstract—User stories are well established in agile software development processes. However, user stories should not be seen as detailed requirements specifications. In agile processes it is accepted that the end users do not know all the requirements at once. Therefore, user stories only give hints about the expectations of an end user. In order to get more details in a communications process, a computer supported strategy is proposed in this paper. This strategy focuses on the agile development of information systems. Namely, it is proposed that additional information (not found in a user story) is extracted from natural language queries. Afterwards, this information is compared with the already existing work in progress model. Thus, the natural queries should help to find gaps and misunderstandings in the current work in progress model.

Keywords—*agile process; user stories; domain models; natural language queries;*

I. INTRODUCTION

In agile processes, user stories are a common way to gather the necessary information from the end user (e.g., an on-site customer). In a user story, the end user typically specifies what a certain actor (i.e., a system or person, which plays a role with respect to the system under development) can do with the system. Though, user stories are a well established technique in the early phase of agile software development, a user story is not a finished and well defined requirement or written contract. Instead, user stories are seen as short description of a piece of functionality, which act as a reminder for a communication process between end users and developers. In this process, details have to be negotiated [27]. So, the question is: How can this communication and negotiation process look like? Usually, developers can use questionnaires to ask for further information. They can make observation. If a first prototype is already developed, the communication is based on the prototype implementation. The software developers might ask what end users can do wrong if they are in a certain state of the prototype.

For information system development (ISD) an additional information gathering technique will be introduced here. Namely, natural language queries will be used. It will be shown how both steps

- extraction of model information from user stories and
 - extraction of additional details from queries
- can be supported by a tool.

The paper is therefore structured as follows. Firstly, an overview of the related work is given (Section II). Afterwards, an overall description is given, of how an agile process can be achieved in domain modeling. The next two sections (Section IV and Section V) focus on the two important parts of this agile process (user stories and natural language queries) and focus on computer supported information extraction. In Section VI, it will be argued, that this approach fits into the Agile Software Development Manifesto. Afterwards, it will be described how the computer support was tested. Finally, this section gives an overview of the prototype implementation. In Section VII conclusions and an outlook to future work are drawn.

II. RELATED WORK

Apart from user stories, some other research fields must be considered in the context of this work. These research topics will be described in the following sections.

A. Quality of Conceptual Models

The validation of artifacts in requirements engineering is always based on several techniques like inspections, walkthroughs, scenarios, verbalization, prototype evaluation, [16][21] or colored Petri Nets [22].

In [17] three dimensions for conceptual model quality are defined. They are syntax, semantics and pragmatics. If the model follows the rules and grammar defined in its meta-model then it is syntactically correct. Semantic quality is given if the model only contains true statements of the domain and is complete (no important concepts or statements are missing). Lastly, pragmatic quality relates the model to the interpretation of the user. A pragmatic quality is given, if it is understandable to the user.

An extensive research on quality of models was also made in [19]. It was proposed that conceptual modeling must shift from an art to an engineering discipline. Any engineering discipline aims at continuous quality checks of products and intermediate products.

There are also many other research results how to check and improve model quality [1], [4], [5], [8], [18]. The several research activities focused on model transformations [1], graphical aspects on conceptual models [18], verbalization strategies of conceptual models [4] [8] and viewpoints [5].

B. Queries

Rumbaugh et al. use queries for checking model completeness [35]. In their book [35], the authors give the exercise to check a partially completed object model with natural language queries. However, no computer support was found for this kind of task.

Current research on natural language query processing [2], [9], [12], [14], [13], [20], [23], [24], [15], [7],[26] only focuses on the retrieval of data or the generation of SQL. That means, these approaches expect that a finished and stable database already exist. A work in progress model and the consequences for natural language query processing was not the focus of these approaches. In these approaches, it is thus not the task of the natural language query to validate if something in the model is missing. Research works that describe visual queries [3], [10], [11] and form based query languages [6], [25] are based on the same assumption (i.e., existing and finished conceptual model or database). In visual query languages an SQL statement is generated by navigating through a final conceptual model.

C. Test Driven Development

Test driven development [31] is another agile method. The main idea is to write a first test case before a requirement is implemented. Afterwards, the developer tries to develop an implementation, which can pass the test case. Both the test cases and implementations to successfully pass the test cases are refined and improved iteratively in test driven development. The paradigm behind this is pointed out by Kent Beck: “**Failure is progress**” (see [31] p. 5).

User stories and natural language queries match very well to this paradigm. User stories represent the initial expectations (“requirements”) of the end users. Natural language descriptions represent the test cases.

D. Linguistic Instruments

In the succeeding sections and sub sections the linguistic instruments tagging and chunking are needed. A tagger is a tool, which takes as input a text and returns a list of sentences with tagged (categorized) words (i.e., words categorized as noun, verb, adjective etc.). The chosen Stanford Tagger categorizes the words according to the Penn-Treebank Tagset [32]. In this tagset the word categories together with some important syntactical features of a word are encoded. For instance if a noun is in plural then the category NNS is chosen. If a proper noun is detected then NNP is used. Current taggers can achieve about 97 % percent correctness [30]. This means, that in at least 3% of the categorization cases, a word is wrongly categorized. This has to be considered if a tagger is used.

Chunking is based on the tagger result. Chunking is useful to group words to so called chunks that can be seen as a phrase (e.g., a verb phrase or a noun phrase). This grouping is based on patterns found in the preceding tagging result. For instance the following groups of word categories can be subsumed to a noun phrase: Noun + noun; article + noun; article + adjective + noun. Details of chunking are described in [32].

III. OVERVIEW OF THE AGILE DEVELOPMENT SCENARIO

Before details of the domain concepts extraction and the completion process will be explained, this section gives an overview. The main idea is to combine user stories and natural language queries. Whereas user stories are needed to get a first initial model, natural language queries are used as test cases.

A. User Stories

A user story is a small piece of text. According to [27] it describes a functionality that will be valuable to either an end user or purchaser of a system or software. A user story follows a certain pattern. Currently two patterns are discussed and used. The first pattern is a declarative sentence in active voice that follows the SPO (subject, predicate, object) style. Examples for the first pattern are: “A user can fill out a resume”; “A customer can place an order”. A possible tool support for such a pattern is mentioned in [36]. The second pattern [34] emphasizes that in a user story sentence an actor is involved. Therefore, the pattern looks as follows “As a(n) <role/actor> , I can <feature>”. The above examples would look as follows in the second style: “As a user, I can fill out a resume”; “as a customer, I can place an order”. With the words “as a”, the speaker makes it more explicit, that with customer or user respectively not a concrete thing inside the system is described. Instead, the role of an external thing with respect to the future system is defined. Although, user stories are well accepted in agile processes, they provide minimal information. The developer must either strongly communicate with the end user or he has to rely on his personal experiences in a certain domain. Since it is always good that an end user is involved, the focus of agile development processes is on communication! In this paper a natural way of doing this in the area of information system development (ISD) and data centric applications is proposed. Namely, additional natural language queries should help to get more information about the model.

B. Completing the Story with Natural Language Queries

User stories help to get a first impression of what the user wants. For instance, for the user story “a customer can place an order” the following information can be extracted:

- The noun “customer” might be an actor in this story
- The noun “order” might be the domain class
- The verb “to place” is an operation or service, which probably belongs to order.

In agile processes, the developer now has to design and implement this user story. In order to implement this properly, additional information is necessary. For instance: Which attributes does order have? Can the notion customer be treated only as an actor or can it be treated as a full domain class? Usually this information is collected in a communication process with an end user. The question now is: Can this communication process be somehow supported? Here, it is proposed to use natural language queries. Continuing the given scenario with the user story “A customer can place an order”. The developer can now ask the end user, which queries should be executed later on in the

information system. Particularly, he should ask, which queries will be needed later for a database table, which is currently only represented as the domain class “order”. Since it cannot be expected that the end user knows SQL, natural language queries should be used instead.

Now let us suppose that only “order” is currently collected as a domain class. At the moment, customer is only seen as an (external) actor, which can communicate with the system. Furthermore let us assume that the end user states the following natural language queries:

- Which customer has placed Order 123
- Tell me the order items that belong to an order.

From these queries, the following can be learned.

Customer is not only an actor. Since he is mentioned in a query, which will be later on used as a database query, customer information is also needed in the database. In other words the information about customers represented by the word customer must be also modeled as a domain class.

Since both words “customer” and “order” are mentioned in the query, a path between them must exist. In its simplest form an association between customer and order must be modeled. Not any order is mentioned in the first query but a special order, which has a certain value. Since order is a class and values are instances of attributes the developer must ask the right attribute. An answer of the end user can be a revised and improved query (e.g., which customer has placed the order with order number 123). With this information the first domain model, which only contains “order” can be extended. Order gets the attribute order number. Furthermore “customer” is inserted as a domain class in the model. An association can be created between order and customer. The same procedure is applied on the second query. From this query, the developer can derive the information, that there will be a domain concept (i.e., the domain class “order item”), which can be related to order. Hence, the initial model, which had only “order” as its model element is iteratively refined.

C. Summary of the Overview

In the above two subsections it was explained how an initial domain model was generated using a user story. Since, such a model is still very incomplete; natural language queries can be used to complete it.

In the next Sections IV and V it will be shown how certain needed information (e.g., concepts) can be extracted automatically from user stories and natural language queries. Section IV focuses on the automatic extraction of actors and domain concepts (domain classes and attributes) from user stories. Section V will focus on the domain concept extraction from natural language queries.

IV. EXTRACTING CONCEPTS FROM USER STORIES

Domain concepts can be extracted from user stories by using the linguistic instruments tagging and chunking.

A. Tagging

At the first level, a tagger analyzes the user story text. The several word tokens are analyzed. Since taggers do not work 100 % correctly and a failure rate of 3 % must be considered,

the result of the tagger is analyzed once again for failures. This is done by broadening the context window. In this step a certain categorized word is compared with its previous and its succeeding neighbors. If a certain pattern appears, which can be seen as a wrong combination of word categories (i.e., the tagger has detected a noun but in this context a verb is the correct categorization), then the categorization is changed.

B. Chunking

After the tagging step, the chunking reanalyzes the tagger output. Chunking subsumes certain combination of categorized words (e.g., noun + noun; article + noun; article + adjective + noun) to noun phrases. Chunking helps to reduce the pattern variations and therefore is a good basis for the next step (interpretation).

C. Interpretation

In the interpretation step the concepts are extracted from the linguistically analyzed user story. The SPO pattern that is introduced in Section III looks like follows with the support of the chunker: <noun phrase> <verb phrase> <noun phrase>. After chunking, the second mentioned pattern (“as a <actor/role>, I can <feature>”) follows the linguistic pattern: <preposition> <noun phrase>, <personal pronoun> <verb phrase> <noun phrase>. The interpretation collects the noun phrases of the sentence. Since a user story itself is based on patterns, the interpretation can consider this for noun phrase selection and categorization. The first <noun phrase> is always treated as the actor/role that is mentioned in the user story. The second <noun phrase> contains the domain concept (class or attribute). The domain concept and the actor respectively are extracted from the noun phrases by ignoring the article.

V. EXTRACTING CONCEPTS FROM NATURAL LANGUAGE QUERIES

The query analyzer that extracts concepts from natural language queries is also based on the linguistic instruments tagging and chunking. Upon these, an interpretation and matching component is built.

A. Tagging

The same tagger that is used for analyzing user stories, is also used for analyzing the queries. In addition to the general optimizations which were introduced, the query tagger also have some additional optimization rules. These rules are necessary since query sentences might start with a verb. Furthermore, noun phrases are more complex than noun phrases which appear in the user story patterns.

B. Chunking

The chunker module has also an extra mode for query sentences. If this mode is switched on, one exception exists regarding the grouping of words and word categories. If the words “many” or “much” follow the word “how” (e.g., “how many persons”) then the implemented chunker behaves slightly different. A word like “many” is not chunked with “person” to a noun phrase but it is grouped with “how”.

Hence, instead of the output [how] [many persons] the query chunker generates the output [How many] [persons].

C. Interpretation

Interpretation of linguistically analyzed query sentences is a combination of noun phrase extraction and more refined parsing of specific patterns.

In a first step the query interpreter extracts all the noun phrases. This guarantees that at least query notions can be extracted, even if a more specific pattern cannot be detected. The found query notions are used to check if they match against existing concepts, views or examples (see subsection D).

More specific patterns are constraint sentences (e.g., "The age must be greater than 20"). These sentences can be used within a query text to constrain the concept. Such constraint sentences can also have adjectives at the end (e.g. "must be old"). If such adjectives are found, then these adjectives are collected as value descriptor candidates. Constraint patterns can also appear within a query (e.g., "Which customer has placed Order 123"). In this example not any order is meant but a specific order (Order 123).

Another task of the interpretation module is to filter out most often used meta-information (e.g., "list of ...", "set of ...").

D. Matching

If all the notions are extracted the system tries to match the notions found in the query with elements in the model. The matching procedure also checks if a constraint is applied on an attribute in the model. If this is not the case (e.g., Order 123), a warning is given to the user. If all the notions in the queries are found in the model or in model related information, then the query is successful. To achieve this, the extracted notions are firstly compared with the concepts in the model (i.e., can the extracted notion, or its singular form be found in the model). If this does not work then the extracted notion is searched in the list of similar words or examples which can be stored during modeling as additional information. Since the similar notions as well as the examples are related to a model concept, these notions can be traced back. Therefore, in any of the above mentioned cases, the notion found in the query can be replaced by the concept in the model to accomplish the next step (path finding). If all the notions extracted from the query are found in the model, then the tool can determine a path between these model concepts. Path finding is done by checking if all the concepts, which are necessary for the query belong to the same connected component within the conceptual model graph.

VI. DISCUSSION, TEST AND PROTOTYPE

A. Discussion

The four important factors of agile software development are [33]

1. Individuals and interactions over process and tools.
2. Working software over comprehensive documentation.

3. Customer collaboration over contract negotiation.

4. Responding to change over following a plan.

Comparing the approach with the above manifesto, the following can be said. Yes, there is a tool and tool support was one aim of this approach. However, it should be clear from the previous sections, that the tool does not dictate any process. In contrary, the aim of the tool is to enforce the interaction between individuals (stakeholders). If a query is not successful, then the stakeholders must discuss the failures and the communication process between them is improved!

Natural language queries are created for a very special purpose. From natural language queries, developers can manually and easily derive SQL queries, which can be embedded into an implementation of the future information system. Hence, these natural language queries represent parts of the future software. Thus, this approach fulfills "working software over comprehensive documentation".

Since the queries represent expectations of the customers, these customers will not understand themselves as a part in a contract negotiation but as an important part of a collaboration.

Finally, the need for responding to a change request is not restricted by using natural language queries as test cases.

B. Tests

Natural language queries, which were found in literature and own created queries were taken as test cases to test and improve the linguistic instruments. Among these test cases, the greatest set of natural language queries came from the Geo Query Project [28]. In this project 880 query sentences are used. These sentences can be categorized in queries starting with "What", "How", "Which", "Where" and other queries. These other queries do not start with an interrogative but start with a verb (e.g., "list", "give me", "name the", etc.) or they neither start with a verb nor with an interrogative (e.g., only a noun phrase is used for the query). The majority of query sentences is provided for queries starting with "What" followed by "How" and "Which". With the Geo Query Corpus a substantial test set was used. This high number of test cases is also important to get a good impression about the several different possibilities to express queries. All the queries in the test sets were applied on the query analyzer.

The user story interpreter currently accepts the two user story patterns as described in Section IV.

C. Tool Prototypes

The tools are implemented in Java. Currently, there are three sorts of tools. The first tool accepts a user story and extracts the actors and domain concepts (e.g., customer, order, order number, order item, etc). It stores this extracted concepts into an XML file, which can then be imported to the second tool, the concept editor. The concept editor graphically displays domain concepts (i.e., domain classes and attributes) but not actors. The third tool is the query analyzer tool. It is an add-on of the concept editor. A query analyzer window is opened if the user presses the Q-Button

in the concept editor. The query analyzer tool has a text area for the query and an error and warnings display area. Figure 1 and Figure 2 show the concept editor and the query analyzer interface. Figure 1 shows the model after the end user has stated the user story example: “As a customer, I can place an order”. Since “customer” firstly is seen as an actor and not as a domain class, it does not appear in the model presented by the concept editor. Only “order” is presented. Let’s continue with the already known natural language query examples and suppose, the end-user would like to execute the following two queries:

- Tell me the order items that belong to an order.
- Which customer has placed Order 123.

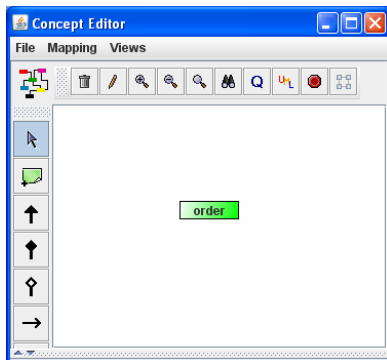


Figure 1: Initial model

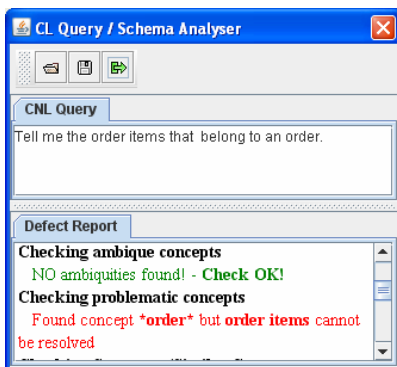


Figure 2: Applying Query on the model

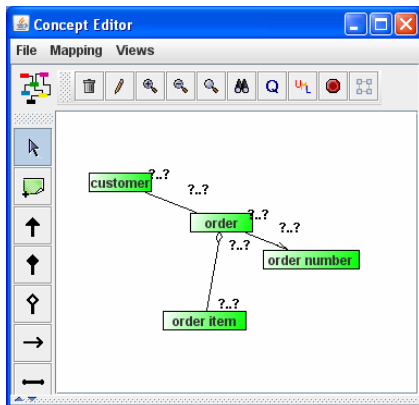


Figure 3: Result of an interaction process after 2 queries

After applying the first query on the query analyzer, the query analyzer returns failures (Figure 2). In addition, also the query “which customer has placed Order 123” is applied on the initial model. For the second query, the tool returns the information, that a constraint can only be defined on an attribute. The stakeholders must discuss what is missing (e.g., order number). If afterwards a refinement step is done, then the improved model might look as follows (see Figure 3). This iterative refinement by testing with natural language queries is similar to the paradigm of test driven development.

VII. CONCLUSION AND FUTURE WORK

In this paper the tool supported extraction of concepts out of user stories were described. Since such user stories are not detailed requirements but should give an idea what an end user might expect from the future systems, these stories must be complemented by additional information. In this paper one strategy of gathering additional information was presented. Particularly, natural language queries can be used even in an early phase of information system development. It was also shown how this strategy itself can be supported by a tool using linguistic instruments.

This work will be continued with a technical refinement of the query analyzer. Beside the explained interactive mode (see Figure 2), a batch mode will be implemented. In the batch mode many queries will be executed on the actual model. All problems will be stored in a report file.

Refinement of the user story interpreter is another future task. Though, the above two mentioned patterns are the most famous ones and are very often mentioned in the context of user stories, variations of these patterns exist. For instance, the “As a <role> I can <feature>” pattern can be extended to “As a <role> I can | want <feature> (so that | because) <reason>”. Attention will also be paid on these variations and automatic extraction of necessary information from these variations.

REFERENCES

- [1] P. Assenova and P. Johannesson, “Improving the Quality in Conceptual Modelling by the Use of Schema Transformations,” Proceedings of the 15th International Conference on Conceptual Modeling, Cottbus, Germany, Lecture Notes in Computer Science (LNCS), Vol. 1157. Springer Verlag Berlin Heidelberg New York, 1996, pp. 277 – 291.
- [2] H. Berger, M. Dittenbach, and D. Merkl, “Quering Tourism Information Systems in Natural Language,” Information Systems Technology and its Applications – Proceedings of the 2nd Conference ISTA 2003, GI Lecture Notes in Informatics, Vol. p-30, Koellen Verlag,, Bonn, 2003, pp. 153 – 165.
- [3] A.C. Bloesch and T.A. Halpin, “ConQuer: A Conceptual Query Language,” Proceedings of the 15th International Conference on Conceptual Modeling, Cottbus, Germany, Lecture Notes in Computer Science (LNCS), Vol. 1157. Springer Verlag Berlin Heidelberg New York, 1996, pp. 121 – 133.
- [4] H. Dalianis, “A method for validating a conceptual model by natural language discourse generation,” Proceedings of the Fourth International Conference CAiSE’92 on Advanced

- Information Systems Engineering, Lecture Notes in Computer Sciences (LNCS) Vol. 594, Springer Verlag, pp. 425 - 444.
- [5] St. Easterbrook, E. Yu, J. Aranda, Y. Fan, J. Horkoff, M. Leica, and R.A. Quadir, "Do Viewpoints Lead to Better Conceptual Models? An Exploratory Case Study," Proceedings of the 13th IEEE Conference on Requirements Engineering (RE'05). IEEE Press, pp. 199 - 208.
- [6] D. W. Embley, "NFQL: The Natural Forms Query Language," ACM Transactions on Database Systems, Vol. 14(2), 1989, pp. 168 - 211.
- [7] R. Ge and R.J. Mooney, "A Statistical Semantic Parser that Integrates Syntax and Semantics," Proceedings of the Ninth Conference on Computational Natural Language Learning, Ann Arbor, MI, 2005, pp. 9-16.
- [8] T. Halpin and M. Curland, "Automated Verbalization for ORM 2," Proceedings of OTM 2006 Workshop - On the Move to Meaningful Internet Systems 2006, Lecture Notes in Computer Science (LNCS 4278), Springer Verlag, pp. 1181 - 1190.
- [9] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide, "Exploring Fact Verbalizations for Conceptual Query Formulation," Proceedings of the Second International Workshop on Applications of Natural Language to Information Systems, IOS Press, Amsterdam, Oxford, Tokyo, 1996, pp. 40 - 51.
- [10] H. Jaakkola and B. Thalheim, "Visual SQL - High Quality ER Based Query Treatment," Proceedings of Conceptual Modeling for Novel Application Domains, Lecture Notes in Computer Science (LNCS), Vol. 2814, Springer Verlag, Berlin, Heidelberg, New York, 2003, pp. 129 - 139.
- [11] K. Järvelin, T. Niemi, and A. Salminen, "The visual query language CQL for transitive and relational computation," Data & Knowledge Engineering, Vol. 35, 2000, pp. 39 - 51.
- [12] Z.T. Kardovác, "On the Transformation of Sentences with Genitive Relations to SQL Queries" Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB 2005), Lecture Notes in Computer Science (LNCS), Vol. 3531, pp. 10 - 20.
- [13] M. Kao, N. Cercone, and W.-S. Luk, "Providing quality responses with natural language interfaces: the null value problem," IEEE Transactions on Software Engineering, Volume 14 (7), 1988, pp. 959 - 984.
- [14] E. Kapetainos, D. Baer, and P. Groenewoud, "Simplifying syntactic and semantic parsing of NL-based queries in advanced application domains," Data & Knowledge Engineering Journal, Vol. 55, 2005, pp. 38 - 58.
- [15] R.J. Kate and R.J. Mooney, "Using String-Kernels for Learning Semantic Parsers," COLING/ACL Proceedings, Sydney, 2006, pp. 913-920.
- [16] G. Kotonya and I. Sommerville, Requirements Engineering, Wiley Publ. Company, New York, 1998.
- [17] O. Lindland, G. Sindre, and A. Solvberg, "Understanding Quality in Conceptual Modeling," IEEE Software, March 1994, pp. 29 - 42.
- [18] D. Moody, "Graphical Entity Relationship Models: Towards a More User Understandable Representation of Data," Proceedings of the 15th International Conference on Conceptual Modeling, Cottbus, Germany, Lecture Notes in Computer Science (LNCS), Vol. 1157. Springer Verlag Berlin Heidelberg New York, 1996, pp. 227 - 245.
- [19] D. Moody, "Theoretical and practical issues in evaluating quality of conceptual models: current state and future directions," Data & Knowledge Engineering Volume 55, 2005, pp. 243 - 276.
- [20] V. Owei, H-S. Rhee, and Sh. Navathe, "Natural Language Query Filtration in the Conceptual Query Language," Proceedings of the 30th Hawaii International Conference on System Science, Vol. 3. IEEE Press, 1997, pp. 539 - 550.
- [21] K. Pohl, Requirements Engineering, Grundlagen, Prinzipien, Techniken, dPunkt Publ. Company, Heidelberg, 2007.
- [22] O. R. Ribeiro and J. M. Fernandes, "Validation of Scenario-based Business Requirements with Coloured Petri Nets," Proceedings of the 4th International Conference on Software Engineering Advances, IEEE Press (IEEE Digital Library), 2009, pp. 250 - 255.
- [23] N. Stratica, L. Kosseim, and B.C. Desai, "Using semantic templates for a natural language interface to the CINDI virtual library," Data & Knowledge Engineering Volume 55, 2005, pp. 4 - 19.
- [24] L.R. Tang and R.J. Mooney, "Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing," Proceedings of the 12th European Conference on Machine Learning (ECML-2001), 2001, pp. 466 - 477.
- [25] J.F. Terwillinger, L.M. Delcambre, and J. Logan, "Querying through a user interface," Data & Knowledge Engineering, Volume. 63, 2007, pp. 774 - 794.
- [26] Y.W. Wong and R.J. Mooney, "Learning for Semantic Parsing with Statistical Machine Translation," Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL-2006), New York, 2006, pp. 439-446.
- [27] M. Cohn, "User Stories Applied - for Agile Software Development", Addison Wesley Publ.Company, 2004.
- [28] Geo Query Project <http://www.cs.utexas.edu/users/ml/geo.html> (last access: 31. May 2011)
- [29] Penn-Treebank TagSet: <http://www.cis.upenn.edu/~treebank> (last access: 31. May 2011).
- [30] C. Manning and H. Schütze, "Foundations of Statistical Natural Language Processing," MIT Press, 2003.
- [31] K. Beck, "Test Driven Development by Example," Addison Wesley Publishing Company, 5th Printing, 2004.
- [32] E.F.T. K. Sang, and S. Buchholz, "Introduction to the CoNLL-2000 Shared Task: Chunking," Proceedings of CoNLL-200 and LLL-2000, 2000, pp.127-132.
- [33] Manifesto for Agile Software Development: <http://agilemanifesto.org/> (last access: 31. May 2011)
- [34] User Stories: <http://www.codesqueeze.com/the-easy-way-to-writing-good-user-stories/> (last access 31. May 2011)
- [35] J. Rumbaugh, M. Blaha, W. Premelani, F. Eddy, and W. Lorensen, "Object-Oriented Modeling and Design," Prentice Hall International Inc. Publ. Company, 1991
- [36] M. Smialek, J. Bojarski, W. Nowakowski, and T. Strazak, "Writing Coherent User Stories with Tool Support," H. Baumeister, M. Marchesi, M. Holcombe (eds), 6th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2005), LNCS, Vol. 3556, 2005, pp. 1217 - 1221.

Performance Evaluation of a Generic Deployment Infrastructure for Component-based S/W Engineering

Abdelkrim BENAMAR

Department of computer sciences
University of Abou Bekr Belkaid
Tlemcen, Algeria
a_benamar@mail.univ-tlemcen.dz

Noureddine BELKHATIR

Adele S/W Eng. Team,
LIG Laboratory
University of Grenoble, France
Noureddine.Belkhatir@imag.fr

Abstract—We present a generic deployment infrastructure for distributed component-based applications. This infrastructure is based on OMG's deployment and configuration specification and model driven architecture paradigm. Even though our approach is experimented for enterprise Java beans model, it can be extended to other specific models. We suggest the use of a classical measurement method in decision making for the proposed generic deployment platform of component-based applications. This method is based on graph theory and k-median algorithm. It allows optimization of the cost of any transaction in component deployment planning.

Keywords—*deployment and configuration specification; model driven architecture; computer network graph.*

I. INTRODUCTION

Software deployment [6] is a very complex and important process covering many activities. This complexity becomes more significant with the evolution of networks and component based systems. Many component based systems [13] are used both by industries and academics. We illustrate our approach on currently used industrial component systems, such as Enterprise Java Beans (EJB), Microsoft corporation .Net and OMG's CORBA Component Model (CCM).

In the following, we present a generic deployment infrastructure for distributed component-based applications. Furthermore, we layout a general method to design made-to-measure distances for any given deployment transactions. The optimal distances are computed with classical graph algorithms such as, k-median and contribute to the improvement of the decision making process for deployment of component-based applications.

The remainder of this paper is structured as follows: section II presents the state of the art on deployment of component-based systems. Section III focuses on the state of the practices. In section IV we synthesize our previous work [3] [17] on defining a generic deployment framework for component-based applications. In Section V, the main approaches to assessing the performance of distributed applications are reviewed. They are followed by a measurement method we apply to the deployment specifically for deployment planning. Finally the main

achievements and perspectives are summarized in the concluding section.

II. STATE OF THE ART

Recently, due to the availability of high-speed networks and advances in packaging and interface technologies, there has been considerable interest in building deployment platforms for component-based applications [8] and evaluating the performance of distributed applications [9].

A. Building Deployment Platforms

Hnětynka [13] introduces the Deployment Factory (DF) and model-oriented environment, based on Deployment and Configuration (D&C) specification for deploying software components. Since the DF is based on a plug-in thought, the deployment of the existing component technologies becomes more easier.

Merle and Belkhatir [17] propose a distributed environment called ORYA, for deploying ordinary applications. In fact, ORYA supports the basic stages of deployment process, such as install, configure, reconfigure and uninstall. Nevertheless, the planning stage of ORYA is very simple, because it supports only the deployment of ordinary applications.

Deng et al. [7] introduce a deployment engine called DAnCE based on D&C specification. This environment is now under construction and supports just the deployment of CCM components. However, it does not provide functionalities of D&C specification.

III. STATE OF THE PRACTICE

In this section we survey the main deployment platforms for component-based applications (e.g., EJB, .Net and CCM) developed by industrials and used in practice. The complete comparative study presented in our previous work [3] proves the robustness of these models and therefore the rationality of selecting only them.

A. Corba Component Model (CCM)

CCM [20] is a component specification proposed by the international consortium called OMG. The objective of CCM is to facilitate the development of heterogeneous distributed

components. In fact, the first specification of Corba was entirely oriented towards interoperability, so that all features related to the deployment were omitted. Nevertheless, the CCM 4.0 standard supports all the functionalities of software deployment and distribution. Precisely, this specification includes four models that are summarized hereafter:

- Abstract model: it designs the component interfaces and their interactions.
- Programming model: it designs the component code sources and their non-functional properties (e.g., transaction, persistence and security).
- Deployment model: it defines the component system assemblies.
- Execution model: it is represented by containers.

B. Enterprise Java Bean (EJB)

The EJB [18] is a framework developed by Sun Microsystems. The purpose of EJB is to allow the development of distributed and object-oriented applications in the Java language. Components in EJB are called beans. The bean interface is directly implemented in Java language. Each bean has two interfaces (e.g., remote, home). The remote interface allows performing the component business. The home interface allows the production of a novel component, or getting an existing component. Unlike CCM, the EJB specification includes two models that are summarized above:

- Abstract model: It represents the specification of component interfaces.
- Deployment model: It allows to assemble a component-based application, pack it into a package, and install it on selected sites.

C. Microsoft's .Net

The .Net is a framework developed by Microsoft Corporation. The objective of this framework is to provide the development of distributed applications. The .NET framework is based on the concept of class that is also called component. The class code is developed in classical programming languages (e.g., C#, visual Basic, Java...). The manifest file is created thanks to the classes' compilation process. All these files are packaged into another file called assembly that is manually deployed through network. In fact, the concept of assembly was introduced by Microsoft. They try to determine the versioning and deployment problems that were cause by the DLLs. Those one were known as DLL hell. The versioning problem appears like when a new application installs a new version of a shared component that is not backward compatible with the version already installed on the machine.

IV. TOWARD A GENERIC DEPLOYMENT PLATFORM

Although there are many environments for making unified the deployment of software component. None of them is generic sufficiently, and they do not perform automatically the deployment of heterogeneous applications. Furthermore, we suggest to use a generic methodology that makes unified the deployment component systems. More precisely, this methodology is based on a model

transformation approach that employs suitable Platform Independent Model (PIM) and Platform Specific Model (PSM).

A. Model Transformation Overview

There are several projects aiming to make generic the deployment of software component. None of them fulfills completely the required features (e.g., release, install, activate, update, adapt) [6]. OMG contributes to the resolution of this problem with its D&C specification [19]. This specification matches the Model Driven Architecture (MDA) paradigm. This paradigm proposes a methodology to software development through modeling and transformation of models to code implementations. Among other approaches to model transformation, providing tools, we can mention VIATRA [23], Tefkat [10], AMW [5], ATL [14][4], Kent [1], and C-SAW [12].

B. Implementation

We outline in this section some implementation details. The prototype we developed relies on the D&C application meta models as PIM and EJB meta model as PSM. We use the Eclipse SDK,

In the following, we summarize the main tools (see Figure 1) used in the prototype. More details are given in [3]

- The Eclipse Modeling Framework (EMF) is used to develop the main project named 'EJB2DnC'.
- The Atlas Transformation Language (ATL) is EMF plug-in that is used for mapping meta models of EJB to D&C application.
- The Eclipse Web Tools Platform (WTP) is used to develop a specific EJB application named stock management.
- The Ant Build Tool (ABT) is tool used in EMF to run java applications.

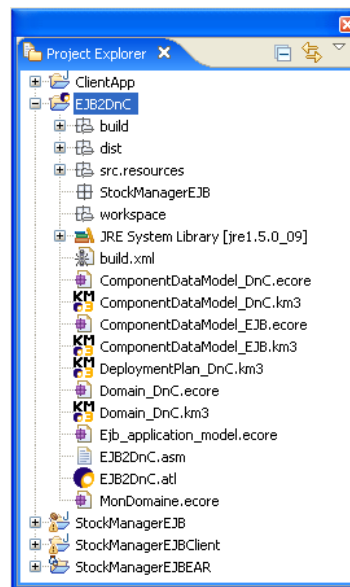


Figure 1. Project explorer view.

V. PERFORMANCE EVALUATION

This section highlights the currently used measurements to determine the performance of generic deployment platform. It proposes to use a classical measurement methodology for component-based applications, and proves the utility in decision making for deployment planning.

A. Motivation

As stated in [3] [6] [13], the deployment is a complex process constituted of many steps and activities, starting with the installation stage. Generally, the component is inserted into the target site (i.e, repository). The configuration stage succeeds the installation stage, and provides several configurations for further utilizations. During this stage, no deployment decisions (i.e, optimal placement and instance number) for components are performed. Naturally, these decisions are achieved in the planning stage. Therefore, we will contribute in planning stage by using measurement methodology in decision making for component deployment. Within the scope of this methodology, several decisions are carried out:

- Which component instance will be used?
- How many component instances will be deployed ?

B. Related works

There are several projects aiming to use end-to-end distances to achieve decision-making in computer networks. Nevertheless almost these works represent the end-to-end distance thanks to raw network metrics measurement. In below, we survey briefly some relevant examples:

Wolski et al. [25] present the Network Weather Service (NWS), which capture the condition of both network and hosts. It can provide the raw measurements of the classical metrics (e.g., bandwidth, latency, connection time, CPU availability) as well as forecasts based on aggregations of the set of raw measurements.

In AppLes project, Berman et al. [2] assume that each application is integrated with its own AppLes agent, which uses the performance model and dynamic information regarding resources to predict the run-time of its application on a given set of resources. Among a set of available possible candidate schedules, AppLes agent selects the one that is predicted to provide the best performance.

In Network Measurements Working Group (NMWG), Lowekamp et al. [16] highlight the used measurements to determine network performance for grid applications. They focus on a set of indicators as bandwidth, latency, throughput and CPU availability. They present also the characteristics of several measurement methodologies.

Seymour et al. [22] build a NetSolve infrastructure for providing domain-specific high-end network services. NetSolve provides a complete run-time infrastructure, as well as server management tools and client interfaces to languages as C, Fortran, Java, and MATLAB.

Karlsson and Mahalingam [15] present an illustration of using raw metric (e.g., latency and number of hops) for decision-making. More precisely, they propose an evaluation

framework for replication algorithms. Moreover, they present a survey on replica placement algorithms with comparison study. Nevertheless, the used raw metrics seem to be quite irrelevant for monitoring the performances of high-level applications.

Qiang Xu [21] presents a use case of other raw metrics (e.g., latency and Round Trip Time) in grid environment. Furthermore, he proposes an approach for automatic hosts clustering, by mapping them to a geometric space. Even though the used raw metrics have advantages which are their stability and easiness of its measurement, they appear to be insufficient to supervise the performance of computer networks.

Gossa and Pierson [11] propose a novel technique to represent derived distances (e.g., computation task cost and data transfer cost) for any transaction in pervasive grid environment. The computation of these distances is based on the measurement of different raw metrics (e.g., latency and bandwidth) that can be provided by any monitoring systems. This work is set apart because it uses the derived metrics which are hard and expensive to measure. They appear to be pertinent on the topic of to data transfer concerns. In addition, the metric computation has been implanted in a grid service, called Network Distance Service (NDS) and developed with Globus Toolkit 4.

Therefore, we were very motivated by the last work [11] because authors use a derived distances which are well-suited with decision making for deployment planning.

C. Overview of Measurement Representation

Since networks are constituted of hosts and links, they can be represented in graph form. We define a network as a graph $G = (\upsilon, \varepsilon)$ where:

- υ is a set of vertices representing the hosts.
- ε is a set of edges representing paths between vertices that are labeled with measurements from source to destination hosts.

According to Lowekamp et al. [16], a metric is a quantity corresponding to the performance of computer networks. There are kinds of measurements (e.g., raw or derived). Raw measurements are something that can be measured easily such as measuring latency using pings. Derived measurements might be an aggregation on a set of low-level measurements. The main useful metrics are:

- the bandwidth (BW) in Megabits/second,
- the latency (L) in Milliseconds,
- the CPU availability (CPU_a) in percents,
- the free memory space (RAM) in Megabytes.

These observations can be represented by matrices called BW, L, CPU_c, CPU_a and RAM. We note $m_{i,j}$ the measurement of the metric m from the host i to the host j . Here, we assume that: $BW_{i,i} = \infty$ and $L_{i,i} = 0$ (i.e, the cost of local data transfer is null).

D. Experimentation

The objective of this section is to take the best decision for components placement related in the generic deployment platform (presented in the previous section). More precisely, our experimentations are made on a test network which is

composed of four hosts (e.g., H_1, H_2, H_3 and H_4) and that is shown in Figure 2. We have evaluated our proposition on a classic planning scenario of component deployment. We forecast the effect of component data size with respect to their locations. Therefore, we consider different component sizes increasing from 1 (or 10^0) to 10^{10} with multiplier factor equal to 10.

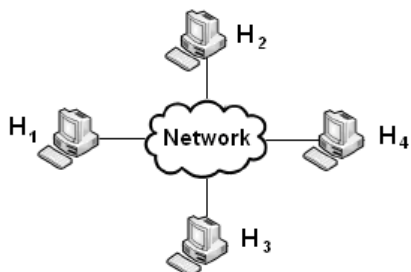


Figure 2. Deployment infrastructure.

Besides, we will assume that:

- all network hosts (e.g., H_1, H_2, H_3 and H_4) undertake the deployment of components,
- the component software is deployed on three hosts (e.g., $H_1, H_2,$ and H_3)

These hypotheses are illustrated in Figure 3.

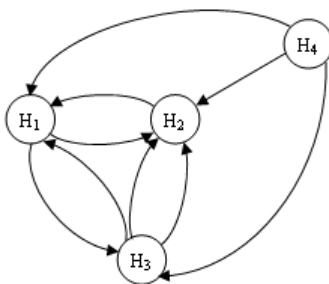


Figure 3. Graph of deployment infrastructure.

In order to optimize the time of component deployment, we only opt for transfer time. Thus, we use a compound metric called the Data Transfer Cost (DTC) [9], and represented by the formula:

$$DTC_{i,j}(\text{dataSize}) = \frac{\text{dataSize}}{BW_{i,j}} + 3(L_{i,j} + L_{j,i})$$

We use two measurement matrices corresponding to bandwidth (BW) and latency (L) (as shown in the Table 1) for computing the matrices DTC with respect to component sizes (as shown in the Table 2).

TABLE I. MEASUREMENT MATRICES

$$BW = \begin{pmatrix} \infty & 1.56 & 2.48 & 2.17 \\ 5.37 & \infty & 3.17 & 3.14 \\ 3.36 & 3.27 & \infty & 87.68 \\ 3.44 & 3.24 & 87.19 & \infty \end{pmatrix}$$

$$L = \begin{pmatrix} 0 & 16.5 & 10.0 & 9.5 \\ 16.5 & 0 & 15.6 & 15.2 \\ 9.8 & 15.7 & 0 & 15.2 \\ 10.0 & 15.7 & 0.6 & 0 \end{pmatrix}$$

TABLE II. REPRESENTATIVE RESULTS OF DTC COMPUTATION.

$$DTC(1) = \begin{pmatrix} 0 & 0.099 & 0.059 \\ 0.099 & 0 & 0.093 \\ 0.059 & 0.093 & 0 \\ 0.058 & 0.092 & 0.047 \end{pmatrix}$$

$$DTC(10^3) = \begin{pmatrix} 0 & 0.104 & 0.062 \\ 0.100 & 0 & 0.096 \\ 0.061 & 0.096 & 0 \\ 0.060 & 0.095 & 0.047 \end{pmatrix}$$

$$DTC(10^5) = \begin{pmatrix} 0 & 0.611 & 0.381 \\ 0.247 & 0 & 0.346 \\ 0.297 & 0.338 & 0 \\ 0.291 & 0.339 & 0.056 \end{pmatrix}$$

$$DTC(10^7) = \begin{pmatrix} 0 & 51.38 & 32.31 \\ 14.99 & 0 & 25.33 \\ 23.86 & 24.55 & 0 \\ 23.31 & 24.78 & 0.964 \end{pmatrix}$$

$$DTC(10^{10}) = \begin{pmatrix} 0 & 51282 & 32258 \\ 14897 & 0 & 25236 \\ 23809 & 24464 & 0 \\ 23255 & 24691 & 917.58 \end{pmatrix}$$

We will present distance computation which is based on graph algorithm, in addition of that, we will implement the classical algorithm to solve the k-median problem. The k-median problem (its implementation is designed in the subsequent Algorithm) is simply stated as: "Given a graph $G=(v, \epsilon)$, find $v_k \subseteq v$ such that $|v_k|=k$, where k may either be variable or fixed, and that the sum of the shortest distances from the vertices in $\{v/v_k\}$ to their nearest vertex in v_k is minimized".

Algorithm *kmedians* (k, σ, δ, d): *best_solution*

Input data

- k : integer (number of hosts)
- σ : set of source vertices
- δ : set of destination vertices
- $P_k(\delta)$: sub-set of source vertices such that $|P_k(\delta)| = k$
- d : matrix of $|\sigma| \times |\delta|$ real (DTC in our case)

Output data

$best_solution$: set of vertices (k best locations)

Method

```

best_criterion ← ∞
for all solution ∈ Pk(δ) do
    criterion ← 0
    for all hs = 1 to |σ| do
        min_dist ← ∞
        for all hd = 1 to k do
            if d(hs, solution(hds, solution(hd

```

E. Synthesis

The optimal values of DTC related to component deployment are computed using the k-median algorithm, then we forecast their variations according component locations (see Figure 4).

Here are some observations based on these graphs:

- The performances should be as expected improved with more instances of components.
- If we want to limit the network to a single host, H3 appears to be the best location for components.
- The DTC with $k=2$ is roughly the half of the DTC for $k=1$. But the value with $k=3$ corresponds to a real improvement.
- If we consider all the sizes together, a real impact of DTC appears from 107. This is obvious because the cost of the transfer of very small data is negligible face to the cost of a large data transfer.

Therefore, we decide to place the component on the three hosts H_1, H_2 and H_3 , since it is the best solution to ensure good performances of the generic deployment platform.

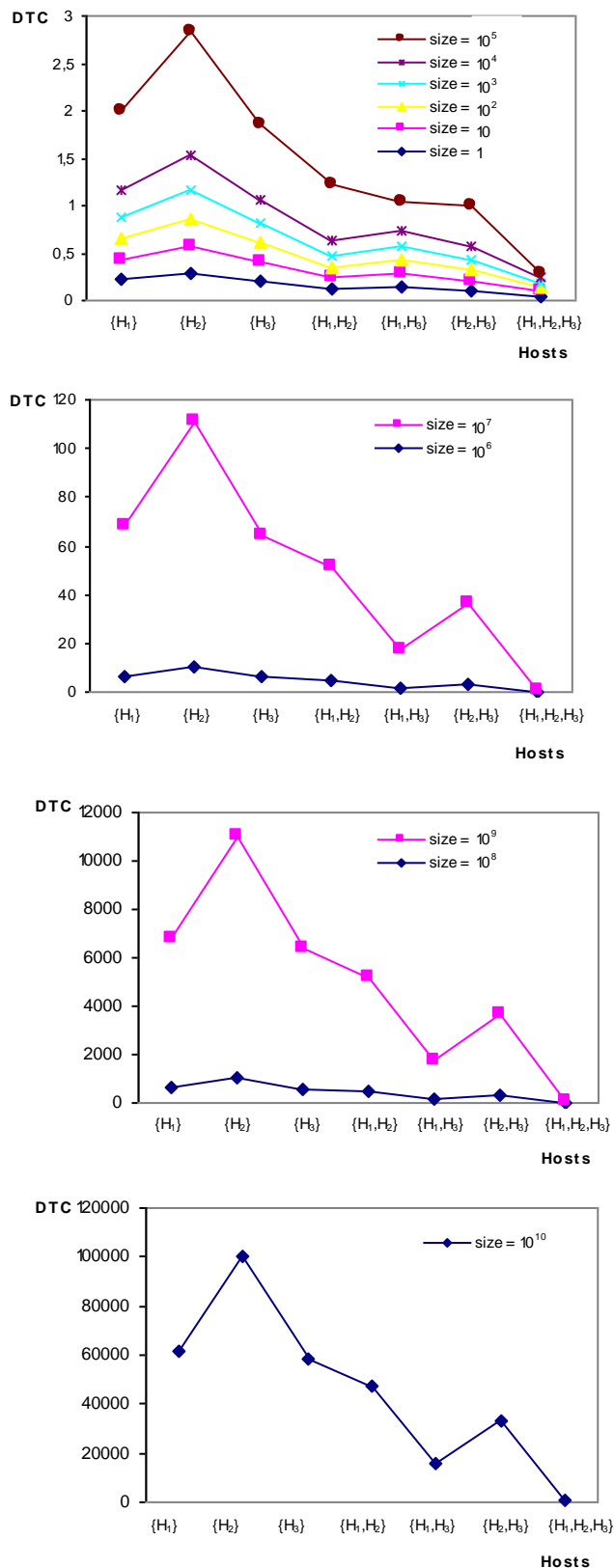


Figure 4. Variation of Component DTC According to their Locations.

VI. CONCLUSION

With the evolution of networks and software component, the deployment process becomes more complex and must cover the classical deployment activities (e.g., release, install, activate, update, adapt, de-install, de-release). Many component systems (e.g., EJB, .Net and CCM) currently exist. Therefore, a generic deployment model that wraps all these component systems would be indispensable. The main contributions of this study are twofold:

- Proposing a generic deployment infrastructure based on D&C specification and MDA approach. The proposed approach is tested with EJB model, but it can be obviously extended to other specific model.
- Applying a method designed to define made-to-measure distances for any given transaction network. The relevance of this provided distance is clearly enhanced by using graph algorithm.

Actually, experimentation is made by testing and evaluating the performance of EJB model deployment. Future research can be performed in various viewpoints.

We selected the most interesting ones:

- Integration of new component software and application architectures such as (e.g., CCM, service oriented architecture...).
- A
- Extending to others performance parameters such as Computation Task Cost (CTC) which take into account the complexity of the computation according to the request data size, the provider capacity and load.

REFERENCES

- [1] D. H. Akehurst and S. J. H. Kent, "A Relational Approach to Defining Transformations in a Metamodel", Proc. Unified Modelling Language (UML 05), Springer Berlin/Heidelberg, 2005, pp. 243-258.
- [2] F. Berman, "Adaptive Computing on the Grid using AppLes," IEEE Transactions on parallel and distributed systems, vol. 14, 2003, pp. 369-82.
- [3] A. Benamar, N. Belkhatir, and F. T. Bendimerad, "A Proposition of Generic Deployment Platform for Component-based Applications," Journal of Software Engineering, Academic Journals Inc, vol. 2, 2008, pp. 23-38.
- [4] J. Bézivin and F. Jouault, "Using ATL for checking models," Proc. Graph and Model Transformation (GraMoT 06), 2006, pp. 69-81.
- [5] J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez, "Modelling in the Large and Modelling in the Small," Proc. MDA Workshops Foundations and Applications (MDAFA 04), Springer Berlin/Heidelberg, 2004, pp. 33-46.
- [6] A. Carzaniga, A. Fuggetta, R. S. Hall, D. Heimbigner, A. Van der Hoek, and A. L. Wolf, "A Characterization Framework for Software Deployment Technologies," Technical Report CU-CS-857-98, University of Colorado, 1998.
- [7] G. Deng, J. Balasubramanian, W. Otte, D. C. Schmidt, and A. Gokhale, "DAnCE: A QoS-enabled Component Deployment and Configuration Engine," Proc. Component Deployment (CD 05), Springer Berlin/Heidelberg, 2005, pp. 67-82.
- [8] M. Dibo and N. Belkhatir, "Defining an Unified Meta Modeling Architecture for Deployment of Distributed Components-based Software Applications," Proc. International Conference on Enterprise Information Systems, (ICEIS 10), SciTePress, vol. 1, 2010, pp. 316-321.
- [9] M. Faerman, A. Su, R. Wolski, and F. Berman, "Adaptive Performance Prediction for Distributed Data-Intensive Applications," Proc. High Performance Networking and Computing (HPNC 99), ACM/IEEE, 1999.
- [10] A. Gerber, M. Lawley, K. Raymond, J. Steel, and A. Wood, "Transformation, the Missing Link of MDA," Proc. Graph Transformation (GT 02), Springer Berlin/Heidelberg, 2002, pp. 90-105.
- [11] J. Gossa and Jean-Marc Pierson, "End-To-End Distance Computation In Grid Environment by NDS, the Network Distance Service," Proc. European Conference on Universal Multiservice Networks (ECUMN 07), IEEE Computer Society, 2007, pp. 210-222.
- [12] J. Gray, Y. Lin, and J. Zhang, "Automating Change Evolution in Model-Driven Engineering," Special issue on Model-Driven Engineering, IEEE Computer Society, vol. 39, 2006, pp. 51-58.
- [13] P. Hnětynka, "Making Deployment of Distributed Component-based Software Unified," Proc. Automated Software Engineering (ASE 04), Computer Society, 2004, pp. 157-161.
- [14] F. Jouault and I. Kurtev, "Transforming Models with ATL," Proc. Model-Driven Engineering Languages and Systems (MODELS 05), Springer Berlin/Heidelberg, 2005, pp. 128-138.
- [15] M. Karlsson and M. Mahalingam, "We Need Replica Placement Algorithms in Content Delivery Networks?" Proc. Web Content Caching and Distribution Workshop (WCW 02). Boulder Editions, 2002, pp. 117-128.
- [16] B. Lowekamp, B. Tierney, L. Cottrell, R. Hughes-Jones, T. Kielmann, and M. Swany, "A Hierarchy of Network Performance Characteristics for Grid Applications and Services," Proposed Recommendation Global Grid Forum (GGF), Network Measurement Working Group (NMWG), 2004.
- [17] N. Merle and N. Belkhatir, "Open Architecture for Building Large Scale Deployment Systems," Proc. Software Engineering Research and Practice (SERP 04), 2004, pp. 930-936.
- [18] R. Monson-Haefel and B. Burke, Enterprise JavaBeans 3.0, O'Reilly Media, Inc, 5th Edition, USA, 2006
- [19] OMG, "Deployment and Configuration of Component-based Distributed Applications Specification," 2004, <http://www.omg.org/docs/ptc/04-08-02.pdf>
- [20] OMG, "CORBA Component Model: CCM version 4.0," 2006, <http://www.omg.org/spec/CCM/4.0/PDF>
- [21] J.S. Qiang Xu, "Automatic Clustering of Grid Nodes," Proc. Grid Computing (GC 05), IEEE/ACM, 2005, pp. 227-233.
- [22] K. Seymour, A. YarKhan, S. Agrawal, and J. Dongarra, NetSolve: Grid Enabling Scientific Computing Environments, Grid Computing and New Frontiers of High Performance Processing, Lucio Grandinetti eds., Elsevier, Advances in Parallel Computing, vol. 14, 2005.
- [23] D. Varró, G. Varró, and A. Pataricza, "Designing the automatic transformation of visual languages," Science Computing Programming, vol. 44, 2002, pp. 205-227.
- [24] A. J. A. Wang and K. Qian, Component-oriented Programming, 1st edition, John Wiley and Sons Inc., Chichester, UK, 2005.
- [25] R. Wolski, N. T. Spring, and J. Hayes, "The Network Weather Service: a Distributed Resource Performance Forecasting Service for Meta-computing, Future Generation," Computer Systems, vol. 15, 1999, pp.757-768.

A Proof-based Approach for Verifying Composite Service Transactional Behavior

Lazhar Hamel
MIRACL, ISIMS, TUNISIA
lazhar.hamel@gmail.com

Mourad Kmimech
MIRACL, ISIMS, TUNISIA
mkmimech@gmail.com

Mohamed Graiet
MIRACL, ISIMS, TUNISIA
mohamed.graiet@imag.fr

Mohamed Tahar Bhiri
MIRACL, ISIMS, TUNISIA
Tahar_bhiri@yahoo.fr

Walid Gaaloul
Computer Science Department Télécom SudParis
walid.gaaloul@it-sudparis.eu

Abstract— Web services are software components accessible via Internet. Web services are defined independently from any execution context. A key challenge of Web service compositions is how to ensure reliable execution. Due to their inherent autonomy and heterogeneity, it is difficult to reason about the behavior of service compositions especially in case of failures. In this work, we propose an approach to formalize a model of Web services composition to check and ensure reliable execution. To achieve this, we propose a proof oriented approach for the formalization and verification of transactional behavior of web services composition using Event-B.

Keywords— web service composition; Event-B; transactional web service; proof; verification.

I. INTRODUCTION

Web services are emergent and promising technologies for the development, deployment and integration of applications on the internet. One interesting feature is the possibility to dynamically create a new added value service by composing existing web services, eventually offered by several companies. Due to the inherent autonomy and heterogeneity of web services, the guarantee of correct composite services executions remains a fundamental problem issue. An execution is correct if it reaches its objectives or fails properly according to the designer's requirement or users needs. The problem, which we are interested in, is how to ensure reliable web services compositions. By reliable, we mean a composition where all the executions are correct.

Some web services are used in a transactional context, for example, reservation in a hotel, banking, etc.; the transactional properties of these services can be exploited in order to answer their composition constraints and the preferences made by designers and users. However, current

tools and languages do not provide high-level concepts for express transactional composite services properties. The execution of composite service with transactional properties is based on the execution of complex distributed transactions which eventually implements compensation mechanisms. A compensation is an operation the goal of which is to cancel the effect of other transaction that failed to be successfully completed. several transactions models previously proposed in databases, distributed systems, collaborative environments. In order to manage with this focus many specifications proposed to response to this aspects. WS-Coordination [1], WS-AtomicTransaction [2] and WS-BusinessActivity [3]. Many research in this field aiming for instance to guarantee that an activity is cancellable and / or compensable. The verification step will help ensure a certain level of confidence in the internal behavior of an orchestration. Several approaches have been proposed in this direction, based on work related to the transition system [4], process algebras [5], or the temporal theories [6].

Our work deal with the formal verification of the transactional behavior of web services composition. In this paper, we propose to address this issue using proof and refinement based techniques, in particular the Event-B method [7] used in the RODIN platform [8]. Our approach consists on a formalism based on Event-B for specifying composite service (CS) failure handling policies. This formal specification is used to formally validate the consistency of the transactional behavior of the composite service model at design time, according to users' needs. We propose to formally specify with Event-B the transactional service patterns. These patterns formally specified as events and invariants rule to check and ensure the transactional consistency of composite service at design time. Most previous work is based on the model checking technique and does not support the full description of transactional web services. Refinement and proof techniques offered by Event-

B method are used to explore it and in section 5 we discuss this approach.

This paper is organized as follows. In Section 2, we introduce a motivating example. Section 3 presents the Event- B method, its formal semantics and its proof procedure and introduces our transactional CS model. In Section 4, we present how we specify a pattern-based of the transactional behavior using the Event-B. An overview of the validation methodology is given in Section 5.

II. MOTIVATING EXAMPLE

In this section, we present a scenario to illustrate our approach we consider a travel agency scenario (Figure 1). The client specifies its requirement in terms of destinations and hotels via the activity “Specification of Client Needs” (SCN). After SCN termination, the application launches simultaneously two tasks “Flight Booking” (FB) and “Hotel Reservation” (HR) according to customer’s choice. Once booked, the “Online Payment” (OP) allows customers to make payments. Finally travel documents (air ticket and hotel reservations are sent to the client via one of the services “Sending Document by Fedex” (SDF) ,”Sending Document by DHL” (SDD) or “ Sending Document by TNT” (SDT). To guarantee outstanding reliability of the service the designers specify that services FB, OP and SDT will terminate with success. Whereas on failure of the HR service, we must cancel or compensate the FB service (according to his current state) and in case of failure of the SDF, we have to activate the SDD service as an alternative.

The problem that arises at this level is how to check / ensure that the specification of a composite service ensures reliable execution in accordance with the designer’s requirements. To do so, the verification process should cover the composite service lifecycle. Basically, at design time the designer should respect the transactional consistency rules.

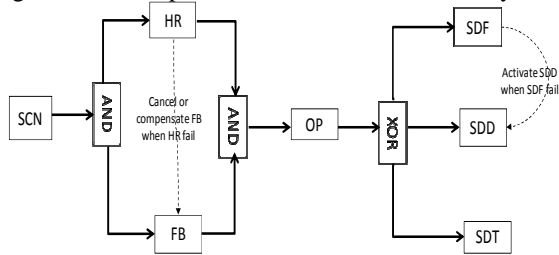


Figure 1. Motivating example

III. FORMALIZING TRANSACTIONAL COMPOSITE SERVICE WITH EVENT-B

To better express the behavior of web services we have enriched the description of web services with transactional properties. Then we developed a model of Web services composition. In our model, a service describes both a coordination aspect and a transactional aspect. On the one hand it can be considered as a workflow services. On the other hand, it can be considered as a structured transaction when the services components are sub-transactions and interactions are transactional dependencies. The originality

of our approach is the flexibility that we provide to the designers to specify their requirements in terms of structure of control and correction. Contrary to the ATMs [9], we start from designers specifications to determine the transactional mechanisms to ensure reliable compositions according to their requirements. We show how we combine a set of transactional service to formally specify the transactional CS model in EVENT-B.

A. Event-B

B is a formal method based on he theory of sets, enabling incremental development of software through sequential refinement. Event-B is a variant of B method introduced by Abrial to deal with reactive system. An Event-B model contains the complete mathematical development of a discrete system. A model uses two types of entities to describe a system: machines and contexts. A machine represents the dynamic parts of a model. Machine may contain variables, invariants, theorems, variants and events whereas contexts represent the static parts of a model .It may contain carrier sets, constants, axioms and theorems.

Refinement: The concept of refinement is the main feature of Event-B. it allows incremental design of systems. In any level of abstraction we introduce a detail of the system modeled. A series of proof obligations must be discharged to ensure the correction of refinement as the proof obligations of the concrete initialization, the refinement of events, the variant and the prove that no deadlock in the concrete and the abstract machine.

Correctness checking: Correctness of Event-B machines is ensured by proving proof obligations (POs); they are generated by RODIN to check the consistency of the model. For example: the initialization should establish the invariant, each event should be feasible (FIS), each given event should maintain the invariant of its machine (INV), and the system should ensure deadlock freeness (DLKF). The guard and the action of an event define a before-after predicate for this event. It describes relation between variables before the event holds and after this. Proof obligations are produced from events in order to state that the invariant condition is preserved. Let M be an Event-B model with v being variables, carrier sets or constants. The properties of constants are denoted by $P(v)$, which are predicates over constants, and the invariant by $I(v)$. Let E be an event of M with guard $G(v)$ and before-after predicate $R(v, v')$. The initialization event is a generalized substitution of the form $v: \text{init}(v)$. Initial proof obligation guarantees that the initialization of the machine must satisfy its invariant: $\text{Init}(v) \Rightarrow I(v)$. The second proof obligation is related to events. Each event E , if it holds, it has to preserve invariant. The feasibility statement and the invariant preservation are given in these two statements[10].

- $I(v) \wedge G(v) \wedge P(v) \Rightarrow \exists v' R(v, v')$
- $I(v) \wedge G(v) \wedge P(v) \wedge R(v, v') \Rightarrow I(v')$

An Event-B model M with invariants I is well-formed, denoted by $M \models I$ only if M satisfies all proof obligations.

B. Transactional web service model

By Web service we mean a self-contained modular program that can be discovered and invoked across the Internet. Each service can be associated to a life cycle or a statechart. A set of states (*initial, active, cancelled, failed, compensated, completed*) and a set of transitions (*activate(), cancel (), fail(), compensate (), complete()*) are used to describe the service status and the service behavior. A service *ts* is said to be *reliable(r)* if it is sure to complete after finite number of activations. *ts* is said to be *compensatable(cp)* if it offers compensation policies to semantically undo its effects. *ts* is said to be *pivot(p)* if once it successfully completes, its effects remain and cannot be semantically undone. Naturally, a service can combine properties, and the set of all possible combinations is $\{r; cp; p; (r; cp); (r; p)\}$ [11].

The initial model includes the context *ServiceContext* and the machine *ServiceMachine*. The context *ServiceContext* describes the concepts *SWT* which represents all transactional web services and *STATES* represents all the states of a given *SWT*. These states are expressed as constants. A set named *STATES* is defined in the SETS clause which represents the states that describe the behavior of such a service. A set named *TWS* is defined in the SETS clause which represents all transactional web services.

```
CONTEXT ServiceContext
SETS
SWT
STATES

AXIOMS
Axm1: STATES= {active, initial, aborted, cancelled, failed,
completed, compensated}
```

The service state which is represented by a functional relation *service_state* defined in VARIABLES clause gives the current state of such a service. The transactional behavior of a transactional web service is modeled by a machine. *Inv1* the invariant specifies that *service_state* is a total function, and that each service has a state.

```
MACHINE ServiceMachine
SEES ServiceContext
VARIABLES
Service_state
SWT_C
SWT_P
SWT_R
INVARIANTS
Inv1: service_state ∈ SWT → STATES
Inv2: SWT_C ⊆ SWT
Inv3: SWT_R ⊆ SWT
Inv4: SWT_P ⊆ SWT
```

In our model, transitions are described by the event. For instance the event *activate* changes the status of a service and pass it from initial status to active. The event *compensate* enables to compensate semantically the work of a service and pass it from completed status to compensate. The event *retry* changes the status of a service and activate it after his failure and pass it from failed status to active. The event

complete enables to finite the execution of a service with success and pass it from active status to completed.

```
Activate ≜ ANY s WHERE
grd1 : s ∈ SWT
grd2 : service_state (s) = initial
THEN
act1 : service_state (s) := active
END

Compensate ≜ ANY s WHERE
grd1 : s ∈ SWT_C
grd2 : service_state (s) = completed
THEN
act1 : service_state (s) := compensated
END
```

C. Transactional composite service

A composite service is a conglomeration of existing Web services working in tandem to offer a new value-added service [12]. It orchestrates a set of services, as a composite service to achieve a common goal. A transactional composite (Web) service (TCS) is a composite service composed of transactional services. Such a service takes advantage of the transactional properties of component services to specify failure handling and recovery mechanisms. Concretely, a TCS implies several transactional services and describes the order of their invocation, and the conditions under which these services are invoked.

To formally specify in Event-B the orchestration we introduced a new context *CompositionContext* which extends the context *ServiceContext* that we have previously introduced. The first refinement includes the context *CompositionContext* and the machine *CompositionMachine* which refine the machine introduced at the initial model. In this section we show how formally the interactions between CS are modeled. We introduce the concept of dependencies (*depA, depANL, depCOMP...*).

```
MACHINE CompositionMachine
REFINES ServiceMachine
SEES CompositionContext
```

Dependencies are specified using Relations concept. It is simply a set of couples of services. For example *depA* represents the set of couples of services that have an activation dependency.

```
Axm1 : depA ∈ SWT ↔ SWT
Axm2 : depAL ∈ SWT ↔ SWT
...
```

These dependencies express how services are coupled and how the behavior of certain services influences the behavior of other services. Dependencies can express different kinds of relationships (inheritance, alternative, compensation, etc.) that may exist between the services. We distinguish between “normal” execution dependencies and “exceptional” or “transactional” execution dependencies which express the control flow and the transactional flow respectively. The

control flow defines a partial services activations order within a composite service instance where all services are executed without failing cancelled or suspended. Formally, we define a control flow as TCS whose dependencies are only “normal” execution dependencies. Alternative dependencies allow us to define forward recovery mechanisms. A compensation dependency allows us to define a backward recovery mechanism by compensation. A cancellation dependency allows us to signal a service execution failure to other service(s) being executed in parallel by canceling their execution. Activation dependencies express a succession relationship between two services s_1 and s_2 . But it does not specify when s_2 will be activated after the termination of s_1 . The guard added to the *activate* event which refines the activate event of the initial model expresses when the service will be active as a successor to other (s) service (s) (only after the termination of these services). For example, our motivating example defines an activation dependency from HR and FB; to OP such that OP will be activated after the completion of HR and FB. That means there are two normal dependencies: from HR to OP and from FB to OP.

At this level, the refinement of the compensate event is a strengthening of the event guard to take into consideration the condition of compensation of a service when a service will be compensated. The guard *grd4* in the *compensate* event in expresses that the compensation of a service *s* is triggered when a service *s0* failed or was compensated and there is a compensation dependency from *s* to *s0*. Therefore compensate allows to compensate the work of a service after its termination, the dependency defines the mechanism for backward recovery by compensation, the condition added as a guard specifies when the service will be compensated.

```

Compensate  $\triangleq$  REFINES Compensate
...
grd4:  $\exists s_0, s_0 \in \text{SWT} \wedge s_0 \rightarrow s \in \text{depCOMP} \Rightarrow ((\text{service\_state}(s_0) = \text{failed}) \vee (\text{service\_state}(s_0) = \text{compensated}))$ 
THEN
act1 : service_state (s) := compensated
END
    
```

IV. TRANSACTIONAL SERVICE PATTERNS

The use of workflow patterns [13] appears to be an interesting idea to compose Web services. However, current workflow patterns do not take into account the transactional properties (except the very simple cancellation patterns category). It is now well established that the transactional management is needed for both composition and coordination of Web services. That is the reason why the original workflow patterns were augmented with transactional dependencies, in order to provide a reliable composition [14]. In this section, we use workflow patterns to describe TCS’s control flow model as a composition pattern. Afterwards, we extend them in order to specify TCS’s transactional flow, in addition to the control flow they are considering by default. Indeed, the transactional flow is

tightly related to the control flow. The recovery mechanisms (defined by the transactional flow) depend on the execution process logic (defined by the control flow).

The use of the recovery mechanisms described through the transactional behavior varies from one pattern to another. Thus, the transactional behavior flow should respect some consistency rules (INVARIANT) given a pattern. These rules describe the appropriate way to apply the recovery mechanisms within the specified patterns. Recovering properly a failed composite service means: trying first an alternative to the failed component service, otherwise canceling ongoing executions parallel to the failed component service, and compensating the partial work already done. The transactional consistency rules ensure transactional consistency according to the context of the used pattern. In the following we formally specify these patterns and related transactional consistency rules using Event-B.

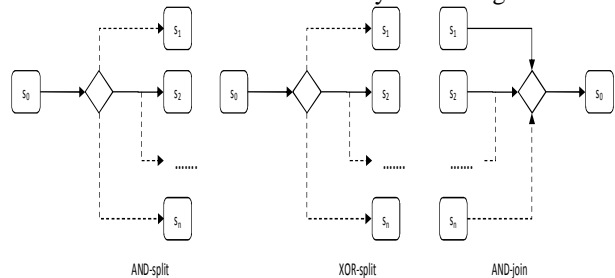


Figure 2. Studied patterns

Our model introduces a new context *And-patternContext* which extends the context *Composition-Context* and a machine transactional patterns which refines the machine *CompositionMachine*. To extend these patterns we introduce new events that can describe them. For example, to extend the pattern AND-split the machine introduces a new event *AND-split* which defines the pattern AND-split. Due to the lack of space, we put emphasis on the following three patterns AND-split, AND-join and XOR-split to explain and illustrate our approach, but the concepts presented here can be applied to other patterns.

An AND-split pattern defines a point in the process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing services to be executed simultaneously or in any order.

```

AND-split  $\triangleq$ 
ANY
S0
SWToutside
WHERE
grd1 : SWToutside  $\subseteq$  SWT_AS
grd2 : S0  $\in$  SWT_AS
grd3 : S0  $\notin$  SWToutside
grd4 : service_state(S0) = complete
THEN
act1 : stateSWTOut := activated
END
    
```

The *SWToutside* represent the set of services (s_1, \dots, s_n) and s_0 is represented by sAS .

To verify the transactional consistency of these patterns we add predicates in the INVARIANT clauses. These invariants ensure transactional consistency according to the context of use. These rules are inspired from [14] which specifies and proves the potential transactional dependencies of workflow patterns. The transactional consistency rules of the AND-split pattern support only compensation dependencies from *SWToutside* (Inv 23).

- Inv 23: $\forall s.s \in SWToutside \Rightarrow sAS \rightarrow s \notin depCOMP$

The compensation dependencies can be applied only over already activated services. The transactional consistency rules supports only cancellation dependencies between only the concurrent services. Any other cancellation or alternative or compensation dependencies between the pattern's services (Inv 11, 12) are forbidden.

- Inv 11: $\forall s.s \in SWT_AS \Rightarrow s \rightarrow sAS \notin depANL$
- Inv12: $\forall s, s_1.s \in SWT_AS \wedge s_1 \in SWT_AS \Rightarrow s \rightarrow s_1 \notin depAL$

Our example illustrates the application of AND-split pattern to the set of services (SCN, HR, FB) and specifies that exist a dependency of compensation from HR to FB and a cancellation dependency also from HR to FB. The guard of the *AND-split* event represents the conditions of activation of the pattern. In our example SCN must terminates its work before activating the pattern. In order to ensure a normal execution of the event an invariant must be preserved by *AND-split* event that express that all *SWToutside* services have an activation dependency from sAS

- Inv 13: $\forall s.s \in SWToutside \Rightarrow sAS \rightarrow s \notin depA$

An AND-join pattern defines a point in the process where multiple parallel subprocesses/services converge into one single thread of control, thus synchronizing multiple threads. To extend the pattern AND-join, the machine introduces a new event *AND-join* which defines the control flow of the AND-join pattern.

```

AND-join  $\triangleq$ 
ANY
S0
SWToutside
WHERE
grd1 : SWToutside  $\subseteq$  SWT_AJ
grd2 : S0  $\in$  SWT_AJ
grd3 : S0  $\notin$  SWToutside
grd4 :  $\forall s.s \in SWToutside \Rightarrow service\_state(s) = complete$ 
THEN
act1 : service_state(S0) := active
END
    
```

Our example illustrates the application of AND-join pattern to the set of services (HR, FB, OP). The guard of the AND-join event represents the conditions of activation of the pattern. HR and FB must terminates its work before activating the pattern. The termination of HR is necessary and not efficient to activate the pattern. All *SWToutside*, HR and FB, services must complete their work.

The transactional consistency rules of the AND-join pattern supports only compensation dependencies for *SWToutside*, sAJ can not be compensated by *SWToutside* services as they are executed after (inv 24).

- Inv 24: $\forall s.s \in SWToutsideAJ \Rightarrow s \rightarrow sAJ \notin depCOMP$

The transactional consistency rules of the AND-join pattern support also cancellation dependencies between only the concurrent services. Any other cancellation or alternative or compensation dependencies between the pattern's services are forbidden.

- Inv25: $\forall s.s \in SWToutsideAJ \Rightarrow s \rightarrow sAJ \notin depANL$

An XOR-split pattern defines a point in the process where, based on a decision or control data, one of several branches is chosen. To extend the pattern XOR-split, the machine introduces a new event *XOR-split* which defines the pattern XOR-split.

```

XOR-split  $\triangleq$ 
ANY
S0
SWToutside
sw
WHERE
grd1 : SWToutside  $\subseteq$  SWT_XS
grd2 : S0  $\in$  SWT_XS
grd3 : S0  $\notin$  SWToutside
grd4 : service_state(S0) = complete
grd5 : sw  $\in$  SWToutside
THEN
act1 : service_state(sw) := active
END
    
```

The XOR-split pattern supports alternative dependencies between only the services *SWToutside*, as the alternative dependencies can exist only between parallel and non concurrent flows. The XOR-split pattern support also compensation dependencies from *SWToutside* to sXS .

- Inv18: $\forall s.s \in SWT_XS \setminus \{sXS\} \Rightarrow s \rightarrow sXS \in depCOMP$

Any other cancellation or alternative or compensation dependencies between the pattern's services are forbidden.

- Inv15: $\forall s.s \in SWT_XS \Rightarrow s \rightarrow sXS \notin depAL$
- Inv22: $\forall s.s \in SWT_XS \setminus \{sXS\} \Rightarrow s \rightarrow sXS \notin depCOMP$

Our example illustrates the application of XOR-split pattern to the set of services (OP, SDD, SDF, SDF) and specifies that exist an alternative dependency from HR to FB. The guard of the *XOR-split* event represents the conditions of activation of the pattern. The execution of OP service must be completed for activate XOR-split pattern. After the activation one service from (SDD, SDF, SDF) will be active.

V. VALIDATION

In the previous section, we showed how to formally specify a TCS using Event-B. The objective of this section is to show how we verify and validate our model using proof and ProB animator[15]. In the abstract model the desired properties of the system are expressed in a predicate called invariant, it has to prove the consistency of this invariant

compared to system events by a proof. We find many proof obligations (Figure 3). Each of them has got a compound name for example, « evt / inv / INV ». A green logo situated on the left of the proof obligation name states that it has been proved (an A means it has been proved automatically). In our case shown in Figure 3 the tool generates the following proof obligations « activate / inv1 / INV » and « compensate / inv1 / INV ». This proof obligation rule ensures that the invariant *inv1* in the *CompositionMachine* is preserved by events activate and compensate. Figure 4 show also the proof obligations «compensate / grd2 / WD ». This proof obligation rule ensures that a potentially ill-defined guard is indeed well defined.

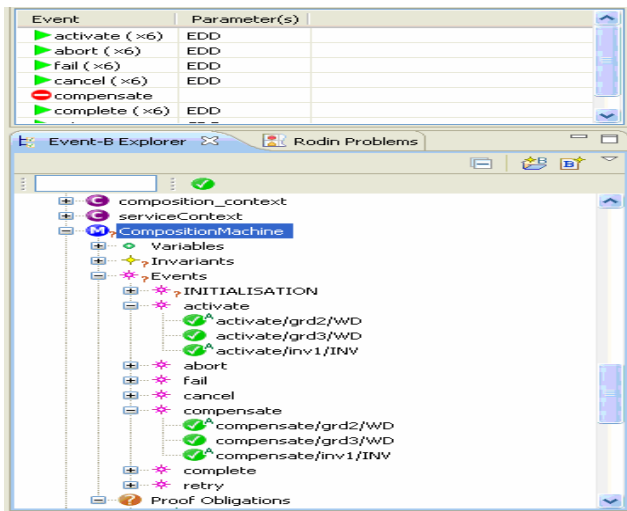


Figure 3. Proof obligations and animation

Our work is proof oriented and covers the transactional web services. All the Event-B models presented in this paper have been checked within the RODIN platform. The proof based approaches do not suffer from the growing number of explored states. However, the proof obligations produced by the Event-B provers could require an interactive proof instead of automatic proofs. Concerning the proof process within the Event-B method, the refinement of transactional web services Event-B models can be performed. This refinement allows the developer to express the relevant properties at the refinement level where they are expressible. The refinement is a solution to reduce the complexity of proof obligations.

In our example the designer can initially specify, as CS transactional behavior, that FB will be compensated or cancelled if HR fails, SDD is executed as alternative of SDF failure. The Event-B formalization of our motivating example defines a cancellation dependency and compensation dependency from HR to FB and alternative dependency from SDF to SDD. For example, by checking the compensation dependency between SCN and HR the RODIN platform mentioned that the proof obligations has not been discharged (Figure 4). As HR is executed after, it

can not exist a compensation dependency from SCN to HR. A red logo with a "?" appear in the proof tree and it means that is not discharged. This basic example shows how it is possible to formally check the consistency of transactional flow using Event-B. To repair this error we can refer to the initialization of the machine and verify the compensation dependencies.

After the initialization of the *ServiceMachine* the compensate event is disabled and after the termination of the execution of a service the event will be enabled. ProB offer to the developer which parameter is used in the animation by clicking right on the event.

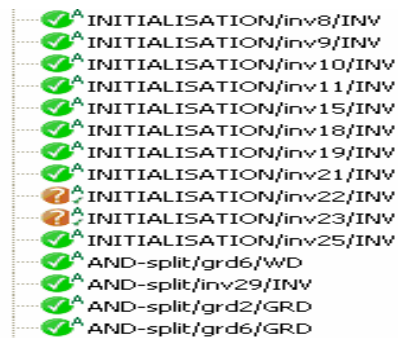


Figure 4. A red logo indicates that the proof obligations is not discharges

In the development of our model some proof obligations are not discharged but the specifications is correct according to our work in [6] which is specified and validated using Event Calculus. To do so, we use ProB animator to verify our specification of transactional web services. This case study has shown that the animation and model-checking are complementary to the proof, essential to the validation of Event-B models. In other case, many proved models (proof obligations are discharged) still contain behavioral faults, which are identified with the animators. The main advantage of Event-B develop that can repair errors during the development. It allows the backward to correct specification. With refinement, the complexity of the system is distributed; the step by step proofs are more readily. Event-B offers more flexibility and expressivity than the input languages of model checkers.

VI. CONCLUSION AND FUTURE WORKS

The paper addresses the formal specification, verification and validation of the transactional behavior of services compositions within a refinement and proof based approach. The described work uses Event-B method, refinement for establishing proprieties. This paper presents our model of web service enriched by transactional properties to better express the transactional behavior of web services and to ensure reliable compositions. Then we describe how we combine a set of services to establish transactional composite service by specifying the order of execution of composed services and recovery mechanisms in case of failure. Finally we introduced the concept of composition

pattern and how we uses it to specify a transactional composite service.

In our future works we are considering the following perspectives:

- Using automation approach of MDE type to verify transactional behavior of services compositions.

REFERENCES

- [1] L.P. Cabrera, G. Copeland, M. Feingold, R.W. Freund, T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, D. Langworthy, M. Little, A. Nadalin, E. Newcomer, D. Orchard, I. Robinson, J. Shewchuk, and T. Storey. Web servicescoordination(ws-coordination), 2005.
- [2] L.P. Cabrera, G. Copeland, M. Feingold, R.W. Freund, T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, D. Langworthy, M. Little, A. Nadalin, E. Newcomer, D. Orchard, I. Robinson, T. Storey, and S. Thatte. Web services atomic transaction (wsatomictransaction), 2003.
- [3] L.P. Cabrera, G. Copeland, M. Feingold, R.W. Freund, T. Freund, S. Joyce, J. Klein, D. Langworthy, M. Little, F. Leymann, E. Newcomer, D. Orchard, I. Robinson, T. Storey, and S. Thatte. Web services business activity framework(ws-businessactivity), 2003.
- [4] R. Hamadi and B. Benatallah, "A petri net-based model for web service composition," Fourteenth Australasian Database Conference (ADC2003), 2003.
- [5] G. Sala`un, A. Ferrara, and A. Chirichiello, "Negotiation among web services using lotos/cadp," European Conference on Web Services (ECOWS 04), 2004.
- [6] W. Gaaloul, S. Bhiri, and M. Rouached, "Event-Based Design and Runtime Verification of Composite Service Transactional Behavior ," IEEE Transactions on Services Computing, 02 Feb. 2010. IEEE computer Society Digital Library. IEEE Computer Society.
- [7] J.R. Abrial: Modeling in Event-B: System and Software Engineering, cambridge edn. Cambridge University Press (2010).
- [8] J.R. Abrial., M. Butler, and S. Hallerstede, "An open extensible tool environment for Event-B," .ICFEM06, LNCS 4260, Springer, pp. 588-605, 2006.
- [9] A. K. Elmagarmid, Database transaction models for advanced applications. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.
- [10] C. Metayer, J. Abrial, and L. Voisin , "Event-B Language. Technical Report D7," z RODIN Project Deliverable, 2005.
- [11] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz, "A transaction model for multidatabase systems." in ICDCS, 1992, pp. 56–63.
- [12] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. K. Elmagarmid, "Business-to-business interactions: issues and enabling technologies," The VLDB Journal, vol. 12, no. 1, pp. 59–85, 2003.
- [13] W. M. P. van der Aalst, A. P. Barros, A. H. M. ter Hofstede, and B. Kiepuszewski, "Advanced Workflow Patterns," in 5th IFCIS Int. Conf. on Cooperative Information Systems (CoopIS'00), ser. LNCS, O. Etzion and P. Scheuermann, Eds., no. 1901. Eilat, Israel: Springer-Verlag, September 6-8, 2000, pp. 18–29.
- [14] S. Bhiri, C. Godart, and O. Perrin, "Transactional patterns for reliable web services compositions," in ICWE, D. Wolber, N. Calder, C. Brooks, and A. Ginige, Eds. ACM, 2006, pp. 137–144.
- [15] M. Leuschel and M. Butler, "ProB: A Model Checker for B," in K. Araki, S. Gnesi, D. Mandrioli (eds), FME 2003: Formal Methods, LNCS 2805, Springer-Verlag, pp. 855-874, 2003.

Certification of MDA Tools: Vision and Application

Oksana Nikiforova, Natalja Pavlova, Antons Cernickins, Tatjana Jakona

Department of Applied Computer Science
Riga Technical University
Riga, Latvia

{oksana.nikiforova, natalja.pavlova, antons.cernickins, tatjana.jakona}@rtu.lv

Abstract—Currently software system development is under the conversion, where traditional code oriented software development is transformed into model driven approach. A lot of methods and tools are proposed to support main statements of the model driven software development. However, there do not exist mechanisms for notification of how much valuable are the Model Driven Architecture (MDA) support tools and how close they are to the main idea of model usage for software development. One of the possible solutions how to solve the problem of selection of appropriate MDA tool can be certification process similar to that in other industries. The paper proposes the framework for such certification and shows an application of it for several MDA support tools.

Keywords-MDA; CASE-tools; certification; modeling.

I. INTRODUCTION

The complexity of software systems permanently increases. It requires from developers carefully select technologies and tools, which will be used during software development process. Researchers and developers try to automate software development process in order to minimize human and material resource costs. Therefore in one hand a big number of CASE-tools were created, each of which support some part of software lifecycle. Functionality of the CASE-tools may be duplicated. The concurrence occurs on this market. Unified opinion which tool is the best and how tools could be evaluated does not exist.

In other hand there are developed different processes, approaches and methods to software development. For example such processes as Rational Unified Process (RUP) [1], Microsoft Solutions Framework (MSF) [2], SCRUM [3], Extreme Programming (XP) [4], etc. exist. Model Driven Architecture (MDA) [5] proposed by Object Management Group (OMG) is popular approach to software development and can be applied within any software development process. Therefore a set of CASE tools, which support also several activities defined by MDA, also appear on the tool market. And it became much more complicated to examine CASE-tools, which also support model and transformation chain of MDA.

The area of the described here research is software development using CASE-tools in the framework of MDA. Software market is crowded with a variety of CASE-tools that automate stages of the development in the framework of MDA. Not developed any standardized procedures or criteria

for how to assess compliance of CASE-tool to standards of MDA, to evaluate what part of the MDA chain considered CASE-tool supports. The goal of this paper is suggest the possibility of certification of CASE-tools based on a considered here evaluation criteria of compliance to the MDA.

The second section describes the difference between the term of Model Driven Software Development (MDSD) and principles of MDA applied for the software development. The third section describes existing researches in the area of MDA tool certification. The fourth section shows example of CASE-tool evaluation for functionality and portability aspects. In the last section the described research is concluded.

II. MODEL DRIVEN APPROACH IN SOFTWARE DEVELOPMENT

Requirements of customers and hence the software becomes more sophisticated and complex with the time. Therefore, developers should be more qualitative and should have tools to satisfy the needs of quality of the software on the level, required by clients, to respect deadlines and to deliver software that functions properly. According to Standish Group [6], only 29% of projects have been succeeded in 2004 (i.e. done in time, met client's expectations, while being not over budget). Paying attention to the development processes of the typical software development company, similar steps, tools and tests will be seen [6]. In order to optimize these activities, model-based approaches may be used, providing manipulations with models under meta-modeling process, as well as the usage of CASE tools for model transformation and generation. This approach to software development is realized with MDD [7].

Model Driven Development appears because there was a necessarily to decrease efforts, to create and use analysis and design models at each stage of the software development process and to automate the transformation of the models [6]. The separation of concerns is another foundation of MDD that provides the separation of high-level business logic from system's architecture and deployment platform. MDA initiative, the primary example of MDD, was introduced by OMG in 2001 to satisfy the needs of the modern software industry [5], [8].

MDA proposes to use models on every stage of software development, specifying a set of tools that supports

construction of models with design and architectural patterns [8]. According to traditional software development life cycle, the application of Model Driven Development should consider the modeling approach as such. Unlike MDD, the MDA approach considers models as central part of the development process (assuming that model represents a set of diagrams, used to express the whole software system) [9]. MDA may be considered as a next stage in the evolution of software development process, which tends to bring some improvements into each step of the software development life cycle [8]. MDA is a framework, which contains technical standards developed in the supervision of OMG (in this case, OMG provides the guidelines of MDA application to software development) [8]. There are four principles that underlie the MDA approach [10]:

1. Models constructed with a well-defined notation are a milestones of system representation for enterprise-scale solutions;
2. System development is performed with construction of a set of models and execution of model transformations;
3. Models and transformations among them are described in a formal form with meta-models on MOF this description could serve as a basis for automation through different CASE tools;
4. The broad usage of model-based approaches requires standards to provide satisfaction of costumers and highest qualification of developers.

One of the milestones of MDA considers the text description of the models, formal descriptions of a system, models and code and possibility to apply the formal transformations on every model of a system, to refine it and obtain model, which is closest to user needs [10]. Considering the resources needed for software development, there is a need to reduce the overall production costs, making the software development process more profitable [11]. Here, the reuse of the existing models, patterns or code may be used (thought, it may be a way too complex or impossible). MDA proposes the following set of activities,

which may improve the software development process and make an easier reuse of some components [11]:

1. Choose application model that corresponds with a problem domain;
2. Subset the model as necessary;
3. Choose models in accordance with the implementation technology platform;
4. Define the interconnection between models;
5. Generate the program code for software system.

In many cases, the necessity of introducing some changes into developed system (or system under development) appears. From this point, changes are introduced into the application model only (1) — changes will be automatically provided to the lower models. When the environment of system development should be changed, models for the new environment should be selected (3); program code should also be regenerated (5) [12]. Therefore, the application models are not changed, meaning that costs are lower, productivity is higher, as well as the maintenance of the system becomes much cheaper. With this approach each model, which is constructed in the framework of MDA guidelines, can be subsequently reused [11]. Fig. 1 shows the supporting component model of Model Driven Architecture. Components in Fig. 1 are depicted into the framework of MDA models and its transformations within the authors defined levels for system domain abstractions [9].

The MDA proposes to construct four basic models for developed system (Fig. 1):

1. Computation Independent Model (CIM) that reflects to business and its models— defined at problem domain level in Fig. 1;
2. Platform Independent Model (PIM) that reflects to analysis and design models of software system to be developed—defined at solution domain level in Fig. 1;
3. One or many Platform Specific Models (PSM) that reflect to detailed design models of software system under construction—defined at software domain level in Fig. 1;

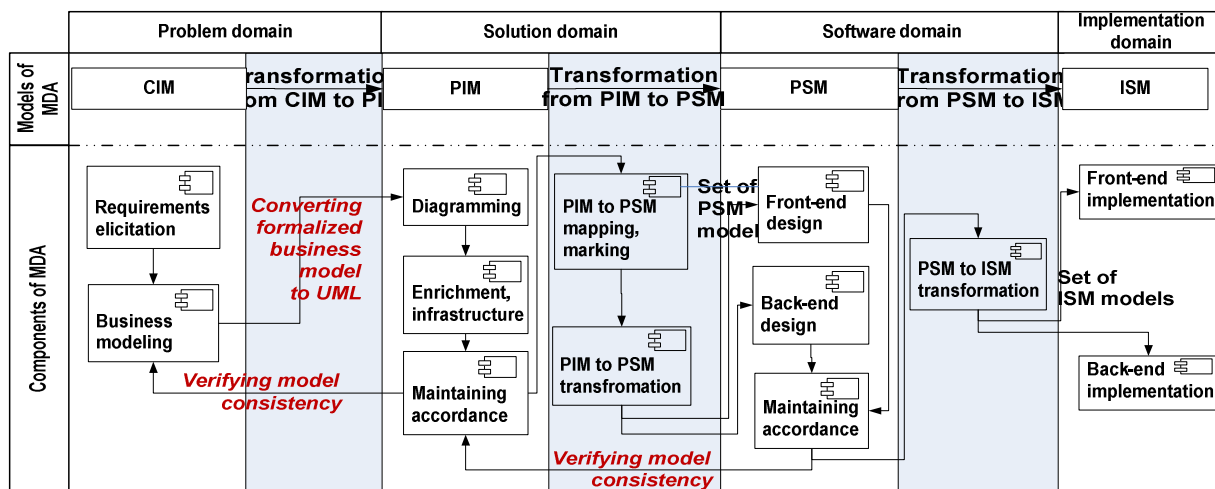


Figure 1. The component model of MDA (adopted from [13])

4. One or many Implementation Specific Models (ISM) that reflect to implementation and runtime models – defined at implementation domain level in Fig. 1.

Also, MDA components may be reflected to the main blocks of Model Driven Development. These blocks are the following [14]:

1. A model repository;
2. One or more domain modeling languages;
3. One or more workbench environments;
4. One or more modeling tools;
5. One or more transformation tools.

The components of MDA, shown on Fig. 1 are representing all of the activities included in the MDA-driven software development process. Dependence on information exchange, which is imported/exported from one component to another, is written on the arrows between components on Fig. 1.

Regular font on arrows between components means that the ability of import/export of models is possible. Transitions between components, which can be performed only by the human at this time, are expressed in italic. Authors' research [15] discusses different types of model transformations (e.g. formal, semi-formal, based on hints or manual) to satisfy the statements of MDA on model transformations. Up to the year 2006 the conclusion stated that there is no solution available to define the complete transformation CIM-to-PIM-to-PSM-to-Code. The weakest link here is exactly the construction of PIM or transformation from CIM to PIM. Solutions focused on construction of CIM and CIM-to-PIM transformations cannot insure that PIM is containing all the necessary information, as well as that the presentation of PIM is formal enough to be able to transform it into correct PSM [15]. Authors' efforts to find CASE tools up to date for CIM-to-PIM transformation and to state the component to support that activity carry to the point, that still there is no guarantee result of CIM-to-PIM transformations. Also, the verification of model consistency is under investigations and the role of model interchange and interchange standards become more and more important.

III. MAIN CONCEPTS OF MDA TOOLS CERTIFICATIONS

The idea lying behind the research is to provide a set of guidelines on the actual implementation of the MDA for the purpose of promoting it as a holistic approach for software development across the IT community. A branch of standards provided within MDA is defined in a form of specification, meaning that the specification-based testing may be used as a basis for compliance assessment [16]. In particular, the conformance statement for CORBA provided by The Open Group [17] is done this way. In fact, the compliance itself is nothing else but the satisfaction of software implementation to the standard specification [16]. [16] comes with an idea of considering the compliance test suite generation as a branch of constraint satisfaction problem, in which the first-order predicate is given and processed to find models that satisfy it. Following this work, instead of starting from a concrete set of constraints and trying to find the appropriate models, the construction (as

well as the further classification) of all possible models is considered.

When it comes to development of a new certification scheme, the first and the foremost task is to define the object of certification [18]. According to [18], the following types of certification are possible:

- Product certification (accordance with particular technical standard);
- Process certification (accordance with ISO 9000 or similar standard);
- Personnel certification;
- Accreditation of certification bodies (the certification of certifiers).

[18] summarizes the study on various certification schemes and categorizes them into several groups, also providing a general structure of certification process itself, as well as presenting a new certification scheme used in space technology.

In fact, the type of certification procedure for current research can be determined as a combination of both the product and the process certification. Such a mixture of types will provide a more detailed outlook on various options to be considered in the certification scheme.

Basically, the former type of certification is considered, as software development tools (i.e., software products) are involved in the research. This may also include the specification of the most common features and options defined to clarify the accordance level of each tool from various perspectives (discussed in [19]).

As far as MDA-oriented software development life cycle represents the process, the latter type of certification should also be considered.

In order to provide a solid background for the certification scheme, as well as to clarify the means of the MDA tool as such, [20] is considered. [20] reviews the MDA approach within the variety of the CASE tools, which are proposed as supporting for MDA activities. The provided specification of MDA tools consists of seven categories, which definition and details are described in [20]:

1. Accordance with MDA-oriented life cycle—the accordance level of software development life cycle supported by a tool, which includes MDA-oriented activities combined into such subcategories as knowledge formalization (CIM), system model refinement (PIM), PIM-to-PSM mapping, system model implementation (PSM), and transformation support;
2. Functional capabilities—the functional capabilities of a tool in such fields as environment, modeling, implementation, testing, documenting, project management, configuration management;
3. Reliability—the capability of a tool to maintain the appropriate level of performance under certain conditions for a certain period of time, including repository management, automatic backup capabilities, data access management, error processing capabilities, as well as fault analysis capabilities;
4. Usability—usage efforts and individual assessments of such usage, including user interface, licensing and

localization options, ease of use, quality of documentation etc.;

5. Efficiency—the amount of resources needed to maintain the appropriate level of performance under certain conditions, including technical requirements, workload efficiency, as well as performance;

6. Maintainability—efforts needed to make specified modifications;

7. Portability—ability of a tool to be transferred to another environment.

The mentioned criteria involve several aspects of the features of the CASE-tool, such as usability and application. Current research is devoted to evaluation of CASE-tools regarding to modeling and implementation capabilities as it is important development property in the framework of MDA [21].

In order to clarify a vision on a certification scheme to assess the compliance of MDA tools, a conceptual framework is proposed. In fact, this framework should be used to verify the output produced by MDA tools. Whereas a wide variety of the tools intended for specific purposes (e.g., mapping definition) may be used [19], an additional specification-based assessment of these tools is considered (discussed in [19]).

In short, models defined by MDA are used to describe the MDA-oriented software development life cycle [11], [19], [5], namely they are CIM, PIM, PSM and ISM.

However, the only models to be specified and promoted by OMG (i.e., described in details) are PIM and PSM [11]. In fact, OMG does not provide any specific requirements for CIM (meaning that it is not “computational,” not formal enough, etc.), as well as ISM itself — the actual source code generated from PSM—from modeling perspective looks out of scope. Despite this, all four layers are somehow covered by various software development tools.

The conceptual framework considers these four models as individual blocks, each of them having their own input and output. The origin of this idea has come from black box testing [22]: whereas software system is considered as a black box, the only thing to be analyzed is the output produced by specific input. Therefore, developer does not need to understand why the compiled code does what it does; here, the requirements are used to determine the correct output of black box testing.

In fact, the main artifacts for the conceptual framework are inputs and outputs. As far as CIM and ISM are out of scope from the perspective of OMG standards, the conceptual framework does not cover the according artifacts. The actual tool use in each block (i.e., what operations are performed) is also not the matter of high importance.

However, the main concern for each tool is the support of XMI standard [23]. In order to perform a transition from raw output to qualified input, the conceptual framework assesses the output from each tool. If tool conforms to OMG standards, then the output from this tool should be opened in other tool with no problems. If not, the conceptual framework would provide an appropriate suggestion on where the root of the problem lies.

While OMG does not provide any constraints (i.e., does not restrict) on the modeling language notation used with MDA (however, the use of UML is strongly recommended) [11] [5], the use of XMI for assessment of software development tools seems to be the only valuable option. This assessment is considered to be formal: a specification is said to be formal when it is based on a language that has a well-defined semantic meaning associated with each of its constructs [24]. It is this formalism, which allows the model to be expressed in a format such as XML, in accordance with a well-defined schema (XMI).

The specification of XMI standard as such is used to create the XML Schema of XMI standard [25], which provides a means by which the syntax and the semantics of an XMI document can be validated. XMI Schemas must be equivalent to those generated by the XMI Schema production rules specified in [23]. Equivalence means that XMI documents that are valid under the XMI Schema production rules would be valid in a conforming XMI Schema; in turn, those XMI documents that are not valid under the XMI Schema production rules are not valid in a conforming XMI Schema [23].

After the XML Schema of XMI standard is created, the developed tool creates a document data model, which consists of [25]:

- Vocabulary (element and attribute names);
- Content model (relationships and structure);
- Data types.

This model is used for further validation of XMI documents. Validation can determine whether the XML elements required by [23] are present in the XML document containing model data, whether XML attributes that are required in these XML elements have values for them, and whether some of the values are correct.

IV. EXAMPLE OF TOOL CERTIFICATION PROCESS

In order to examine the modeling and implementation capabilities of tools, a scope of correspondence should be defined first. Considering the information from previous Sections, the main concern is concentrated on PIM, its refinement, as well as further transition to PSM with similar concentration, accordingly. In addition, the specification of MDA tools provided in Section 3 should also be considered.

Based on [26], the following tools have been selected for evaluation:

- ArgoUML 0.28;
- Altova UModel 2009;
- Sparx Systems Enterprise Architect 7.5.843;
- IBM Rational Enterprise Architect 7.0.0;
- MyEclipse Enterprise Workbench 7.1.1.
- MS Visual Studio 2010

[26] considers these tools as UML tools, which provide source code generation capabilities from UML diagrams, as well as reverse engineering capabilities. However, the only use of UML does not guarantee that tool is “MDA compliant”. That is why the most important features of UML tools should be mapped to the appropriate features of the MDA tools.

The compliance to MDA is defined in Section 3 with 7 view points: accordance with MDA-oriented life cycle; functional capabilities; reliability; usability; efficiency; maintainability and portability. Currently selected CASE-tools are evaluated according some of functional capabilities, namely, modeling and implementation, which are represented with UML support and programming languages support. As well as for portability, this is presented with supporting of different platforms, interchange format and programming languages in source code generation and reverse engineering. Therefore, a model defined in appropriate modeling notation (as was mentioned before, the use of UML is suggested), a model enrichment (transition) to meet the specifics of selected platform, generation of platform-specific source code, as well as support for MOF/XMI should be considered as the most important features of these tools.

Other features like configuration management, testing, project management, etc. are the matter of secondary importance.

These tools feature a source code generation approach based on template definition, meaning that a file (i.e., template) describing the use of meta-data information should be defined first. If several tasks are considered, it is possible to define a set of templates, where each template deals with an appropriate task (here, a nested hierarchy is considered, where main template contains information about complementary templates). Certain tools (such as UModel and Enterprise Architect, namely) provide an ability to redefine the set of supplied generation templates, whereas other tools are unable to provide such a feature.

Table 1 provides an outlook on several features declared by tool vendors that are important for correspondence with the proposed approach (based on [26]).

TABLE I Declared features of corresponding UML tools (based on [26])

	ArgoUML	Altova UModel	Sparx Systems Enterprise Architect	IBM Rational Rose Enterprise	MyEclipse Enterprise Workbench	MS Visual Studio 2010
<i>Common features</i>						
UML	1.4	2.2	1.3, 1.4, 2.0, 2.1	1.4	2.1	2.0
UML Profiles	•	•	•		•	•
MOF/XMI	1.1, 1.2	2.1	1.1, 1.2, 2.1		1.0	2.1
XMI import/export	•	•	•		•	•
<i>UML Diagram support</i>						
Class	•	•	•	•	•	•
Component	•	•	•	•	•	•
Composite structure		•	•		•	
Deployment	•	•	•	•	•	
Object	•	•	•	•	•	•
Package		•	•			•
Profile		•	•		•	•
Activity	•	•	•	•	•	•

State machine		•	•		•	
Statechart UML 1.x	•		•	•		
Use case	•	•	•	•	•	•
Communication			•			•
Collaboration UML 1.x	•		•	•		
Interaction overview			•		•	
Sequence	•	•	•	•	•	
Timing			•		•	
<i>Source code generation capabilities</i>						
CORBA IDL			•	•		
Java	•	•	•	•	•	
C++	•	•	•	•	•	•
C#	•	•	•	•		•
VB.NET		•	•	•		•
PHP	•		•			
Other			Ada, Python, ActionScript			J#, JScript
<i>Reverse engineering capabilities</i>						
CORBA IDL	•		•	•		
Java	•	•	•	•	•	•
C++		•	•	•		•
C#		•	•	•		•
VB.NET		•	•	•		•
PHP			•			
Other			C, Python, Visual Basic, ActionScript			J#, JScript

To sum up, UModel and Enterprise Architect provide the richest set of functional features, with the latter being the most functional one in terms of source code generation and reverse engineering capabilities. However, when it comes down to interoperability among the tools—the main concern for the proposed conceptual framework—even those with same version of XMI standard fail. In theory, the project developed in ArgoUML should be operable in Enterprise Architect easily due to the same version of XMI standard used in both tools (and vice versa). Similar arguments are also exposed on such tools as UModel and Enterprise Architect for the same reason. The most common error relates to incorrect syntax in XMI files, which clearly outlines the problems with proper implementation of standards from the side of vendors.

Microsoft Visual Studio could be used as logical sequel of previously examined tool. This tool does not support modeling activities, but support different programming languages for software development.

V. CONCLUSIONS

The paper discusses possibility of certification of MDA CASE-tools, to find out some standard in existing assortment of tools. Basic principles of MDA were examined to achieve this goal. The paper defines components of MDA, and relationships among them. During this research basic

principles of certification and defined properties of certification corresponding to MDA were found. Tool certification principles are important milestone to understand how certification could be performed and which result we want to obtain.

During this research 6 tools were examined within the correspondence to modeling capabilities in the framework of MDA: ArgoUML, Altova UModel, Sparx System Enterprise Architect, IBM Rational Enterprise Architect, My Eclipse Enterprise Workbench and MS Visual Studio 2010. The first five tools are pure modeling tools, and the last one is MS Visual studio positioned as development tool with modeling capabilities.

The main contribution of the paper is the stressed necessity for CASE tools certification. The paper shows possibility of CASE tools certification in the context of existing concepts. Possibility of tool verification in accordance to proposed framework is shown in example of 6 CASE tools analysis.

With such an abundance of various CASE-tools, both commercial and open-source, their certification is required. Due to the fact that MDA is now the most widely used approach in software development, it makes sense to certify the CASE-tools in the framework of MDA. It is important to identify the main criteria to determine conformance of CASE-tool to the standards of MDA, and in the same time these criteria should display conformance of CASE-tool to the tasks facing the developer. The entire certification process as a whole will only improve the quality of CASE-tools and provide the ability to track information about new features in the developed CASE-tools.

ACKNOWLEDGMENT

The research reflected in the paper is supported by Grant of Latvian Council of Science No. 09.1245 "Methods, models and tools for developing and governance of agile information systems" and by ERAF project "Evolution of RTU international collaboration, projects and capacity in science and technologies".

REFERENCES

- [1] I.Jacobson, G.Booch, J.Rumbaugh, "The Unified Software Development Process", Addison-Wesley, 1999.
- [2] Microsoft Solution Framework: <http://technet.microsoft.com/en-us/library/bb497059.aspx>
- [3] Introduction to Scrum - an Agile Process: <http://www.mountaingoatsoftware.com/topics/scrum>
- [4] Extreme Programming: A gentle introduction: <http://www.extremeprogramming.org/>
- [5] MDA Guide, version 1.0.1: <http://www.omg.org/docs/omg/03-06-01.pdf>
- [6] R.Bendraou, P.Desfray, M.Gervais, A.Muller, "MDA Tool Components: a proposal for packaging know-how in model driven development," Software and Systems Modeling, Vol.7, No.3., Springer, Berlin 2008, pp.329-343.
- [7] J. Krogstie, "Integrating enterprise and IS development using a model driven approach," Proc. 13th International Conference on Information Systems Development—Advances in Theory, Practice and Education, Springer. New York, 2005, pp.43-53.
- [8] M.Guttman, J.Parodi, "Real-Life MDA: Solving Business Problems with Model Driven Architecture," Morgan Kaufmann, San Francisco, 2007.
- [9] O.Nikiforova, V.Nikulshins, U.Sukovskis, "Integration of MDA Framework into the Model of Traditional Software Development," Frontiers in Artificial Intelligence and Applications, Vol.187, IOS Press. Amsterdam, 2009, pp.229-239.
- [10] A.Brown, J.Conallen, D.Tropeano, "Models, Modeling, and Model Driven Development," S.Beydeda, M.Book, V.Gruhn, (eds.) Model-Driven Software Development, Springer, Berlin, 2005, pp.1-17.
- [11] S.Mellor, K.Scott, A.Uhl, D.Weise, "MDA Distilled: Principles of Model-Driven Architecture," Addison-Wesley, San Francisco, 2004.
- [12] A.Cernickins, O.Nikiforova, "An Approach to Classification of MDA Tools," The 49th Scientific Conference of Riga Technical University, Computer Science, Applied Computer Systems. Riga, 2008, pp 72-83.
- [13] O.Nikiforova, A.Cernickins, N.Pavlova, "Discussing the Difference between Model Driven Architecture and Model Driven Development in the Context of Supporting Tools," The 4th International Conference on Software Engineering Advances (ICSEA), International Workshop on Enterprise Information Systems (ENTISY), IEEE Computer Society, 2009, pp.1-6.
- [14] A.Uhl, "Model-Driven Development in the Enterprise," IEEE Software, Vol.25, IEEE Press, Washington, 2008, pp.46-49.
- [15] O.Nikiforova, M.Kuzmina, N.Pavlova, "Formal Development of PIM in the Framework of MDA: Myth or Reality," The 46th Scientific Conference of Riga Technical University, Computer Science, Applied Computer Systems, Riga, 2006, pp. 42-53.
- [16] P.Bunyakiati, A.Finkelstein, D.Rosenblum, "The Certification of Software Tools with respect to Software Standards," IEEE International Conference on Information Reuse and Integration, 2007.
- [17] CORBA 2.3 Conformance statement template: <http://www.opengroup.org/csq/csqdata/blanks/OB1.html>
- [18] H.Schäbe, "A Comparison of Different Software Certification Schemes": <http://www.sipi61508.com/ciks/schabel.pdf>
- [19] A.Cernickins, O.Nikiforova, "On Foundation for Certification of MDA Tools: Defining a Specification," RTU 50th International Scientific Conference, Computer Science, Applied Computer Systems, 2010, pp.45-51.
- [20] A.Cernickins, "An analytical review of Model Driven Architecture (MDA) tools," Master's thesis. Riga, 2009.
- [21] A.Cernickins, "Clarifying a Vision on Certification of MDA Tools," Scientific Papers, University of Latvia. Vol.757. Computer Science and Information Technologies, Latvia, Riga, 5.-7. July, 2010, pp 23-29.
- [22] I.Sommerville, "Software Engineering" (8th edition), Addison-Wesley, Wokingham, 2006.
- [23] MOF 2.0/XMI Mapping, Version 2.1.1: <http://www.omg.org/spec/XMI/2.1.1/PDF>
- [24] Implementing Model Driven Architecture using Enterprise Architect. Mapping MDA Concepts to EA Features: http://www.sparxsystems.com/downloads/whitepapers/EA4MDA_White_Paper_Features.pdf
- [25] XML Schema: <http://www.w3.org/XML/Schema>
- [26] A.Cernickins, O.Nikiforova, K.Ozols, J.Sejans, "An Outline of Conceptual Framework for Certification of MDA Tools," Model-Driven Architecture and Modeling Theory-Driven Development, Greece, Athens, 22.-24. July, 2010. - pp 60-69.

Automatic Generation of GUI from VDM++ Specifications

VDM++ GUI Builder

Carlos A. L. Nunes

Department of Informatics Engineering
Faculty of Engineering, University of Porto
Porto, Portugal
ei05095@fe.up.pt

Ana C. R. Paiva

Department of Informatics Engineering
Faculty of Engineering, University of Porto
Porto, Portugal
apaiva@fe.up.pt

Abstract—The Vienna Development Method is supported by several tools. These tools allow generating Java code from a VDM++ specification but do not generate a graphical user interface (GUI). This paper describes a generic approach and tool to automatically generate a GUI in Java from a VDM++ specification. The generated GUI calls methods of the VDM++ specification, which allows testing the specification itself in order to increase confidence that it is an accurate description of the intended behaviour. This GUI may evolve to interact with the already supported generation code in Java (for the API) in order to obtain a complete application from a VDM++ specification based on a fully automatic code generation process.

Keywords—Formal Methods; Graphical User Interfaces; Vienna Development Method; Automatic Code Generation

I. INTRODUCTION

In the development of a VDM++ specification, interaction with the underlying model is usually done by the use of an interpreter – VDMTools [1] or VDMJ [2]. Although current tools provide an API to externally use the interpreter [1, 2], they offer little more than a way to establish the connection. As this stands, in order to create a Graphical User Interface (GUI) to interact with a VDM++ specification, a developer is forced to design and implement it from the ground up, and also create the necessary “glue” between the VDM interpreter/tool and the GUI.

Using automatic code generation techniques from a formal specification, this research work puts forward an approach that allows users to interact with a VDM++ specification through an automatically generated GUI. Enabling the developer to execute and test the VDM++ specification without the direct use of an interpreter.

Additionally, the generated GUI may be considered as an evolutionary prototype and be connected with the API code generated by current tools, in the following steps of the development process, in order to provide a complete application obtained by a fully automatic code generation process.

This paper is organized as follows: Section II introduces related work; Section III presents basic concepts related to the context of this work; Section IV describes the GUI generator tool and its approach; Section V presents a case

study; Section VI discusses the results of a case study; and Section VII presents conclusions and future work.

II. STATE OF THE ART

The development of Graphical User Interface (GUI) is, currently, tied to the use of tools and techniques that support the design and implementation of the user interfaces. These tools and techniques vary according to the main problem they focus on and use different approaches in order to achieve the common goal of assisting the developer.

A. Interactive Graphical Tools

Also called GUI builders, this type of tool makes it possible to “drag and drop” interface components into place, in order to create windows and dialogs. Leaving to the developer the task of coding the actions associated to a given interface.

In this manner, the developer can instantly see the final result. Something that is not always straightforward when coding the GUI.

This kind of tool gained its momentum with the NeXT Interface Builder [3].

Two examples of such tools, currently in use, are the Glade interface builder [4] and the interface builder component of the NetBeans integrated development environment [5].

B. Graphical User Interface Markup Languages

Conventional programming methods to develop a GUI use a specific programming language, and often lead to the creation of repetitive, sometimes error prone, and frequently complex code. User Interface Markup Languages address these problems by describing the GUI in a markup language, usually dialects of XML. Relying on sub-applications to interpret and transform the GUI description into program code. This approach, besides reducing the amount of written code, makes it easier for the developer to concentrate on user interface design, instead of functionality [6].

Examples of user interface markup languages include UsiXML [7], XAML [8], XUL [9] and SwiXML [10].

However these languages still rely on the developer to insert functionality using a more conventional approach.

C. Property Models

Graphical user interfaces usually possess dependencies between values manipulated by the user interface, that lead to conditionally enabled GUI elements. The implementation of this aspect of a user interface is time consuming and once again leads to repetitive code. This is the problem property models address.

By maintaining an explicit model of dependencies between parameters of a command, property models can then be used by reusable algorithms to implement enabling or disabling of user interface elements.

But as stated, the model needs to be explicitly defined, requiring the use of a special purpose language or similar construct [11].

D. Formal Language-Based Tools

The motivation behind the use of existing formal method based techniques is a strong emphasis on dialog management. Which for example, in typical graphical installation user interfaces is indeed a very important aspect. However, outside of this user interface style, dialog management by the system does not contribute to having a shortest path between windows.

Other problems with this kind of tool are the difficulty of expressing unordered operations, thus the interface would have a very rigid sequence of required actions; and the need for the developer to learn a new special purpose language [3].

E. Constraints

“A constraint can be thought of intuitively as a restriction on a space of possibilities (...). Mathematical constraints are precisely specifiable relations among several unknown (or variables), each taking a value in a given domain (...)” [12]. This concept can be used to implement several different aspects of a user interface. Two examples of such a tool are Amulet [13, 14] and Subarctic [3].

Relying on constraints, a user interface designer can, for example, easily define that a line has to be attached to a button. In the same way, the colour, position and size of an object can be derived from a relationship with another object expressed by a constraint. At the end, a constraint solver is used to find a solution.

These types of systems offer a simple and declarative specification for implementing a user interface however, as far as we know, they are not used beyond research environments. One of the reasons for this is the inherent unpredictability of the resulting user interface.

The solver will try to find a solution that satisfies all constraints. When several solutions exist, the solver may find one that was not expected by the interface designer.

Another difficulty lies in the debugging of a set of constraints, as locating the bug may not be easily done. A related problem is the need by some solvers, to build the set of constraints in a particular form (for example, in a linear form), or the need for the developer to know some details of how the solver works. Also, it can prove to be difficult to master the declarative programming paradigm of constraints as most developers are used to imperative programming

languages – in which the way to approach problems is different [3].

Nevertheless, constraints are widely used for layout control. NeXTStep, for example, provided a limited form of constraints that could be used to control layout [3]. This form of constraints gained a fair share of usage as the results were more predictable to developers, and was also easier to use. The Java platform also makes use of constraints in the form of layout managers [15].

F. Automatic Model-Based Techniques

The goal of these tools is to free the developer from GUI implementation details, allowing him to focus on developing functionality.

The motivation for this kind of tools may be the rapid development of quality user interfaces; endowing programmers with little to no experience in building user interfaces, the capacity to create high quality user interfaces; automatically creating user interfaces suited for a wide range of platforms, without the need of additional work.

Early examples of such tools are UIDE [3] and HUMANOID [16]. These systems used heuristic rules to select the suitable elements and layout, as well as other details of the user interface specified by the model. A more recent example of an automatic model-based technique generates user interfaces from UML domain and use case models [17].

A common disadvantage in the use of these techniques is the degree of unpredictability. When heuristics are involved, the final result of the user interface specification may be difficult to predict. Another common disadvantage is the need to learn a special purpose modelling language. And due to the inherent difficulty of automatically generating user interfaces, this kind of tools typically place significant limitations on the type of user interfaces they can produce. This usually leads to the generated user interface being not as good as one created by more common programming techniques [3, 18].

G. Summary

The tools or techniques, described above, focused on a specific aspect or problem within GUI development. For this work, the main problems are: user interface design, defining the look and feel; assigning functionality to the interface; and automatic GUI generation. As the basis for the GUI generation process is a formal model, this approach can be considered an “Automatic Model-Based Technique”, with the distinguishing features of not relying on a special purpose modeling language, and the removal of unpredictability. Another new aspect is the use of a XML markup language to describe the user interface, giving a greater degree of freedom to make alterations after the automatic generation. No attention to user interface functionality is required from the developer.

III. BASIC CONCEPTS

A. Formal Methods

Formal methods, in the context of software engineering, are a set of mathematical based languages, techniques and tools to specify and verify systems, in order to develop reliably systems despite their complexity [19]. The use of formal methods does not guarantee correctness, but can reveal system inconsistency, ambiguity, and omissions that otherwise could pass undetected.

For specifying the system and its properties in great detail, a formal method uses a specification language, with mathematical based syntax and semantics. As for system verification, formal reasoning techniques are used [19, 20] [21, 22].

B. The Vienna Development Method (VDM) Language

The Vienna Development Method is one of the oldest formal methods [23]. Initially the method only possessed a meta-language for specification, but evolved to include the VDM++ specification language. The VDM++ language is an object-oriented version of the VDM-SL formal language. Apart from classes, the VDM++ language includes instance variables, operations, functions, types, operators and expressions. As with the VDM-SL, VDM++ allows the definition of invariants, pre-conditions and post-conditions.

Besides basic types, such as Boolean and numeric, the language includes three collection types – set, seq and map. A set consists of a unordered collection without repeated elements of the same type; a seq consists of an ordered collection of elements, allowing repetition; and a map is a finite function relating elements of type A with elements of type B [24] [25].

IV. VDM++ GUI BUILDER

The VDM++ GUI builder generates a GUI from a VDM++ specification. VDM++ can be used to model virtually any kind of system. So the GUI generation approach should be generic enough to work on any kind of modelled system.

A. Architecture

The VDM++ GUI Builder is integrated with the tools developed in the context of Overture Tool Project [26] – an open source project to develop a set of high quality formal modelling tools, built on top of the Eclipse Platform [27].

As such, the VDMJ engine [2] is used to execute and evaluate VDM instructions, as well as providing the bulk of the information about the VDM++ specification necessary by the GUI generator.

The other major external tool (not part of Overture) used is the SwiXML Engine [10]. This engine is used to render the GUI elements from a XML description generated by the VDM++ GUI Builder. This tool was chosen because it is specifically designed for Java applications and possesses a very simple mechanism for UI element search. The tool optionally assigns an id for each UI element, which can be used in runtime mode to retrieve the corresponding UI element.

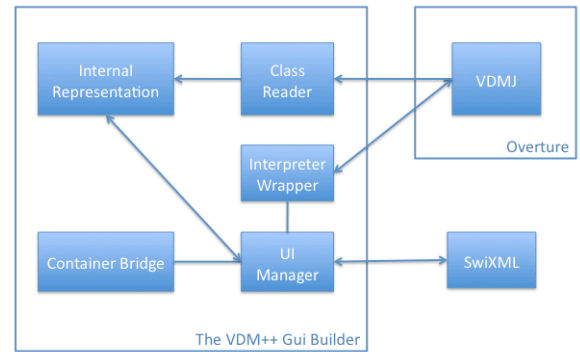


Figure 1. Diagram of the architecture

As shown in Figure 1, the architecture has five major modules:

- The interpreter wrapper,
- The class reader,
- The UI manager,
- The container bridge,
- The internal representation.

The Interpreter Wrapper serves to establish a link between the external VDMJ engine and the VDM++ GUI Builder. It allows calling VDM++ specification methods and retrieving the result.

The Class Reader is used to collect/maintain an internal representation of the information about the VDM++ classes inside the specification, for instance, their operations, functions, constructors and other elements, tailored for the purposes of GUI generation. This module relies heavily on VDMJ to extract such information. Even though, this module can be replaced with another one in order to use the VDM++ GUI Builder with other tools different from the ones available within the Overture project.

The UI Manager is used to create the windows of the GUI, and serves as an intermediary to the functionality of the underlying VDM specification during runtime.

The Container Bridge, serves as a backend to a window. Basically providing actions during runtime to the events of the user interface and a wrapper for a generated window.

Finally, the Internal Representation is an internal depiction of the VDM++ specification from which the GUI will be generated.

B. Annotations

In order to provide extra information not extractable from a pure VDM++ specification, some annotations were defined. These annotations are written within VDM++ comments (starting with "--") so that it does not require an extension to the VDM++ grammar. The annotations take the form of "--@name=value" or "--@name" and are handled separately by the approach.

The annotations are intended for VDM++ classes, operations and functions. There are two specific annotations for methods (operations or functions), "--@press" and "--@check=<value>" and one for classes "--@nowindow". The press annotation is intended to identify methods that describe

possible user action, and “--@check=<value>” is used to retrieve information – value is used to name the state variable with the required information. This annotation can only be applied to methods without arguments. As for the “--@nowindow” class annotation, it serves to mark classes that are to be ignored by the GUI generation process. These would be auxiliary classes in the specification that are not converted to windows on the generated GUI.

C. GUI Generation Strategy

As previously stated, a VDM++ specification is the basis for the GUI generation process. As this formal specification can be used to describe almost any kind of system, and lacks any intentional GUI oriented elements, the generation strategy relies primarily on signature analysis of methods to create the GUI elements.

The GUI generation strategy supports two different generation modes. One ignoring annotations and another one using annotations to guide the GUI generation process.

The strategy assumes that each class is a valid basis for a single window. In a specification with *n* classes (not annotated with “--@nowindow”), the resulting GUI will have *n*+2 windows – two additional windows, one with *n* buttons to give access to the other windows (Figure 8), and another to show all the class instances created in each moment of the execution, for debugging purposes (Figure 10).

Apart from annotations, the GUI elements are generated from the analysis of the signatures of the methods of the underlying class.

Not relying on annotations, a method will lead to the generation of input data GUI elements for the arguments, a button with the name of the method, and in cases where there is a return value, an output data GUI element (Figure 2). In cases where the parameter is a class, the generated GUI provides a combo box with the class instances created until that moment.

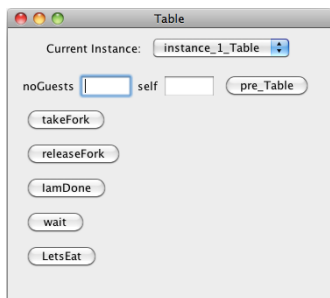


Figure 2. Example of a generated window from the “Dining” VDM++ example (//overture.svn.sourceforge.net/)

Relying on extra information provided by annotations, the generation process adopts a different approach. When a method is annotated with “--@press” the generation strategy will be the same as the one previously described. The annotation serves only to explicitly define that the method is to be parsed in the context of GUI generation. If the method is annotated with “--@check=<value>”, two labels will be generated. The first label will show the string defined by

<value>, the second will have the return value of the corresponding method.

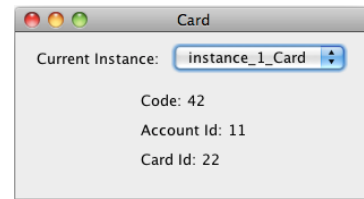


Figure 3. Example of a window generated from the class Card (with ‘check’ annotated methods) of the Dispenser system used in the case study.

All windows generated from VDM++ specification classes have a drop-down list. This list (labelled “Current Instance” in Figure 3) contains all the instances of such class.

Such list also contains a “new” option to allow the construction of new instances. This option leads to the immediate creation of a new instance of the class when it does not have a constructor, or to a new window (Figure 4) when there is a constructor with arguments.

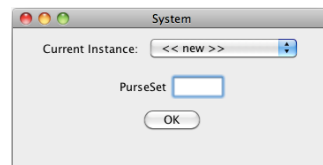


Figure 4. Example of a window generated from the class System of the “ElectronicPurse” specification found in //overture.svn.sourceforge.net .

D. Dependency Graph

There may exist GUI elements disabled at a given time. For example, when a method has a parameter of the type Class X and there is no instance of such class, this method is disabled. In order to address this issue, the approach keeps track of the dependencies of a given method and checks if they are satisfied.

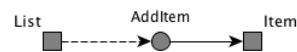


Figure 5. Graph representing the dependencies of simple list system.

The above graph (Figure 5) represents the dependencies obtained from a specification of a list. The specification has two classes, “Item” and “List”, the latter possessing one operation, “AddItem” (represented by a dashed arrow in Figure 5). This operation requires the existence of an “Item” instance to be enabled (dependency represented by a solid arrow in Figure 5).

Extending the previous example, so that a “List” requires a “Person”, would generate the dependency graph in Figure 6 which means that it will be possible to construct List instances only after creating Person instances.

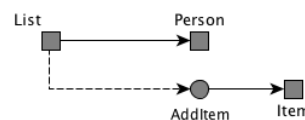


Figure 6. Graph representing the dependencies of the extended system.

V. CASE STUDY

The Overture Project provides several examples of VDM++ specifications ([//overture.svn.sourceforge.net/](http://overture.svn.sourceforge.net/)). In order to evaluate the approach, including the use of annotations, the Cash Dispenser system was selected. The specification describes a system that allows the withdrawal money from accounts using a card and a till. The system keeps record of issued cards, cardholders and current accounts, and can issue card statements to the cardholders. The VDM++ specification used in this experiment includes the class “SimpleTest” used as a test case. Since this class does not specify additional behaviour of the system being modelled, the “--@nowindow” annotation was added to it.

The following figure depicts the dependencies that the specification has, according to the previously described approach. Note that in Figure 7, only classes, operations and functions with dependencies are represented.

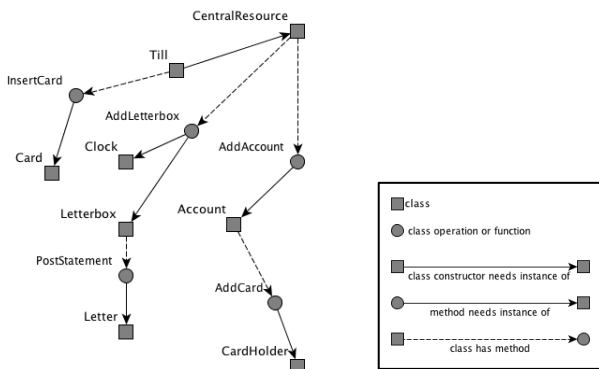


Figure 7. The Cash Dispenser system dependency graph.

The generated main window is shown in Figure 8. The Till button is initially disabled because there is a dependency between Till class and CentralResource class (represented by a solid arrow in Figure 7) which means that an instance of CentralResource is needed in order to construct a Till instance.

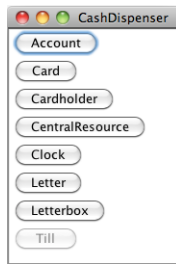


Figure 8. The main window with the Till button disabled

An instance of CentralResource class is immediately constructed when opening the corresponding window (Figure 9) because such class has no defined constructor. The window has two buttons disabled, “AddLetterbox” and “AddAccount” – their dependencies are not yet satisfied, as illustrated in Figure 7. “AddLetterBox” method is enabled after creating instances of “Clock” and “Letterbox” classes. “AddAccount” method is enabled after constructing instances of the “Account” class.

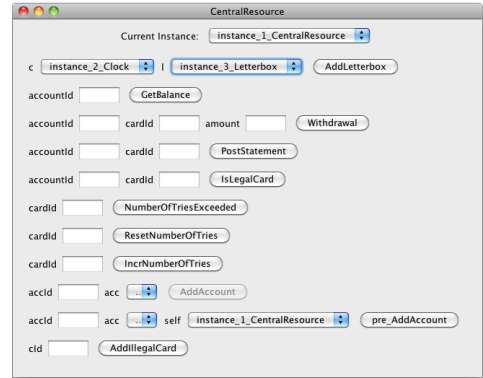


Figure 9. The “CentralResource” window with AddLetterBox and AddAccount buttons disabled

After creating the “Clock” and the “Letterbox”, the “AddLetterbox” operation becomes enabled, with the appropriate controls now populated with the constructed instances of “Clock” and “Letterbox” classes.

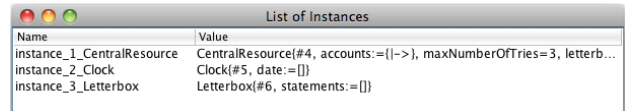


Figure 10. The list of instance window, after creating the instances.

VI. DISCUSSION

As the case study shows, the described approach is able to generate a fully functional GUI to interact with a VDM++ specification, with minimal additional effort from the part of the developer. It enables calling methods present in the specification and displaying the return value.

However, the generated GUI is unsophisticated, due to the inherent difficulty of implementing a GUI generation process based on a formal language not specific for GUI modelling (apart from the annotations introduced by the approach). More annotations could be introduced, but they would require additional modelling effort, which could put into question the goal of this research work: generate a GUI from a generic VDM++ specification with minimal additional effort.

The GUI element enabling/disabling previously described can check argument availability but does not validate it. For example, a method that takes as argument a class instance would still be accessible, even if the available instances themselves possessed undefined or invalid required values. But this is not necessarily a limitation of the approach. As it could serve to help the developer identify situations where function or operation pre-conditions are missing.

VII. CONCLUSION AND FUTURE WORK

The described approach is able to generate a fully functional GUI from a VDM++ specification. The generated GUI is also capable of enabling/disabling GUI buttons based on a dependency graph extracted from the analysis of GUI specification methods. The approach achieves this while following the grammar of the VDM++ formal language and

without requiring the user active participation in the GUI generation process.

Furthermore, by adding annotation with additional information to the VDM++ specification, it is possible a better adjustment of the GUI elements generated.

Taking into account the results and features of available VDM++ tools, the approach could be improved in the following ways:

- Adding different user interface patterns to choose from. Based on the design pattern terminology in [28], this approach uses a user interface pattern [29] that focuses on guaranteeing that the GUI will be adequate for a VDM++ specification, whichever it may be. But in terms of an evolving UI prototype, it could be useful to try different interface patterns.
- Taking advantage of the available pre-conditions in a VDM++ specification. The dependencies that check for GUI element enabling/disabling could also be extended to include the evaluation of pre-conditions.
- Implementing the connection of the generated GUI with the API code generated automatically by existing VDM tools. VDM Tools are capable of generating Java code from a VDM++ specification. The integration of the GUI with this code would lead to a standalone java GUI application created with no user intervention from a VDM++ specification. This could be achieved by making the UI Manager module aware of the proper VDM methods equivalents in the generated Java code. Thus 'redirecting' the GUI calls to such methods in Java instead of VDM++ methods like what happens now.
- Make the class reader dependent on the Overture AST when the development of this tool is completed. Currently the tool depends directly on VDMJ for extracting class information, but this is not a recommended method. Ideally the tool should use a purposely built Abstract Syntax Tree.

REFERENCES

- [1] C.S.K. Corporation, *The VDM Toolbox API 1.1*, 2008.
- [2] N. Battle, *VDMJ Tool Support: User Guide*, 2011.
- [3] B. Myers, S.E. Hudson, and R. Pausch, *Past, present, and future of user interface software tools*. ACM Trans. Comput.-Hum. Interact., 2000. 7(1): pp. 3-28.
- [4] D. Aitel, *A beginner's guide to using pyGTK and Glade*. Linux J., 2003. (113): p. 5.
- [5] Oracle. *Lesson: Using the NetBeans GUI Builder*. [2011 6/3/2011]; Available from: <http://download.oracle.com/javase/tutorial/javabeans/nb/>.
- [6] J. Bishop, *Multi-platform user interface construction: a challenge for software engineering-in-the-small*, in *Proceedings of the 28th international conference on Software engineering*, 2006, ACM: Shanghai, China. pp. 751-760.
- [7] Q. Lambourg, et al., *USIXML: A Language Supporting Multi-path Development of User Interfaces*, in *Lecture Note in Computer Science* 2005. pp. 134-135.
- [8] Microsoft Corp., *XAML Overview (WPF)*. [29/06/2011]; Available from: <http://msdn.microsoft.com/en-us/en-us/library/ms752059.aspx>.
- [9] M.D.N., *The Joy of XUL*. [28/05/2011]; Available from: https://developer.mozilla.org/en/The_Joy_of_XUL.
- [10] W. Paulus, *SwiXML* [03/06/2011]; Available from: <http://www.swixml.org/>.
- [11] J. Jarvi, et al., *Algorithms for user interfaces*, in *Proceedings of the eighth international conference on Generative programming and component engineering 2009*, ACM: Denver, Colorado, USA. pp. 147-156.
- [12] P.V. Hentenryck, and V. Saraswat, *Strategic directions in constraint programming*. ACM Comput. Surv., 1996. 28(4): pp. 701-726.
- [13] B.A. Myers, et al., *The Amulet user interface development environment*, in *CHI '97 extended abstracts on Human factors in computing systems: looking to the future 1997*, ACM: Atlanta, Georgia. pp. 214-215.
- [14] B.T.V. Zanden, et al., *Lessons learned about one-way, dataflow constraints in the Garnet and Amulet graphical toolkits*. ACM Trans. Program. Lang. Syst., 2001. 23(6): pp. 776-796.
- [15] I. Darwin, *GUI Development with Java*. Linux J., 1999. 1999(61es): pp. 4.
- [16] P. Szekely, P. Luo, and R. Neches, *Facilitating the exploration of interface design alternatives: the HUMANOID model of interface design*, in *Proceedings of the SIGCHI conference on Human factors in computing systems 1992*, ACM: Monterey, California, United States. pp. 507-515.
- [17] A. Rosado and J.P. Faria, *A metamodel-based approach for automatic user interface generation*, in *Proceedings of the 13th international conference on Model driven engineering languages and systems: Part I 2010*, Springer-Verlag: Oslo, Norway. pp. 256-270.
- [18] J. Nichols, D.H. Chau, and B.A. Myers, *Demonstrating the viability of automatically generated user interfaces*, in *Proceedings of the SIGCHI conference on Human factors in computing systems 2007*, ACM: San Jose, California, USA. pp. 1283-1292.
- [19] E.M. Clarke and J.M. Wing, *Formal Methods: State of the Art And Future Directions*. ACM Comput. Surv., 1996. 28: pp. 626-243.
- [20] D. Björner, *The Vienna development method (VDM): Software specification and program synthesis*, in *Proceedings of the International Conference on Mathematical Studies of Information Processing 1979*, Springer-Verlag. pp. 326-359.
- [21] *VDMTools: advances in support for formal modeling in VDM*. SIGPLAN Not., 2008. 43(2): pp. 3-11.
- [22] J. Woodcock, et al., *Formal methods: Practice and experience*. ACM Comput. Surv., 2009. 41(4): pp. 1-36.
- [23] P.G. Larsen and J.S. Fitzgerald. *Recent Industrial Applications of Formal Methods in Japan*. in *BCS-FACS Workshop on Formal Methods in Industry*. 2008. British Computer Society.
- [24] J.S. Pedersen and K.H. Shingler, *Software Development Using VDM*, 1989.
- [25] P.G. Larsen, et al., *VDM-10 Language Manual*, 2011.
- [26] P.G. Larsen, et al., *Tutorial for Overture/VDM++*, 2010.
- [27] A. Wolfe, *Eclipse: A Platform Becomes an Open-Source Woodstock*. Queue, 2003. 1(8): pp. 14-16.
- [28] L. Aversano, et al., *An empirical study on the evolution of design patterns*, in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering 2007*, ACM: Dubrovnik, Croatia. pp. 385-394.
- [29] A. Granlund, D. Lafrenière, and D.A. Carr. *A Pattern-Supported Approach to User Interface Design Process*. in *HCI International'2001*. 2001. New Orleans: Lawrence Erlbaum Associates.

An Approach to Model, Configure and Apply QoS Attributes to Web Services

Ahmed Al-Moayed
 Department of Computer Science
 Furtwangen University of Applied Science
 Furtwangen, Germany
 ahmed.almoayed@hs-furtwangen.de

Bernhard Hollunder
 Department of Computer Science
 Furtwangen University of Applied Science
 Furtwangen, Germany
 bernhard.hollunder@hs-furtwangen.de

Abstract—Service-oriented architectures has become a commonly accepted solution for integrating enterprise applications around the globe. As the SOA environment grows, its complexity and the complexity of modelling, configuring and applying quality of service attributes to it increases. Though there are some tools supporting these activities, they are either limited to certain quality of service domains or dependent to specific development environments. In this paper, we elaborate a concept for managing quality of service attributes for Web services. In particular, our approach covers the modelling of arbitrary quality attributes based on a meta-model for quality attributes. It also covers the generation of a graphical user interface to configure the modelled quality of service attributes and the transformation of the modelled QoS attribute into policy descriptions. Finally, our approach outlines the assignment of the policies to the target Web services. This approach offers a solution, which reduces the cost and effort by the creation of QoS-aware Web services.

Keywords-Service-oriented architecture, meta-model, model-model transformation, QoS-aware Web services

I. INTRODUCTION

Service-oriented architectures (SOA) refers to a system architecture that provides a variety of different and possibly incompatible methods and applications as reusable services. As an enterprise may have a huge variety of service providers, which offer the same functionality, the services may differ in the non-functional requirements. A well designed application should have a precise functional goal and a set of non-functional requirements such as security and performance, which must be full-filled during execution time.

Applying quality of service (QoS) to distributed Web services is an important process as the demand for high quality Web services in terms of non-functional attributes raises. One way to apply QoS to Web services is by associating it with the Web Service Definition Language (WSDL). The WS-Policy Framework [7] is an OASIS standard, which allows Web services to express their capabilities, requirements and general characteristics in an XML form. However, in order to create such policies, a certain policy grammar knowledge is needed, which is not always acquired by Web service developers. In this paper, we present an approach, which offers an easy way to model quality of services for

Web services and applies them to any Web service without being dependent on any kind of IDE (Integrated Development Environment), implementation language or previous knowledge of the implementation source code of a Web service.

We distinguish two kinds of developers: a QoS developer and Web service developer. The first one is responsible for modelling and implementing new QoS attributes; the latter one is responsible for applying the required QoS attributes to Web service once they have been developed.

There are a few tools, which allow the Web service developer to select certain QoS and apply them to a Web service. The problem with existing tools is, they are either hard to extend, mostly restricted to a certain policy domain, such as WS-SecurityPolicy [9] and WS-ReliableMessaging [8], or bundled with a specific IDE. To our best knowledge, there is no tool support, which offers both QoS developers and Web service developers the following features:

- A flexible, extended and simple meta-model for QoS modelling.
- An easy way to model and create QoS attributes for Web services and place them under a Web service developer's disposal.
- A dynamic graphical user interface, which allows the developer to easily and separately configure the modelled QoS attributes for each designated Web service.
- A QoS editor, which is not platform-, IDE- or language-specific. An editor, which supports SOA as an architecture and not Web service as a language-specific implementation.
- Automatic transformation of the configured QoS model into an adequate policy and automatically associate the created WS-Policy with the Web service.
- Association of the created policy description with the Web service.

While developing a Web service, an IDE is more than just a source code editor, which helps the Web service developer to write source code and to offer the developer code suggestions. It also automates many processes during development such as code compiling, generation of proxies and stubs and code deployment. However, there is only limited IDE

support for developing QoS-aware Web services. This paper is one step to improve this support. It offers a solution, which automate and simplify the modelling, configuring and applying of QoS attributes to Web services. The Web service developer will no longer be required to directly configure the required set of low-level policy assertions and manually configure steps in order to apply the modelled QoS to Web services. This will reduce the developing effort and the costs of creating a QoS-aware Web service.

This paper is structured as follows: Section II gives an overview on our approach. Section III describes the QoS meta-model used in this paper. The QoS model will be explained in Section IV. Section V describes the dynamic QoS graphical user interface, which is derived from the QoS model. Section VI will shed a light on QoS model to policy transformation. Related work will be discussed in Section VII. Section VIII discusses future works. In the final section, we will conclude this paper.

II. APPROACH

The first component is the meta-model. It describes exactly how the QoS model is created or defined. There are many QoS meta-model proposals, which can be used to define and apply QoS models for Web services. Malfatti [6] introduced a suitable meta-model for our approach. It is simple, extensible, easy to understand and expressive enough to model arbitrary QoS attributes. The meta-model was created in Eclipse Modelling Framework (EMF) (Core) meta-model, a powerful tool for designing models and their runtime support.

The QoS model offers the QoS developer a way to model QoS attributes within certain QoS categories. These categories are already defined in the QoS meta-model. As we will see in Section IV, with this meta-model, we will be able to model different QoS attributes including some standardized QoS attributes such as reliable messaging and security. The QoS model is expressed in XML format. EMF, however, provides Java interfaces, Java implementation classes and a factory and package (meta data) implementation class, which provides support for building and modifying EMF models. The QoS developer will be able to use an EMF editor to manually add or remove QoS attributes.

Once a QoS model has been created, a graphical user interface will be automatically generated. The QoS model includes essential information on how the graphical user interface (GUI) should look like. Depending on certain elements in the QoS model, the GUI will be able to adapt to new changes. For example, depending on how many QoS categories are modelled, the GUI will generate a tab for each category. Within the same category, the QoS attribute will be presented. The GUI will also collect additional data from the Web service, such as the service endpoint interface methods, which will be needed for later steps in order to generate QoS policies. The main purpose of the GUI is to enable

the Web service developer to easily configure the modelled QoS attributes and apply them to a Web service. Once this step has been done, the GUI will write the configured QoS attributes back to the QoS model in order to be transformed into a policy representation. For the moment, the GUI is implemented in Java as an Eclipse plug-in. It is, however, our intention to support other GUI frameworks in the future.

Once all data for the generation of the QoS policies has been configured in the GUI, component four in Figure 1 will automatically transform the QoS model into a user defined QoS policy. This component will be able to transfer the QoS model to different QoS policies. In this approach, we have used the WS-Policy to demonstrate this work. Figure 1 summarizes our approach.

III. QoS META-MODEL

The QoS meta-model in Figure 1 has two purposes; saving monitoring data of existing SOA environments with existing QoS attributes and the modelling of QoS. QoS monitoring is not the focus of this work. However, it has a great importance for future works. The meta-model for QoS modelling was slightly modified for better flexibility. The following changes were made in the meta-model:

- The `CATEGORY` attribute in the `QOSPARAMETER` was modified to include only predefined values specified in the enumeration class `QOSCATAGORY`.

The following elements were added:

- The `QOSVALUE` element was expanded to include a new attribute `DATATYPE`. The new attribute is necessary for the creating of the QoS graphical user interface described in Section V.
- A new enumeration class `QOSDATATYPE` was added. A predefined values, which are needed for defining the `DATATYPE` attribute.
- A new enumeration class `QOSCATAGORY` was added. A list of pre-defined categories the QoS model supports.
- A new `QOSPROPERTIES` element was added. A list of properties, which could be used to add more information to either the `QOSPARAMETER` or `QOSVALUE`.

The following element was not considered in the modified meta-model:

- The `QOSLEVEL` was not considered in this work since the modelled QoS is always fulfilled.

The meta-model enables the QoS developer to model QoS attributes for Web services for different business domains. A main characteristic of the QoS model is its simplicity and the ability to model QoS attributes. The QoS model has the following relationships:

- Every Web service or Web service method has 0..* QoS parameters.
- Every `QOSPARAMETER` has exactly one QoS metric. As described in [6], a metric specifies a measurement unit used for describing the `QOSVALUE`.

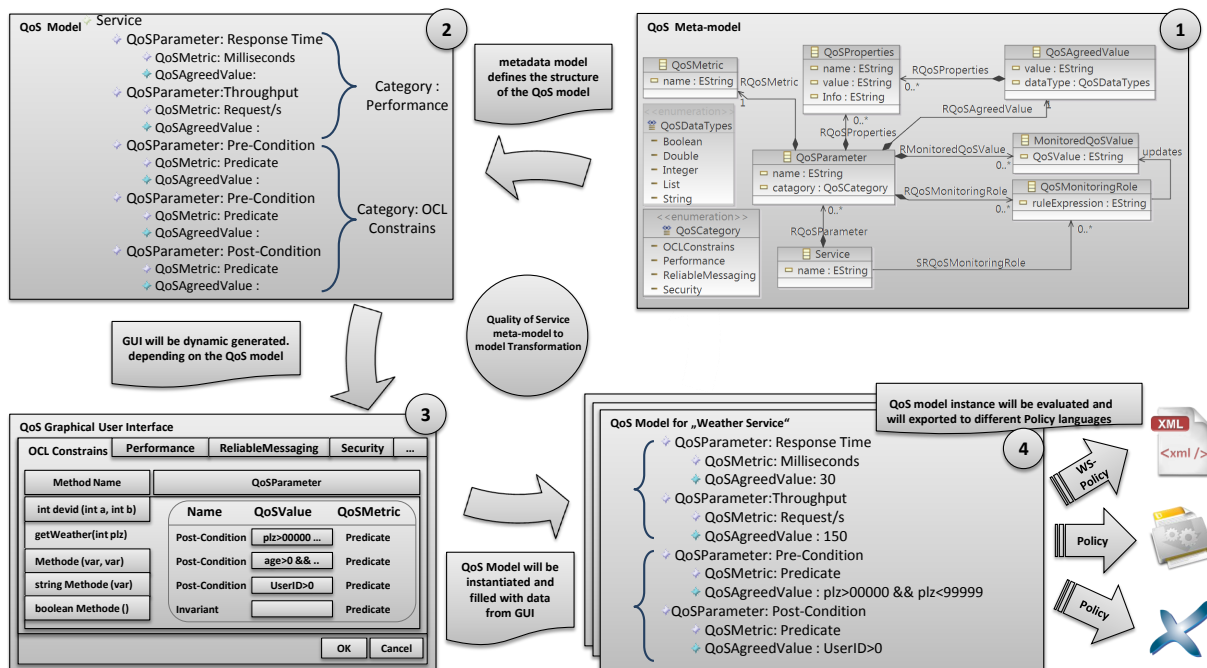


Figure 1. An approach to model, configure and apply QoS attributes to Web services

- Every QOSPARAMETER has exactly one QoS value.
- Every QOSPARAMETER and QOSVALUE have 0..* QoS properties. It is used to add extra information about the QOSPARAMETER or QOSVALUE.
- Every QOSPARAMETER has 0..* monitored QoS values. The monitored QoS values are series of values taken by the monitoring system in order to be either compared to the QOSAGREEDVALUE or to be used to compute instant or average values.
- Every QOSPARAMETER has 0..* QoS monitoring rules, a rule defines how the QoS attributes in SOA should be monitored.

This meta-model is based on EMF (Core), a modelling framework and code generation facility, which is used to building tools and applications based on a structured data model.

IV. QoS MODEL

In this section, we will model three QoS attributes to demonstrate the flexibility of the model. We will present a standardized QoS attribute from the WS-* family and introduce two non-standardized QoS attributes. The first QoS attribute is from WS-ReliableMessaging. Listing 1 models QoS attributes described as a RM policy assertion example in [8], Section 2.4. In the following example, lines (2) - (6) indicate that if the idle time exceeds ten minutes, the sequence will be considered as terminated by the Service Endpoint. lines (7) - (11) express that an unacknowledged message will be transmitted after three seconds. Lines (12) - (16) express that the exponential backoff algorithm will be

```

<qossoa:Service xmi:version="2.0"
2 <RQoSParameter name="InactivityTimeout"
3   category="ReliableMessaging">
4   <RQoSMetric name="Millisecond"/>
5   <RQoSAgreedValue value="600000" dataType="Integer"/>
6 </RQoSParameter>
7 <RQoSParameter name="BaseRetransmissionInterval"
8   category="ReliableMessaging">
9   <RQoSMetric name="Millisecond"/>
10  <RQoSAgreedValue value="3000" dataType="Integer"/>
11 </RQoSParameter>
12 <RQoSParameter name="ExponentialBackoff"
13   category="ReliableMessaging">
14   <RQoSMetric />
15   <RQoSAgreedValue />
16 </RQoSParameter>
17 <RQoSParameter name="AcknowledgementInterval"
18   category="ReliableMessaging">
19   <RQoSMetric name="Milliseconds"/>
20   <RQoSAgreedValue value="200" dataType="Integer"/>
21 </RQoSParameter>
22 ...
23 </qossoa:Service>
    
```

Listing 1. QoS model for reliable messaging

used to retransmitted the message if the message was not acknowledged. Lines (17) - (21) indicate that an acknowledgement could be buffered up to two-tenths of a second by the RM destination.

The following example models a QoS attribute, which is not standardized. Listing 2 describes OCL constraints, a well-known formalism for expressing constraints on classes variables, methods parameters or methods return values. The OCL constraints set preconditions, postconditions and invariants for the Web service class or Web service methods. The pre-condition in line (4), for example, indicate that the given age must be within a range, a minimal age of 18

```

1 <qosoa:Service xmi:version="2.0"
2   xmlns:xmi="http://www.omg.org/XMI"
3   xmlns:qosoa="http://qosoa/1.0">
4   <RQoSParameter name="preConditions"
5     category="OCLConstrains">
6     <RQoSMetric name="predicate" />
7     <RQoSAgreedValue value="age >=18 && age <=120"
8       dataType="String" />
9   </RQoSParameter>
10  <RQoSParameter name="preConditions"
11    category="OCLConstrains">
12    <RQoSMetric name="predicate" />
13    <RQoSAgreedValue value="zipCode >01000 &&
14      zipCode <99999 && zipCode.size==5"
15      dataType="String" />
16  </RQoSParameter>
17  ...
18  <RQoSParameter name="postCondition"
19    category="OCLConstrains">
20    <RQoSMetric name="predicate" />
21    <RQoSAgreedValue value="userID >0"
22      dataType="String" />
23  </RQoSParameter>
24  <RQoSParameter name="Invariant"
25    category="OCLConstrains">
26    <RQoSMetric />
27    <RQoSAgreedValue />
28  </RQoSParameter>
29 </qosoa:Service>

```

Listing 2. QoS model for OCL constraints

years and maximum of 120 years. The pre-condition in line (10) indicates that the given zip code must be within rang between 01000 and 99999. It also implies that the zip code must be of five digits since these pre-conditions must comply with the zip code rules in Germany. The post-condition in line (18) defines that the return value USERID shall not be a negative number. The model offers also the possibility to specify an OCL invariant condition as shown in line (24).

Another example of modeling QoS attributes is performance. Response time and throughput are QoS attributes, which are two of the most common used attributes in order to measure performance. As response time refers to the duration, which starts from the moment a request is sent to the time a response is received, throughput refers to the maximum amount of requests that the service provider can process in a given period of time without having effect on the performance of the Web service endpoint [10]. Listing 3 shows an example of how performance can be modelled. Line (4) defines the QoS attribute "ResponseTime", which indicated that the Web service shall guarantee a response time within 10 milliseconds. Line (9) indicated that the Web service will be able to handle up to 120 request/second without having any change on the Web service performance.

V. GRAPHICAL USER INTERFACE

The graphical user interface is a component, which uses the QoS model to create its representation. Its main purpose is to offer the Web service developer a graphical tool to configure the QoS values of the modelled QoS attributes and associate them with the Web service.

There are two factors, which decide how the GUI should look like; the first factor is the QoS model. The QoS model

specifies how many categories shall be represented. Every QoS category is represented by a GUI tab, where all QoS attributes under the represented category will be represented. For example, if the QoS model includes four QoS categories; performance, OCL constraints, reliable messaging and security. The QoS model will be transformed into a GUI, which has four tabs. Each tab will represent a category. If the QoS constraints, for example, has four QoS attributes, the QoS constraints tab on the GUI will represent these four QoS attributes as shown in Figure 1.

The element QOSMETRIC helps the GUI engine to determin, how the QOSAGREEDVALUE shall be presented. For example, if the QOSMETRIC indicates that the QoS attribute is a string, the GUI engine will use a text field. If the QOSMETRIC is a predicate, then the GUI engine will use a check box for the presentation of this attribute.

The second factor is the Web service endpoint. A list of the Web service methods will be extracted either directly from the Web service endpoint interface (SEI) or from the WSDL. Each extracted method has its own list of QoS attributes. If, for example, two Web service methods have two different "ResponseTime" values, a policy for each method will be created. This will result in creating a separate policy for each selected method. The created policy could be also applied Web service wide. These possibilities give the Web service more flexibility and dynamic.

VI. QOS POLICY

As a Web service developer configured the QoS attributes on the graphical user interface, all QoS values will be assigned to the QOSAGREEDVALUE element in the QoS model. Once the QoS values has been assigned, a QoS policy will be generated. If, for example, every Web service method has different QoS attributes, a separate policy will be created for every Web service method. A WS-Policy may include the description of more than one QoS attribute depending on the user input in the graphical user interface.

Listing 4 shows the modelled QoS in Listing 1 after being transformed into reliable messaging policy assertion (also described in [8]). The following transformation rules are applied:

```

<qosoa:Service xmi:version="2.0"
2   xmlns:xmi="http://www.omg.org/XMI"
3   xmlns:qosoa="http://qosoa/1.0">
4   <RQoSParameter name="ResponseTime"
5     category="Performance">
6     <RQoSMetric name="Millisecond" />
7     <RQoSAgreedValue value="10" dataType="Double" />
8   </RQoSParameter>
9   <RQoSParameter name="Troughput" category="Performance">
10    <RQoSMetric name="Requests/s" />
11    <RQoSAgreedValue value="120" dataType="Double" />
12  </RQoSParameter>
13 </qosoa:Service>

```

Listing 3. QoS model for performance


```

1 <wsp:Policy wsu:Id="MyPolicy">
2 <wsm:RMAssertion>
3 <wsm:InactivityTimeout Milliseconds="600000" />
4 <wsm:BaseRetransmissionInterval
5 Milliseconds="3000" />
6 <wsm:ExponentialBackoff />
7 <wsm:AcknowledgementInterval
8 Milliseconds="200" />
9 </wsm:RMAssertion>
10 </wsp:Policy>

```

Listing 4. Reliable messaging policy assertion

- The CATEGORY attribute in the QoS model declares the name of the policy. For example, the category “ReliableMessaging” is transformed into a *wsm:RMAssertion* element declaring a Reliable Messaging policy.
- The element RQOSPARAMETER in the QoS model declares a QoS attribute. The RQOSPARAMETER element indicates the reliable messaging quality attribute “InactivityTimeout” and is therefore transformed into “wsm:InactivityTimeout” element.
- The element RQOSMETRIC in the QoS model declares a property or how the QoS should be measured. The “Milliseconds” is transformed into “Milliseconds” attribute within the “wsm:InactivityTimeout” element.
- The element RQOSAGREEDVALUE in the QoS model declares a QoS value. The value “600000” will be mapped as a value for the QoS attribute.

Listing 5 shows the modelled QoS in Listing 2 after the transformation into an OCL policy assertion. The same transformation rules apply as already described above.

Once the policies have been created, component four in Figure 1 will assign the created policies to the Web service endpoint interface. In our proof of concept, we use the CXF policy engine to attach the corresponding policy to either the selected Web service methods or the Web service endpoint interface. CXF uses the @POLICY annotation to signal the compiler that there are policies, which should be considered and assigned to the correspond at Web service while creating the Web service WSDL.

VII. RELATED WORK

In our research for related work, a recent approach, which nearly investigates our approach or even a part of it was not found. Most of the recent works on QoS-aware Web services focus on QoS-aware Web services compositions. They investigate methods, algorithm or frameworks in order to better compose Web services according to their QoS

```

1 <OCLConstrains context="RegistrationService">
2 <Precondition predicate="zipCode >01000 &&
3 zipCode <99999 && zipCode.size==5"/>
4 <Precondition predicate="age >=18 && age <=120"/>
5 ...
6 <Postcondition predicate="UserID >0">
7 <Invariant/>
8 </OCLConstrains>

```

Listing 5. OCL policy assertion

attribute. Such works could be found in [1] [2] [5]. In this section, we will describe papers, which propose either QoS meta-models or policy editors.

Tondello et al. [12] proposes a QoS-Modelling Ontology, which allows QoS requirements to be specified in order to fully describe a Web service in terms of quality. However, this proposal focuses on using QoS specification for semantic Web services description and Web service search. This approach, however, contains many variables and many characteristics in ontology for semantic Web services, which does not flow in the same direction as this work intends to.

Suleiman1 et al. [11] addresses the problem with Web service management policies during design. The authors presented a solution, which uses a novel mechanism. It generates W-Policy4MASC policies from corresponding UML profiles semi-automatically and feedback information monitored by the MASC middleware into a set of UML diagram annotations.

D’Ambrogio [3] introduced a WSDL extension for describing the QoS of a Web service. It uses a meta-model transformation according to MDA standards. The WSDL meta-model is extended and transformed into a new WSDL model called Q-WSDL, which supports QoS description. As D’Ambrogio favour an approach, which does not support introducing a new additional language on top of WSDL, our approach uses standards for the description of QoS attribute in Web services.

WSO2 WS-Policy editor [14] offers an integrated WS-Policy editor with the WSO2 application server. The editor offer two policy views; a source view and a design view. The source view shows the policy in its XML format and the design view shows the policy as a tree view. The user will be able to add and remove element to and from the policy. However, this policy editor only offers support for WS-Security and WS-ReliableMessaging. A support for new QoS attributes is not mentioned.

NetBeans offers a graphical tool, which allows users to graphically configure security and reliable messaging to a Web service. Extending this tool, however, is complex due to the lack of documentation and its dependability to NetBeans API and Glassfish.

All the these works discuss QoS attributes after the Web services is developed. Our approach offers a solution to develop a QoS-aware Web service.

VIII. FUTURE WORK

In [4], we presented the design of a comprehensive tool chain that facilitates development, deployment and testing of QoS-aware Web services. This paper is a part of the work presented in the tool chain, which elaborates a concept for managing quality of service attributes for Web services. Future works will include different tasks, which will be individually explained in this section.

In Section V, we introduced a GUI, which is dynamically generated depending on the QoS attributes modelled in the QoS model described in Section IV. However, the generation of the GUI is platform-specific. This GUI is only a proof of concept in order to demonstrate the feasibility of this approach. Our goal is to create a GUI using MDA as a base for our approach, which will allow the dynamic GUI generation to different platform.

Section IV indicates that the QoS model will be transformed to a QoS policy. In this paper, we have only considered WS-Policy as a policy language in order to prove that the concept really works. It is the intention of this approach to offer QoS model transformation support to more than one policy language. This will increase the flexibility of our approach.

In [13], we offered a solution architecture, which collects real time data about applied QoS attributes from the SOA environment: The purpose of this architecture is to evaluate the compliance of the entire SOA with the QoS attributes described in the SOA QoS policy. It is our intention to use the meta-model mentioned in Section IV for the evaluation and monitoring of the SOA environment.

This paper presents an approach of how QoS attributes could be easily modelled and transformed into an adequate policy language. However, a policy without a handler, which enforces the policy on the Web service is only half the solution. Future works include a repository component, which is designed to store QoS handlers. This repository will include everything a Web service developer needs to implement a Web service Handler. This includes handler implementation, handler configurations and test cases.

IX. CONCLUSION

There are tools and IDEs, which help developers to ease the process of creating programs and minimizes their error rates. Nowadays, it is hardly imaginable to start designing and implementing complex systems without them. To create a QoS policy and conjugate it with a Web service requires a good knowledge of its grammar and its mechanism. Tools, which help developers to model QoS attributes, simplify the configuration and automate applying QoS attributes to Web services still has a long way to completion. In this paper, an approach, which relieve a Web service developer with this burden, was presented. It offers an easy way to model QoS attributes. It also supports the modelling of new QoS attributes, simplifies the configuration and automatize applying QoS attributes to Web services. It is a step forward to completing a tool chain for constructing QoS-aware Web services and reducing a lot of development effort and cost.

X. ACKNOWLEDGMENT

This work has been supported by the German Ministry of Education and Research (BMBF) under research contract 017N0709.

REFERENCES

- [1] M. H. Agdam and S. Yousefi. A Flexible and Scalable Framework For QoS-aware Web Services Composition. In *Proc. 5th Int Telecommunications (IST) Symp*, pages 521–526, 2010.
- [2] P. Bartalos and M. Bielikova. QoS Aware Semantic Web Service Composition Approach Considering Pre/Postconditions. In *Proc. IEEE Int Web Services (ICWS) Conf*, pages 345–352, 2010.
- [3] A. D’Ambrogio. A Model-driven WSDL Extension for Describing the QoS of Web Services. In *Web Services, 2006. ICWS ’06. International Conference on*, pages 789–796, sept. 2006.
- [4] B. Hollunder, A. Al-Moayed, and A. Wahl. *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, chapter A Tool Chain for Constructing QoS-aware Web Services, pages 172–188. IGI Global, 2011.
- [5] H. Kil and W. Nam. Anytime Algorithm for QoS Web Service Composition. In *Proceedings of the 20th international conference companion on World wide web, WWW ’11*, pages 71–72, New York, NY, USA, 2011. ACM.
- [6] D. Malfatti. A Meta-Model for QoS-Aware Service Compositions. Master’s thesis, University of Trento, Italy, 2007.
- [7] OASIS. Web Services Policy Framework - Version 1.5, September 2007. <http://www.w3.org/TR/ws-policy/>. Last Access: 17.04.2011.
- [8] OASIS. Web Services Reliable Messaging Policy - Version 1.2, February 2009. <http://docs.oasis-open.org/ws-rx/wsrml/v1.2/wsrml.pdf>. Last Access: 23.04.2011.
- [9] OASIS. Web Services Security Policy - Version 1.3, April 2009. <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/>. Last Access: 17.05.2011.
- [10] OASIS. Web Services Quality Factors Version 1.0, July 2010. <http://docs.oasis-open.org/wsqm/wsqf/v1.0/WS-Quality-Factors.html>. Last Access: 17.05.2011.
- [11] B. Suleiman and V. Tosic. Integration of UML Modeling and Policy-Driven Management of Web Service Systems. In *Proc. ICSE Workshop Principles of Engineering Service Oriented Systems PESOS 2009*, pages 75–82, 2009.
- [12] G Tondello and F. Siqueira. The QoS-MO Ontology For Semantic QoS Modeling. In *Proceedings of the 2008 ACM symposium on Applied computing, SAC ’08*, pages 2336–2340, New York, NY, USA, 2008. ACM.
- [13] A. Wahl, A. Al-Moayed, and B. Hollunder. An Architecture to Measure QoS Compliance in SOA Infrastructures. In *Proceedings of the Second International Conferences on Advanced Service*, pages 27–33, Los Alamitos, CA, USA, 2010. IEEE Computer Society.
- [14] WSO2. WSO2 WSAS: The WS-Policy Editor 3.2.0 - User Guide, April 2010. <http://www.wso2.org/project/wsas/java/3.2.0/docs/policyeditor/docs/userguide.html>. Last Access: 17.05.2011.

Transformation of Composite Web Service for QoS Extension into ACME\Armani

Amel Mhamdi

MIRACL, ISIMS, TUNISIA
amel.mhamdi1@yahoo.fr

Raoudha maraoui

MIRACL, ISIMS, TUNISIA
maraoui.raoudha@gmail.com

Mohamed Graiet

MIRACL, ISIMS, TUNISIA
mohamed.graiet@imag.fr

Mouard Kmimech

MIRACL, ISIMS, TUNISIA
mkmimech@gmail.com

Mohamed Tahar Bhiri

MIRACL, ISIMS, TUNISIA
tahar_bhiri@yahoo.fr

Eric Cariou

Université de Pau et des pays de l'Adour
Eric.Cariou@univ-pau.fr

Abstract— After the great proliferation of Web services, we can find many services that have the same answer. The Quality of the Service QoS of Web services has become the famous criterion to choose one of many responses. The effective instantiation of solution is provided by the ADL (Architectural Description Language) with an architectural style. The Acme with the ARMANI design language provides software architects with a rich language for describing software architecture designs. Recently, the application of Model Driven Architecture (MDA) to Web services has received a considerable attention. This paper focuses on the extension of the meta-model of the transactional composite Web service TCWS to the QoS of Web service. This paper presents a transformation of the meta-model of TCWS with QoS to the meta-model of acme, in order to facilitate the development of an architectural style with the Acme ADL.

Keywords- Web Services Composition, QoS, MDE, Transformation, ACME/ARMANI ADL.

I. INTRODUCTION

More businesses are planning to build their future solutions on Web service technology. Currently, SOAP, WSDL and UDDI have become standards in the field of a reliable execution of Web service. Like most Web services will need to establish and maintain the standards, the quality of service will become a point of differentiation of these services. Recently, there have been attempts to find a standardized participation form to describe the QoS with which the services are performed. At any time, it is necessary to combine a set of Web services into a more complex Web service to respond to more complex requirements. To ensure a reliable Web service composition and resolve the problem of heterogeneities, the work in [1] browses to describe a protocol for mediation using the concept of architectural styles of ACME and refers to ARMANI to detect incompatibilities of the software architectures. In this paper, we focused, on the one hand, on formalizing a reliable composition of a Web service based on non-functional properties of Web services; that is the

quality of the service. To achieve this, we describe a Web service composition using the ACME concept of the architectural style and ARMANI, to detect architecture's software disparities. Then, we automate partially our proposed formalization methodology using an MDE (Model-Driven Engineering) approach. In this context, we recover the meta-model of the proposed composite Web services and we elaborate the ACME meta-model. These meta-models respectively play the role of source and target meta-models for the exogenous transformation of composite Web services to ACME. In addition, we implemented SWC2ACME, a tool for transforming a composite Web service software architecture into on ACME using the MDE language ATL (ATLAS Transformation Language). We are then able to check the composition of the Web services through the ACME verification tools.

The paper is organized as follows. We shall start by the related works. Next, we describe in Section 3 our automatic MDE approach for an exogenous transformation from Web services to an ACME. Section 4 describes the specification of the QoS of the Web service and we sketch a meta-model for the composite Web service with the QoS. Then, we formalize a reliable composition of Web services in ACME/ARMANI. After that and to translate the source meta-model to the target, a set of transformations is introduced. The final part of Section 5 applies the transformation to the running example. Finally, Section 6 represents a conclusion.

II. RELATED WORKS

Although, there are many researches which tried to identify and classify the QoS parameters; there is no specific consensus on all the important QoS for Web services. Most of the work [2][3] took into consideration these parameters to which other parameters are associated. There are several proposals of the QoS model for Web services. We can classify the models into three classes. That which suggests a classification based on attributes that are independent of the

service environment (functional part) and attributes depending on the service environment (non-functional part) [2]. This model provides a general approach that some attributes of the QoS must be measured by examining the service implementation. Another modelling identified and organized by the QoS attributes of the Web services into categories (attributes related to the execution, to the transaction support, to security and the price and configuration management) [3]. It is likely that the consumer of a service does not require all the categories of the service quality. Other works [4] have classified the QoS attributes into two parts: the specific services and the generic QoS. These are divided into measurable parameters and immeasurable ones. This classification takes into consideration the specific qualities of services that are related to the business logic of applications. In our work, we try to model the QoS of the non-functional parameters; these parameters are divided into measurable and immeasurable parameters. We formalize the quality of service for Web services with an architecture description language. Yet, most approaches that formalize the Web services, with an ADL, ignore the specification of the non-functional properties such as integrity and performance. We must be able to define the QoS of the Web services through the ADL specifications since the ADL techniques are a way to check the properties of the Web services. We can, then, check the properties of the composition and the QoS of the Web services through the ADL ACME.

To achieve this goal, we rely on the MDE approach defined in the following section.

III. PROPOSED APPROACH

Transformations are the heart of the MDA approach. They can get different views of a model, refine or abstract. In addition they can move from one language to another. In MDA, each model is based on a specific meta-model, which defines the language that the model is created in. The Meta Object Facility (MOF) represents the only basis of the meta-model for which any new meta-model. Therefore, the transformation rules between two MOF compliant meta-models; the source and the target define the transformation model to model. In this paper, the source meta-model is the composite Web service for QoS extension, this composition reifies all non-functional properties: the transactional properties and quality of the service, and the destination meta-model is the Acme (Fig.1).

Transformation rules define a mapping between a source and destination meta-model that preserves an equivalent or similar semantic. A transformation engine executes the rules of transformation on the source model (input) to generate the equivalent model of destination (output).

Figure 1 illustrates the principle of an automatic translation of the Web services composition for the QoS extension in the ACME\ARMANI. We distinguish two levels of specification: M2 (a meta-model level) and M1 (a model level), as defined by the MDA approach. An M1 level model is said to be conform to an M2 meta-model if it

satisfies the consistency rules described in the meta-model in addition to the specific rules outlined at the M1 model level. In our approach, the M2 level contains the Web services composition for the QoS extension meta-model on one hand and the ACME/ARMANI one on the other hand. The M1 level allows the definition of Web services models conform to the Web services composition meta-model [5].

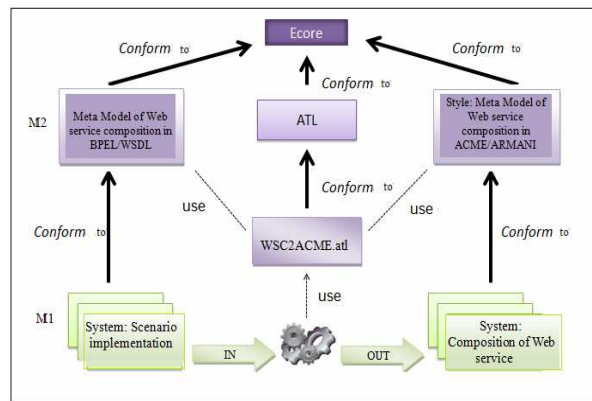


Figure 1. The proposed approach for an automatic transformation of a composite Web service for QoS extension.

These models will be automatically transformed into the ACME models (conform to the ACME\ ARMANI meta-model). We aim at checking the conformity of these transformed models to specific constraints. These constraints are defined at the model level (M1) and are checked thanks to the ACME Studio environment, which enables the evaluation of the ARMANI constraints [6]. To achieve the formalization of the Web service composition for the QoS extension in the ACME and check the consistency of this composition; we proceed to the automatic translation of this composition onto the ACME. This approach of translation covers all ACME constructs including the notion of style. These constructs are: a system, a component, a connector, a port, etc. The source and target models (Web service composition for QoS and architectural style of Web services described in ACME) and the tool WSC2ACME are consistent with their meta-models for the Web services, ACME and ATL. These meta-models are consistent with the MOF meta-model.

IV. METAMODELING OF QUALITE OF SERVICE OF WEB SERVICE

In this section, we provide an overview of the meta-model of quality of service we have defined. This meta-model reifies all the characteristics of a reliable composition of the Web services. It provides the description of the QoS and this by integrating a set of specifications as a slight extension of the WSDL. We modeled in an earlier work [7] the manager of mediation for a Web service composition. The manager is seen as a set of service integration of Web services which aims at resolving heterogeneities between

Web services and explicitly contains the non functional service manager, a set of adaptive interface service for all functional properties and a set of data mediation service on the heterogeneity of data exchanged between Web services compounds. In our work, we modeled the non-functional QoS parameters. This is because they should also think about non-functional requirements and their integration with functional requirements to provide better quality Web services. The non-functional QoS parameters are divided into specific parameters (SQoS) and generic parameters (GQoS). The generic parameters are also divided into measurable parameters (SMP) and immeasurable parameters (SIP) (Fig. 2).

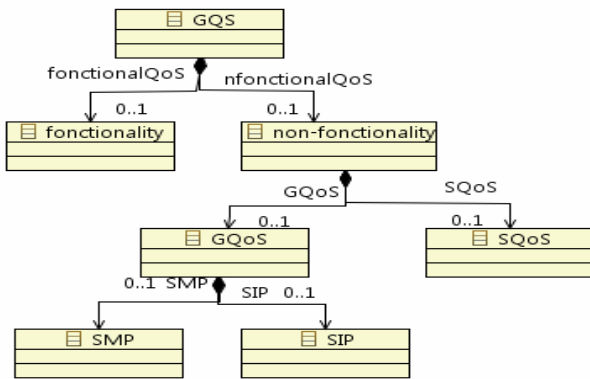


Figure 2. The meta model of the manager of QoS.

Then, we focus on a measurable service manager. These specifications are the most used. They define the quantitative attributes that could be measured. The QoS meta-model will give benefits to both service providers and requesters. The new QoS meta-model is a lightweight extension to the WSDL.

The details of these factors are:

- **Integrity:** is the quality feature that refers to the maintaining of correct and consistent interaction to the source and for transaction completeness [8] (Fig. 3).

$$\text{Integrity} = \text{ExpectedResult} - \text{ProvidedResult}$$

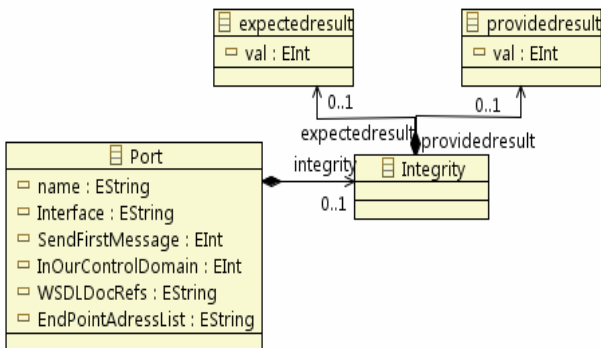


Figure 3. The meta model of Integrity property.

- **Availability:** ensures the Web service is that present or ready for instantaneous use. TimeToRepair (TTR) and TimeBetweenFailure (TBF) can be applied to measure it. Besides, we would like to add two dimensions StartTime (the start time of a service when it is available to end users) and EndTime (the last time when of a service is available to end users) [8]. It can be measured and specified as shown in Figure 4:

$$\text{Availability} = \text{TBF} / (\text{TBF} + \text{TTR})$$

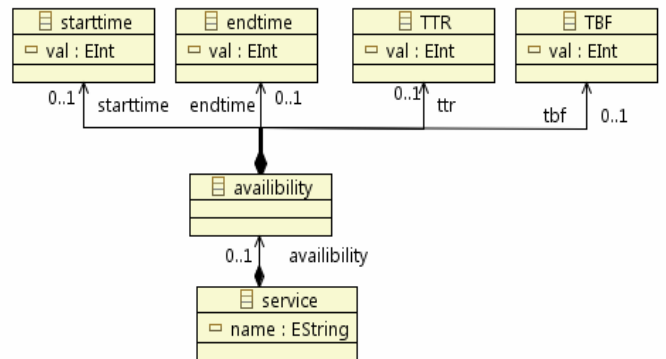


Figure 4. The meta model of Availability property.

- **Accessibility:** is quantified by MaxNumberOfResponse (is the maximum number of responses that can be processed) and NumberOfCorrectResponse (the number of response that fulfil user's requirements) [8] (Fig. 5).

$$\text{Accessibility} = (\text{NumberOfCorrectResponse} / \text{MaxNumberOfResponse}) * 100\%$$

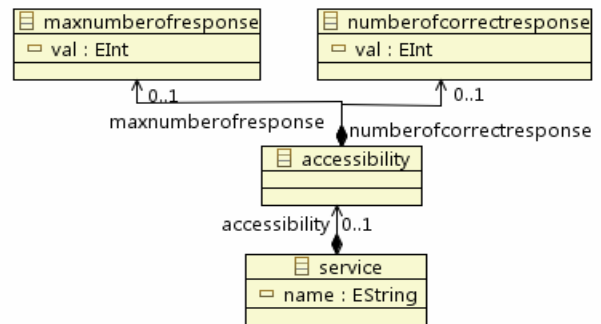


Figure 5. The meta model of Accesibility property.

- **Reliability:** represents the ability of a Web service to perform its required functions under stated conditions for a specified time interval. FlowControl and InactivityTimeOut [8] can be applied to measure it (Fig. 6).

$$\text{Reliability} = \text{FlowControl} + \text{InactivityTimeOut}$$

- **Service Time:** It is the sum of the time when the service provider receives a request for a Web service (ReceiveRequest) and the time when the service provider sends the response to requester (SendResponse). It can be specified as shown in Figure 7.

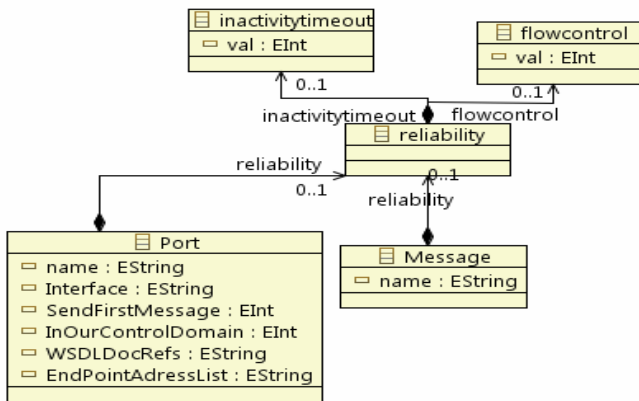


Figure 6. The meta model of the Reliability property.

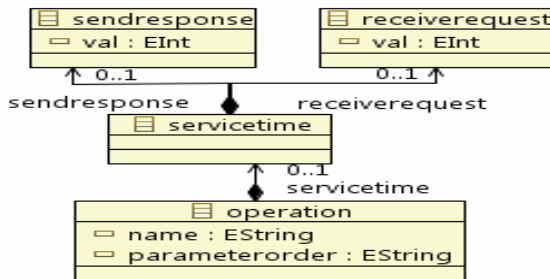


Figure 7. The meta model of Service Time property.

V. FORMALIZATION OF QOS FOR WEB SERVICES

A. The ADL ACME\Armani

The ADL ACME [9] [10], developed at Carnegie Mellon, is a common foundation for architecture description languages. The ARMANI language allows describing architectural properties in the invariant or heuristics forms attached to any architectural element (component, family, system, connector, etc.). Such properties are achievable within the ACME Studio environment [11]. In the same way, the ADL ACME supports the type concept. One can define the types of architectural elements (component type, connector type, role type, port type and style type). The concept property of ACME used in the type and instance levels allows attaching non-functional properties to the

architectural elements. Lastly, the ACME provides basic types (int, float, Boolean and string) and type builders (enum, record, set and sequence).

B. Formalization with ACME/ARMANI

Our work began with the improvement of an existing style. We have studied the work of [12] dealing with the composition of Web services without mediation approach, or control over the execution flow of services. We have formalized this protocol mediation to ensure reliable composition of Web services [1]. Figure 8 shows an ACME description of style implementing the transactional aspect of the composition of Web services.

```

Family WSM = {
Property Type Interfaces = Enum {Client,Service};
Property Type legalSoapVersions = Enum {SOAP1_1, SOAP1_2};
Property Type EndPoint = Record [Transport: legalTransportProtocols; Encoding: legalSoapVersions; ];
Property Type EndPoints = Set {EndPoint};
Component Type CompTWSCommon = {
Rule NameUnique = invariant forall p1:
PortTWSCommon in self.PORTS | forall p2:
PortTWSCommon in self.PORTS | (p1 != p2) -> p1.name
!= p2.name; }
Component Type CompTWSClient extends
CompTWSCommon with {
rule rule25 = invariant forall p : Port in self.PORTS |
satisfiesType(p, PortTWSClient) ;
rule rule26 = invariant size(self.PORTS) > 0; }
Connector Type ConnTWSAct extends ConnTWS with {
rule CondActivation = invariant forall r1 : Role in
self.ROLES | forall r2 : Role in self.ROLES | forall p1 :
PortTWSClient in r1.ATTACHEDPORTS |
forall p2 : PortTWSService in r2.ATTACHEDPORTS |
(r1 != r2 AND attached(r1, p1) AND attached(r2, p2)) ->
(p1.Prec == terminate AND p2.Prec == activate) OR
(p2.Prec == terminate AND p1.Prec == activate); ..... }

```

Figure 8. The ACME description of the style with non functional properties

The contributions of different added properties of quality of the services were formalized as constraints and properties with ACME. Indeed, ACME disposes of concepts for expressing properties and constraints with verifiable tools. These properties and constraints can be expressed on each entity or on the overall behavior of the architecture. For example, ResponseTime was formalized with a property of type int in the component service. This property takes into consideration two other parameters of the type int ResponseCompletionTime and ConsumeRequestTime. The calculation of the response time is formalized using the Design invariant concept of ACME.

However, to calculate the factor successability, it took two rules to define it as a form of invariant to calculate respectively the number of messages sent and the number of that received successfully. Note that earlier, we defined a property SendsFirstMessage of a type boolean in a port in order to accumulate the number of messages sent in case that the property SendsFisrtMessage was evaluated to be true. Similarly, we define a property of a type boolean InOurControlDomain in a port in order to accumulate the number of messages received in case that InOurControlDomain property was evaluated to be true. Figure 9 shows an excerpt of the formalization of these properties at a client component.

```

Component Type CompTWSCClient extends
CompTWSCCommon with {
  Property ResponseComopletionTime : int;
  Property UserRequestTime : int;
  Property ResponseTime : int ;
  Property succ : int ;
  rule rule25 = invariant forall p : Port in self.PORTS |
  satisfiesType(p, PortTWSCClient) << label : string =
  "External ports are all Client type"; errMsg : string = "Only
  client type ports are allowed"; >>;
  rule rule26 = invariant size(self.PORTS) > 0 << label :
  string = "Component has at least one port"; errMsg : string
  = "Component should have at least one port"; >>;
  //
  Design Invariant ResponseTime == (
  ResponseComopletionTime - UserRequestTime);
  rule MsgReq = invariant forall p : PortTWSCCommon in
  self.PORTS | p.SendsFirstMessage == Yes -> p.NbMsgReq
  == p.NbMsgReq + 1 ;
  rule MsgRes = invariant forall p : PortTWSCCommon in
  self.PORTS | p.InOurControlDomain == Yes ->
  p.NbMsgRes == p.NbMsgRes + 1 ;
  rule successibility = invariant forall p
  :PortTWSCCommon in self.PORTS | succ == p.NbMsgRes /
  p.NbMsgReq;}
    
```

Figure 9. The ACME description of QoS of Web service .

Now we consider the example of the throughput factor that is a constraint expressed at the component level through the External Analysis concepts of ACME by using the rate Analysis function of Armani. Here is the formalization of this property:

```

External Analysis throughputRate(comp :Component) :
int = armani.tools.rateAnalyses.throughputRate(comp);
    
```

VI. EXOGENOUS TRANSFORMATION OF COMPOSITE WEB SERVICE TO ACME

This section demonstrates how a composite Web service for extension can be modeled as an ACME\ARMANI ADL and how it can be mapped to an equivalent MOF based model representing an ACME\ARMANI. In this part of the paper, we aim at automatically transforming composite Web services for the QoS extension into ACME. To achieve this automation, we get the meta-model of composite Web service proposed elaborate the partial ACME\ ARMANI meta-model. In addition, we implemented WSC2ACME, a tool for transforming a composite Web service software architecture to an ACME using the MDE transformation language ATL [13].

A. An Overview of the tool WSC2ACME

Our model transformation, which defines the generation of a target model from a source model, is described by a transformation definition, consisting of a number of transformation rules that are executed by a transformation case tool. There are various methods of specifying the model transformation [14].

In this Section, we present in a detailed way the WSC2ACME tool written in ATL allowing the transformation of the software architecture of the Web services towards an ACME model. In order to design and develop our WSC2ACME tool, we used the following constructions: standard rule, defined in the context of models element offered by the model transformation language ATL.

An ATL module corresponds to the transformation of a set of source models into a set of target models in accordance with their meta-models. Its structure is formed by a header section, an optional import section, a set of helpers and a set of rules. The header section defines the names of the transformation module and the variables according to the source and target models. It also encodes the module execution mode that can be either normal (defined by the keyword form) or refining. The syntax of the header section is defined as follows:

```

module WSC2ACME; -- Module Template
create OUT : acme from IN : Webservice;
    
```

OUT and IN are the names of the source and target models. They are not used thereafter. Both model types are respectively Web Service and ACME. Thus, they must conform to the meta-model defining their type.

• Translating of functional Web service properties (WSDL)

A Web service is translated into the ACME. We start by the functional property. To achieve this transformation we based our rules to the meta-model of the WSDL. We define the example of rule which allows us to transform

a WSDL and reifies all correspondences between the source component and the target system component.

```

rule definition2System {
from
s : Webservice!definition
to
t : acme!System (name <- s.name,Connector <-
s.dependance,Component <- s.services,links <-
s.bindings,Property<- Sequence { targetnamespace,
xmlns,msg, imports, Types,porttype}),...)
    
```

• Translating of composite Web service

A composite Web service is translated into the ACME. This composition presents an empty structure. We define the rule which allows us to transform a composite service and reifies all correspondences between the source component and the target system component.

```

rule Port2Port {
from
s : Webservice!Port
to
t : acme!Port (name <- s.name, Property <- Sequence {
Integrity,
s.reliability,Interface,SendFirstMessage,InOurControlDoma
in,WSDLDocRefs,EndPointAdressList, s.SOAP, s.prec,
s.reliability}),
Integrity:acme!Property( name <- 'Integrity',val <-
s.integrity.getIntegrity().toString()),
Interface:acme!Property( name <- 'Interface',val <-
s.Interface),
SendFirstMessage:acme!Property( name <-
'SendFirstMessage',val <- s.SendFirstMessage. toString()),
InOurControlDomain:acme!Property( name <-
'InOurControlDomain',
val <- s.InOurControlDomain.toString()),
WSDLDocRefs: acme!Property( name <-
'WSDLDocRefs',val <- s.WSDLDocRefs),
EndPointAdressList:acme!Property( name <-
'EndPointAdressList',val <- s.endpointadresslist)}
    
```

• Translating of transactional Web service properties

A transactional Web service is translated into the ACME. To achieve this transformation we based our rules to the meta-model of Web service for the transactional extension proposed in [7]. We define the example of rule which allows us to transform a meta-model of Web service for the transactional extension and reifies all correspondences between the source component and the target system component.

```

rule Prec2Property {
from
    
```

```

s : Webservice!precondition
to
t : acme!Property(name <- 'Prec ' ,val <- s.name)}
    
```

```

rule Dependance2Connector {
from
s : Webservice!dependance
to
t : acme!Connector(name <- s.name)}
    
```

• Translating QoS of Web service properties

A composite Web service for the QoS extension is translated into the ACME. To achieve this transformation we based our rules to the meta-model of the QoS of Web service. We define the example of rule which allows us to transform a meta-model of the QoS of Web service and reifies all correspondences between the source component and the target system component.

Example of helper:

```

helper context Webservice!Responsetime def :
getResponseTime() : Integer =
self.ResponseCompletionTime.val-
self.ConsumeRequestTime.val ;
    
```

Example of rule:

```

rule reliability2Property {
from
s : Webservice!reliability
to
t : acme!Property(name <- 'Reliability',val <-
s.getreliability().toString())
}
    
```

VII. CONCLUSION

The work deals with the modeling of the QoS of the Web service extension. This paper studies a model transformation of composite Web services for the QoS from the PSM, created Acme\Armani style architecture, into PSM. We have presented a specification of the QoS. We have also introduced the meta-model for the QoS of the Web service and the meta-model for Acme\Armani. To translate the composite Web services for the QoS extension to the Acme\ARMANI, we have introduced a set of the ATL transformation rules to implement WSC2ACME tools in order to transform a Web services model conform to its meta-model to a partial ACME model conform to the meta-model of the ACME.

In our future works we are considering the following perspectives:

- Improve the efficiency of our WSC2ACME using the large logistic problem.

- Assessing the *WSC2ACME* quality using verification techniques applicable on the model transformation: structural tests, mutation analysis, statistical analysis, contracts [15], [16], [17], [18].

REFERENCES

- [1] M. Graiet, R. Maraoui, M. Kmimech, M.T. Bhiri , W. Gaaloul: Towards an approach of formal verification of mediation protocol based on Web services, 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS2010), Paris-France, November 2010.
- [2] S. Araban and L. S. Sterling. Measuring quality of service for contract aware Web services. In First Australian Workshop on Engineering Service-Oriented Systems, pages 54–56, 2004
- [3] R. Shuping A model for Web services discovery with qos. SIGecom Exch., 4(1) :1–10, 2003.
- [4] R. Ben Hlima , Conception, implantation et expérimentation d'une architecture en bus pour l'auto-réparation des applications distribuées à base de services Web, l'Université Toulouse III - Paul Sabatier et la Faculté des Sciences Économiques et de Gestion – Sfax, Le Jeudi 14 Mai 2009,
- [5] M. Kmimech, M. Tahar Bhiri, M. Graiet and P. Anierté.: Checking component assembly in ACME: an approach applied on UML 2.0 components model. In 4th IEEE International Conference on Software Engineering Advances (ICSEA'2009), Portugal, IEEE CS Press, Septembre 2009.
- [6] Garlan, D., R. Monroe, and D. Wile (2001). *ACME: Architectural Description of Component-based (2001). Capturing software architecture design expertise with Armani*. Technical Report CMU-CS-98-163, Carnegie Mellon University School of Computer Science.
- [7] Maraoui R., Graiet M., Kmimech M., Bhiri M.T., and Elayeb B., Formalisation of protocol mediation for Web service composition with ACME/ARMANI ADL, Service Computation IARIA 2010-Lisbon-Portugal, Nov 2010.
- [8] Wan Nurhayati AB. R., UML QoS Profile exploration for the specifications of a generic QoS metamodel for designing and developing good quality Web services , School of Computing, Science & Engineering University of Salford, Salford, UK, March 2010.
- [9] D. Garlan, R.T Ronroe, D. Wile ACME.: An Architecture Description Interchange Language , Proceedings of CASCON 97, Toronto, Ontario, November, 169--183, 1997.
- [10] D. Garlan, R. T. Monroe, and D. Wile. ACME: Architectural Description of Composed-Based Systems. Gary Leavens and Murali Sitaraman, ed.s Kluwer, 2000.
- [11] Group2006, <http://www.cs.cmu.edu/~ACME/ACME Studio/>.
- [12] C. Gacek. C, and C. Gamble (2008): Mismatch Avoidance in Web Services Software Architectures. Journal of Universal Computer Science, vol. 14, no. 8 (2008), 1285-1313.
- [13] Combemale. B, Approche de méta-modélisation pour la simulation et la vérification de modèle, *application à l'ingénierie des procédés* : Thèse de Doctorat, Toulouse, (11 juillet 2008).
- [14] Belzad B. and Anthanasios S., On behavioural model transformation in Web service, school of computer science, University of birmingham.
- [15] Küster, J. M. Definition and validation of model transformations. Software and systems Modeling, in press 2006.
- [16] Mottu, J. M., Baudry, Brottier. E and LeTrao. Y (2005) Génération automatique de tests pour les transformartions de modèles. Première journée sur IDM, Paris, 2005.
- [17] Baudry. B, Ghosh. S, Fleurey. F, France. R, Le Traon. Y and Mottu. J. M. Barries to systematic model transformation testing. Communications of the ACM, Vol. 53, No. 6 (2009).
- [18] E. Cariou, N. Belloir, F. Barbier, and N. Djemam. OCL Contracts for the Verification of Model Transformations. In Proceedings of the Workshop the Pragmatics of OCL and Other Textual Specification Languages at MODELS 2009, volume 24. Electronic Communications of the EASST, 2009.

ATL Transformation of UML 2.0 for the Generation of SCA Model

Soumaya Louhichi

MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
louhichi.soumaya@gmail.com

Mohamed Graiet

MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
mohamed.graiet@imag.fr

Mourad Kmimech

MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
mkmimech@gmail.com

Walid Gaaloul

Computer Science Department Télécom SudParis
9, rue Charles Fourier 91 011 Évry Cedex, France
walid.gaaloul@it-sudparis.eu

Mohamed Tahar Bhiri

MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
Tahar_bhiri@yahoo.fr

Eric Cariou

Université de Pau et des pays de l'Adour
Avenue de l'Université BP 1155 64013
PAU CEDEX France
Eric.Cariou@univ-pau.fr

Abstract— Service Component Architecture specification (SCA) is an emerging and promising technology for the development, deployment and integration of Internet applications. This technology supports the management of dynamic availability and treats the heterogeneity between the components of distributed applications. However, this technology is not able to solve all problems. Currently, software systems are evolving. This factor makes development and maintenance of systems more complex than before. One solution to remedy this was the use of the Model Driven Engineering (MDE) approach in the development process. The aim of this paper is to apply an MDE automation type ensuring the passage from an UML 2.0 model to SCA model. To achieve this, we study two metamodels: the UML 2.0 component metamodel and the SCA meta-model. To ensure traceability between these two meta-models, we have defined transformation rules in ATL language.

Keywords-UML 2.0, SCA, MDE, ATL

I. INTRODUCTION

Nowadays, Service Oriented Architecture (SOA) [1] can be seen as one of the key technologies to enable flexibility and reduce complexity in software systems. SOA is a set of ideas for architectural design and there are some proposals for SOA frameworks including a concrete architectural language: the Service Component Architecture (SCA) [2].

SCA is a new promising programming model for constructing service-oriented application which facilitates the development of business integration in Service Oriented Architecture (SOA). SCA technology supports the management of dynamic availability and treats the heterogeneity between the components of distributed applications. In spite these advantages, SCA application are incomprehensible by stakeholders who have not enough knowledge in the SOA field. For this, we decide to use the modelling languages to describe SCA concepts.

The most adopted modelling language to SCA is the UML 2.0 which approved itself as a powerful tool for modeling components and services.

Recently, the application development process becomes more and more complex. To remain competitive, companies must significantly reduce their development and maintenance costs. A solution for this is the use of MDE approach, a new discipline of software engineering, which has emerged to deal with complexity, growth, rapidly changing and heterogeneity in software applications.

The increasing use of MDE solves the problem of complexity in the development process at a high level of abstraction. Thus, an application can be generated automatically from high level models.

The goal of this paper is to apply an MDE automation type to develop a tool that transforms an UML 2.0 component model to an SCA model. The result of this transformation is an XMI [3] file, which then can be used as a template to produce the source code of an SCA application. The transformation is expressed in ATL language (Atlas Transformation Language) [4].

This paper is organized as follows: in section 2, we present the MDE approach, the metamodeling and transformation languages. In Section 3, we study our two metamodels for UML 2.0 and SCA. In the next section, we develop the transformation rules. Section 5 is the subject of the implementation and execution of those rules. We end with a conclusion.

II. MODEL DRIVEN ENGINEERING

The Model Driven Engineering has become in recent years the most used approach for developing quality software. This approach more abstract than the programming one allows focusing on concepts independently of platforms, focusing on one or more concerns abstract and study them to obtain a complete system by composition and by transformation.

The concept of model is at the heart of the device, in MDE a model is considered as entity of first class in the software development [5], it serves not only to better understand and reason about the system we built, but also to be in position to transform models into other abstract models or into practical implementation one. In the rest of this section, we will present the main artifacts of Engineering Models, languages expressing the metamodels and model transformations. A metamodel is a model that describes all the kind of elements and their relationships that can be instantiated for forming models. For instance, the UML metamodel describe all the kinds of UML diagrams and their elements (Class, State, Component, Activity, Use Case,...). In MDE, each model is conformed to a metamodel. Metamodel are key constructions because they make models automatically handable by tools. A metamodel defines concretely a modeling language.

The most widely used MDE platform is EMF (Eclipse Modeling Framework) which provides a metametamemamodel (the metamodel allowing the definition of metamodels) called Ecore. Ecore is aligned on the MOF (Meta Object Facilities) which is the standard metametamemamodel from the OMG [6]. EMF is a modeling and code generation framework used to support the creation of model driven tools and applications.

Model transformations are at the heart of Model Driven Engineering, and provide the essential mechanism for manipulating and transforming models. The transformation of models plays an important role in the Model Driven Engineering. Indeed, several studies have been done to define transformation languages that ensure effectively the passage between models. We will use the ATL free tool [7]; it quickly seems to us as the best suited tool to the problem of transformation. In fact, ATL is a proposal submission in response to the RFP call delivered by the OMG. ATL is one of the most popular and widely used model transformation languages. ATL is a hybrid model transformation language containing a mixture of declarative and imperative constructs based on Object Constraint Language (OCL) [8] for writing expressions. ATL transformations are unidirectional, operating in on read-only source models and producing write-only target models (Figure 1). During the execution of a transformation, source models can be navigated but changes are not allowed. Target models can not be navigated.

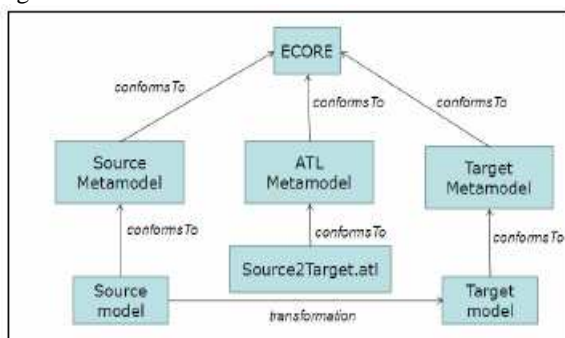


Figure 1. ATL model transformation schema

III. UML 2.0 AND SCA METAMODELS

The transformation process requires initially the presence of two metamodels:

- Source metamodel: the UML 2.0 metamodel.
- Target metamodel: SCA metamodel.

A. Source Metamodel: UML 2.0 Metamodel

The UML 2.0 metamodel definition consists of two parts: UML 2.0 Superstructure, which defines the user vision and UML 2.0 Infrastructure, which specifies the metamodeling architecture of UML and its alignment with MOF (Meta-Object Facility) [9]. In the remainder of this section, we focus firstly on UML 2.0 Superstructure which is simply denoted UML 2.0 [10] and then we study the behavioral part of a component model.

1) Structural concepts of UML 2.0

The main structural concepts of UML 2.0 component model are: component, port, connector [11].

- The component: represents a modular part of a system that encapsulates its contents and which is replaceable within its environment. Its description may include a set of ports and a set of connectors.
- Port: allows the component to communicate with its environment, a port can be equipped with provided or required interface used to specify the expected operations of the environment or to specify provided operations of the component.
- Connector: A connector defines a relationship between two ports. We find two types of connectors: The Delegation Connector and the Assembly Connector. The Delegation Connector represents the forwarding of messages between a port of a component and a port of one of its part. The Assembly Connector must only exist between a provided Port and a required one.

UML 2.0 metamodel represents the different relationships between UML 2.0 concepts (structural and behavioral concepts). Relations between these concepts are defined in the following points:

- A component inherits the metaclass Class. It also inherits EncapsulatedClassifier. So, it can have ports typed by provided and required interfaces.
- The metaclass EncapsulatedClassifier inherits StructuredClassifier. Therefore, a component can have an internal structure and may define connectors.
- The metaclass Property models the properties of an instance of StructuredClassifier.
- The metaclass Port represents an interaction point between a classifier and its environment.
- EncapsulatedClassifier is a classifier with port typed by interfaces.
- The metaclass connector represents a link that allows instances to communicate with each other.
- Every extremity of connector named ConnectorEnd represents a distinct role of the communication represented by the connector.

- The metaclass ConnectorEnd represents an endpoint of a connector, which attaches the connector to a connectable element. Each connectorEnd is a part of one connector.

2) *Behavior concept of UML 2.0*

UML is a popular representation and methodology for characterizing software. In fact, UML supports the modeling of system behavior through the use of state machines.

UML has two types of state machines:

- Behavioral state machines.
- Protocol state machines.

In UML 2.0, the state machines can be used to specify the behavior of several elements of the models described in UML 2.0, such as instances of an UML 2.0 class. While protocols state machines may be used profitably to express protocols related to scenarios of use of services offered by interfaces or ports[12]. Behavioral and protocol state machines share common elements like state, region, vertex, pseudostate, transition...

- State: models a situation during which some invariant conditions holds.
- Region: is an orthogonal part of either a composite state or a state machine. It contains states and transitions.
- Vertex: is an abstraction of a node in a state machine graph, it can be the source or destination of any number of transitions.
- Pseudostate: is an abstraction that encompasses different types of transient vertices in the state machine graph.
- Transition: it shows the relation ship, or path, between two states or pseudostates. Each transition can have a guard condition that indicates if the transition can even be considered (enabled), a trigger that causes the transition to execute if it is enabled, and any effect the transition may have when it occurs.

A protocol state machine has the characteristics of a generic state machine (composite states, concurrent regions...) with the next restrictions on states and transitions [13]:

- States cannot show entry actions, exit actions, internal actions, nor do activities.
- State invariants can be specified.
- Pseudostates cannot be deep or shadow history kinds; they are restricted to initial, entry point and exit point kinds.
- Transitions cannot show effect actions or send events as generic state machines can.
- Transitions have pre and post-conditions; they can be associated to operation calls.
- A protocol state machine may contain one or more regions which involve vertices and transitions. A protocol transition connects a source vertex to a target vertex. A vertex is either a pseudostate or a state with incoming and outgoing transitions. States may contain zero or more regions.

- A state without region is a simple state; a final state is a specialization of a state representing the completion of a region.
- A state containing one or more regions is a composite state that provides a hierarchical group of (sub) states; a state containing more than one region is an orthogonal state that models a concurrent execution.
- A submachine state is semantically equivalent to a composite state. It refers to a submachine (sub Protocol State Machine) where its regions are the regions of the composite state.

Figure 2 corresponds to the UML 2.0 metamodel for describing components illustrating the different relationships between concepts (structural and behavioral concepts) in a component UML 2.0.

B. *Target Metamodel: SCA Metamodel*

In this section, we describe the different structural and behavioral concepts of SCA model necessary for the construction of its metamodel.

1) *Structural concepts of SCA*

Services Component Architecture (SCA) is a set of specifications describing a model for building applications and systems using Service Oriented Architecture SOA [14]. SCA complete previous approaches in the implementation of services, and focuses on open standards such as Web services.

SCA provides an application code based on components and divides the deployment of a service-oriented application into two stages:

- The implementation of components that provide and consume services.
- The assembly of sets of components to deploy applications, by connecting the references to services.

An SCA implementation represents a reusable service component that encapsulates the business logic that supports one or more services. Implementations can be in a variety of languages, including Java, BPEL4WS [15], C, and COBOL. Implementations also define the references dependencies on other components' services that the implementation must invoke during normal operation as well as configuration properties. Interface types (typically WSDL portTypes) describe both services and references. Services and references use SCA bindings to configure the interaction protocol used for providing or using a service. Examples of bindings are the Web services binding (the Web services protocol stack) or a messaging backbone.

Services, references, and properties define an SCA implementation's configurable aspects.

An SCA component is a configured SCA implementation that sets property values and resolves the references to other SCA components by specifying the component wires (interconnections). An SCA composite (or SCA assembly) is a packaged set of components and wires that define the structural composition.

The SCA composite can provide for the interaction between internal components and external applications by defining

composite services, references, and properties. This means that an SCA composite can be an SCA component within another SCA composition, with the first SCA composite providing that component's implementation. In SCA, this is called recursive service composition.

2) Behavior concepts of SCA

SCA specification are based on services which are becoming more and more popular as means for decoupling systems from each other while at the same time making functionality and data available to all authorized applications on the network.

Behavioral descriptions of services can be defined using higher level standards such as BPEL (Business Process Execution Language). BPEL is an XML-based language that models a business process as a composition from a set of elementary web services.

The main concept of BPEL [16] is BPEL process which defines several concepts like basic and structured activities, variables, partner links, and handlers. In a simple case, a BPEL process defines partner links, variables, and activities.

- Partner links represent message exchange relationships between two parties. Via a reference to a partner link type the partner link defines the mutual required endpoints of a message exchange: the myRole and a partnerRole attributes defines who is playing which role. Partner links are referenced by basic activities that involve Web Service requests.
- Variables are used to store workflow data as well as input and output messages that are exchanged by Web Services activities via partner links.
- Handlers specify responses to unexpected behavior like time or message events, faults, compensation, or termination.
- Nesting of structured activities is used to express control flow in BPEL. There are specific structured activities for loops (while, forEach, repeatUntil), sequential execution (sequence), conditional branching based on data (if) or events (pick), and concurrent branches (flow).

- Basic activities specify the actual operations of a BPEL process. There are three activities involving Web Services: invoke for synchronous or asynchronous calls to a remote Web Service, receive to wait for the receipt of a specific message, and reply for responding to a remote request.

All these activities reference a partner link and specify input and/or output variables for messages.

3) SCA metamodel

Figure 2 corresponds to the SCA metamodel illustrating the different relationships between SCA concepts (structural and behavioral concepts). Relations between these concepts are defined in the following points:

- An SCA component may have zero or more than one service.
- An SCA component may have zero or more than one reference.
- An SCA component may have many properties used to configure its implementation
- A service is defined by only one interface and it may have multiple bindings and it may have also multiple BPEL process to describe it's behavior.
- A reference is defined by only one interface and it may have multiple bindings and it may have also multiple BPEL process to describe it's behavior.
- An interface describes the set of operations offered by the service or used by the reference.
- A composite may be considered as a set of components, having many properties, services, references and wires.
- A BPEL process is a set of partners, partnerLinks, variables and activities.
- A partner may have zero or more than one partnerLink.
- A partnerLink may have zero or one partnerLinkType which may contain one or two Role.

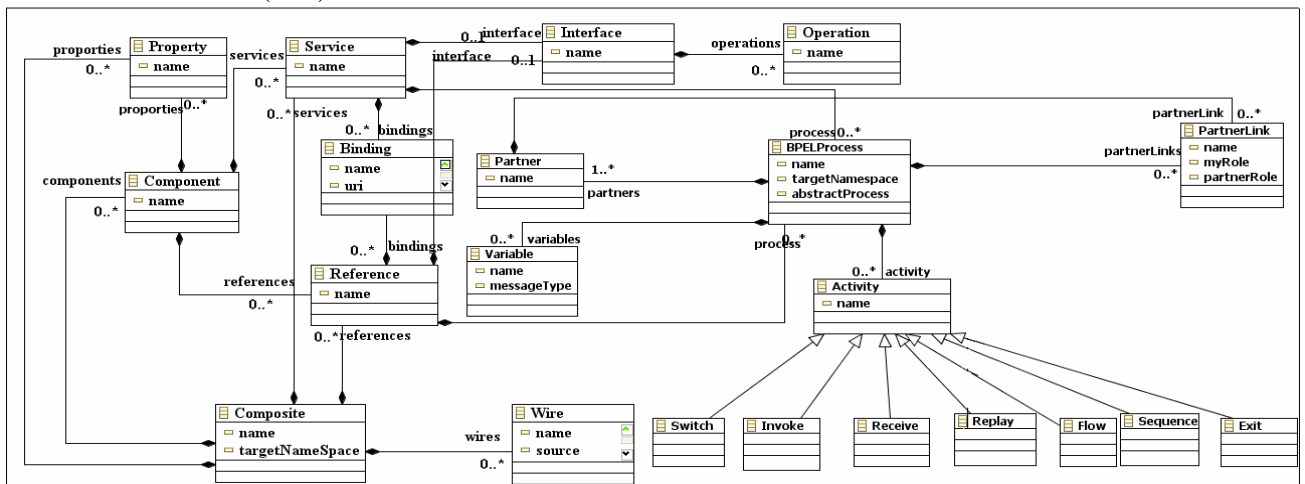


Figure 2. Ecore metamodel of SCA

Figure 3 presents the UML 2.0 metamodel for describing components. It is used to describe the different relationships

between structural and behavioral concepts of UML 2.0 component model:

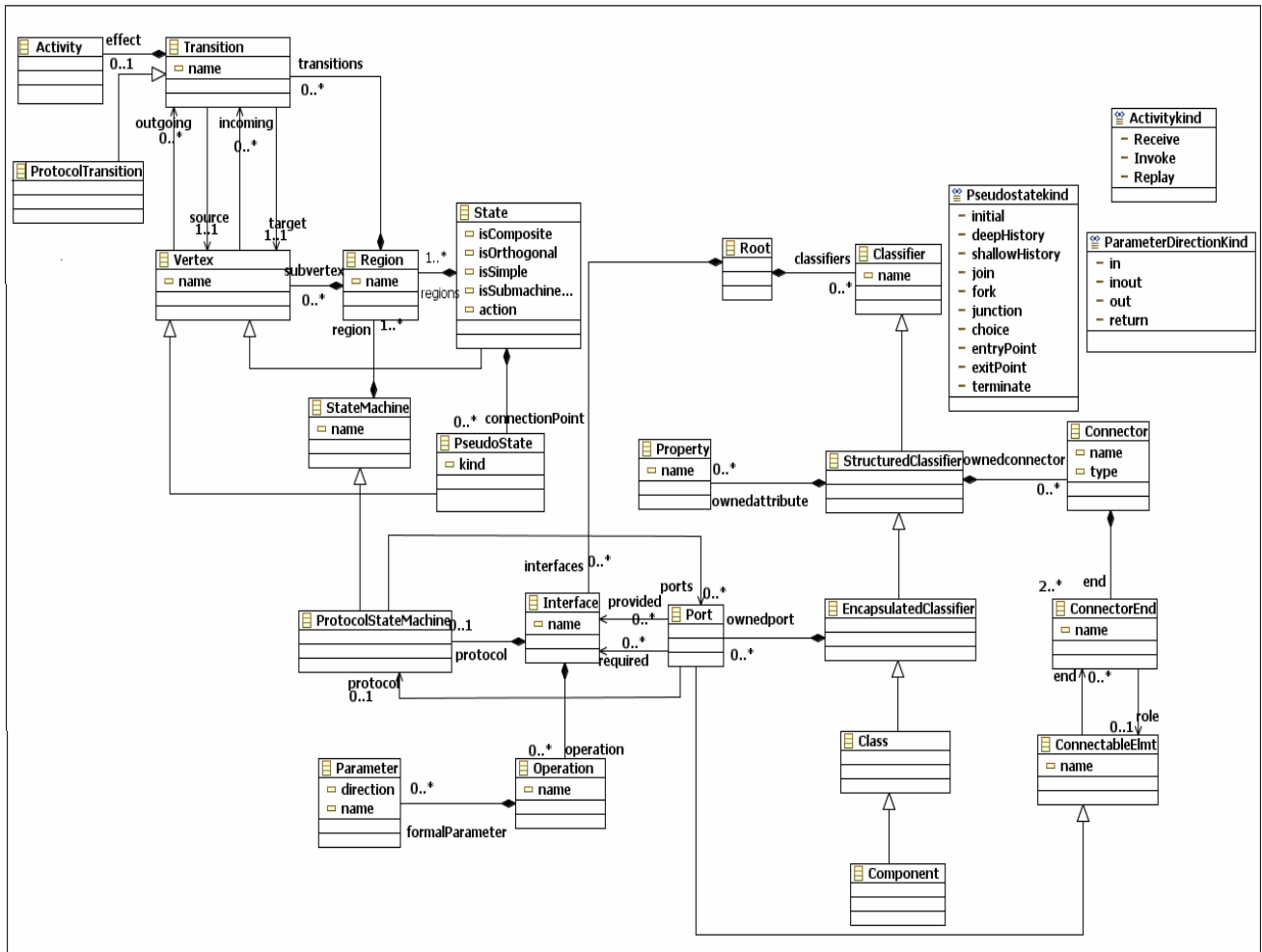


Figure 3. Ecore definition of the UML 2.0 component part

IV. THE TRANSFORMATION RULES

In this section, we present the transformation rules allowing the passage from an UML 2.0 component model to an SCA model. The transformation rules are established between source and target metamodels, in other words between all the concepts of source and target models (structural and behavior concepts). These rules are briefly explained in the following table in natural language and then formulated using the ATL syntax previously introduced.

TABLE I. SUMMARY OF THE TRANSFORMATION RULES

UML 2.0 concepts of source model	SCA concepts of target model
Component	Component SCA
	Partner
Port with provided interface	Service
	PartnerLink

Port with required interface	Reference
Interface	Interface
Operation	Operation
Property	Property
ConnectorEnd	Binding
Connector	Wire
Protocol State Machine	Process BPEL
Parameter	Variable
Region	Sequence
State	Basic Activity (Receive,Invoke,Replay)
PseudoState (kind= choice)	Switch
PseudoState (kind= fork)	Flow
PseudoState (kind= exitPoint)	Exit

Before starting to define some transformation rules, we will give the general form of these:

```
rule ForExample {
  from
    i : InputMetaModel!InputElement
  to
    o : OutputMetaModel!OutputElement(
      attributeA <- i.attributeB,
      attributeB <- i.attributeC + i.attributeD
    )
}
```

Figure 4. An example of ATL rule

- ForExample: is the name of the transformation rule.
- i (resp. o) is the name of the variable representing the identified element source that in the body of the rule (resp. target element created).
- InputMetaModel (resp. OutputMetaModel) is the metamodel in which the source model (resp. the target model) of the transformation is consistent.
- InputElement means the metaclass of elements of source model to which this rule will be applied.
- OutputElement means the metaclass from which the rule will instantiate the target elements.
- The exclamation point ! used to specify to which meta model belongs a meta class.
- attributeA and attributeB are attributes of the meta class OutputElement, their value is initialized using the values of attributes i.attributeB, i.attributeC and i.attributeD of the meta class InputElement.

We will now proceed to the definition of some of our transformation rules using the ATL language:

- Rule that transforms an UML 2.0 component to an SCA component, here an SCA component takes the same name as a UML 2.0 component. This rule also allows the transformation of each instance of an UML 2.0 component in a BPEL Partner in SCA model.

```
rule component2componentsca
{
  from c:UML!Component
  to cs:SCA!Component(
    name<-c.name+'_serviceComponent',
    properties<-c.ownedattribute),
    p:SCA!Partner(name<-c.name,
    owner<-c.ownedport->first().protocol)
}
```

- Rule that transforms an UML 2.0 port with provided interface to an SCA Service. This rule allows also the transformation of each port in UML 2.0 into aPartner Link BPEL in SCA model.

```
rule port2service{
  from p:UML!Port(
    p.provided->notEmpty())
  to ps:SCA!Service(
    name<-p.name+'_service port',
    interface<-p.provided->first(),
    component<-p.owner,
    bindings<-p.end,process<-
    p.protocol),
    pl:SCA!PartnerLink(name<-p.name,
    myRole<- 'ITF_'+p.name+'Provider',
    partnerRole<-',
    owner<-p.protocol)
}
```

- Rule that transforms a Protocol State Machine to a BPEL process.

```
rule psm2BPELprocess{
  from ps: UML!ProtocolStateMachine
  to p: SCA!BPELProcess(name<-ps.name,
  targetNamespace<-
  'http://'+ps.name+'.org/',
  abstractProcess<- false)}
```

V. IMPLIMENTATION AND EXECUTION OF THE TRANSFORMATION RULES

At first, we have developed two ECORE models corresponding to source metamodel and target metamodel, after we have implemented the transformation rules in the ATL language. Once the transformation program UML2SCA.atl is created, then we can start the execution. The general context of the ATL transformation is illustrated in Figure 5 below.

The engine of transformation allows generating the SCA model, which is consistent to SCA metamodel, from the UML 2.0 model which is consistent to UML 2.0 metamodel using UML2SCA.atl program which must be also consistent to metamodel that defines the semantics of ATL transformation. All metamodels must be consistent to the Ecore metamodel.

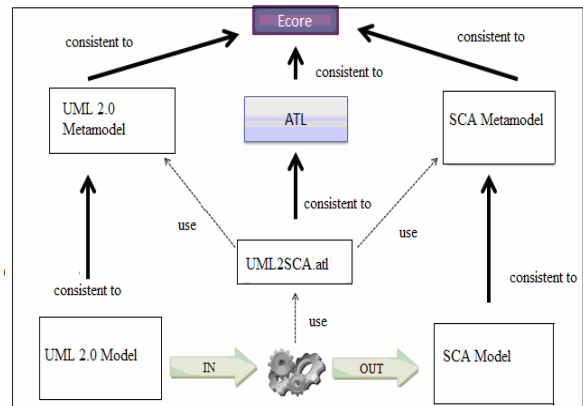


Figure 5. General context of ATL transformation

To validate our transformation rules, we completed several tests. As an illustration, we consider the example below (Figure 6). The example studied is an example of an automated banking machine (ABM). Any person with an appropriate card can use the ABM to take money. To take the money, a customer must be identified.

Our example can be modeled in UML 2.0 as follows: a customer is modeled by a Customer component with a port named abm typed by a required interface named identify. The ABM is modeled by an ABM component having port named customer typed by provided interface named authenticate. These two components are connected by a connector named Customer-ABM.

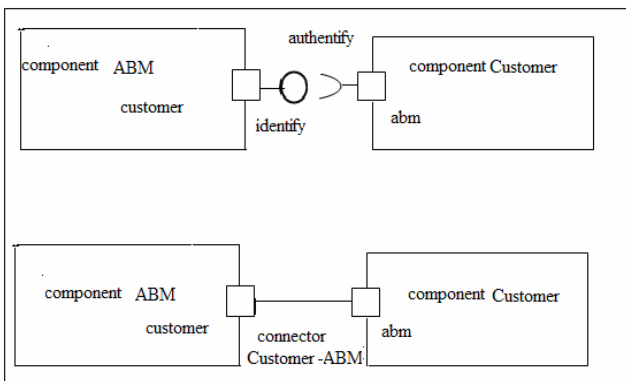


Figure 6. Source model

Behavior of ABM component is described using its interface identify. Behavior of this last one is described using a protocol state machine named identification.

We get the following input model as shown in Figure 7.

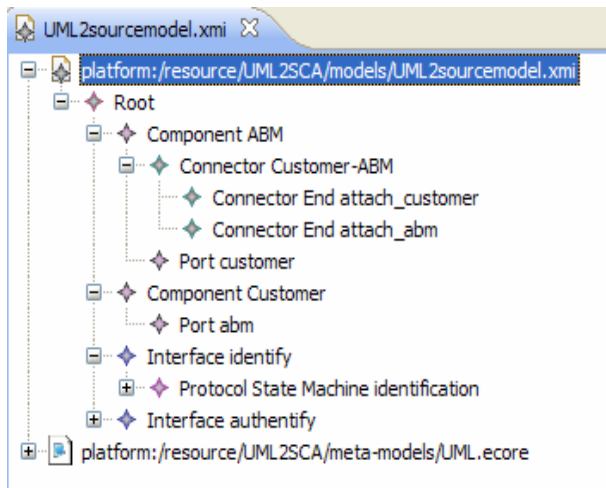


Figure 7. Source Model in Text Editor View

When the model is validated and there are no errors, the user can run the ATL model transformation to transform the UML 2.0 model into SCA model and the SCA Ecore model is created. The result of this transformation is shown in Figure 8 below.

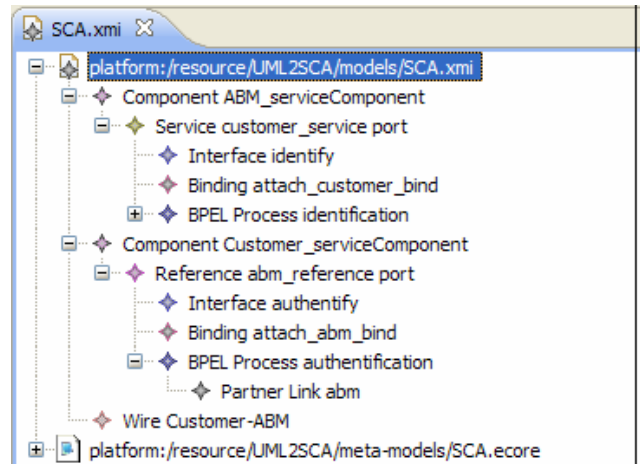


Figure 8. Target Model in Text Editor View

We can see from Figure 8 that each UML 2.0 component has been transformed into an SCA component, each port in UML 2.0 typed with provided interface has been transformed into an SCA service, each port with required interface has been transformed into an SCA reference and each instance of an assembly connector (in our example Customer-ABM) has been transformed into an SCA wire (wire Customer-ABM). Concerning the behavioral part, each instance of Protocol State Machine in UML 2.0 has been transformed into a BPEL Process in SCA.

VI. CONCLUSION

We applied the MDE approach to service-oriented applications engineering. It is a question of generating the ingredients of an SCA application from an UML 2.0 component diagram. To reach there, we elaborated at first time the source metamodel representing an UML 2.0 component diagram. At the level of target metamodel, we try to design all the metaclasses needed to generate a PSM model respecting the SCA architecture. Transformation rules have been developed in ATL language. The transformation process allows generating an XMI file containing a structural and behavioral description of the SCA application: SCA components, services, references, interface, operations, bindings as well as the process BPEL used to describe the behavior of SCA application.

As future work, we intend to more improve the behavioral aspect of SCA application and to try to treat the composite aspect in SCA.

REFERENCES

- [1] OSOA, Open Service Oriented Architecture, the Home Page, 2007. <http://www.osoa.org/>
- [2] Open SOA Collaboration, Service Component Architecture (SCA), SCA Assembly Model v1.00 specifications, 2007.
- [3] B. Combemale, " Approche de Métamodélisation pour la Simulation et la Verification de Modèle". Toulouse, 2008.

- [4] J. Troya and A. Vallecillo, “Towards a Rewriting Logic Semantics for ATL”. ISUM/ A tenea Research Group. Universidad de Maalaga, Spain.
- [5] J. Bézivin , “Sur les Principes de L’ingénierie des Modèles”, RSTI_L’objet 10, ou sont les objets?, 2004.
- [6] OMG, Meta Object Facility (MOF) specification –version 1.4 formal, April 2002.
- [7] The LINA website. Available: <http://www.sciences.univ-nantes.fr/lina/atl>
- [8] B. Combemale and S. Rougemaille, “–ATL- Atlas Transformation Language”, 2005.
- [9] OMG, “ Meta Object Facility (MOF) specification, version 1.3,” Mars 2000.
- [10] X. Blanc, “MDA en Action Ingénierie Logicielle Guidée Par les Modèles”, Eyrolles 2005.
- [11] M. Graiet, “Contribution à une Démarche de Vérification Formelle d’Architectures Logicielles”, 2007.
- [12] OMG, “Unified Modeling Language : Superstructure version 2.0”.
- [13] C. Dumez, NAIT-sidi-moh, J. Gaber and M. Wack, “Modeling and specification of Web services composition using UML-S”.
- [14] F. Curbera, “Component Contracts in Service-oriented Architectures”, IBM T.J. Watson Research Center, 2007.
- [15] OASIS, “Service Component Architecture WS-BPEL Client and Implementation Specification Version 1.1”, 2009.
- [16] T. Ambuhler, “UML 2.0 Profile for WS-BPEL with Mapping to WS-BPEL”.

Towards the Development of Integrated Reuse Environments for UML Artifacts

Moataz A. Ahmed

Information and Computer Science Department
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
moataz@kfupm.edu.sa

Abstract—Systematic software reuse is recognized to achieve better software, faster and at a lower cost. The benefits of reuse can be maximized if types of early stage software artifacts can be easily reused. In early-stage reuse, once a match is found, all related later stages artifacts for the match can also be reused. However, the development of integrated reuse environments to allow managing and reusing repositories of early stage artifacts has not caught adequate attention of researchers yet. In response to this problem, we propose an approach to the development of environments integrated with CASE tools and capable facilitating early-stage artifacts reuse. Successful implementation of such environments is expected to improve the software quality and developers productivity.

Keywords—early-stage artifacts; design reuse; integrated reuse environment; similarity metrics; multi-view similarity.

I. INTRODUCTION

Systematic software reuse has clear benefits to include reduction in overall development costs, increased reliability, reduced process risk, effective use of specialists, standards compliance, and accelerated development [1]. Features of the object-oriented (OO) software development paradigm, such as abstraction and encapsulation, encourage reuse of software by enabling building reusable blocks of *code*. However, it has been recognized for long that reuse of *early-stage* artifacts are particularly more beneficial than reuse of later-stage artifacts [10]. Types of early-stage reusable artifacts include [2]:

Domain Models: These can be reused at the earliest stage of the software development process, the domain analysis stage. Very few systems exist that exploits the reuse of artifacts at this stage. An example of such a system is the work of Blok and Cybulski [3]. Another example is the generic application frames in the ITHACA development environment [4]. Yet, a more recent example can be found in the software product lines approach, which was often touted as a silver bullet for actualizing software reuse goals [5][7][9].

Requirement Specifications: These artifacts can be reused during the requirements analysis phase of the software lifecycle [10]. An example of how a requirements specification reuse may be assisted by a software tool is described by Cybulski and Reed's [11].

Design: These artifacts can be reused during the design phase. An example of a design repository is the SPOOL Design Repository [12]. Another example is the work of Lee and Harandi [13].

In early-stage reuse, once a match is found, all related later stages artifacts for the match can also be reused. For instance, if an analysis model for a previous project is found to match the analysis model of a current project, then the previous project's design, code, test data, and relevant documentation may be reused in the current project.

Early-stage artifacts reuse, despite its clear benefits, suffers from a few problems though. Reuse problems include increased maintenance costs, the not-invented-here syndrome, *lack of tool support*, *difficulty of maintaining a library of reusable artifacts*, and *the cost of locating and adapting reusable artifacts* [1].

A step towards a solution to the problems above could be the development of effective tightly-knit tools to allow finding and reusing exiting design artifacts and what follows based on matching requirements specification. For maximal utilization within the day-to-day activities, such tools should be offered through a reuse environment integrated with some prominent CASE tools; hence, Integrated Reuse Environment (IRE). In this paper, we present an approach to the development of IREs to maximize the designer's productivity. The approach will be focusing on reusing Unified Modeling Language (UML) artifacts. The rationale behind focusing on UML is that it is considered the de-facto standard for expressing early-stage OO artifacts (e.g., analysis and design models) [8]. Accordingly, tools and techniques to support reusing UML artifacts would facilitate and encourage more early-stage reuse. However, to the best of our knowledge, there is IREs that allow finding and reusing exiting UML design artifacts and what follows based on matching requirements specification.

The rest of the paper is organized as follows. Section II presents a framework for assessing the similarity between UML artifacts of different projects. Section III discusses related work. Section IV lists some research questions to be answer in future work in order to realize effective IREs. Section V concludes the paper.

II. MULTI-VIEW UML MODELS SIMILARITY

During the requirement-analysis phase of the software development life cycle, the system requirements are typically modeled and analyzed from related but different viewpoints where each view represents one aspect of the software system to be developed. The division into different views is arbitrary and typically includes at least three views namely *structural view*, *functional view*, and *behavioral view* [21][22][23][24], each capturing important

aspects of the system, but all required for a complete description of the system. One or more kinds of diagrams provide a visual notation for the concepts in each view. A typical software procedure incorporates all three aspects [21]. Models from different views are meant to be compared to discover requirements that would be missed using a single view [1][22]. Structure describes static objects relevant to the domain in question, their relationships, attributes and their possible states; functions describe the input-output transformations, and behavior the instantiation and dynamics of the transformations with time.

It is worth noting here, though, that there is a general paucity of concepts for specifying the *functionality* of object communities [22]. The UML taxonomy of diagrams provides a logical organization for the various major kinds of diagrams into only two major categories: structure and behavior; with no category to represent the functional aspect [39]. Nevertheless, *use cases* can be interpreted as one means of specifying functionality, as according to Jacobson et al. [16], *they define the functionality inside the system and constitute a specific way of using some part of this functionality*. Clearly, a use case has also a flavor of behavior abstraction, as it is a special sequence of related transactions in the interaction between the actor and the system [22].

During the requirements engineering phase, the view-points analysis technique relies on these multi-view models where they are compared to discover requirements that would be missed using a single view.

We propose that developing IREs to allow early-stage OO artifacts reuse would require a framework of consistent multi-view similarity metrics that considers similarity across the three system views: functional view, structural view and behavioral view. For effective reuse of available designs of completed projects, the IRE should facilitate assessing the combined similarity between new requirements to the requirements of completed projects to provide closest match so that their design counterparts can be reused with minimal effort.

Considering UML, as the de-facto standard, we consider *Use Cases* as representative of the functionality (i.e., the services) that users require of the object oriented system. Use cases describe the typical interactions between the users of a system and the system itself, providing a narrative of how a system is used. During the requirements phase of a software project, analysts can take use cases to the next level by providing a more formal level of refinement in a form of *sequence diagrams*. Each use case is realized by one or more sequence diagrams that depict how the objects interact and work together to provide services. We propose considering the development of sequence diagrams similarity metrics in the functional view.

UML Structure diagrams show the static structure of the objects in a system. Examples of UML structure diagrams include the Class Diagram, Component Diagram, Object Diagram, Deployment Diagram, Package Diagram, Composite Structure Diagram, and the Profile Diagram. However, most of these diagrams are mainly used during the architecture and design phase to express artifacts at

different design levels. During the requirements engineering phase, the static structure of the system is mainly captured using instances of the Class Diagram and Object Diagram. The Class Diagram shows the building blocks of any object-oriented system: the classes that make up a system. The potential for collaboration among these classes, through message passing, is shown in the relationship between these classes. Object diagrams show instances instead of classes. They are useful for explaining small pieces of class diagrams with complicated relationships, especially recursive relationships. We propose considering the development of class diagrams similarity metrics in the structural view.

Behavior diagrams can be used at two different levels: system level and object level. At the system level, behavior diagrams (mainly the State Machine Diagram, which is an object-based variant of Harel's statecharts [39]) is used to show the system behavior in response to user actions, as in user interface design [38]. At the object level, they show the dynamic behavior of the objects, including their methods, collaborations, activities, and state histories. The dynamic behavior of a system can be described as a series of changes to the system over time. Examples of UML behavior diagrams include Activity Diagram, Interaction Diagram, and State Machine Diagram. However, most of these diagrams are mainly used during the architecture and design phase to express artifacts at different design phases. During the requirements engineering phase, the system-level behavior is of interest and mainly captured using the State Machine Diagrams [38]. They are used to more formally describe the flows within or between use cases. We propose considering the development of state machine diagrams similarity metrics in the behavioral view.

The left-hand side part of Fig. 1 shows that design reuse is achieved by comparing the requirements of the new system to requirements of existing systems in a repository. The comparison is meant to assess the level of similarity. If the level of similarity between the best matching old requirements and the given new requirements is greater than a given threshold, corresponding design artifacts can be reused as a starting point for the new requirements. The right-hand side of the figure shows that the similarity assessment between the new and old requirements should consider the three views (depicted as dimensions): use cases representing the functional view, class diagrams representing the structural view, and state machine diagrams representing the behavioral view. It is worth noting here that sequence diagrams are used in the view-points analysis technique during the requirements engineering phase to double check the consistency between the structural view and the functional view as shown in the figure. Moreover, as stated above, sequence diagrams can be used to provide a more formal level of refinement of use cases. Accordingly, sequences diagrams could be used as a better representative of the functional view.

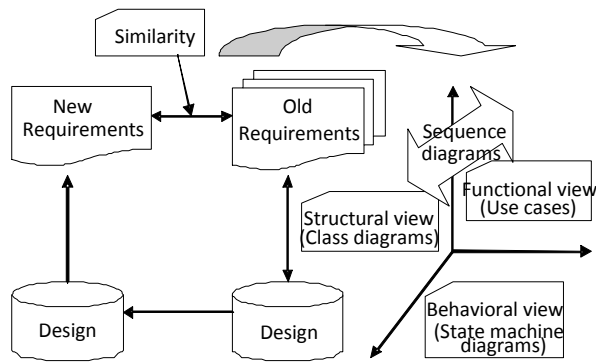


Figure 1. Various UML views in the context of similarity measurement.

In the sequel, we give a literature survey to identify the technology gap and corresponding research questions.

III. RELATED WORK

Developing with reuse consists of the following activities [15]: locating reusable artifacts (retrieval), assessing their relevance to current needs (assessment), and adapting them to those needs (adaptation). Locating reusable artifacts often involves some form of comparison of a query with candidate models in the repository. In every search, a *search space*, a *search goal*, and a *comparison function* are always defined. In software retrieval, the search space is known as the *software repository*. The search goal is called the *query*. The comparison function is called a *similarity metric*. How well the retrieved artifacts match the query depends on the soundness of the *similarity metric*.

The following subsection presents a literature review on reuse environments and similarity metrics.

A. Reuse Environments

Braga et al. [9] present the odyssey environment to support reuse-based software development environment based on component based software development. Odyssey is a framework where conceptual models, architectural models and implementation models are specified for pre-selected application domains. Similarly, eColabra [14] is a reuse environment that takes advantage of highly precise information retrieval techniques and graph visualization techniques to customize reuse during the classification and retrieval stages. Furthermore, eColabra applies information retrieval techniques to object oriented resources while exploiting the object oriented languages semantics and characteristics.

Correa et al. [18] present an approach to object oriented design reuse by integrating design patterns and anti-patterns into an object oriented design workbench. This allows the reuse of knowledge about good and bad object oriented design practices. Furthermore, a tool (OOPDTool) was developed to support the approach.

Cybulski et al. [11] combine keyword-based and faceted classifications of requirements and design. The keywords are extracted from the body of requirements text and are

then translated into design terms of a faceted classification. Facets are subsequently used to determine affinity between requirements and design artifacts, which are used for reuse based refinement of requirements documents. Beyer et al. [19] present a success story in establishing an architecture-centric approach at a small development organization. They evolved the development organization towards systematic reuse by introducing an architecture-centric strategy for product development.

Recently, Martins et al. [20] have applied data mining techniques improve the search of reusable assets. They have applied association rules and clustering techniques to aid the knowledge extraction. They used the concept of log files to extract a historic pattern and facilitate the overall search process.

COTS-aware requirement engineering (CARE) [35] approach for component identification has the focus on the utility of knowledge base. The goals and requirements are specified as enterprise goals which are further sub specialized into component goals. CARE points out the importance to keep requirements flexible as they have to be constrained by the capabilities of available components. In this approach, requirements are classified as native requirements acquired from customers and foreign requirements of the COTS components. The method puts emphases on narrowing the gap between customer and component requirements by using knowledge base. The process model describes the activities performed to define the system agents, goals, system requirements, software requirements and architecture. The product model describes the format of the product created using the process. The meta-model describes the knowledge content and structure for the CARE approach. The method highlights the importance of mapping system requirements and product specification; however, it does not support the possible mismatch between both specifications.

The COTS usage risk evaluation (CURE) [36] is a ‘front-end’ analysis tool that predicts the areas where the use of COTS products will have the greatest impact on the program. CURE is designed to find risks relating to the use of COTS products and report those risks back to the organization. Ideally, CURE is performed on both the acquiring and the contracting organization, but this is not necessary. The evaluation consists of four activities: preliminary data gathering, on-site interview, analysis of data and presentation of results. The CURE method has proven to be a useful tool for organizations that acquire or develop COTS-based systems. However, there are several limitations of the current version of CURE ranging from considerable amount of manual analytical work performed by evaluators and training required by the evaluators.

B. Similarity Metrics

Retrieval is one of the activities in a *software reuse process*, which takes in a *query* as input and returns *reusable artifacts* (or objects of reuse) as output. Because the goal of software retrieval is to return most similar reusable software artifacts, we propose considering a

framework for multi-view similarity assessment as discussed above.

In the sequel, we discuss related works in a chronological order starting with the latest. We conclude with a summary table (TABLE I) focusing on five aspects namely *artifacts considered* for reuse, *artifact internal representation*, *criteria for matching* artifacts (syntactic vs. semantic), use of extra-artifact *annotations* to guide matching, and the *search algorithm* used.

Ahmed [6] proposed a similarity metric to measure the similarity of UML models from the functional view using sequence diagrams. He used heuristic search techniques such as Genetic and Greedy Algorithms to assess similarities. His work did not consider consistencies with other views though.

Rufai [2] proposed a set of structural similarity metrics to measure the similarity in structural view for UML models. Different metrics capture the different structural aspects of the UML model.

William Robinson and Woo [25][26] present an automatic technique that provides assistance for use cases reuse. Given an initial description of the use case by software analyst, an automated graph based relational learner retrieves a set of similar use cases from a database. Their work uses automated graph based relational learner called SUBDUE. It represents an interaction diagram (sequence diagram) which provides a semantic structure that includes objects and methods of a use case as labeled graph that consists of vertices and edges. It uses a structure similarity measure to determine the distance between query structure and the structures available in repository. The technique does not make use of extra annotations and can be incorporated into tools like rational rose. In their other work, the authors have developed a CASE tool called REUSER which uses SUBDUE algorithm to automatically retrieve related UML sequence diagrams for reuse.

Saeki [27] has made an investigation into which parts of a use case description can be catalogued as reusable patterns and template for requirement analysis process. He listed the following parts as candidates for reuse:

- Use case templates for describing use cases.
- Use case patterns for providing the reusable and changeable structures for use cases.
- Use case frameworks that are large scale combinations of use case patterns for an application domain.
- Aspect patterns for wearing non-functional requirements with functional requirements.

Alspaugh et al. [28] have provided an approach to scenario management and evolution. They defined scenarios as a sequence of events with associated attributes. They defined a similarity measure as the sum of the number of common attribute values in each attribute list, divided by the sum of sizes of each attribute list. A variation was also proposed where attributes are assigned weights. Annotations are used to guide matching. The authors have not taken into account the relation between the attributes that might arise from semantic structures.

TABLE I. EARLY SOFTWARE ARTIFACT REUSE EFFORT

Work	Artifact Support	Internal Rep.	Matching Criteria	Extra Annotation	Search Algorithm
[6]	Sequence Diagrams	Message flow	Both Syntactic and Semantic Similarity (Macro and Micro Levels)	No	Depth First B&B Genetic Algorithm
[25] [26]	Use-Cases, Sequence Diagrams, Class Diagrams	Conceptual Graphs	Semantic Similarity or Structural relationship similarity (graph matching)	No	Subdue Algorithm
[2]	Class Diagrams	Class Vectors	Syntactic	No	Greedy & Hungarian Algorithm
[27]	Use Cases	Use Case Templates, Use Case Patterns, Use Case Framework, Aspect Patterns	Analogy-based matching	No	N/A
[28]	Scenarios	Attributes or facets constituting the use case scenario (Actor, go, purpose etc)	Syntactic similarity between scenarios	Yes	N/A
[3]	Use Cases, Sequence diagrams	Event flow vectors	Both Syntactic and Semantic Similarity	Yes	N/A

Bloch and Cybulski [3] have considered event flows in use-cases in measuring the use-case similarity. The authors represented the event flows present in the use-cases as follows. They classified the various events that are part of a particular domain model into a set of clusters. For a new use case they associate its events to these clusters based on the lexical description of the event. The entire event flow in the use case is represented as a vector where each dimension of the vector represents the number of events in a particular cluster.

Ryan and Mathews [29] have facilitated the reuse of previously developed requirement specifications for same or similar domains by identifying and encoding the types of knowledge used during requirement acquisition. They developed a tool (ReqColl) to aid in the process of requirement collection using conceptual graphs with semantic relationships. They used a conceptual graph matching algorithm to compare requirements expressed as conceptual graphs. The similarity is assessed based on matching nodes and arcs inside the conceptual graphs.

Reubenstein and Waters [30] developed a tool, Requirement Apprentice (RA), to assist users in

documenting consistent and complete requirements. The tool maintains a cliché library and uses it to retrieve similar requirements based on hybrid reasoning system. The cliché library holds bulk of the general information related to the domain. This allows users to document only specific requirement information; remaining general information is completed from the library.

In conclusion, even though the UML has become more or less the *de facto* standard modeling language for representing analysis as well as design artifacts, researchers have done little in proposing a similarity metric for UML models. The above discussed works consider comparison of artifacts taking only a single view into consideration. However, as mentioned earlier, a single view cannot comprehensively capture the software requirements and considering only one view while assessing similarity for reuse will not be as effective approach. Accordingly, for more effective reuse of software artifacts the similarity must be assessed considering all views.

Effective IREs should be built on top of strong foundations for a multi-view similarity metric, i.e., a similarity metric that considers multiple views of software in its similarity assessment.

IV. RESEARCH QUESTIONS

The problem of multi-view similarity assessment thus can be stated as to map classes of class diagrams, sequence diagrams, and state machine diagrams of the input model to particular counterparts in the repository such that the substitutions (e.g., class mappings from input model to repository model) have least conflicts. This problem is in essence a constraint satisfaction combinatorial optimization problem in a possibly large, but finite, space [6]. Accordingly, finding optimal substitution for maximal similarity of UML artifacts represents a NP-hard problem. The applicability of search heuristic algorithms, to include but not limited to Genetic Algorithms, Tabu Search, and Simulated Annealing, should be investigated [31][32][33][34]. Performance comparison against exact exhaustive search techniques, e.g., Depth-first Branch and Bound, should also be considered.

Similarity assessments and corresponding measurements should be used in ranking repository projects models according to their similarities to a current project models.

Cornerstone to the similarity assessment is the multi-view similarity metrics. Effective metrics, according to some measures such as *precision* and *recall*, should be developed.

A major focus of the IRE should be to offer tools to facilitate reusing design and later artifacts based on matching requirements. However, the IRE should also offer ad-hoc semantic-based UML artifacts repository search. Metadata and ontology along with indexing and storing schemes to allow for time-wise efficient retrieval of previous artifacts from the repository should also be developed. Standard representation in line with standards such as the *resource description framework schema* (RDFS) should be investigated for repressing the metadata and the

ontology [17]. Text/data mining algorithms should be researched and developed to support such search activities.

For maximal utilization within the day-to-day activities, best ways for integration with CASE tools should be researched. The market is glutted with UML modeling tools such as Rational Rose, Enterprise Architect, Together J, Visio, Microsoft Visual Modeler, Advanced Tech GD-Pro, Visual UML, Object Domain, Object Team, etc. The standard interchange format adopted for the UML is the XML Metadata Interchange (XMI) format [37]. Using XMI in representing UML artifacts in the repository should be investigated.

It is also important to maintain the quality of the artifacts maintained in the repository; and in the same line offer recommendations to re-users with regard to such artifacts. In order to do so, the IRE should facilitate the collection of reviews from re-users with respected to re-used/inspected artifacts. Algorithms should be developed to allow synthesizing recommendations with regard to existing artifacts based on a diversity of reviews as well as some other statistics.

Last but not least, intelligent user interface for easy artifacts management and reuse within the context of the day-to-day development activities should be developed. It should be intelligent in the sense that it can offer suggestions with regard to the modeling task at hand.

V. CONCLUSION AND FUTURE WORK

This paper presented an approach towards the development of integrated UML reuse environment. The paper surveys current state of the art and identifies gaps along with corresponding research questions for future work.

The author has been supervising a group of graduate students in an effort that aims at laying the foundations for multi-view similarity metrics and assessment along with the development of IREs proof of concept through a set of case studies. The effort has started earlier [2][6], was suspended for some time, and got resumed recently as a focus for future work. The successful realization of such IREs to facilitate early-stage reuse of UML artifacts is expected to improve the software quality and developers productivity.

ACKNOWLEDGMENT

The author wish to acknowledge King Fahd University of Petroleum and Minerals (KFUPM) for the use of various facilities in carrying out this research.

The author also thanks Jarallah AlGhamdi, Raheem Rufai, Awes Ahmed, and Sajjad Mahmoud for many helpful discussions, comments, and different contributions to the origin of this work.

REFERENCES

- [1] I. Sommerville, *Software Engineering*, 9th ed.: Addison-Wesley, 2010.
- [2] R. Rufai, "New Structural Similarity Metrics for the UML," MS Thesis, King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia, 2003.

- [3] M. C. Blok and J. L. Cybulski, "Reusing UML Specifications in a Constrained Application Domain," in *Proceedings 5th Asia Pacific Software Engineering Conference (ASPEC'98)*, 1998, pp. 196-202.
- [4] V. De Mey and O. Nierstrasz, "The ITHACA Application Development Environment. Visual Objects (ed. D. Tsichritzis," Centre Universitaire d'Informatique, University of Geneva 1993.
- [5] D. C. Rine, "Success factors for software reuse that are applicable across domains and businesses.," in *Proceedings of the ACM Symposium on Applied Computing*, 1997, pp. 182-186.
- [6] A. Ahmed, "Functional Similarity Metric for UML Models," MS Thesis King Fahd University of Petroleum & Minerals, 2006.
- [7] V. Sugumaran, *et al.*, "Software product line engineering," *Communications of the ACM*, vol. 49, pp. 29-32, 2006.
- [8] G. Booch, *et al.*, *Object-Oriented Analysis and Design with Applications*: Addison-Wesley, 2007.
- [9] R. M. M. Braga, *et al.*, "Odyssey: A Reuse Environment Based on Domain Models," in *Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology*, ASSET '99., 1999.
- [10] J. L. Cybulski, "Introduction to Software Reuse," Technical Report TR96/4, University of Melbourne, Melbourne, Australia 1996.
- [11] J. L. Cybulski and K. Reed, "Requirements Classification and Reuse: Crossing Domain Boundaries," in *Proceedings of the 6th International Conference on Software Reuse: Advances in Software Reusability*, London, UK, 2000.
- [12] R. K. Keller, *et al.*, "Design and Implementation of a UML-Based Design Repository.," in *Proceedings 13th International Conference on Advanced Information Systems Engineering (CAiSE2001)*, Interlaken, Switzerland, 2001.
- [13] H.-Y. Lee and M. T. Harandi, "An Analogy-Based Retrieval Mechanism for Software Design Reuse.," in *Proceedings of the 8th Knowledge-Based Software Engineering Conference (KBSE'93)*, Chicago, 1993, pp. 152-159.
- [14] O. Edelstein, *et al.*, "eColabra: An Enterprise Collaboration & Reuse Environment," in *Proc. Fourth Int'l Workshop (NGITS '99)*, 1999, pp. 229-236.
- [15] H. Mili, *et al.*, "Reusing Software: Issues and Research Directions," *IEEE Transactions on Software Engineering*, vol. 21, 1995.
- [16] I. Jacobson, *et al.*, *Object-oriented software engineering: a use case driven approach*: Addison-Wesley, 1992.
- [17] W3C, "RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004, <http://www.w3.org/TR/2004/REC-rdf-schema-20040210> last accessed Jan. 2011.," ed, 2004.
- [18] A. L. Correa, *et al.*, "Object Oriented Design Expertise Reuse: An Approach Based on Heuristics, Design Patterns and Anti-Patterns," in *6th International Conference on Software Reuse*, 2000, pp. 33 - 191.
- [19] H.-J. Beyer, *et al.*, "Introducing Architecture-Centric Reuse into a Small Development Organization," in *Proceedings of the 10th international conference on Software Reuse: High Confidence Software Reuse in Large Systems* Berlin, Heidelberg, 2008.
- [20] A. C. Martins, *et al.*, "Suggesting Software Components for Reuse in Search Engines Using Discovered Knowledge Techniques," in *SEAA '09 Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications 2009*, pp. 412-419.
- [21] J. Rumbaugh, *et al.*, *Object-Oriented Modeling and Design*, First ed.: Prentice Hall, 1991.
- [22] J. Iivari, "Object-orientation as structural, functional and behavioural modelling: a comparison of six methods for object-oriented analysis," *Information and Software Technology*, vol. 37, pp. 155-163, 1995.
- [23] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*: John Wiley & Sons, 2000.
- [24] J. Mylopoulos, "Multiple Viewpoints Analysis of Software Specification Process," *IEEE Transactions on Software Engineering*, 1995.
- [25] W. N. Robinson and H. G. Woo, "Finding Reusable UML Sequence Diagrams Automatically," *IEEE Software*, vol. 21, pp. 60-67, 2004.
- [26] H. G. Woo and W. N. Robinson, "Reuse of Specifications Using an Automated Relational Learner: A Lightweight Approach," in *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, 2002, pp. 173-180.
- [27] M. Saeki, "Patterns and Aspects for Use Cases: Re-use Techniques for Use Case Descriptions," in *Proceedings of the 4th International Conference on Requirements Engineering (ICRE'00)*, 2000.
- [28] T. A. Alspaugh, *et al.*, "An Integrated Scenario Management Strategy," in *Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, 1999.
- [29] K. Ryan and B. Mathews, "Matching Conceptual Graphs as an aid to Requirements Re-use," *Proceedings of IEEE International Symposium on Requirements Engineering*, pp. 112-120 1993.
- [30] H. B. Reubenstein and R. C. Waters, "The Requirements Apprentice: Automated Assistance for Requirements Acquisition," *IEEE Transactions on Software Engineering*, vol. 17, 1991.
- [31] R. Poli, *et al.*, "Particle swarm optimization: An overview," *Swarm Intelligence*, vol. 1, pp. 33-57, 2007.
- [32] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed.: Prentice Hall, 2010.
- [33] S. Sait and H. Youssef, *Iterative Computer Algorithms with Applications in Engineering*: IEEE Computer Society, 1999.
- [34] D. Teodorovic, *et al.*, "Bee Colony Optimization: Principles and Applications," in *8th Seminar on Neural Network Applications in Electrical Eng.*, NEUREL-2006, IEEE CNF, 2007.
- [35] L. Chung and K. Cooper, "Knowledge based COTS aware requirements engineering approach," in *14th Int. Conf. Software Eng. Knowledge Eng*, 2002, pp. 175 - 182.
- [36] D. J. Carney, *et al.*, "Identifying Commercial off the Shelf Product Risks: The COTS Usage Risk Evaluation," *Carnegie Mellon Software Engineering Institute (SEI)* 2003.
- [37] Object Management Group (OMG). *OMG XML Metadata Interchange (XMI) Specification (V. 2.11/Beta 2.4)*. Jan. 2002. <http://www.omg.org/spec/XMI/> (Accessed 2011-05-28).
- [38] B. Shneiderman and C. Plaisant, *Designing the User Interface*, Pearson Education, Inc., 2005.
- [39] Object Management Group (OMG). *OMG Unified Modeling Language (OMG UML), Superstructure (V. 2.3)*. May, 2010. <http://www.omg.org/spec/UML/2.3/Superstructure/PDF/> (Accessed 2011-09-23).

An Automated Translation of UML Class Diagrams into a Formal Specification to Detect UML Inconsistencies

Khadija El Miloudi Younès El Amrani Aziz Ettouhami

Laboratoire Conception et Systèmes FSR
University MohamedV-Agdal
BP 1014 RP Rabat. Morocco
{elmiloudi,elamrani,touhami}@fsr.ac.ma

Abstract— In view of the informal semantic of UML, there is a high risk of introducing ambiguities and contradictions in the modelled software. A considerable amount of literature has been published on UML inconsistencies. These studies have demonstrated the absence of any rule in UML to prevent such inconsistencies from being introduced in UML designs. This article describes a systematic translation of UML Class Diagrams into a formal specification to uncover most of the UML inconsistencies published to date. Examples of inconsistent UML class diagrams presented in previous research studies were used to validate the approach. The formal model obtained from UML class diagrams helped to uncover inconsistencies without any further proof. In order to relieve the user from writing a much rigorous and precise formalism, a tool that automatically generates the formal model from the UML class diagram was developed.

Keywords- Z; UML; UML inconsistencies; Formal Specification; Software Model Checking.

I. INTRODUCTION

There are numerous off-the-shelf software proposing to automatically translate UML Class Diagrams into several implementations. Nonetheless, there are very few translations into a formal notation to detect UML inconsistencies. Therefore, UML inconsistencies are insidiously injected into any generated implementation, when not removed. According to a definition provided in [7], inconsistency “denotes any situation in which a set of descriptions does not obey some relationship that should hold between them”. This paper will use this definition to identify most of contradictions in UML class diagram using Z notation [1]. The Z notation was chosen for the various benefits that it offers. Hall [5][6] identifies several advantages of formal methods and concluded that they “contributes to demonstrably cost-effective development of software with very low defect rates”. This study makes use of Anthony Hall’s model [2][4] of specification and interpretation of class hierarchies to express UML [2] class diagram in Z [1]. The obtained model uncovers inconsistencies of a given UML class diagram. We selected Anthony Hall’s model because it is referenced by most of works published to date and it models all needed concepts for the inconsistencies studied, namely: class hierarchy, multiplicity and association between classes. A prototype was devised to automatically generate formal specifications

based on Anthony Hall’s model [2][4]. All presented examples were automatically generated then type-checked with Z/EVES [13].

The structure of the rest of this paper is as follows. In Section 2, we present the related work. Section 3 provides a summary of Z notation [1] used in this paper. Section 4 summarizes Anthony Hall’s model [2][4]. Section 5 illustrates how UML [2] inconsistencies identified in published previous studies are uncovered in Z [1]. Section 6 draws some conclusions and future works.

II. RELATED WORK

There are several researches that are closely related to our work. A method for the automatic detection of the contradictions in UML Class Diagram has been introduced in [10]. Two kinds of inconsistencies were detected: contradictory multiplicities and the disjoint constraint violation. A semantic of UML in terms of first order logic was used to translate the class diagram into a program in logic. Our work inspires by this approach and chooses to formalize all the UML class model into the Z notation, both contradictions studied in [10] trivially surfaced. The strength of our approach takes root into the simplicity and elegance of Anthony Hall’s class hierarchies model. Also, the use of the Z notation made it possible to foster the Z/EVES [13] system for future investigations of the UML design robustness and to automatically process the model.

In [8], a definition of a production system language and rules specific to UML software designs is proposed. The system aims at detecting inconsistencies, notifying users and automatically fixing the inconsistency during the design process. The production system uses the Jess rule Engine [15]. In our approach, we use the Z notation [1] based upon set theory and mathematical logic. In the generated model, an inconsistency appears as two inconsistent predicates as it will be illustrated in Section 5. Our approach provides more visibility on the generated predicates, which enables further investigations on the software correctness. In the same context, the RoZ tool [11] has been developed to automatically generate formal specifications from UML class diagram. The UML design is completed by annotations in Z. However, this tool is different from ours, on the one hand, RoZ does not tackle inconsistency detection in UML class diagram. On the other hand, this tool requires the

designers to annotate in Z the UML class diagram to proceed with the generation of specifications, hence, the tool RoZ [11] requires Z specification at the UML stage, whereas in our method, we generate a complete translation of UML class diagram without any preliminary Z annotations. In our approach the separation of UML and Z allows to work on UML designs provided by software engineers without Z knowledge. When the model is generated, it offers the choice between an automatic processing to detect inconsistencies or a human static checking by a Z literate for further investigations.

In [16] and [17], a formal representation of UML models is proposed. The formal specification obtained is used to express and check some properties, called conjectures, on the model. Whereas in our approach we check the structural inconsistencies of a UML Class Diagram in general, focusing on generalization and multiplicities.

III. SUMMARY OF Z NOTATION

Z [1] is a formal specification language created by J.R. Abrial based upon set theory and mathematical logic. In Z notation, a specification uses the notion of schema to structure the underlying mathematics and allow an easy reuse of its subparts. According to [12], a schema is a “structure describing some variables whose values are constrained in some way”. A schema consists of two parts: the declaration part which contains the declaration of state variables and the predicate part which consists in a set of predicates constraining the variable state values. These predicates express properties on the state variables and introduce relationships between them. The name of the Z schema enables its re-use. A Z schema may be used or re-used as a declaration, a type or a predicate. When the specification requires a composite type, a schema is used to denote it. For example, the following schema denotes the type Rider, which is composed of four state variables with their types.



At an early stage of the specifications, the new types are introduced as given sets. New introduced types serve as basic types in the specification. A given set is introduced between square brackets. For example, to introduce a given set named OBJECT, we write:

[OBJECT]

The symbol \mathcal{P} is used to denote all subsets of a set. For example, to denote RIDER a subset of the set OBJECT we write:

$$RIDER: \mathcal{P} OBJECT$$

Several subsets can be defined at once, for example the following declaration

$$MAN, WOMAN: RIDER$$

introduces two subsets of RIDER. To denote that the two sets are disjoint we write

$$MAN \cap WOMAN = \emptyset$$

It could be abbreviated to:

$$disjoint \langle MAN, WOMAN \rangle$$

To denote a partial function named *idRider* from RIDER to Rider we write:

$$idRider: RIDER \rightarrow Rider$$

dom *idRider* denotes the domain of the partial function *idRider*, and ran *idRider* denotes its range. We can also define a function by set comprehension, example:

$$idRider = \{ rider : Rider \cdot rider.self \mapsto rider \}$$

The function *idRider* is the set of all mappings *rider.self* \mapsto *rider*.

We can also define a function using lambda notation which is: (λ declaration | constraint \cdot result)

For example, the relation *f* on the set of natural numbers \mathbb{N} associates to each natural number *m*, the unique number $2 \cdot m + 1$ as follow:

$$\left| \begin{array}{l} f: \mathbb{N} \leftrightarrow \mathbb{N} \\ \hline f = (\lambda m: \mathbb{N} \cdot 2 \cdot m + 1) \end{array} \right.$$

IV. UML CONSTRUCTS IN Z

In order to detect inconsistencies, a formal translation of UML constructs is used. The Z notation is the selected formal notation. The translation must meet a published model widely referenced. The model used is Anthony Hall’s [2][4]. The most needed constructs are the class construct and the generalization relationship. The model is presented through the example of the riding school from [4] illustrated in Figure 1.

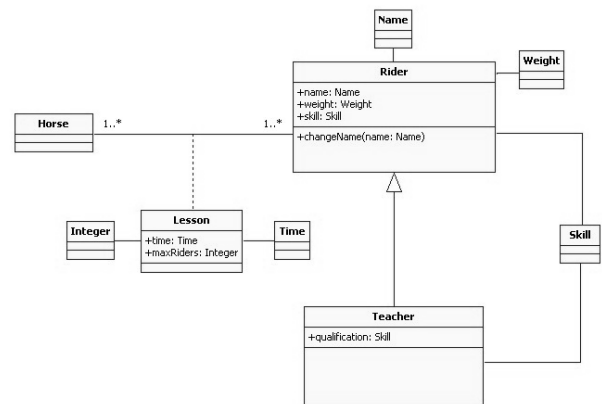


Figure 1. A Riding School UML Class Diagram

In Object-oriented modeling, a class describes the state and behavior of the class objects. The objects of a class are also called the class instances. The set of all object identities in Anthony Hall’s model is introduced as the given set [OBJECT]. To model the set of all classes we introduce the given set [CLASS]. A class is a particular member of CLASS. In the example of the riding school, the UML rider class is translated into a schema containing the attributes and their types. An attribute *self* represents the identifier of the current instance. The name of the schema in Z is the concatenation of name of the class with the text ‘CoreClass’. Then a free type, with the same name as the class, is defined. It adds an optional nil value to be used in initializations.

In our example, a schema called *SRider* represents all instances of the class. The state variable *riders* represents the set of the riders identified by the system. The state variable *ridersIds* is the set of their identities. A function *idRider* binds each unique instance identifier to the corresponding rider.

| RIDER: P OBJECT

RiderCoreClass

self: RIDER
name: Name
weight: Weight
skill: Skill

Rider ::= nilRider | ridercoreclass rider «RiderCoreClass»

SRider

riders: P Rider
idRider: RIDER → Rider
riderIds: P RIDER

idRider
 = { *ridercoreclass*: RiderCoreClass
 • *ridercoreclass*.*self* → *ridercoreclass* *rider*
ridercoreclass }
riderIds = dom *idRider*

An initialization schema is generated for each class to indicate the initial value of each attribute. Two types of initialization are proposed: an initialization by default which allows assigning nil values defined above to all attributes and the second method is used to initialize the attributes with values provided by the user.

InitRiderCoreClassByDefault

RiderCoreClass'

name' = nilName
weight' = nilWeight
skill' = nilSkill

InitRiderCoreClassWithValues

RiderCoreClass'

name? : Name
weight? : Weight
skill? : Skill

name' = *name*?
weight' = *weight*?
skill' = *skill*?

The following example illustrates the way to formalize a method using the Z notation. Each method is translated into an operation schema. Each operation includes a schema that indicates whether the system state will be changed (RiderOp below) or remains unchanged (RiderGet below). This schema also guarantees us that the object identifier (*self*) remains unchanged.

Since the formal model is automatically generated from the UML Class Diagram, only the method signature is defined (ChangeNameRider below).

RiderOp

ΔRiderCoreClass

self' = *self*

RiderGet

∃RiderCoreClass

self' = *self*

changeNameRider

RiderOp

name? : Name

In Object-oriented programming, the setters/getters methods are often used. The setter method takes a new value as an input parameter to modify the private attribute. The getter method returns the value of the private attribute. In our tool, the getters/setters methods are automatically generated for each class.

setskillRider

RiderOp

skill? : Skill

$$\begin{array}{|l} \hline \mathbf{RsetskillRider}: Skill \rightarrow Rider \rightarrow Rider \\ \hline \mathbf{RsetskillRider} = \{ \textit{skill}: Skill \\ \bullet \textit{skill} \rightarrow \{ \textit{setskillRider} \mid \textit{skill}? = \textit{skill} \\ \bullet (\textit{ridercoreclasstorider} \theta \textit{RiderCoreClass} \\ \mapsto \textit{ridercoreclasstorider} \theta \textit{RiderCoreClass}') \} \} \end{array}$$

$$\begin{array}{|l} \hline \mathbf{setskillRiderSystem} \\ \hline \mathbf{\Delta SRider} \\ \textit{rider}?: RIDER \\ \textit{skill}?: Skill \\ \hline \textit{idRider}' = \textit{idRider} \oplus (\{\textit{rider}?\} \triangleleft \textit{idRider} \wp \\ \mathbf{RsetskillRider} \textit{skill}?) \end{array}$$

$$\begin{array}{|l} \hline \mathbf{getskillRider} \\ \hline \mathbf{RiderGet} \\ \textit{skill}!: Skill \\ \hline \textit{skill}' = \textit{skill} \end{array}$$

The example below illustrates the transformation rule of the inheritance relationship between Teacher Class that inherits from Rider Class. The inheritance relationship between two classes is translated into Z by the inclusion of the schema of the super-class in the declaration part of the schema of subclass. In any inheritance relationship, the set of object identities of the subclass is a subset of the object identities of the super-class. To express this relationship, we define the schema called RiderTeacherHierarchy. The lambda function $(\lambda \textit{Teacher} \cdot \theta \textit{Rider})$ used in the predicate part of RiderTeacherHierarchy denotes the projection function from Teacher state to Rider state.

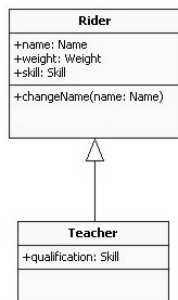


Figure 2. Example of Inheritance Relationship

$$\begin{array}{|l} \hline \mathbf{TeacherCoreClass} \\ \hline \mathbf{RiderCoreClass} \\ \textit{qualification}: Skill \\ \hline \textit{self} \in \textit{TEACHER} \end{array}$$

$$\begin{array}{|l} \hline \mathbf{Teacher} ::= \textit{nilTeacher} \\ \mid \textit{teachercoreclasstoteacher} \langle \textit{TeacherCoreClass} \rangle \end{array}$$

$$\begin{array}{|l} \hline \mathbf{STeacher} \\ \hline \textit{teachers}: \mathbb{P} \textit{Teacher} \\ \textit{idTeacher}: \textit{TEACHER} \rightarrow \textit{Teacher} \\ \textit{teacherIds}: \mathbb{P} \textit{TEACHER} \\ \hline \mathbf{idTeacher} \\ = \{ \textit{teachercoreclass}: \textit{TeacherCoreClass} \\ \bullet \textit{teachercoreclass.self} \\ \mapsto \textit{teachercoreclasstoteacher} \textit{teachercoreclass} \} \\ \textit{teacherIds} = \text{dom} \textit{idTeacher} \end{array}$$

$$\begin{array}{|l} \hline \mathbf{RiderTeacherHierarchy} \\ \hline \mathbf{SRider} \\ \mathbf{STeacher} \\ \hline \textit{teacherIds} = \textit{riderIds} \cap \textit{TEACHER} \\ \forall \textit{t}: \textit{teacherIds} \\ \bullet (\lambda \textit{TeacherCoreClass} \cdot \theta \textit{RiderCoreClass}) \\ (\textit{teachercoreclasstoteacher} \sim (\textit{idTeacher} \textit{t})) \\ = \textit{ridercoreclasstorider} \sim (\textit{idRider} \textit{t}) \end{array}$$

To get an overview of all classes of the system and relationships that bring them together, the schema System is introduced.

$$\begin{array}{|l} \hline \mathbf{System} \\ \hline \mathbf{SRider} \\ \mathbf{STeacher} \\ \mathbf{RiderTeacherHierarchy} \end{array}$$

This model is used in Section 5 to illustrate how common UML inconsistencies [9][10] are translated into inconsistent predicates.

V. UML INCONSISTENCIES IN Z

In this section, a formalization of UML inconsistencies is presented using Z Notation. This formalization is based on the model presented in Section 4. Each inconsistency is presented through an illustrative example previously published.

A. Generalization and Disjointness

In a UML Class Diagram, the disjoint constraint means that an instance of the super-type may not be a member of more than one sub-type, it is denoted in UML between brackets near the inheritance arrow, i.e., multiple inheritance of disjoint classes is forbidden. The example studied in the papers [9][10] and illustrated in Figure 3. shows a diagram where {disjoint} constraint is violated.

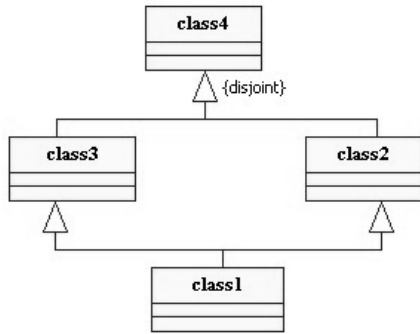


Figure 3. Inconsistent Class Diagram

The formalization of disjointness is given by the inclusion of a predicate which guarantees the disjointness of classes mentioned with the constraint {disjoint}.

```

System
Class1
Class3
Class2
Class4
class3class1Hierarchy
class2class1Hierarchy
class4class2Hierarchy
class4class3Hierarchy
disjoint < class2Ids , class3Ids
    
```

The predicate $disjoint\langle class2Ids, class3Ids \rangle$ is equivalent to the predicate:

$$class2Ids \cap class3Ids = \emptyset \quad (1)$$

The constraints

$$(class1Ids = class2Ids \cap CLASS1) \wedge (class1Ids = class3Ids \cap CLASS1) \quad (2)$$

are introduced in the schema *System* from *class2class1Hierarchy* and *class3class1Hierarchy*.

$$(1) \text{ And } (2) \text{ implies that } class1Ids = \emptyset \quad (3)$$

If *class1* is instantiated then $class1Ids \neq \emptyset$, hence the inconsistency.

To check the consistency of the specification, a disjointness theorem is generated when a disjoint property is reported on the UML model.

theorem disjointness
 $\exists Class1 \mid class1Ids \neq \emptyset \cdot System$

If the theorem cannot be proved, then the *System* is inconsistent.

B. Completeness and Disjointness

We found also in the UML class diagram the {complete} constraint.

The {complete} constraint means that each instance of the super-type must be a member of one of the sub-types. Here is an example from [9].

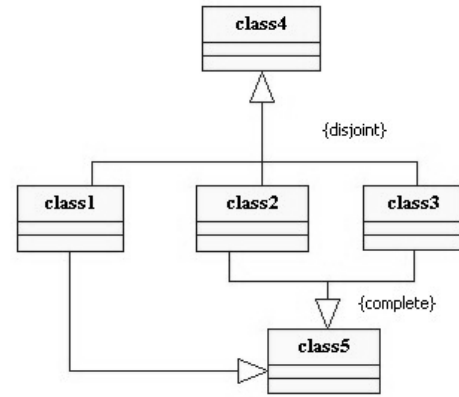


Figure 4. Inconsistent Class Diagram

The {complete} constraint used in the Figure 4. imposes on the *class5* to be specialized either as *class2* or *class3*. The completeness constraint is translated by the following predicate in the system schema:

$$class5Ids = class2Ids \cup class3Ids$$

This predicate expresses that all instances of *class5* belong either to *class2* or *class3*. The system obtained is:

```

System
Class1
Class2
Class3
Class4
Class5
class4class1Hierarchy
class4class2Hierarchy
class4class3Hierarchy
class5class1Hierarchy
class5class2Hierarchy
class5class3Hierarchy
disjoint < class1Ids, class2Ids, class3Ids >
class5Ids = class2Ids \cup class3Ids
    
```

On the one hand in the final schema *System*, the predicate $disjoint\langle class1Ids, class2Ids, class3Ids \rangle$ translates the disjoint constraint in UML and the predicate $class5Ids = class2Ids \cup class3Ids$ translates the complete constraint in UML.

On the other hand, the generalization between *class1* and *class5* is translated into the following predicates:
 $class1Ids = class5Ids \cap CLASS1$ from the schema *class5class1Hierarchy*.

where $class5Ids: \mathbb{P} CLASS5$ from the schema *Sclass5*. Therefore $class1Ids \subseteq CLASS5 \cap CLASS1$ implies that $class1Ids = \emptyset$.

If *class1* is instantiated, then $class1Ids \neq \emptyset$. Hence the inconsistency.

The following theorem is used to check the consistency of the schema *System* when using disjoint and complete constraints simultaneously.

theorem completeness

$$\exists Sclass1 \mid class1Ids \neq \emptyset \cdot System$$

In the same way, it detects when there is no such a *System*.

C. Multiplicities

Consider the following class diagram used in the article [9][10] representing a multiple inheritance:

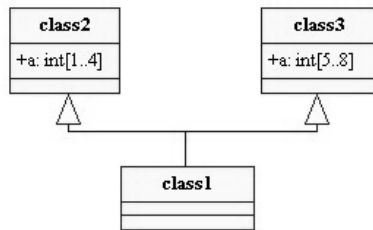


Figure 5. Inconsistent Class Diagram

In this example, we have three classes named *class1*, *class2* and *class3*. *Class1* inherits from both *class2* and *class3*. The multiplicity of an attribute indicates the number of values that attribute can contain. In the example, *class2* has an attribute with a multiplicity maximum of 4 and minimum of 1, *class3* has an attribute with the same name but different bounds: the multiplicity maximum is 8 and minimum is 5.

The following schema illustrates the multiplicity formalization:

<i>class2</i>
<i>self</i> : CLASS2
<i>a</i> : $\mathbb{P}Z$
$1 \leq \# a \leq 4$

<i>class3</i>
<i>self</i> : CLASS3
<i>a</i> : $\mathbb{P}Z$
$5 \leq \# a \leq 8$

We use Anthony Hall’s modelling [2][4] summarized in Section 4 to represent the inheritance relationship. We include the schema of the super-class in the declaration part of the schema of subclass. In this example, we have a multiple inheritance. *class1* is represented by the following schema:

<i>class1</i>
<i>class3</i>
<i>class2</i>
<i>self</i> \in CLASS1

In Z, the introduction of a schema S1 into another schema S2 introduces all the state variables and predicates of S1 into S2. In this example, the inconsistency is immediately detected in Z because *class1* inherit two attributes with the same name from two different super-classes *class2* and *class3*. The Z/EVES [13] immediately uncovers such a redundant declaration. Even if we keep only one declaration in the variable part of the schema, the two predicates remain inconsistent. It is worth saying here that the UML standard [2] is ambiguous in the case of multiple inheritance of the same attribute. Therefore, it is up to the designer to provide a semantic in that case.

VI. CONCLUSION AND FUTURE WORK

A. Concluding Remarks

This article illustrated most frequent UML inconsistencies published so far using Anthony Hall’s model [2][4] most of these are translated into contradictory predicates. In some ambiguous cases, UML must be supplemented by an additional formal semantic. Typically UML lacks a semantic for multiple inheritance of attributes with the same name. A prototype has been developed to automatically translate UML designs into their formal counterpart. The Z notation makes the formal translation of the design particularly suitable for further investigation in Z.

B. Future Work

There are two ways to build on this work. First we are developing an automated and interactive verifier of the inconsistencies using Z/EVES [13] meanwhile a formalization of the Object Constraint Language is prepared in order to translate UML Class Diagrams using OCL [14] into a more precise Z counterpart. The current prototype is completed to automatically generate formal specifications from UML Class diagrams annotated by OCL constraints.

REFERENCES

[1] J. M. Spivey: The Z Notation: A Reference Manual, Prentice Hall, Englewood Cliffs, NJ, Second Edition, 1992.
 [2] Object Management Group (OMG). Unified Modeling Language: Superstructure. Version 2.3, May 2010. <http://www.omg.org/spec/UML/2.3/Superstructure/PDF/>. Sept 11, 2011.

- [3] A. Hall, "Using Z as a Specification Calculus for Object-Oriented Systems". In Bjorn D. Langmaack H (eds), Proceedings of VDM 90, Lecture Notes in Computer Science No. 428, pp. 290 - 318. Springer Verlag, 1990.
- [4] A. Hall, "Specifying and Interpreting Class Hierarchies in Z", Z User Workshop, Cambridge 1994, ed. J. P. Bowen and J. A. Hall, Springer, 1994.
- [5] A. Hall, "Realising the Benefits of Formal Methods", Formal Methods and Software Engineering, LNCS 3785, Springer, pp. 1-4, 2005.
- [6] A. Hall, "Seven Myths of Formal Methods", IEEE Software, September 1990, pp. 11-19.
- [7] B. Nuseibeh, S. Easterbrook and A. Russo, "Leveraging Inconsistency in Software Development". IEEE Computer, vol. 33, pp. 24-29, April, 2000.
- [8] W. Liu, S. Easterbrook and J. Mylopoulos: Rule-based Detection of Inconsistency in UML Models; Proc. Workshop on Consistency Problems in UML-Based Software Development, pp. 106-123, 2002.
- [9] K. Kaneiwa and S. Satoh, "Consistency Checking Algorithms for Restricted UML Class Diagrams". In Proceedings of the Fourth International Symposium on Foundations of Information and Knowledge Systems, vol. 3861, pp.219-239, 2006.
- [10] K. Satoh, K. Kaneiwa and T. Uno, "Contradiction Finding and Minimal Recovery for UML Class Diagrams using Logic Programming". Proceeding of 21st IEEE International Conference on Automated Software Engineering (ASE'2006), pp. 277-280, 2006.
- [11] Yves Ledru. RoZ tool. 22 february 2000. <http://vasco.imag.fr/RoZ/index.html>. Sept 11, 2011.
- [12] Jim Woodcock and Jim Davies.: Using Z Specification, Refinement, and Proof. University of Oxford, 1995.
- [13] Irwin Meisels. Software Manual for Windows Z/EVES Version 2.3. ORA Canada Technical Report TR-97-5505-04h, June 2004.
- [14] Object Management Group (OMG). Object Constraint Language. Version 2.2, February 2010. <http://www.omg.org/spec/OCL/2.2/PDF/>. Sept 11, 2011.
- [15] Jess, the Rule Engine for the JavaTM Platform. <http://www.jessrules.com/>. Sept 11, 2011.
- [16] N. Amalio, F. Polack and S. Stepney. "UML + Z: UML augmented with Z". In Software Specification Methods: an Overview Using a Case Study. Marc Frappier and Henri Habrias, editor. Hermes Science Publishing. 2006.
- [17] N. Amalio, S. Stepney and F. Polack, "Formal Proof from UML Models". In et al, J. D.,ed., ICFEM 2004, volume 3308 of LNCS, pp. 418-433. Springer .2004.

UML 2.0 Profile for Structural and Behavioral Specification of SCA Architectures

Wided Ben Abid
MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
benabidwided@hotmail.com

Mohamed Graiet
MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
mohamed.graiet@imag.fr

Mourad Kmimech
MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
mkmimech@gmail.com

Mohamed Tahar Bhiri
MIRACL, ISIMS
BP 1030, Sfax 3018, TUNISIA
Tahar_bhiri@yahoo.fr

Walid Gaaloul
Computer Science Department Télécom SudParis
9, rue Charles Fourier 91 011 Évry Cedex, France
walid.gaaloul@it-sudparis.eu

Eric Cariou
Université de Pau et des pays de l'Adour
Avenue de l'Université BP 1155 64013
PAU CEDEX France
Eric.Cariou@univ-pau.fr

Abstract— Service Component Architecture (SCA) aims to simplify the construction of service oriented architecture (SOA) to encourage a better reuse and to be independent from used technologies. In the other hand, UML 2.0 is the de-facto standard for graphical notation and modelling in software engineering. To face this situation we recommend an adaptation of UML 2.0 to SCA. It is in this context that we have defined a profile UML 2.0 for SCA containing a set of stereotypes applied to metaclasses stemming from the metamodel UML 2.0. These stereotypes are completed by formal constraints in OCL. Our profile introduces new elements to reflect the architectural concepts of SCA.

Keywords—Software architecture, SCA, UML 2.0, OCL, Profile and Metamodel.

I. INTRODUCTION

Nowadays, software engineering aims to decrease the complexity of application development by reusing heterogeneous and distributed software components. Thanks to the Web technologies, to the SOA architecture (Service Oriented Architecture) [1] and the SCA Architecture (Service Component Architecture) [2], the opening of the company to the world is made possible. The use of the standard SCA as the model of specification of the service oriented components architectures produces concepts and notations which are not readable and easily understandable, especially in the industrial circles. Using a graphical model seems a way that could overcome this disadvantage.

The UML language being a modelling standard which supplies, on one hand readable graphic representations and on the other hand proposes diagrams to specify workflows, seems a relevant way to model SCA Architectures. To face this situation, we recommend an adaptation of UML 2.0 to the SCA. It is in this context that we defined a profile UML 2.0 of specification of the architectures SCA. Our profile UML 2.0-SCA is a set of stereotypes applied to metaclasses stemming from the UML 2.0 metamodel. The proposed stereotypes are endowed with the constraints

of use expressed formally in OCL [3]. Such a profile is defined to favor:

- Recovery (or reuse) of software architecture described in SCA from the academic world.
- Design and implementation of software systems having explicit and documented software architectures.
- The transformation of model according to the approach MDA [4] [5] [6]. For example, the transformation of a PIM (Platform Independent Model) described in this profile to another PIM or PSM (Platform Specific Model) described in UML 2.0 or using others profiles.

Then, we partially automate our proposed formalization methodology using an MDE (Model Driven Engineering) approach. For this, we will transform the metamodel of the proposed UML 2.0-SCA profile to SCA metamodel. These metamodels respectively play the role of source and target metamodels for the exogenous transformation of the profile UML2 to SCA. In addition, we implemented ProfilUML2SCA, a tool for this transformation using the MDE language ATL (ATLAS Transformation Language) [7].

This paper has four main sections besides an introduction and a conclusion. The first and second section will position our contribution with respect to different approaches of modelling software architectures and initializes an SCA metamodel to express in a semi-formal way the SCA concepts to be modeled in UML 2.0 to establish correspondences. The third section describes an extension of the proposed UML profile. In Section 4, we present our automatic MDE approach for Exogenous transformation from our profile to SCA application.

II. SOFTWARE ARCHITECTURE IN UML

UML is a modelling language which is generalist, semi-formal and widely used in the industrial world. However, several researchers [8] [9] studied the possibility of modelling software architecture by using UML. Two approaches corresponding to the standard UML are

proposed. The first strategy uses UML as it is, to represent the architectural concepts of the ADLs, such as component, connector, role, port and configuration. The major advantage of this approach is the understanding of this modelling by every user of UML. But this strategy has limitations on the inability of UML, especially UML1.x to translate architectural concepts explicitly. For this reason, we use a second approach which consists in defining profiles. UML can be adapted to every domain through the extensibility mechanisms offered by this language such as stereotypes, tagged values and constraints. These mechanisms offered by UML extend UML without changing the UML metamodel. The advantage using profiles consists in clarifying the representation of the architectural concepts. So, we define this profile based on the strategy of using extensibility mechanisms of UML 2.0 to constrain the UML metamodel in order to adapt to the architectural concepts of SCA.

III. METAMODELLING OF THE SCA ARCHITECTURE

A. Structural aspects of SCA

SCA provides a programming model for building applications and systems based on a SOA. The main idea behind SCA is to be able to build distributed applications, which are independent of implementation technology and protocol. SCA is the result of a collaborative project OSOA (Open Service Oriented Architecture) [10] which aims to provide a set of specifications including firstly a model for creating components and also a programming model for building software applications based on architecture services.

In this section, we introduce only the model for creating software components. SCA provides an assembly model representing a network of services and allows building the SCA components in different languages, while ensuring integration with existing models. The basic unit of deployment of an SCA application is composite. A composite is an assembly of heterogeneous components, which implement particular business functionality (see Figure 1 below).

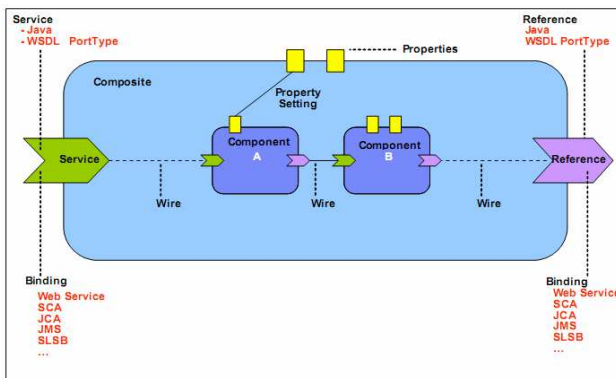


Figure 1. Diagram of an SCA composite [11]

A SCA composite is an assembly, which can contain components, services, references of services, declarations of properties allowing the configuration of its components, and

links specifying the connections between components. Independently of whatever technology is used, every component relies on a common set of abstractions including services, references, properties, and bindings.

A component is the basic entity for the construction of SCA application. This element has an implementation that must be either Java class or a BPEL process. Independently of the technology used for its implementation, the component is based on a common set of abstractions such as services, references and properties. Figure 2 shows an example of an SCA component:

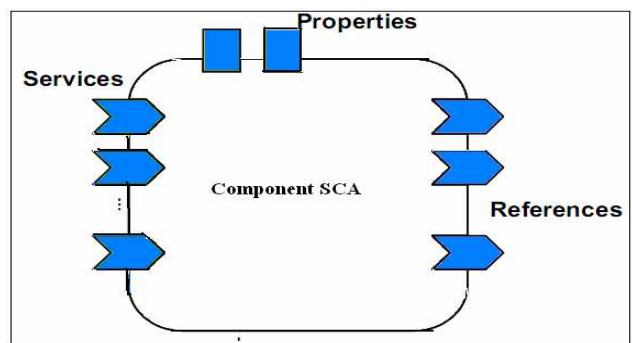


Figure 2. Example of SCA component

Each SCA component implements a business logic exposed by one or more services. A service describes what a component provides, i.e., its external interface. A reference specifies what a component needs from the other components or applications of the outside world. Services and references are matched and connected using wires or bindings. A component also defines one or more properties.

To provide distant communications between services, SCA offers the possibility of using a protocol described in the binding specified within the service and/or within the implementation, for example the protocols JMS (Java Message Service), RMI (Remote Method Invocation) or SOAP (Simple Object Access Protocol) to perform synchronous or asynchronous communications. A single service or reference can have multiple bindings, allowing different remote software to communicate with it in different ways.

B. Behavioural aspects of SCA

The web services technology is widely used as support of the interoperability between applications. In this context, the interactions between components of the SCA Architecture are made through its service interfaces. The communication is realized by means of message exchanges. A web service defines the functionality it provides and the required information that must be met to perform its function. The functionality of the web service can be implemented in any number of ways and languages such as XLANG [12], Web Services Flow Language(WSFL) [13] and Business Process Execution Language(BPEL) [14].

BPEL is a language of composition which is spirit to become a standard. This language describes a business process who specifies the execution order between a

numbers of constituent activities, the partners involved, the message exchanged between these partners and the fault and exception handling mechanisms, to achieve a commercial goal.

The main concept of BPEL is the BPEL process. It uses several concepts as Partner links, handlers, variables, correlation sets, and activities for the process logic. The atomic element of a process is an activity, which can be the “send of a message” (activity: reply), the “reception of a message” (activity: receive), the “call of an operation” (activity: invoke) or “manipulate data” (activity: assign). Structured Activities prescribe the order in which a collection of activities take place like “execute these structured activities prescribe the order in which a collection

of activities take place like “execute these activities sequentially” (activity: sequence), “repeat the execution of this activity” (activity: while) or “parallel execution of activities” (activity: flow).

In this section, we thus decided to elaborate a metamodel for SCA Architecture representing most of the concepts stemming from this specification. This metamodel allows, in our context, to express in a semi-formal way the concepts SCA both structural and comportmental to be modelled in UML 2.0. Our metamodel is built as an extension of the metamodel proposed by the community OASIS (Organization for the Advancement of Structured Information Standards). Our metamodel is illustrated in Figure 3.

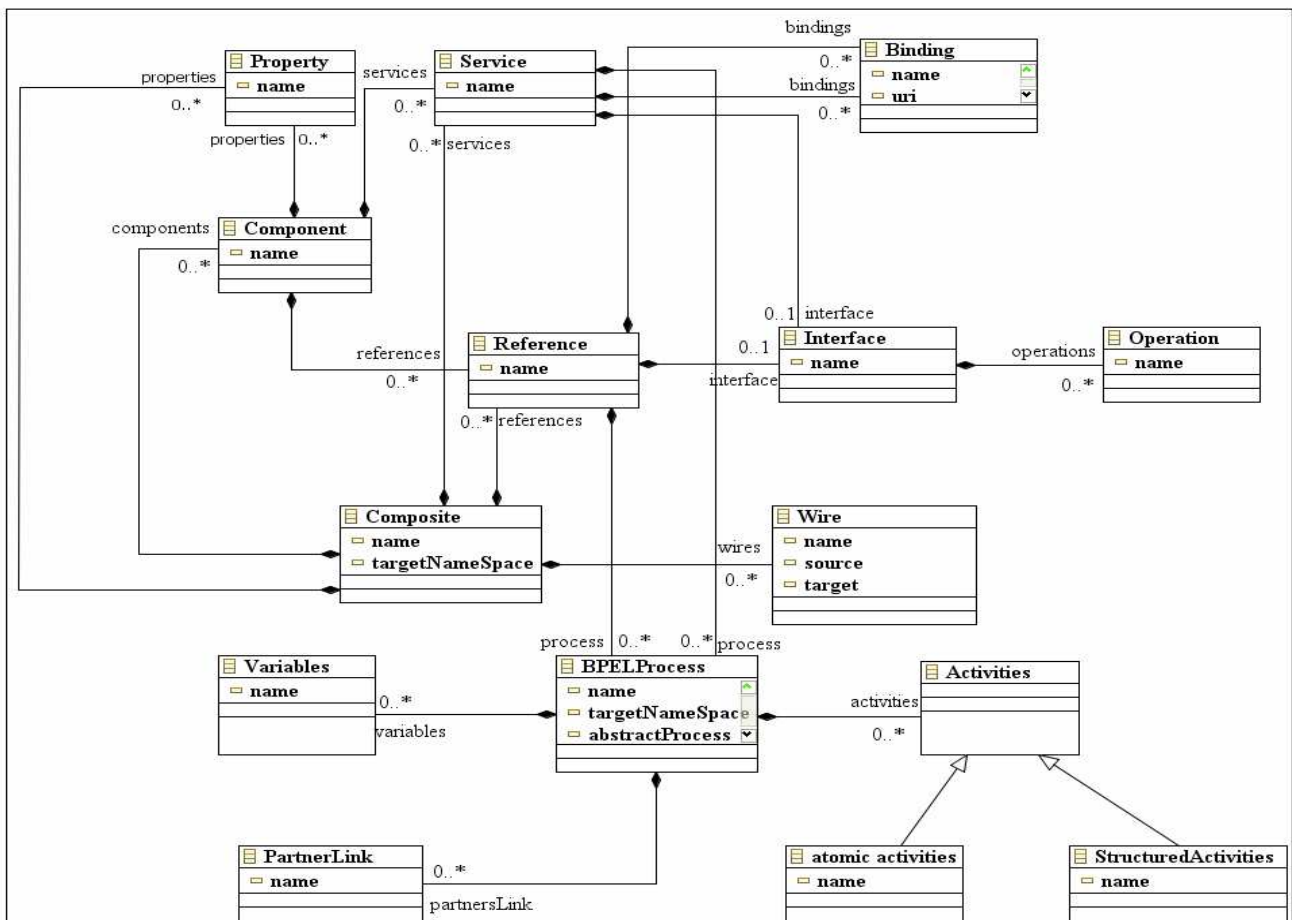


Figure 3. A metamodel of SCA

The behavioral aspect is represented in this metamodel by the BPEL process. While being a powerful language for implementing processes, BPEL is difficult to use. Its XML representation is very verbose and only readable for the trained eye. Several vendors offer a graphical interface that generates BPEL code. However, the graphical

representations are a direct reflection BPEL code and not easy to use by end-users. Therefore, we provide a mapping from UML to BPEL. In the following section, we are going to establish stereotypes to model respectively behavioral and structural concepts of the SCA Architecture.

IV. UML PROFILE FOR SPECIFYING SCA ARCHITECTURES

This part is dedicated to the technical definition of the profile SCA-UML. Such a profile contains a set of stereotypes applied to metaclasses UML 2.0 and defined by a set of constraints OCL. In UML 2.0, the state machines can be used to specify the behavior of several elements of the models described in UML 2.0, such as instances of a class UML 2.0. While the state machine description of protocols can be used with profit to express protocols related to scenarios of use of services offered by interfaces or ports (Figure 4).

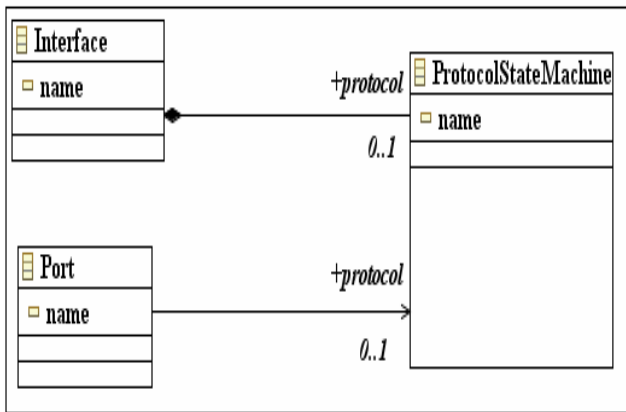


Figure 4. State machine description of protocols associated at the interfaces or ports

The concept of state machine UML2 .0 is used as a basis for stereotyping behavioral aspects of SCA or more precisely BPEL activities.

In the rest, we will establish stereotypes to model structural and comportemental aspects of SCA such as BPEL process, BPEL activities, component, ports services, ports references and connectors. We provided particular care to the development of formal constraints in OCL related to stereotypes. This gives a better idea for the context of use of these stereotypes.

A. SCA Components

An SCA component is described by an UML 2.0 component stereotyped by <<SCAComponent>> (Figure 5). The stereotype <<SCAComponent>> is defined by the following OCL constraints:

- No provided or required interface is associated with <<SCAComponent>>. **self.provided -> isEmpty () and self.required -> isEmpty ()**
- All ports associated with <<SCAComponent>> are <<SCAPortService>> or <<SCAPortReference>> and must be of type port. **self.ports -> forAll (p| p. stereotype = SCAPortService and p.SCAPortServiceType = #port) or (p| p. stereotype = SCAPortReference and p.SCAPortReferenceType = #port)**
- <<SCAComponent>> has at least one port.

self.ports -> size () >= 1 and (self.ports.oclAsType (service).stereotype = SCAPortService or self.ports.oclAsType (reference).stereotype = SCAPortReference)

- One and only one <<SCAProtocolStateMachine>> is associated with <<SCAComponent>>. **self.stateMachine -> size () = 1 and self.stateMachine.oclAsType (ProtocolStateMachine). Stereotype =SCAProtocolStateMachine)**

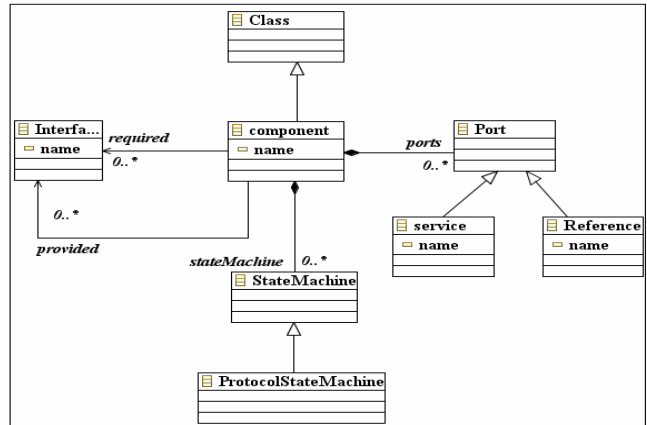


Figure 5. The Component metaclass in UML 2.0 metamodel

B. Services and references

A service from an SCA component provides a set of business functionality to other SCA components whereas a reference represents the services offered by other components. For it a SCA service is described by an UML 2.0 port (Figure 6) stereotyped by <<SCAPortService>>. A SCA reference is described by an UML 2.0 port (Figure 7) stereotyped by <<SCAPortReference>>.

A port is the element of a component used to interconnect components via connections between ports. A port realizes an interface of services.

The stereotype <<SCAPortService>> is defined by the following OCL constraints:

- All the offered interfaces associated in <<SCAPortService>> are SCAInterface. **self.provided -> forAll (i | i.stereotype = SCAInterface)**
- <<SCAPortService>> has at most one interface provided and no interface required. **self.provided -> size () <=1 and self.required-> isEmpty ()**
- One and only one <<SCAProtocolStateMachine>> is associated with <<SCAPortService>>. **self.protocol -> size () = 1 and self.protocol -> forAll (psm| psm.stereotype = SCAProtocolStateMachine)**

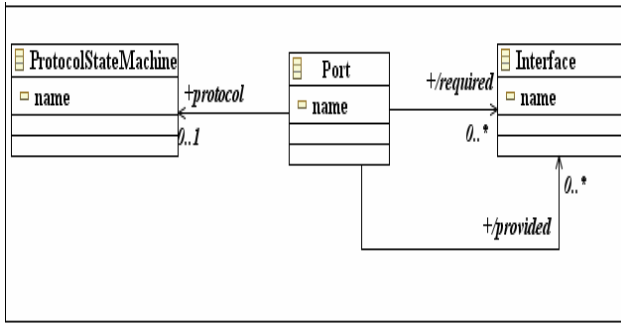


Figure 6. The metaclass Port in the metamodel UML 2.0

The following OCL constraints are defined for the stereotype <<SCAPortReference>>:

- All required interfaces associated with <<SCAPortReference>> are SCAInterface.
self.required -> forAll (i| i.stereotype = SCAInterface)
- <<SCAPortReference>> has at most a required interface and no interface provided.
self.required -> size () <=1 and self.provided-> isEmpty ()
- One and only one <<SCAProtocolStateMachine>> is associated with <<SCAPortReference>>
self.protocol -> size () = 1 and self.protocol -> forAll (psm| psm.stereotype = SCAProtocolStateMachine)

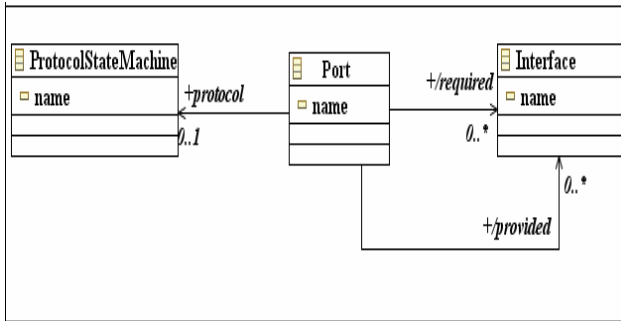


Figure 7. The metaclass Port in the metamodel UML 2.0

C. The interfaces of components

Every SCA interface (a port of a component) possesses one or several operations. An SCA interface is described by an UML 2.0 interface (Figure 8) stereotyped by <<SCAInterface>> for ports services and interfaces. This one is defined by the following OCL constraints:

- All the operations associated with SCAInterface are operations without parameter.
self.ownedOperation -> forAll (o|o.formalParameter -> isEmpty ())
- No attributes are associated with an SCAInterface.
self.ownedAttribute -> isEmpty ()
- Exactly one and only one <<SCAProtocolStateMachine>> is associated with each <<SCAInterface>>.

self.protocol -> size () = 1 and self.protocol -> forAll (psm| psm.stereotype = SCAProtocolStateMachine)

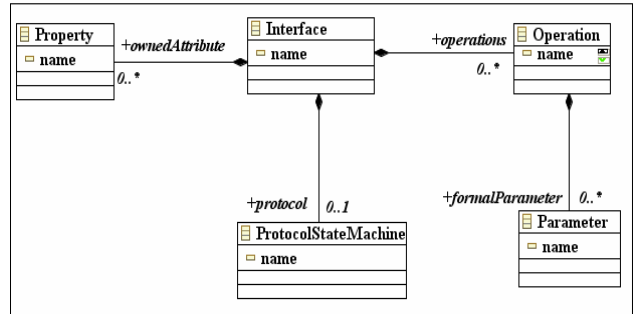


Figure 8. The metaclass Interface in the metamodel UML 2.0

D. BPEL Process

A BPEL process is represented as a protocol state machine describes the comportemental aspect of SCA with the stereotype <<SCAProtocolStateMachine>>. But the definition of the stereotype requires the introduction of other stereotypes such as <<SCAProtocolTransition>>, <<SCARegion>>and <<SCAVertex>> to express more formally the behavioral aspects.

1) <<SCAVertex>> stereotype

Each activity has a descriptive name and an entry action detailing the work performed by the activity. For these, an activity in BPEL can be represented by a state in diagram state machine (see Figure 9), stereotyped by <<SCAVertex>>. This stereotype is defined by the following OCL constraints:

- All transitions incoming <<SCAVertex>> must be SCAProtocolTransition.
self.incoming -> forAll (t | t.oclAsType (ProtocolTransition).stereotype SCAProtocolTransition)
- All outgoing transitions of <<SCAVertex>> must be SCAProtocol transition.
self.outgoing -> forAll (t | t.oclAsType (ProtocolTransition).stereotype = SCAProtocolTransition)

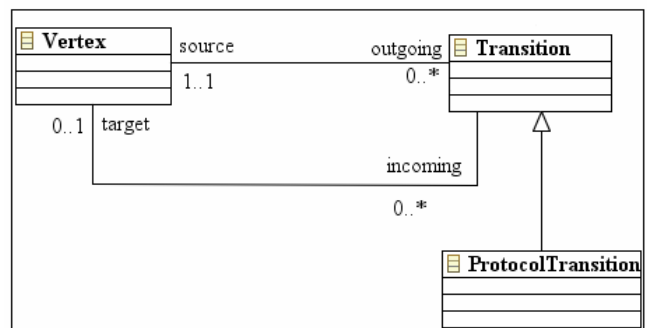


Figure 9. the metaclass Vertex in the metamodel UML 2.0

2) <<SCARegion>>

The stereotype <<SCARegion>> applied to the metaclass Region (see Figure 10) is defined by the following OCL constraints:

- All vertices belonging to <<SCARegion>> are SCAVertex.
self.subvertex -> forAll(s | s.stereotype = SCAVertex)
- All transitions belonging to SCARegion must be SCAProtocolTransition.
self.transitions -> forAll (t | t.oclAsType (ProtocolTransition).stereotype = SCAProtocolTransition)

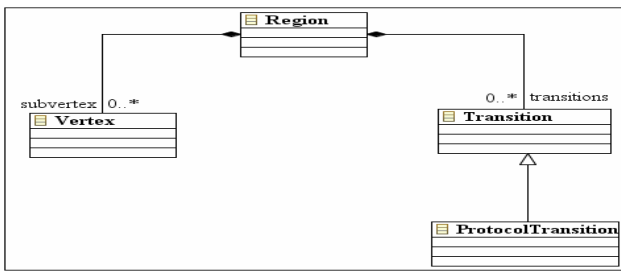


Figure 10. the metaclass Region in the metamodel UML 2.0

- All regions belonging to stereotype SCAProtocolStateMachine must be SCARegion.

self.oclAsType (ProtocolStateMachine).region-> ForAll(r | r.stereotype = SCARegion)

4) <<SCAProtocolStateMachine>> stereotype

The stereotype <<SCAProtocolStateMachine>> applied to the metaclass StateMachine (see Figure 11) is defined by the following OCL constraint:

- All regions belonging to stereotype SCAProtocolStateMachine must be SCARegion.
self.oclAsType (ProtocolStateMachine).region-> ForAll(r | r.stereotype = SCARegion)

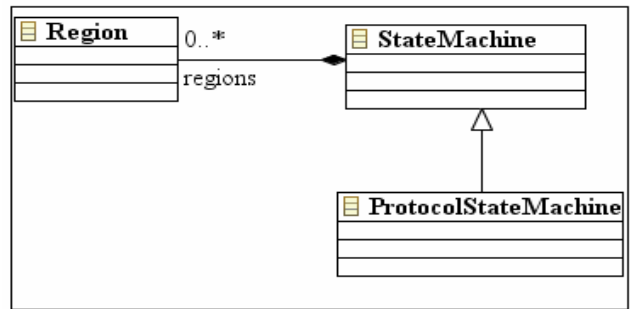


Figure 11. The StateMachine metaclass in UML 2.0 metamodel

3) <<SCAProtocolStateMachine>> stereotype

The stereotype <<SCAProtocolStateMachine>> applied to the metaclass StateMachine (see Figure 11) is defined by the following OCL constraint:

Finally, Figure 12 illustrates our UML2.0 profile for SCA.

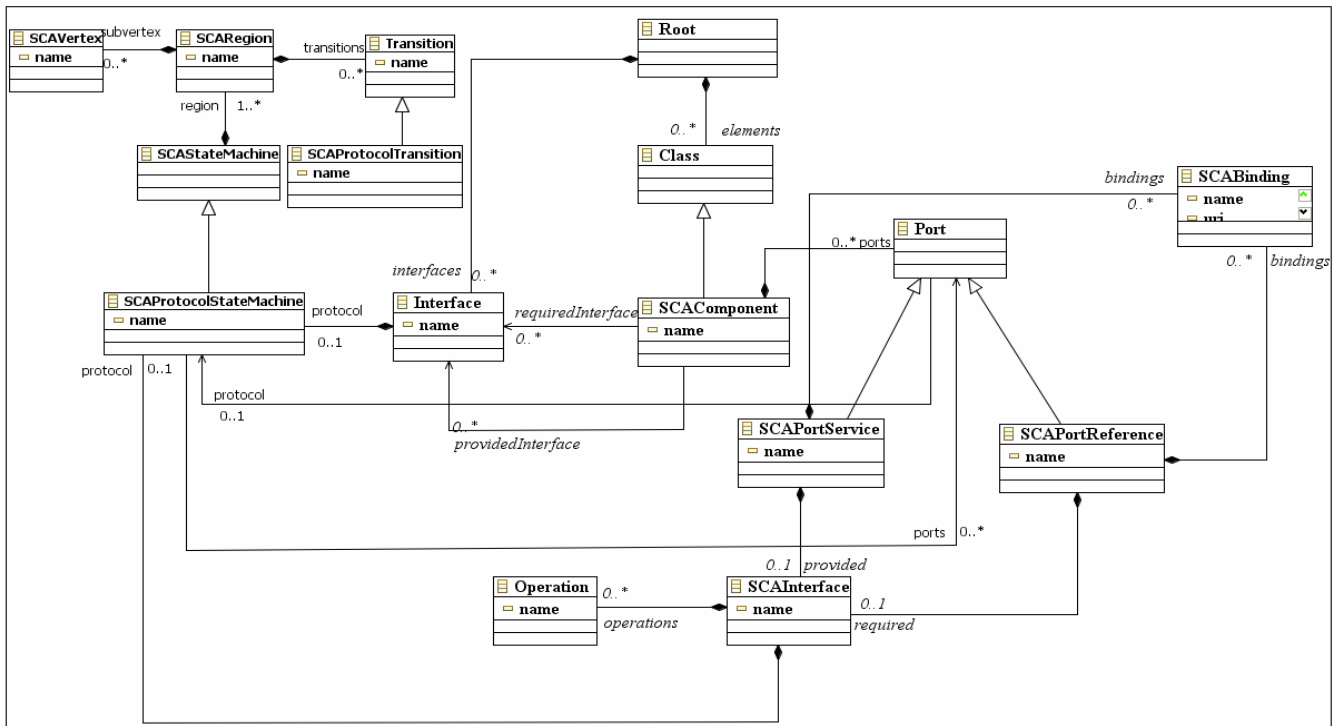


Figure 12. A metamodel of Profile UML 2.0-SCA

V. EXOGENOUS TRANSFORMATION OF PROFILE UML 2.0-SCA TO SCA

In this part of paper, we aim to automatically transform this profile into an application using an MDE approach of automation [15]. Before the transformation of our profile into ecore, we have created its implementation in Domain Specific Language (DSL).

A. Our approach

In this section, we present in a detailed way the ProfilUML2SCA tool written in ATL allowing the transformation of an extension of profile proposed previously towards an SCA application.

Figure 13 illustrates our proposed approach for an automatic transformation of a profile UML 2.0-SCA to SCA. We distinguish two levels of specification: M2 (a Meta model level) and M1 (a model level) as define by the MDA approach. In our approach a transformation model defines how to generate a model (SCA model) according to the metamodel (SCA Metamodel) from the model (Profile model) consistent with the metamodel (Profile Metamodel).

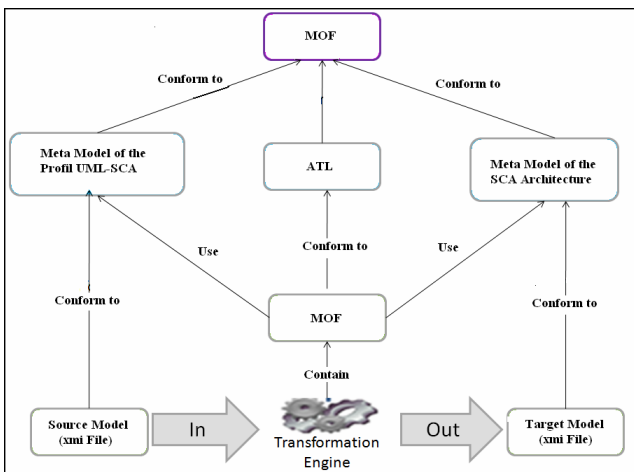


Figure 13. The proposed approach for an automatic transformation profile into SCA

The source and target models (i.e., the Profile UML 2.0-SCA model and the SCA model) and the ProfilUML2SCA tool are consistent with their ProfilUML, SCA and ATL metamodels. These metamodels are also consistent with the Ecore meta-model of the EMF platform [16]. The profile source metamodel, resp. the SCA target metamodel, is represented by an Ecore diagram in Figure 12, resp. Figure 3.

B. Global Overview on the ProfilUML2SCA tool

In the next, we present the standard rules for the development of our tool. Our profile transformation into SCA is based on rules issued from OCL constraints. An ATL module corresponds to the transformation of a set of source models into a set of target models according to their metamodels. Its structure is formed by a section header, an optional import section, a set of helpers and a set of rules.

The header section (Figure. 14) defines the names of the transformation module and the variables of the source and target models. The following ATL source code represents the header of the ProfilUML.atl file, thus the ATL header for the transformation from Profile UML-SCA to SCA application:

```

module profilUML; -- Module Template
create OUT : SCA from IN :
ProfilUML;
    
```

Figure 14. The header section of transformation

- **module** defines the module name.
- **create** introduces the target model declaration.
- **from** introduces the source model Declaration.

In this part of paper, we present the transformation rules of the structural aspect transformation of our profile UML2.0-SCA using the ATL language.

We define the rule which allows us to transform an SCAComponent in the profile to Component in SCA, here an SCA component takes the same name as a SCA.

```

rule SCAComponent2Component{
from scac:ProfilUML!SCAComponent
to c: SCA!Component (
name<-scac.name)}
    
```

- Each instance of a stereotype SCAPortService is transformed into a Service in SCA.

```

rule SCAPortService2Service{
from scaps:ProfilUML!SCAPortService
to s:SCA!Service(
name<-scaps.name,
component<-scaps.component,
interface<-scaps.provided,
process<-scaps.provided.protocol)}
    
```

- Each instance of a stereotype SCAPortReference is transformed into a Reference in SCA.

```

rule SCAPortReference2Reference{
from scapr:ProfilUML!SCAPortReference
to r:SCA!Reference(
name<-scapr.name,
component<-scapr.component,
interface<-scapr.required,
process<-scapr.required.protocol)}
    
```

- Each instance of a stereotype SCAProtocolStateMachine is transformed into a BPELProcess.

```

rule
SCAProtocolStateMachine2ProcessBPEL{
from
psm: ProfilUML!SCAProtocolStateMachine
to bp: SCA!BPELProcess (name<-psm.name) }
    
```

VI. CONCLUSION

This paper proposes a UML profile for specifying the SCA Architectures. This profile is based on the reuse of concepts for the description of the elements of the model which essentially arise from the SCA Architecture. Such a profile will facilitate the work of the developers which are not still familiarized with complex languages and notations.

In a second part, we proposed an MDE approach which allows transforming a metamodel of our extension of a UML profile proposed into an SCA metamodel. To do so, we elaborated two metamodels: the ProfilUML metamodel and the SCA metamodel. Then, we designed and implemented a ProfilUML2SCA tool in order to transform a profile model conform to its metamodel to a SCA model conform to its meta-model.

The extension proposed in this paper provides a special study of the structural and behavioral aspects of the SCA Architecture. So, we intend to extend our profile to take into account the advanced concepts such as SCA connector and composite.

REFERENCES

- [1] Open SOA Collaboration, Service Component Architecture (SCA), SCA Assembly Model v1.00 specifications, 2007.
- [2] OSOA, Open Service Oriented Architecture, the Home Page, 2007. <http://www.osoa.org/>
- [3] J. Warmer and A. Kleppe, "The Object Constraint Language," Addison-Wesley, August 2003.
- [4] X. Blanc, "MDA en action ingénierie logicielle guidée par les modèles," Eyrolles, 2005.
- [5] Object Management Group. MDA Guide, version 1.0.1, 2003. <http://www.omg.org>
- [6] J.Bézivin and X.Blanc, Promesses et Interrogations de l'Approche MDA, Développeur Référence, Septembre 2002.
- [7] F. Jouault, "Contribution à l'étude des langages de transformation de modèles," thèse de doctorat, Ecole Doctorale sciences et technologies de l'information et des matériaux, Nantes, 2006.
- [8] D. Garlan, S.W. Cheng, and A. Kompanek, "Reconciling the Needs of Architectural Description with Object-Modelling Notations," Science of Computer Programming Journal, Special UML Edition Elsevier Science, 2001.
- [9] N. Medvidovic, D.S. Rosenblum, D.F. Redmiles, and J.E. Robbins, "Modelling Software Architectures in the Unified Modelling Language," ACM Transactions on Software Engineering and Methodology, vol. 11, no .1, January 2002.
- [10] OSOA. SCA Service Component Architecture: Assembly Model Specification, March 2007.
- [11] SCA, "Building Your First Application Simplified BigBank," SCA Version 0.9, August 2007.
- [12] S. Thatte, "XLANG Web Services for Business Process Design", October 2005.
- [13] F. Leymann, Web Services Flow Language.WSFL 1.0, October 2005. <http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>.
- [14] T. Andrews, F. Curbera , H. Dholakia , Y. Golland , J.Klein , F. Leymann, K. Liu , D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, Business Process Execution Language for Web Services, October 2005.
- [15] R. Maraoui, M. Graiet, M. Kmimech, M.T. Bhiri, and B. Elayeb, "Formalisation of protocol mediation for web service composition with ACME/ARMANI ADL," Service Computation IARIA 2010-Lisbon-Portugal, November. 2010.
- [16] F. Budinsky, D. Steinberg, and R. Ellersick, "Eclipse Modelling Framework : A developer's Guide," Addison-Wesly Professional, 2003.

Process Improvement and Knowledge Sharing in Small Software Companies: A Case Study

Minna Kivihalme, Anne Valsta, Raine Kauppinen
HAAGA-HELIA
Ratapihantie 13
FIN-00520 Helsinki, FINLAND
{minna.kivihalme, anne.valsta, raine.kauppinen}@haaga-helia.fi

Abstract — Process improvement, knowledge sharing and management are challenging issues for small software companies. In this article, the experiences of three small software business organizations, one research and development (R&D) -project team in hotel business using Taimi-tool [16][26] to support process improvement and knowledge sharing were studied and analyzed with grounded theory. The findings of the case study show that systematic process improvement is not very familiar to small software companies. Modeling and writing down processes are considered old-fashioned, too strict and rules given from above. Process improvement does not fit in to the self-image of an innovative, agile and flexible software company. However, for the R&D-project team, Taimi-tool gave good insights for modeling processes and finding the best ways to improve them. This implies that the problems are not with the tool, but rather with the small software business organizational culture towards process improvement and knowledge sharing. There is a clear need for collaboration with research and development in the field of workplace learning and competence development in small business working life and vocationally oriented educational institutions.

Keywords-process; project; knowledge sharing; education

I. INTRODUCTION

Small organizations as well as the larger ones need software process modeling, development and improvement. However, traditional models and frameworks, such as CMMI or SPICE are often much too complicated and rigorous to be used in small software business. During the earlier study done at HAAGA-HELIA University of Applied Sciences, it was found that in process modeling and process improvement for small businesses the key purpose is to encourage and develop the organizational and individual knowledge [16]. In addition, the need for a tool supporting an agile process development in the small companies was identified. Based on these findings, a prototype of Taimi-tool was introduced.

Taimi-tool is designed to support communication and sharing of best practices which can be found in any company during an excellent project or lessons learned from a troublesome project. These experiences remain literally unique unless they are shared between colleagues. An excellent project is worth modeling and should be distributed as best practices or as a model, and as widely as

possible. In the next section the research approach and methods are described. In the third section Taimi-tool is introduced. The fourth section presents the current state of art of relevant related work. In the fifth section case study companies are introduced briefly and finally findings, conclusions and future work end the paper.

II. RESEARCH APPROACH AND METHODS

In this research the aim was to understand and specify the relations between software process improvement and knowledge sharing in small software companies. The approach is qualitative and the methods used to analyze data are based on grounded theory [9]. The research questions are:

- 1) How important systematic process improvement is for small software businesses? How do small software businesses share knowledge internally and externally?
- 2) What kind of a connection there is between process improvement and knowledge sharing in small software organization?
- 3) What kind of issues the small software businesses have faced in integrating knowledge sharing and management to normal daily-routines? Could a tool like Taimi be helpful in these situations?

The hypothesis based on the previous study [16] is that the small organizations will benefit using a process modeling tool like Taimi. It will make their work more systematic and at the same time it allows them to be flexible and even more agile in their daily work. Process improvement is not only for those who fancy processes. Taimi will encourage everyone in the organization to participate process improvement achievements.

The data collection was done using case-study approach [28] during eight months in 2010. To start the research group organized workshops in every case study company. These workshops were documented as group interviews. In every workshop there were at least two company representatives and two members of the research team. Different roles like management and leadership roles, project management and project team member roles were involved. During the workshops, Taimi-tool was introduced and a brochure and guidelines of use were given to the case study companies. After the workshops, each company could

either install Taimi to their own server by downloading the software from Internet or they could use the service hosted by the research team during the experiment. Research team named a specific contact person for every company to help with the details. After few months, the date was fixed for the final interviews. Everyone involved in the workshops were interviewed except one process developer who was retired. These interviews were semi-structured, recorded and transcript personal interviews.

During the months between the workshops and final interviews, the research team noticed that it seemed as if the companies were not experimenting much with Taimi. This was later confirmed in the interviews. Because of this, an R&D-project team in hotel business was added as a case study company. The R&D-project team had started a project a bit earlier. After they were contacted and introduced to Taimi and this case study they wanted to test Taimi in their project. The research group thought this was a good way to evaluate the suitability of Taimi for a different type of processes and organizations. A workshop was arranged with the project team giving them the necessary material and information.

III. TAIMI – A MODELLING APPLICATION

Taimi is an open source web application used via a web browser such as Firefox or Internet Explorer. Taimi [26] is developed using Ruby on Rails [22] and it runs on a standard web application server such as Apache Tomcat [27]. In Taimi, each process model and project is visualized as a matrix containing phases and tasks to be stored in the database (MySQL). The phases of a process model or a project can be named and colored as the user wants (see Figure 1). The task boxes get a color code based on the phase they belong into, so it is easy to read the matrix. Tasks can be copied to another process model or project, even several times in the same one if necessary. It is also possible to add attachments and comments to the tasks.

In Taimi, a task in a process model can be seen as a best practice with supporting templates or other guidance. The tasks can be completed with as specific or general descriptions as needed for the common benefit of the organization. The users of a process model can comment on their experiences related to the model as well as suggest improvements to the model immediately when an issue arises during a project.



Figure 1. Adding a new phase into a process model

In Taimi, a process model can be copied into a project and vice versa. After that, the new project is a look-alike copy of the original process model. Tasks that are not part of the new project can be erased. It is also possible to add and edit new tasks (see Figure 2) to the project or copy tasks from other projects or process models.

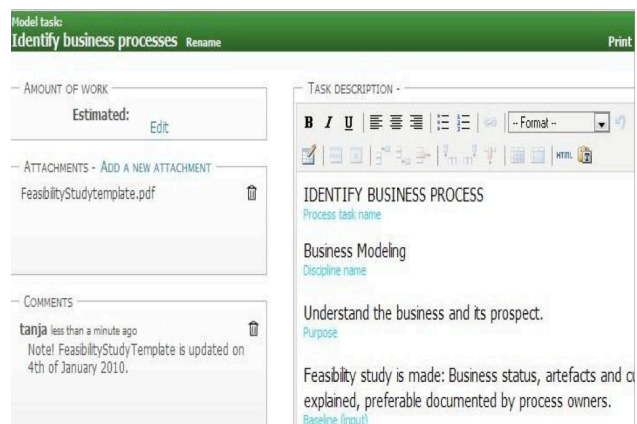


Figure 2. An added task in a process

It is possible to estimate the amount of the work needed to complete a task and define start and end dates for each task. When a task in a project is completed, it can be marked as done. Thus, the progress of the project can be monitored visually (see Figure 3).

The information gained in a project is immediately visible throughout the organization and to the relevant interest groups since a project manager is able to share the project with the necessary interest groups. The members of these groups can also comment on the tasks in the project and add attachment files to them as seen in Figure 2. In this way, Taimi can be used as a collaborative platform supporting the evolution of process models and projects in an agile way based on the shared lessons learned in different projects.



Figure 3. A project with two completed tasks

IV. RELATED WORK

Software process improvement (SPI) is closely related to the knowledge management (KM) strategy of an organization [18]. Especially in IDEAL [8], CMM [17] and SPICE [23] the focus is in achieving a certain level of maturity. The knowledge management strategy should vary according to the maturity level [18]. Knowledge management strategy can be viewed from two different levels: 1) based on coding [11][25] and 2) tactical level [20] According to Dixon [7] there are five different types of knowledge transfer: serial transfer, near transfer, far transfer, strategical transfer and expert transfer. In order to succeed in knowledge transfer and knowledge management, an organization needs to define what activities and operations it will have and choose its knowledge management strategy accordingly [18].

Sulayman and Mendes [24] have studied the small and medium sized organizations specialized in web technologies. They found that issues in SPI, which were important to the success in business operations, were very closely related to the knowledge management strategy of the company as well. According to the survey made for software developers about the web 2.0 tools, those tools have already changed the way software developers communicate with each other, for example, about testing, marketing and developing [1]. Cockburn has identified term 'osmotic communication' in his book about agile software development and emphasized its meaning to software engineering especially in the agile development [3]. It is very common to see SPI as a mean to achieve better quality with less cost. SPI can also be seen as a way of rationalizing the activities in an organization. It is very important that an organization is able to handle uncertainty and instability because "Chaos is often a sign that the implementation process is on its way and that you are about to receive valuable information helping you succeed" [2].

It is important to recognize the need for improvement when the company management anticipates the future. When the company is in its peak condition, the work is often very ambitious, even aggressive and the need for

change can easily be left unnoticed [10]. Company after company has seen its management fixing pieces instead of redesigning the processes applied to get the work done [5][10].

In a study [4] made in Ireland, the target was to figure out why small companies are reluctant to use known best practices in their activities. It was found that process improvement is considered to be so expensive investment that small companies do not see or do not want to see it as a profitable effort [4]. The small software companies have easily adapted principles of lean thinking and the question often raised by them is whether the customer is willing to pay for this kind of work or not. This is a potential conflict, because there is evidence of customers wanting to see proof of quality and stable working habits within the small and especially young companies [16].

The study in Ireland also showed that start-up companies saw the formal SPI as an obstacle to creativity, innovation and flexibility. From their viewpoint, SPI was not about improving the process, but instead a set of strict instructions forcing them to follow the given process and blocking creativity and flexibility. Also, it was found out that the educational background, experience and know-how of the technical management had a lot of influence on the willingness to commit to the best practices.

A similar study [19] was made in Vietnam focusing on the obstacles of applying process models and SPI. In that study it was found that depending on the size of the company the issues were emphasized differently. For the small organizations, the most important issue was the lack of resources. In the bigger organizations, in addition to the previous issue, the lack of communication, the commitment of the management and timetable pressure were identified. These findings were compared to the research made in the UK where the timetable pressure was identified as the key issue. The conclusion was that while in Vietnam, the lack of resources meant that the process improvement was not really happening, in the UK it was, but there still were problems related to the resources in the form of timetable pressure. In addition, the staff in Vietnam was quite young compared to the staff in the UK. The conclusion was that young inexperienced managers did not see process improvement so important that they would have allocated resources for such initiatives. [19]

In a Finnish study [15] about the agile future organization, it was found that organization needs versatile talented people and that the agility needs to be part of the business operation strategy as well. The software development alone cannot be agile if the business around it is not. In addition, an agile process framework is needed for teams to be able to tailor their process for the particular situation [15].

In a study [21] of project managers' knowledge transfer made in the USA, it was found that inexperienced project managers relied more on social networks in order to get the knowledge they were looking for than their more experienced colleagues who used more formal knowledge sources. The conclusion of this study [21] was that inexperienced project managers were sensitive or even timid to search and ask information from knowledge management

systems used in their company. According to the study social norms and organizational culture either encourages IT project manager to share their knowledge or inhibit it.

Wiki technology has been found to be very powerful technique to transfer knowledge to large groups of people in ad-hoc and other dynamic situations [12]. But there is a lot more than just ad-hoc and other dynamic situations in knowledge transfer and knowledge management. Davidson and Rowe [6] defined different levels of knowledge management. At the first level, the team learns from the previous project and answers to questionnaire to find out the lessons learned. This information is delivered to the second level, where it will be analyzed and passed on to the next level, which is a strategic level. It is obvious that this formal knowledge management will benefit from a suitable supporting tool and, in most cases, this process needs a knowledge manager to handle the tool and process [6].

V. CASE STUDY COMPANIES

The companies participating in the case study are small or medium sized Finnish software engineering companies (companies A-C, see Table 1) and a hotel entrepreneur (company D).

Table 1. Some business features in 2009-2010

Company	Personnel	Turnover, MEUR	Established
A	110	7.7	2000
B	50	4.5	2005
C	60	5	2002

A. Company A

Company A is a Finnish software engineering company creating applications ranging from intranets to business intelligence and social services. Their emphasis is on the lean philosophy – removing waste and concentrating only on producing something useful and having a strong focus on usability at the same time. This company has been growing quite fast. The atmosphere of the company is very informal and free-minded. The slogan for this company could be “Things can always be done better”. The company recruits people having the attitude of wanting to give his or her best as an individual and as a team member.

B. Company B

Company B is a Finnish software engineering company specializing in tailor-made customer-specific software projects, consulting, maintenance and support for large information systems. Processes are quite a new concept to the company B. The management has mainly focused on creating innovative environment for people to work in. Company B views process models from the standard point of view: as a way to implement company strategy. However, company B sees processes and modeling very situational.

C. Company C

Company C is a Finnish software consulting company specialized in software quality. The aim of the company C is

to assist their customers to ensure the quality of their IT systems. Because of their service strategy, the process meta models (CMMI, SPICE) are well known and those models are used to improve customers’ processes.

D. Company D

Company D was not a partner at the beginning of the research study and it differs from the previous ones. Company D was actually one team from HAAGA-HELIA’s R&D-project team for developing processes for hospitality and hotel business.

VI. FINDINGS

Systematic process improvement was not very familiar to the companies in this case study. This is the case even in the company having more than 100 employees. Instead, knowledge management and knowledge transfer are very much appreciated. However, they are on the level of near transfer [7] and not on strategic level. Modeling and writing down processes are considered old-fashioned, too strict and as rules given from above. Process improvement does not fit into the image of an innovative, agile and flexible company. According to Mark Kennaley the word process is not valued in the agile approach [14]. Instead of process Kennaley recommends ‘standard work’ in the meaning of how daily work is performed and described.

Process and creativity are seen as opposites and companies want to emphasis creativity, passion for work and freedom to be creative. Similar results were found in the research conducted in Ireland [4]. The daily work is seen too artistic to be modeled as a process. Still, knowledge transfer and knowledge management are highly appreciated and there are lots of different unsystematic methods for transferring knowledge within the company and via networks. These methods are not treated as processes even though they might be seen that way. Nevertheless the atmosphere of knowledge sharing is very free and open. The organizational culture in these case study companies encourages project managers to share their knowledge. The same results were also found in the study made in the USA [21].

In every interview we made, one common denominator was found; at a certain point people do not have the time to do anything in addition to the deliverable result. The deeper you dig into the world of project managers the more you sense that it is hard to invent the wheel all over again. Project managers eventually see the pattern, but they do not have time or resources to make the pattern visible even if they would like to. The results of the research in Vietnam support this as well [19]. Project managers are bound very tightly to the ongoing project. They would like to have more guidance, methods and help instead of being forced to invent the wheel again. However, the management and leaders in the small companies think that there are many ways and opportunities to transfer knowledge, for example, ‘brown bag sessions’, ‘company Fridays’ and study or learning groups. All these methods are actually methods for near transfer and only they work if face-to-face methods are available. But, these methods are unavailable if you cannot be present, for example, if you are working in the customer

premises and the customer will not allow collaboration within the company's social network. This was the case for one of the project managers working at customer premises. Companies use and have been using all sorts of tools to communicate such as discussion forums, wikis, social networks and network drives, but they also feel that the information gets buried into these systems. Major disadvantage for these tools is the fact that you need the time to gain all the visible or hidden information and time is a scarce resource in the small or medium enterprise (SME).

Four groups were identified using the open coding of grounded theory: knowledge management and transfer, creativity, process and process improvement and lack of time. In axial coding, the relationships among these concepts are identified. These relations are illustrated in Figure 4.

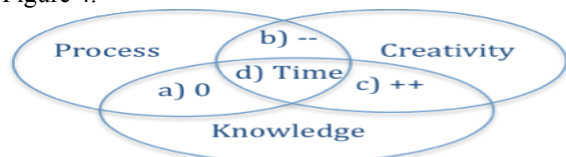


Figure 4. Creativity, process, knowledge and time in a SME

The SME's see knowledge management and transfer worth chasing for while processes and especially process improvement is something SME's are not so keen on doing. One of the managers said, "We should forget analyzing and think about what to do next, because analyzing doesn't really create anything. Process is something given from above".

Processes were even seen as an obstacle to creativity. When the creativity is high, processes are low. Negative correlation between process and creativity is illustrated by letter b) in the Figure 4. Another manager said, "The work we do for our customers is very creative: people get familiar with things and tell about them, so it is very informal. We have not defined our process, and we are talking about our work as artistic work."

Knowledge transfer and management are directly comparable to creativity in SME's self-image. The positive correlation between knowledge and creativity illustrated by letter c) is seen in the Figure 4. One of the project managers said, "We are encouraged to teach each other. We should give small sessions out of our own interest and will. These demo days have been ok. It has been possible to go deeper into some interesting subject area".

According to Sulayman and Mendes [24] and Mathiassen and Pourkomeylian [18] knowledge management and transfer are closely related to processes and their improvement. For the companies in this study this was not the case. In these companies the knowledge transfer happens ad-hoc without visible processes involved. This relationship is illustrated by letter a) in the Figure 4. One of the managers said, "We try to give people as much freedom as possible and encourage them to use their creativity and passion. That's how we create self-organized workplace and knowledge sharing". By this the manager revealed in fact that there was not a process involved in knowledge sharing.

These findings show that SME's are not ready for a tool like Taimi. There were some positive signs especially from

the project managers' point of view: they would have appreciated this kind of a tool if they would have had time to take a bit deeper look at it. This also means that they would have liked to concentrate on the process, process improvement and knowledge sharing as well. One of the project managers said, "I don't need any new features for Taimi. The matrix is just great. The main features of Taimi are really good. And we could have used Taimi for real as a tool between project managers and company management, but the lack of time was my problem". According to Davidson and Rowe [6] companies would benefit from a suitable tool to support the knowledge management and transfer as well as knowledge manager to handle the tool and process.

The project managers in the case study companies were too busy in their ongoing project to be able to take a wider perspective to their work. They needed the support from their superiors, which they did not get this time. One of the project managers said, "I would have liked to test the tool, but in the company there were others who didn't want to test it or they wanted to have free hands and not to use any tool at all. The decision was made by management to not to trial the tool".

In the Figure 4 the letter d) illustrating the lack of time has connections to all the other themes. The lack of time rounds up to the fact of project managers wanting to have the support from their superior even in the atmosphere of informal and innovative company. One of the project manager said, "There could be concrete recommendation what tool to use in projects. Now everybody is using the tools they want and you have to compare the plusses and minuses of the product by yourself. If there were some guidance, it would be nice. Face-to-face meetings are of course nice. A tool, project database or something would be a good bonus for later use or reference." By this the project manager meant that, for some of the staff, it would be helpful and timesaving to have a process and a supporting tool for the process and knowledge sharing.

Findings from the company D differ from the rest of the case study companies. Taimi got good feedback from the company D. The idea of user interface as a process matrix, the idea of being able to link different information to tasks and being able to change the process easily and quickly if necessary were appreciated by the company D. The tool gave good insight to process modeling and finding the best ways of improving the process in the hotel involved in the R&D-project team. Company D wanted to start using Taimi, but they were somewhat worried about the continuity of Taimi's future. These findings about the tool and the idea behind it are promising.

VII. CONCLUSION AND FUTURE WORK

The conclusion of this study is that process improvement is on very early stage in the small and medium sized software engineering companies. Modeling processes is on learning phase and it is not yet something that is seen important and necessary. There is always something more important to be done. Also, the lack of time and resources does not promise more process improvement to happen in

the near future. Sharing knowledge happens ad-hoc or in a project between its members.

A new tool or method needs to be learnt first and it takes time. And spare time is something the small companies do not have. Internal improvement gets left behind. Outside stimulator or supporter, research time and money is needed to get the internal improvement on the road. The first step is always the hardest. Next steps are easier to take. A comprehensive tool would help sharing and transferring knowledge wider in a company.

How could we support SME? According to Illeris [13] radical changes and development have been taking place in recent years concerning work-based and work-related learning and competence development. Globalization, the knowledge society and human competence are becoming an increasingly decisive resource of competition. Illeris notes vary of general qualifications with terms such as organizational learning, experimental learning and spirituality at work. According to Illeris even professionals find it difficult to negotiate these areas, not to speak of small companies with no time and no particular educational function. The situation is different in educational institutions, where time is available and whose function is vocational education. The lesson learned from this study is that there is a need for much tighter collaboration with research and development in the field of workplace learning and competence development in working life with small software development companies and vocationally oriented educational institutions.

During the curricula, students should be adapted to operate with a shared tools and knowledge ware. The young professionals need good understanding and skills on processes and methodologies. Further research is needed to study the attitudes of students choosing software development as their area of expertise and how these attitudes reflect to their future career as professional software specialists. Another, maybe more important, question is what are the mechanisms, including teaching methods, especially in vocationally-oriented education to promote the internal process improvement both in business processes, entrepreneurship and software processes as a part of workplace learning and competence development.

REFERENCES

- [1] Black, S. and Jacobs, J. 2010. Using Web 2.0 to Improve Software Quality, Web2SE'10, May 4, Cape Town, South Africa.
- [2] Börjesson, A. and Mathiassen, L. 2004. Successful Process Implementation, IEEE Software, July/August, pp. 36-44.
- [3] Cockburn, A. 2007. Agile Software Development, The Cooperative Game, Second Edition.
- [4] Coleman, G. and O'Connor, R. 2008. Investigating software process in practice: A grounded theory perspective. The Journal of Systems and Software Vol. 81, pp. 772-784.
- [5] Dangle, K., Larsen, P., Shaw, M. and Zerkowitz, M. 2005. Software Process Improvement in Small Organizations: A Case study. IEEE Software, June, pp. 68-75.
- [6] Davidson, P. and Rowe, J. 2009. Systematising knowledge management in projects. International Journal of Managing Projects in Business, Vol 2, No 4, pp. 561-576.
- [7] Dixon, N. 2000. Common Knowledge, How Companies Thrive by Sharing What They Know?
- [8] McFeeley, B. 1996. IDEAL: a user's guide for software process improvement. The software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Handbook CMU/SEI-96-HB-001.
- [9] Glaser, B.G. 1992. Basics of grounded theory analysis: Emergence vs. forcing. Mill Valley, CA: Sociology Press.
- [10] Hammer, M. and Champy, J. 1993. Reengineering the Corporation. A Manifesto for Business Revolution.
- [11] Hansen, T., Morten, N. and Tierney, T. 1999. What's your strategy form managing knowledge? Harvard Business Review, pp. 106-116.
- [12] Hester, A.J. 2010. Increasing Collaborative Knowledge Management in Your Organization: Characteristics of Wiki Technology and Wiki Users. SIGMIS-CPR'10, May, pp. 113-143.
- [13] Illeris, K. 2003. Workplace learning and learning theory. Journal of Workplace Learning, April, pp. 167-178.
- [14] Kennaley, M. 2010. SDLC 3.0. Beyond a Tacit Understanding of Agile, Towards the Next Generation of Software Engineering. Fourth Medium Press.
- [15] Kettunen, P. and Laanti, M. 2007. Combining Agile Software Projects and large-scale Organizational Agility, Software Process Improvement and Practice, July, pp. 183-193.
- [16] Kivihalme, M. and Valsta A. 2010. Improving Software Development Processes in Small Companies: A Case Study. In Proceedings of the IASTED International Conference on Software Engineering (Innsbruck, Austria, February 16 – 18, 2010) SE 2010. ACTA Press.
- [17] Laryd, A. and Orci, T. 2000. Dynamic CMM for Small organizations. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.5555> [ref. Apr/2011].
- [18] Mathiassen, L. and Pourkomeyliyan, P. 2003. Managing knowledge in a software organization. Journal of Knowledge Management, May, pp. 63-80.
- [19] Niazi, M., Ali Babar, N. and Katugampola, M. 2008. Demotivators of Software Process Improvements: An Empirical Investigation. Software Process Improvement and Practice, March, pp. 331-347.
- [20] Nokana, I. and Takeuchi, H. 1995. The knowledge-Creating Company, Oxford University Press.
- [21] Petter, S. and Randolph, A.B. 2009. Developing Soft Skills to Manage User Expectations in IT Projects: Knowledge Reuse Among IT Project Managers. Project Management Journal, Dec, pp. 45-59.
- [22] Ruby On Rails. <http://www.rubyonrails.org> [ref. Apr/2011].
- [23] SPICE, 2007. Software Process Improvement and Capability dEtermination, ISO 15504, <http://www.sqi.gu.edu.au/spice> [ref. Apr/2011].
- [24] Sulayman, M. and Mendes, E. 2010. Quantitative Assessments of Key Success Factors in Software Improvement for Small and Medium Web Companies. SAC'10, March 22-26, Sierre, Switzerland.
- [25] Swan, J., Newell, S., Scarbrough, H. and Hislop, D. 1999. Knowledge management and innovation: network and networking. Journal of Knowledge Management, Vol 3 No. 3, pp. 262-275.
- [26] Taimi, process and project management tool, 2010. <http://www.taimimap.fi/sivut/material.html> [ref. Apr/2011].
- [27] Tomcat. <http://tomcat.apache.org> [ref. Apr/2011].
- [28] Yin, R.K. 2003. Case Study Research: Design and Methods, Third Edition.

Choosing a Business Software Systems Development and Enhancement Project Variant on the basis of Benchmarking Data – Case Study

Beata Czarnacka-Chrobot
 Department of Business Informatics
 Warsaw School of Economics
 Warsaw, Poland
 e-mail: bczarn@sgh.waw.pl

Abstract—Execution of Business Software Systems (BSS) Development and Enhancement Projects (D&EP) is characterised by the exceptionally low effectiveness, leading to the considerable financial losses. Thus it is necessary to rationalize investment decisions made with regard to the projects of this type. Each rational investment decision should meet two measurable criteria: of effectiveness and of economic efficiency. In order to make *ex ante* evaluation of these criteria, being key to the decision-making process, one may successfully use ever richer resources of benchmarking data, having been collected in special repositories that were created with improvement of software processes in mind. The goal of this paper is to present possibilities of rationalization of investment decision concerning the choice of BSS D&EP execution variant with the use of benchmarking data on the basis of a case study. These issues classify into economics problems of software engineering.

Keywords-business software systems development and enhancement projects variants; rational investment decision; benchmarking data repositories; software engineering economics

I. INTRODUCTION

In practice, execution of Business Software Systems (BSS) Development and Enhancement Projects (D&EP) is characterised by the exceptionally low effectiveness, leading to the considerable financial losses. This may be proved by numerous analyses. As indicated by the results of the Standish Group studies success rate for application software D&EP has never gone beyond 35%, while currently products delivered as a result of nearly 45% of them lack on average 32% of the required functions and features, the estimated project budget is exceeded by approx. 55% on average and the planned project time – by nearly 80% on average [1] (for more details see [2]). Analyses by T.C. Jones plainly indicate that those software D&EP, which are aimed at delivery of business software systems, have the lowest chance to succeed [3]. The Panorama Consulting Group, when investigating in their 2008 study the effectiveness of ERP (Enterprise Resource Planning) systems projects being accomplished worldwide revealed that 93% of them were completed after the scheduled time while as many as 68% among them were considerably delayed comparing to the expected completion time [4]. Merely 7% of the surveyed ERP projects were accomplished as planned. Comparison of actual versus

planned expenses has revealed that as many as 65% of such projects overran the planned budget. Only 13% of the respondents expressed high satisfaction with the functionality implemented in final product while in merely every fifth company at least 50% of the expected benefits from its implementation were said to be achieved. Meanwhile (see also [2]):

- BSS are one of the fundamental IT application areas.
- BSS development or enhancement often constitutes serious investment undertaking.
- In practice, COTS (Commercial-Off-The-Shelf) BSS rarely happen to be fully tailored to the particular client business requirements therefore their customization appears vital.

Low effectiveness of BSS D&EP execution leads to the substantial financial losses, on a worldwide scale estimated to be hundreds of billions of dollars yearly, sometimes making even more than half the funds being invested in such projects. The Standish Group estimates that these losses – excluding losses caused by business opportunities lost by clients, providers losing credibility or legal repercussions – range, depending on the year considered, from approx. 20% to even 55% of the costs assigned for the execution of the analysed projects types (see e.g., [5][6]). On the other hand, analyses of The Economist Intelligence Unit, which studied the consequences of BSS D&EP delay indicate that there is strong correlation between delays in delivery of software products and services and decrease in profitability of a company therefore failures of BSS D&EP, resulting in delays in making new product and services available and in decreasing the expected income represent threat also to the company's business activity [7].

The above studies unequivocally indicate there is a significant need to rationalize investment decisions made with regard to BSS D&EP. To do so, one may successfully use ever richer resources of benchmarking data, having been collected with the intention to support improvement of various IT projects, including BSS D&EP. The goal of this paper is to present possibilities of BSS D&EP investment decision rationalization with the use of benchmarking data, illustrated with an example taken from development practice. This decision concerns choosing variant of BSS D&EP execution – since each project of this type may be executed using one of the three variants, namely: (1)

developing new BSS from scratch, (2) customization of COTS BSS, and (3) modernization of BSS being currently used.

The paper is structured as follows: in Section 2 the author presents the criteria of rational investment decision in the context of BSS D&EP along with the selected results of studies concerning *ex ante* evaluation of these criteria. Section 3 is devoted to the presentation of the considered case study problem. In Section 4 the main conclusions coming from the benchmarking data analysis are pointed out, while in Section 5 the effectiveness and efficiency factors for the recommended BSS D&EP variant are analysed. Finally, in Section 6 the author draws conclusions and some open lines about future work on the rationalization of BSS D&EP investment decision with the use of benchmarking data.

II. RATIONAL INVESTMENT DECISION CRITERIA FOR BUSINESS SOFTWARE SYSTEMS DEVELOPMENT AND ENHANCEMENT PROJECTS

Each rational investment decision should meet three criteria, which in the context of BSS D&EP should be interpreted as follows (for more details see [8]):

- Criterion of consistency, which means that the project undertaken should comply with the environment (economic, organizational, legal and cultural) – unlike the other two criteria, this criterion is not subject to quantitative assessment therefore it is skipped in this paper.
- Criterion of economic efficiency, meaning that the decision should benefit to the maximisation of the relationship between the effects to be gained as a result of project execution and the costs being estimated for the project.
- Criterion of effectiveness, meaning that such decision should contribute to achieving the assumed result, in the case of BSS D&EP usually being considered as delivering product meeting client's requirements with regard to functions and features without budget and time overruns.

Generally speaking, in the case of economic efficiency evaluation, effects are compared against costs necessary to achieve these effects while in the case of effectiveness evaluation these are only the results that are of significance. Thus, economic efficiency is measured by relating total effects to total costs. Meanwhile, effectiveness is measured by the ratio of the achieved result to the assumed result, which is being conveniently expressed as a percentage.

Both economic efficiency criterion as well as effectiveness criterion are based on the obvious assumption that the effects, costs and results are measurable. However, in the case of BSS D&EP this assumption is often treated as controversial. Numerous studies indicate that evaluation of BSS D&EP economic efficiency is made relatively rarely while fundamental reason for this *status quo* are difficulties related to identification, and most of all quantitative expression, of benefits resulting from the execution of such projects (see e.g., [9][10][11][12][13]). These studies reveal

that difficulties related to identification and quantitative expression of BSS D&EP costs too are of significance, which also is of importance to the evaluation of their effectiveness.

Key conclusions coming from the above mentioned studies have also been confirmed by the results of studies carried out by the author of this paper in two research cycles among Polish dedicated BSS providers (for more details see [14]). They revealed that at the turn of the years 2005/2006 the results obtained with the use of the effort estimation methods, employed only by approx. 45% of the respondents, were designed for estimating BSS D&EP costs and time frame while relatively rarely they were used to estimate economic efficiency – such use of these methods was indicated by only 25% of those using effort estimation methods. Heads of IT departments in Polish companies, for which BSS D&EP are executed, still explain the sporadically required calculation of this type of investments efficiency mostly by the necessity to undertake them – most often due to the fact that without such solutions they lack possibility to match competition from foreign companies, as well as to match foreign business partners requirements. While Polish public administration institutions in practice still do not see the need for the BSS D&EP economic efficiency evaluation, in most cases as an argument giving the non-economic purposes of systems being implemented in this type of organizations. On the other hand, at the turn of the years 2008/2009 the results obtained with the use of the BSS D&EP effort estimation methods (approx. 53% of BSS providers surveyed in this cycle declared they commonly employed such methods) were more often used to estimate efficiency: there was an increase to approx. 36% of those using effort estimation methods. This applies to internal IT departments of Polish companies yet still it does not comprise public administration institutions. This increase may be explained first of all by stronger care about financial means in the times of recession, however it still leaves a lot to be desired. Meanwhile, to rationalize various BSS D&EP investment decisions, one may successfully use benchmarking data, having been collected in special repositories with intention to support effective and efficient execution of such projects.

III. CASE STUDY: DESCRIPTION OF THE PROBLEM

A company that was facing the need to choose an appropriate variant of BSS D&EP execution collects and processes, as a part of its basic activity, orders for certain goods from all over the world in a 24-hour mode, 7 days a week through: website, client service centres, fax and electronic mail (description of the case study taken from [15]). All those channels cooperate with the application, having been functioning in the company for a dozen or so years already, that is designed for orders processing and which no longer is able to satisfy present requirements since:

- Large part of processes is not automated, which requires additional work for registering orders and that generates losses.

- Current status of orders is not known therefore they are being lost, as a result of this other losses are also borne, which together with earlier mentioned losses are estimated to be approx. USD 5000 a day.
- System is expensive and difficult to maintain, with frequent malfunctions as it employs obsolete technology.
- System extends the time of delivering new products to the market, increases the risk of losing clients and lack of compliance with their requirements, slows down the growth of competitive advantage.

Thus the company has faced a decision on choosing variant of BSS D&EP execution that would:

- Eliminate the above mentioned drawbacks of the existing solution.
- Contribute to short- and long-term profits – that’s why the costs and duration of project are of great significance.
- Reduce the costs of functioning of both company and technology.
- Contribute to the reduction of risk, both in terms of business and technology.

Offers for each BSS D&EP variant were submitted, having approximate average values as shown in Table I.

TABLE I. PARAMETERS OF OFFERS CONCERNING EXECUTION OF PARTICULAR VARIANTS OF BSS D&EP CONSIDERED

Variant	BSS D&EP variant	Execution cost offered	Execution time offered
1	Development of new BSS from scratch using modern technologies	USD 10 million	3 years
2	Customization of BSS purchased	USD 5 million	2 years
3	Modernization of BSS used currently	USD 3,5 million	1,5 years

Source: Author’s analysis based on [15, p. 2].

Since each variant was backed by certain part of the board and key users, an analysis aimed at supporting decision-making process was carried out.

IV. CONCLUSIONS FROM THE BENCHMARKING DATA ANALYSIS

The analysis used benchmarking data for BSS D&EP having been collected in the following repositories:

- Standish Group, featuring data about over 70 thousands of the accomplished application software D&EP, which were analysed using the tool called *VirtualADVISOR* [15].
- Software Productivity Research (SPR), containing data from approx. 15 thousands of the accomplished application software D&EP, which were used to verify conclusions coming from Standish Group repository analysis with the use of *SPR Knowledge Plan* tool [16].
- International Software Benchmarking Standards Group (ISBSG), having collected data from approx. 5 thousands of the accomplished application

software D&EP [17], also used to verify findings coming from Standish Group repository analysis and also with the use of *SPR Knowledge Plan* tool, which at its present version offers possibility to import data from the ISBSG repository.

Priority was given to the Standish Group data and this being not only due to the size of this repository, objectivity of data (they come solely from clients) or the fact of IT branch appreciating its practical value [5] but also because they take into account an appropriate kind of client (in terms of branch and size of a company), appropriate kinds and size of BSS D&EP as well as appropriate type and size of application. Thus using the Standish Group repository made it possible to match all three kinds of BSS D&EP against the profile, with 90% match of the 120 attributes of more than 100 projects [15].

What’s also important, in their analyses the Standish Group employs clearly defined criteria of project classification, dividing projects into the following three groups (see e.g., [1][6][18]):

- Successful projects – that is projects completed with delivery of product having functions and features being in accordance with client requirements specification and within the estimated time and budget.
- Challenged projects – that is projects completed with delivery of product that is operating yet has fewer vital functions/features comparing to the client requirements specification and/or with overrun of the planned budget and/or duration.
- Failed projects – that is projects that were abandoned (cancelled) at some point of their life cycle or were completed with delivery of product that had never been used.

In the analysis of the Standish Group data, the following criteria were employed as equivalent for particular variants of the BSS D&EP considered:

- 1) Criterion of expected BSS D&EP effectiveness, including:
 - a) chance to succeed
 - b) level of planned costs overrun
 - c) level of planned duration overrun.
- 2) Criterion of expected BSS D&EP efficiency, including:
 - a) return on investment (ROI)
 - b) payback period.

Data presented in Table II clearly indicate that in the case being considered the highest chance to succeed is held by modernization variant, for which success coefficient is several times higher than that characteristic of variant consisting in development of new application, being only 4% (sic!), and significantly higher than that of COTS customization variant. Also in case of variant 3 the lowest percentage of projects ends with being abandoned – it is several times lower than in case of variant 1 and two times lower than in case of variant 2. What seems interesting, the highest percentage of projects that ended in partial failure (challenged projects) occurs in case of the customization of

COTS application. What's more, the average expected overrun of both costs (see Table III) and project duration (see Table IV) is also the highest in case of this project variant.

TABLE II. EXPECTED CHANCE TO SUCCEED FOR PARTICULAR VARIANTS OF BSS D&EP CONSIDERED

Resolution	Variant 1	Variant 2	Variant 3
Successful	4%	30%	53%
Challenged	47%	54%	39%
Failed	49%	16%	8%

Source: [15, p. 4].

Moreover, data in Table III clearly indicate that the average expected overrun of the planned costs for projects that ended in partial failure too is the lowest in case of variant 3. Also the lowest percentage of such projects overruns the costs by more than 50%. If offered costs and average expected overrun of these costs are taken into consideration when calculating the expected cost then it appears evident that the lowest expected cost of project execution applies to modernization variant.

TABLE III. EXPECTED LEVEL OF PLANNED COST OVERRUN FOR PARTICULAR VARIANTS OF BSS D&EP CONSIDERED (CHALLENGED PROJECTS)

Cost overrun	Variant 1	Variant 2	Variant 3
0% to 50%	64%	58%	75%
51% to 50%	36%	42%	25%
Average	44%	47%	34%
Offered cost	USD 10 million	USD 5 million	USD 3,5 million
Estimated cost	USD 14,4 million	USD 7,35 million	USD 4,7 million

Source: Author's analysis based on [15, p. 4].

Analogous conclusions may be drawn on the basis of the analysis of data presented in Table IV. Again, the average expected overrun of the planned duration for projects that ended in partial failure proves being the lowest for variant 3. Also the lowest percentage of such projects overruns the duration by more than 50%. If we take into account the offered duration and average expected overrun of this duration then we can see that the lowest expected duration of project execution applies to modernization variant too.

TABLE IV. EXPECTED LEVEL OF PLANNED DURATION OVERRUN FOR PARTICULAR VARIANTS OF BSS D&EP CONSIDERED (CHALLENGED PROJECTS)

Duration overrun	Variant 1	Variant 2	Variant 3
0% to 50%	57%	59%	80%
51% to 50%	43%	41%	20%
Average	44%	45%	29%
Offered duration	36 months	24 months	18 months
Estimated duration	52 months	35 months	23,5 months

Source: Author's analysis based on [15, p. 4].

Data shown in Table V clearly indicate that the highest percentage of projects characterised by the highest ROI can be found in case of variant 3 again. On the other hand, what's interesting is that projects with average ROI most often are projects consisting in developing new application from scratch while the lowest percentage of projects characterised by the lowest ROI can be found in case of customization variant.

TABLE V. EXPECTED ROI FOR PARTICULAR VARIANTS OF BSS D&EP CONSIDERED

ROI	Variant 1	Variant 2	Variant 3
High	11%	34%	52%
Average	66%	57%	37%
Low	23%	9%	11%

Source: [15, p. 5].

In Table VI both ROI and payback period for particular variants of the considered project were estimated in optimistic and pessimistic version. In the optimistic version it was assumed that the costs were identical with the offered costs while in the pessimistic version - that the costs were exceeded by the average values being expected for each variant analysed (see Table III). Based on these assumptions, both in optimistic and in pessimistic version, the highest 5-year gain applies to the modernization variant; also in case of that variant the payback period proves the shortest. It is worth noting that project in variant consisting in developing the new application would pay off after nearly 5 and half years in the optimistic version and after nearly 7 and half years in the pessimistic version.

TABLE VI. EXPECTED ROI AND PAYBACK PERIOD FOR PARTICULAR VARIANTS OF BSS D&EP CONSIDERED

Variant	Optimistic version			Pessimistic version		
	Costs (in \$ millions)	5-year gain (in \$ millions)	Payback period (in years)	Costs (in \$ millions)	5-year gain (in \$ millions)	Payback period (in years)
1	10	0	5,4	14,4	0	7,3
2	5	7,25	3,2	7,35	2,8	4,4
3	3,5	10,6	2,4	4,69	7,9	3,1

Source: Author's analysis based on [15, p. 5].

The above analysis clearly indicates that what in the considered case would be the best of the three BSS D&EP variants both from the perspective of the expected effectiveness and from the perspective of the expected efficiency is variant consisting in modernization of the application being used (variant 3).

V. THE EFFECTIVENESS AND EFFICIENCY FACTORS FOR THE RECOMMENDED VARIANT

In the analysed case, BSS D&EP consisting in modernization of application being used proves the most effective as well as the most efficient, what results, among others, from (see also [15]):

- Undertaking of such projects as a rule is a result of clearly defined needs of users therefore their goals are comprehensible, what undoubtedly promotes

users' engagement in the project and the board's support for the project, which, according to the list of success factors having been developed by the Standish Group since 1995, are still the two most important success factors [18].

- The fact that modernization projects do not require extensive analysis of requirements, numerous agreements, long-time training, changes of processes that would be destabilizing the work.
- Commonness of such projects thus the skills of executing them are high; what's more, projects of this type do not require additional skills in terms of project management, they rather require technical, the so called „hard”, skills.
- Present structure of project costs in terms of development activities, which due to the increased complexity of projects and ever more developed tools has changed and is now in inverse proportion to the structure as it was 25 years ago: now programming costs make up approx. 20% while other development works make up approx. 80% of the total cost.
- The fact that modernization projects are characterised by the lowest hidden cost (mainly user's time), estimated to be 15% of project costs versus 55% for variant 2 and versus 35% for variant 1.
- The discussed projects may be successfully carried out using agile approach, which also ranks high (sixth position) in the current list of success factors [18].
- Products smaller than those in case of developing application from scratch are developed as a result of the modernization projects and this is what increases their chance to succeed.
- The discussed projects do not have redundant requirements – as this is the case of the COTS customization where, according to the Standish Group data, less than 5% (sic!) of the features and functions get used [15], and of the development of new products (see Figure 1).

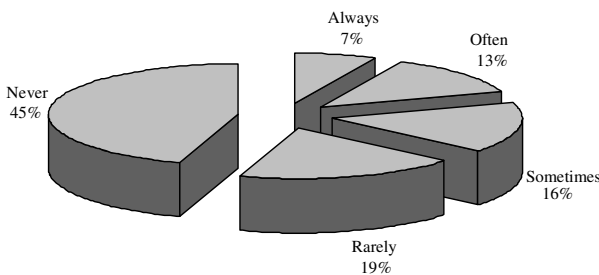


Figure 1. Average use of functions and features in the implemented software systems - custom development applications

Source: Author's analysis based on [15, p. 15].

However, variant recommended in the discussed case is not devoid of drawbacks though. Most of all, it evidently is not suitable for organizations where BSS have not functioned so far (in Poland approx. 95% of small companies do not use BSS – comparing to 50% in developed countries), for new organizations, new departments, and in case of fusion the modernization often ends in failure too. Moreover in modernization variant there are limited possibilities to implement fundamental business changes. What's more, the use of obsolete technologies is being continued, what makes cooperation with modern applications difficult, reduces usability, portability and maintainability of the modified application; performance is usually lower too. It is worth stressing that these attributes are the software product quality attributes of the ISO/IEC 9126 norm [19]. Thus what appears to be open to doubt is reduction of costs and difficulties in maintaining the system as well as technological risk - this being one of the major goals of the solution variant to be chosen (see Section 2). It is also worth mentioning that the ISBSG data indicate lower productivity of such projects: in case of BSS D&EP consisting in developing new BSS from scratch it ranges on average from 9 (for 4GL) to 24.5 (for 3GL) work hours for developing 1 function point (for more details about function points see [20]) whereas in case of modernization projects it takes approx. 27 work hours on average to develop 1 function point [21].

VI. CONCLUSION AND FUTURE WORK

Based on the analysis of benchmarking data coming from the Standish Group repository, having been carried out with the use of *VirtualADVISOR* tool, it was concluded that what proves the best among the three BSS D&EP variants *in the discussed case* is variant consisting in modernization of application being used. Data analysis indicates that choosing the above mentioned variant is rational due to the criterion of both expected effectiveness and expected efficiency of project. This conclusion has been confirmed by the verification based on the repository of the SPR and ISBSG data, having been carried out with the use of *SPR Knowledge Plan* tool.

From the point of view of effectiveness and efficiency, modernization variant has many advantages yet it is not devoid of drawbacks though. What's more, this does not have to be the best solution in other cases, e.g., for real time systems, for small software product development/enhancement projects, or for organizations that specialise in developing specific kind of new software systems where there is possibility to use the already written code. It should be also mentioned that projects of higher risk, i.e., those having lower chance to succeed, often happen to be more efficient.

As indicated by the study results discussed in this paper, in view of exceptionally low effectiveness of BSS D&EP it is necessary to rationalize investment decisions being made with regard to such projects. To do so one may successfully use ever richer resources of benchmarking data having been collected in repositories with intention to support effective and efficient BSS D&EP execution. In the opinion of T.C.

Jones: "For many years the lack of readily available benchmark data blinded software developers and managers to the real economics of software. Now (...) it is becoming possible to make solid business decisions about software development practices and their results (...). [Benchmarking – B.C.C.] data is a valuable asset for the software industry and for all companies that produce software" [22]. This paper presented the possibility of rationalization of investment decision concerning the choice of the BSS D&EP variant execution with the use of such data, illustrated on the basis of a case study.

REFERENCES

- [1] Standish Group, "CHAOS summary 2009", West Yarmouth, Massachusetts, 2009, pp. 1-4.
- [2] B. Czarnacka-Chrobot, "The economic importance of business software systems size measurement", Proc. of the 5th International Multi-Conference on Computing in the Global Information Technology (ICCGI 2010), 20-25 September 2010, Valencia, Spain, M. Garcia, J-D. Mathias, Eds., IEEE Computer Society Conference Publishing Services, Los Alamitos, California-Washington-Tokyo, 2010, pp. 293-299.
- [3] T. C. Jones, Patterns of software systems failure and success, International Thompson Computer Press, Boston, MA, 1995.
- [4] PCG, "2008 ERP report, topline results", Panorama Consulting Group, Denver, 2008, pp. 1-2.
- [5] J. Johnson, "CHAOS rising", Proc. of 2nd Polish Conference on Information Systems Quality, Standish Group-Computerworld, 2005, pp. 1-52.
- [6] Standish Group, "CHAOS summary 2008", West Yarmouth, Massachusetts, 2008, pp. 1-4.
- [7] Economist Intelligence Unit, "Global survey reveals late IT projects linked to lower profits, poor business outcomes", Palo Alto, California, 2007: <http://www.hp.com/hpinfo/newsroom/press/2007/070605xa.html> (11.07.2011).
- [8] B. Czarnacka-Chrobot, "Evaluation of business software systems development and enhancement projects effectiveness and economic efficiency on the basis of functional size measurement", Proc. of the 10th International Conference on Software Engineering Research and Practice (SERP 2011), The 2011 World Congress in Computer Science, Computer Engineering & Applied Computing (WORLDCOMP'11), H. R. Arabnia, H. Reza, L. Deligiannidis, Eds., CSREA Press, Las Vegas, Nevada, USA, July 2011, in press.
- [9] A. Brown, "IS evaluation in practice", The Electronic Journal Information Systems Evaluation, vol. 8, no. 3, 2005, pp. 169-178.
- [10] E. Frisk and A. Plantén, "IT investment evaluation – a survey of perceptions among managers in Sweden", Proc. of the 11th European Conference on Information Technology Evaluation, Academic Conferences, 2004, pp. 145-154.
- [11] Z. Irani and P. Love, "Information systems evaluation: past, present and future", European Journal of Information Systems, vol. 10, no. 4, 2001, pp. 183-188.
- [12] S. Jones and J. Hughes, "Understanding IS evaluation as a complex social process: a case study of a UK local authority", European Journal of Information Systems, vol. 10, no. 4, 2001, pp. 189-203.
- [13] A. J. Silvius, "Does ROI matter? Insights into the true business value of IT", The Electronic Journal Information Systems Evaluation, vol. 9, issue 2, 2006, pp. 93-104.
- [14] B. Czarnacka-Chrobot, "Analysis of the functional size measurement methods usage by Polish business software systems providers", in Software Process and Product Measurement, A. Abran, R. Braungarten, R. Dumke, J. Cuadrado-Gallego, J. Brunekreef, Eds., Proc. of the 3rd International Conference IWSM/Mensura 2009, Lecture Notes in Computer Science, vol. 5891, Springer-Verlag, Berlin-Heidelberg, 2009, pp. 17-34.
- [15] Standish Group, "Modernization – clearing a pathway to success", West Yarmouth, Massachusetts, 2010, pp. 1-16.
- [16] Software Productivity Research: <http://www.spr.com/spr-knowledgeplanr.html> (11.07.2011).
- [17] ISBSG, "Data demographics release 11", International Software Benchmarking Standards Group, Hawthorn, Australia, June 2009, pp. 1-24.
- [18] Standish Group, "The CHAOS manifesto", West Yarmouth, Massachusetts, 2009, pp. 1-54.
- [19] ISO/IEC 9126 Software Engineering – Product Quality – Part 1-4, ISO, Geneva, 2001-2004.
- [20] B. Czarnacka-Chrobot, "The effectiveness of business software systems functional size measurement", Proc. of the 6th International Multi-Conference on Computing in the Global Information Technology (ICCGI 2011), 19-24 June 2011, Luxemburg City, Luxemburg, Constantin Paleologu, Constandinos Mavromoustakis, Marius Minea, Eds., International Academy, Research, and Industry Association (IARIA), Wilmington, Delaware, USA, 2011, pp. 63-71.
- [21] Ch. Symons, "The performance of real-time, business application and component software projects", Common Software Measurement International Consortium (COSMIC) and ISBSG, September 2009, pp. 1-45.
- [22] ISBSG: <http://www.isbsg.org> (11.07.2011).

Towards Functional and Constructional Perspectives on Business Process Patterns

Peter De Bruyn, Dieter Van Nuffel, Philip Huysmans, Herwig Mannaert

Department of Management Information Systems

University of Antwerp

Antwerp, Belgium

{peter.debruyen,dieter.vannuffel,philip.huysmans,herwig.mannaert}@ua.ac.be

Abstract—Contemporary organizations need to be more agile to keep up with the swiftly changing business environment. The Normalized Systems theory has proven to introduce this required agility within an organization, starting at the software level. However, in order to realize an agile enterprise, also business processes have to exhibit this evolvability. Currently, the relevance of Normalized Systems theory at the business process level has been demonstrated, however no equivalent to the software elements at the organizational level have been developed. Therefore, this paper investigates whether it is possible to base such elements on the available business process patterns in literature. After investigating the usefulness of the MIT Process Handbook with regard to this purpose, this paper emphasizes the importance of recognizing the so-called functional–constructional gap and identifies the need for developing modular and evolvable constructional business process design patterns to further extend Normalized Systems theory on the business level.

Index Terms—Normalized Systems, business process patterns, analysis patterns, evolvability, MIT Process Handbook

I. INTRODUCTION

Contemporary organizations need to be more agile to keep up with the swiftly changing business environment. As a consequence, all constructs of an organization—structure, business processes, information systems—have to evolve at an equivalent pace. The Normalized Systems (NS) theory has proven to introduce this required agility within an organization. First, the theory prescribes how to design and implement information systems that are able to evolve over time, and are thus designed to accommodate change [1]. It is based on the systems theoretic concept of stability and on the prevention of so-called combinatorial effects, i.e., changes of which the impact is not only dependent on the kind of the change but also on the size of the system. As such, NS proposes four design principles that need to be adhered at all times [2]:

- *separation of concerns* requires that every change driver or concern is separated from other concerns;
- *data version transparency* requires that data is communicated in version transparent ways between components;
- *action version transparency* requires that a component can be upgraded without impacting the calling components;
- *separation of states* requires that actions or steps in a workflow are separated from each other in time by keeping state after every action or step.

The design principles show that software constructs, such as functions and classes, by themselves offer no mechanisms to accommodate anticipated changes in a stable manner. The NS theory therefore proposes to encapsulate software constructs in a set of five higher-level software elements: action element, data element, workflow element, trigger element, and connector element [3]. These elements are modular structures that adhere to these design principles, in order to provide the required stability with respect to anticipated changes [2]. As these elements themselves are free of combinatorial effects, also the applications based on them are free of combinatorial effects.

However, it does not suffice to introduce agility within the information systems to realize an agile enterprise. Other organizational artifacts have to evolve in the same way as well. Therefore, the NS theory was extended to other organizational elements, such as business processes and enterprise architectures [4]. Regarding the former, business processes, the applicability of the extension is already demonstrated [5].

Nevertheless, the authors have not yet been able to identify an equivalent of the five software elements at the business process level. Although preliminary research findings indicate that Notification and Payment might classify as such a business process element [5], additional research is required. Therefore, this paper investigates whether it is possible to base such elements on the available business process patterns in literature.

When selecting appropriate business process elements, an important distinction needs to be made between patterns from a functional and the constructional perspective. The functional and constructional view on a system are fundamentally different conceptualizations of a system [6]. The *functional perspective* is concerned with the external behavior of the system [7]. This perspective is adequate for the purpose of using or controlling a system. Therefore, knowledge of the required input variables, transfer function and output variables are key components of this perspective. In contrast, the *constructional perspective* describes what a system really is [8]. In this perspective, knowledge about the composition (i.e., which components constitute the system) and structure (i.e., how these components are related) is focused on. The function of a system is brought about by the operation of its construction. However, the construction cannot be deduced from the functional description, since the two perspectives deal

with fundamentally different components. Consistent with the NS elements, we aim to propose process elements using a constructive perspective. Therefore, it is important to consider these perspectives when building on business process patterns found in literature.

The remainder of this paper will be structured as follows: in Section II we will briefly discuss some already existing analysis and design patterns in literature. Next, we will investigate to which extent we can derive corresponding NS conform elements and patterns from those available business process patterns by studying one frequently cited and used framework, being the MIT Process Handbook, in Section III. Afterwards, we will discuss how NS theory extends towards normalized business process design patterns and Section V will end up with some final comments and opportunities for further research.

II. RELATED WORK

The use of patterns in software development and information systems analysis has been increasingly gaining attention during the past decade. One of the first publications on software development patterns that generated considerable interest was probably the so-called Gang of Four (GoF) book of Gamma et al. [9]. Gamma et al. identified patterns as ideas that senior developers have used many times while solving commonly occurring problems [9]. Essentially, the overall meaning or intention of this concept has remained rather unchanged throughout many later publications on design patterns. Further summarizing, patterns are frequently claimed to exhibit the following characteristics:

- starting from a generally occurring problem in the considered problem domain;
- proposing standard and / or best practice solutions to these problems applicable to a myriad of analogous situations;
- incorporating domain knowledge and expertise sometimes requiring multiple years of experience to gather independently;
- exhibiting high-quality and robustness by representing frequently tested solutions;
- increasing pace of the development / modeling process by avoiding to systematically start from scratch and trying to ‘reinvent the wheel’.

Moreover, they are generally claimed to be a sign of a discipline becoming somewhat more mature, in the sense that an accumulation of generally recurring problems and their best-practice solutions becomes identified and documented. As such, existing knowledge from experts can be consolidated, published, and made available for a whole community [10].

After the work of Gamma et al. [9], additional, more analysis-oriented frameworks arised. As such, we will present here a brief illustrative, yet not exhaustive, overview. Given the plethora of available frameworks, a lot of different classification approaches exist as well. Some pattern frameworks for example mainly focus on the data aspects of an organization model, such as Hay [11] and Fowler [12]. Elaborating on

these previous two frameworks, an interesting work was also delivered by Silverston providing domain data models for several industries such as manufacturing, telecommunications, health care, insurance, etc. [13], [14]. More process-oriented patterns can be found in, for example, Larman [15]. Furthermore, some claim that the broad area of workflow patterns are to be considered as some form of process-oriented analysis patterns (see e.g., [16]). Scheer also provided domain models for several functional domains of a typical organization, and combined both data and process related aspects [17]. Moving to more abstract levels, some frameworks claiming to state more generic and universal patterns can be noticed. For example, Dietz models every enterprise as an aggregation of instantiations of one universal transaction pattern [18]. Also REA (resources, events, agents) similarly views an enterprise as an aggregation of transactions representing some kind of economic exchange [19], [20]. Finally, one could argue that also some general reference models could be considered to a certain extent as analysis patterns, such as the value chain model of Porter [21], the eTOM model for the telecommunications sector or the SCOR model representing a reference model for supply chain operations [22].

III. INVESTIGATING CURRENT BUSINESS PROCESS DESIGN AND REFERENCE MODELS

In this section we will discuss the extent to which currently available analysis patterns and reference models can be applied to and serve as a means to extent the previously discussed NS theory to the level of normalized design patterns at the business level (i.e., modular and evolvable business process design patterns). As the number of available frameworks in this regard is rather extensive at first sight (cf. Section II) and due to the limited available space, we chose to focus our attention initially to only one framework that formulates a number of functional patterns at the business process level: the MIT Process Handbook. This approach allowed us to analyze this one specific framework in a rather profound way. The pattern framework was selected mainly because of the fact that it is a generally well known, publicly available framework, extensively discussed and referred to in both academic and practitioners literature. Also, as will be further clarified later on, this framework’s motivation seems to be closely resembling our previously stated purpose in Section I: the formation of a repository consisting out of generally reusable business process patterns. As such, the purpose is to investigate whether these functional patterns can be translated in the required modular and evolvable design patterns.

The MIT Process Handbook initiative originated around 1994 reacting to an identified need of enabling more easily business process redesign and the knowledge management regarding those business processes [23]. As such, the purpose of the project was to identify similarities between and alternatives regarding different business processes at various organizations [23]. This resulted in an online available “process handbook” to exchange ideas regarding organizational practices ending

up with a “repository” of knowledge about business processes and featuring more than 5900 entries in July 2002 [24].

In its very essence, the MIT Process Handbook structures its business process repository around two main dimensions: parts and types. Process *parts* represent the fact that a business process can be subdivided into several “sub” business processes or activities as for example the “sell product” business process is broken down into the more detailed processes like “identify potential customers”, “inform potential customers”, “obtain order”, “deliver product”, etc. Process *types* represent different alternatives or “specializations” of a generic activity, as for example the processes “sell by mail order” and “sell in retail store” can be considered as two specializations of the generic process “sell product”. Using these two dimensions to situate the different business processes then results in the so-called “Process Compass” where processes are depicted on the vertical axe according to their different parts / subactivities and on the horizontal axe according to their different types / alternatives [23].

Clearly, this way of working already implies a certain amount of modularity: in a top-down way, the general, more “high-level” activities are constantly broken down into more detailed constituent subactivities, representing the respective modular “building blocks”. In addition, the different types / alternatives available for certain processes suggest the possibility of being able to compose new processes in a kind of “plug and play” manner: for each subactivity, frequently some “equivalent” types are proposed, apparently allowing the designer of a new or ameliorated process to choose and trade-off between different alternatives or replace on a later time an existing activity by an alternative “version” of that subactivity.

In order to assess whether the Handbook can provide sufficient support regarding our attempt to extend our framework on modularity and evolvability to patterns on the business level, let us start for example on the overall activity of the Process Handbook (i.e., the most generic and high-level one, claimed as being the basis for most business processes): “produce as a business”. A schematic overview of this process is provided in Figure 1. One can notice that the business process “produce as a business” has five parts (i.e., design product and process, buy, make, sell and manage a business). The figure moreover is partially expanded for the parts “design product and process” and “sell”. When trying to leverage these business processes and parts to our constructional building blocks created at the software level in NS theory, three somewhat related issues arise: under-specification, lack of adherence to prescriptive design principles and an inherent top-down approach.

A. Underspecification

The repository of the Process Handbook regularly seems to be lacking highly specified and detailed descriptions of its different processes and activities. Frequently, some relevant subactivities seem to be omitted and the structure of decomposition already stops before one has attained a very fine-grained modular overview of all the needed, broken-down,

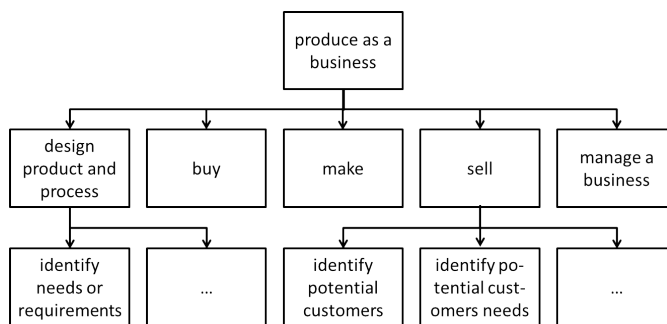


Fig. 1. Partial breakdown of the “Produce as a business” business process based on the MIT Process Handbook

activities which are required for exhaustively executing a more generic high-level process. Consider for example the subprocess “identify needs or requirements” as shown in Figure 1. This subprocess has been given the rather vague essential description that it is a process for “identifying the usability parameters of a resource that is managed in a flow dependency” and it is already situated at the bottom of the Process Compass. Therefore, no further subactivities are identified in the Process Handbook. However, one could reasonably argue that this process can still be refined into more fine-grained activities such as, e.g., conducting a market survey, analyzing preceding sales results, etc, which could then be detailed into even more specific subprocesses. As such, the process descriptions frequently leave considerable room for interpretation about the actual specific activities entailed in certain processes.

Indeed, it has not been the objective of the initiators of the Process Handbook to provide such a very fine-grained overview of each process. While Malone et al. mention in their introduction that their Handbook is expected to be useful in automatically generating software, they argue later on that as their main focus is to support human decision-makers “*there is no requirement that all our process descriptions be detailed or formalized enough to be executable by automated systems*” [23, p. 426]. Explicitly referring to Hammer and Champy’s [25] concept of analysis paralysis, they further claim that it is more important to be able to make a rapid assessment of the basic characteristics of a process, rather than an elaborate and detailed overview. Proposing an approach to further fill in the Process Handbook, Pentland et al. [26, p. 3] also emphasize that it is “*pointless to spend a lot of energy mapping out [in a detailed way] how a particular activity is accomplished*”. However, in order to directly apply a NS approach, it is necessary to break down the action entities up to the point that it enables the identification of each individual concern which can potentially be considered as a separate “change driver” [1]–[3]. Also applying the NS principles at the business level requires the rather fine-grained identification of such individual change drivers related to individual elementary life cycle information objects, albeit that they can depend on time, context and subjective interpretation [4], [5].

B. Lack of adherence to prescriptive design principles

Also few or no applied prescriptive design guidelines can be retrieved towards the design of the process repository. Previous research regarding evolvable modularity at the software level [1]–[3] and the business level [5] however points out that some very stringent principles should consistently be adhered to with this respect. Consider for instance again the example previously outlined and depicted in Figure 1. “Identify needs or requirements” is a subprocess of “design product and process”, which is at its turn a part of the general process “produce as a business”. “Identify potential customer needs” is a subprocess of “sell” which is at its turn a part of the same general process “produce as a business”. While not completely clear from the descriptions delivered by the Handbook, one could argue that certain functionality is common or repeated within these two distinct processes (each having no ‘lower’ subprocesses / parts in the process compass). Specializing the “sell” process towards “sell via electronic store” has a subprocess “identify customer needs in electronic store”, which again seems to cover at least some common functionality, apart from the employed distribution channel. Surprisingly, the “identify customer needs in electronic store” process is subdivided into several other parts, not reused in the processes “identify needs or requirements” or “identify potential customer needs”. Similar instances of common functionality scattered around in various business processes were noticed regarding the delivery of products, advertising via diverse channels, etc. In terms of evolvability, this would imply that the need for e.g., changing something in the way customer needs are identified, could have a possible impact on all business processes involved with this functionality. This evidence suggests that at least one of the four basic principles of NS theory is not adhered to. The principle of Separation of Concerns namely enforces one to separate each individual change driver in its own distinct construct in order to encapsulate and limit the impact of a change driver in its own construct. More specifically, NS theory demands that during the design of evolve software or business processes, common parts of data or business processes should be kept in a non-redundant form, whereas each variation should be kept separated from the common part.

This lack of unambiguous and stringent prescriptive design guidelines to compose the Process Handbook is not only apparent when analyzing the respective processes, but also while studying the guidelines directed to the users of and contributors to the Handbook. For example, it is stated that: “we believe that [... our structure] is comprehensive and intuitive [...] we have, in general, tried to maintain a branching factor of about “7 plus or minus 2” in the specialization hierarchy [...] and use it] primarily as a rough guideline for editing the Process Handbook” [24, p. 250]. Later on, one can read: “wherever possible, we have tried to create groupings that constitute a mutually exclusive and exhaustive partitioning of the possible specializations of that activity” [24, p. 250]. Finally, Malone et al. [23, p. 439] even explicitly mention that their Handbook is primarily a resource to suggest people what

to do rather than proposing “prescriptive rules” in drafting the Handbook and are confident in the “*role of intelligent human “editors” to select, refine, and structure the knowledge represented in the Handbook to tackle the editorial challenge.*”

C. Top-down modeling

Finally, the systematic way of structuring the different business processes in a top-down fashion (i.e., starting from general high-level processes and then refining them into more detailed ones including some possible alternatives) inherently contributes to the functional–constructional gap as mentioned in Section I.

The breakdown of the “produce as a business” business process can illustrate this point. From a functional point of view, one tries to relate the input variables to the output variables through a transfer function. Knowing the transfer function, insight is gained in how the systems responds to various instances of input parameters from the environment. In this case, one tries to understand which resources (input variables) are required to deliver products or services (output variables) to the customers. However, such a transfer function is generally extremely complicated. Therefore, the technique of functional decomposition can be applied to reduce this complexity. Using functional decomposition, the transfer function is replaced by a set of sub-systems of which the transfer function is easier to understand. In the “produce as a business” business process, design, buy, make, sell, and manage are identified as sub-systems. Consequently, one now has to understand the inputs and outputs of, for example, the “sell” transfer function. However, these systems are still considered using a functional perspective: together, they describe in more detail how resources can be converted to products or services. On these functionally decomposed processes, one can again apply functional decomposition, resulting in very detailed descriptions of transfer functions. In traditional architecture literature, this process is referred to as *analysis* [27]. However, this activity is radically different from *designing* a structure which brings about this transfer function. When designing process elements, we aim to describe the structure to bring about the business process functionality. In design studies, this activity is referred to as *synthesis* [28]. Unsurprisingly, the design of a system which brings about the “produce as a business” function will be very complex. Analogously to functional decomposition, constructional decomposition can then be applied. However, the elements which are identified in a constructional decomposition are different in nature than the elements from a functional decomposition. Consequently, it does not make sense to try to relate the elements of a functional decomposition to the elements of a constructional decomposition. In other words, one cannot expect to arrive at essential constructional process building blocks by describing very detailed functionally decomposed elements, as proposed by the MIT Process Handbook.

Indeed, as Kodaganallur [10] seems to be noticing rightfully, few development methodologies seem to be seamlessly integrated with the use of patterns at the analysis level. However,

using a systematic integrated bottom-up approach starting from data and action entities, encapsulating them in higher-order elements with proven stability and finally aggregating them into evolvable business process (patterns) is the inherent rationale in the NS theory. In our view, designing organizations towards (proven) evolvability requires such a bottom-up approach where in a first phase some very fine-grained building blocks exhibiting proven evolvability by adhering to the predefined principles (e.g., “send notification”) are developed. Later on, these fine-grained building blocks can be reused in a safe and black-box way to construct more coarse-grained blocks, again conforming to the predefined design principles (e.g., “decide on customer creditworthness”). Only in a final phase these coarse-grained blocks could be aggregated towards such general processes as depicted in Figure 1 and enabling the transformation of, for instance, “sell product” to NS compatible patterns. Again, this suggests that the MIT Process Handbook is not directly applicable with respect to evolvable and modular constructional business process patterns.

These three issues are obviously highly related to one another and add up to the conclusion that the MIT Process Handbook primarily gives a *functional* overview of possible business processes in an enterprise. When designing new business processes or trying to ameliorate existing ones, the Process Handbook gives of an overview of some available options regarding the general functional elements such a business process should or could incorporate without giving any further compelling guidance on the way in which they should be aggregated or structured. Additionally, it still leaves (consciously) some room for interpretation regarding the specific activities and change drivers incorporated in each activity or business process. As such, the Process Handbook can (and maybe should) be considered as primarily a kind of reference model containing mainly domain knowledge on broad aspects affecting many enterprises: HR, supply chain management, marketing, etc. It can signify a considerable help and contribution when one is indeed looking for the functionalities common business processes have to perform. However, the Process Handbook does not adhere to specific modularity and evolvability guidelines and was essentially also not established with that main purpose in mind.

IV. THE NEED FOR MODULAR AND EVOLVABLE CONSTRUCTIONAL BUSINESS PROCESS DESIGN PATTERNS

In the previous section, we argued that the MIT Process Handbook can have considerable value regarding the functional analysis and decomposition of business processes, but that it can not be simply translated to existing NS software elements for several reasons. Without explicitly discussing several other existing and possibly relevant reference frameworks or claimed business process patterns, it seems reasonable to argue that some of the aforementioned arguments can be applied to multiple other existing frameworks as well: several of them indeed emphasize the modeling of best-practices and functional requirements in a top-down way. As such, a first important

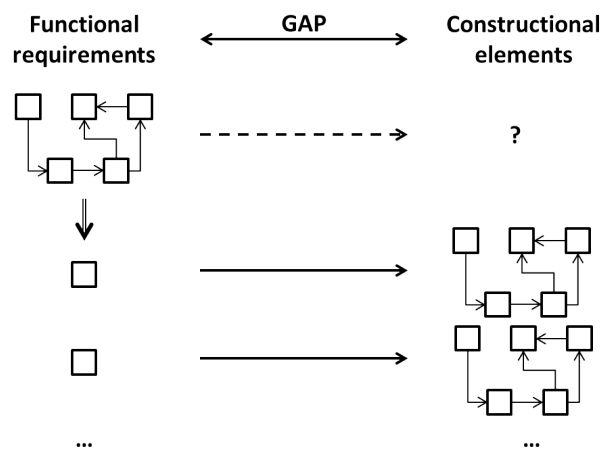


Fig. 2. A visualization of the functional–constructional gap and the need for modular and evolvable constructional business process design patterns

conclusion to be made is that this functional–constructive gap is again clearly been proven to be present and offers considerable challenges to unifying models on the business level with those on the software level. However, the existence of this gap is frequently underestimated or even not mentioned or addressed at all by many current methodologies. Obviously, the question then remains how to indeed develop modular and evolvable business process design patterns, focusing on constructive components instead of functional components.

Based on the NS theory rationale, Mannaert et al. [1] have already emphasized the existence of the functional–constructional gap and the importance of adhering to certain design principles when developing information systems. More specifically they suggest to consider the development of an information system as a *linear transformation* of a set of functional requirements (i.e., data entities, action entities and connectors) into a set of instantiations of software constructs (i.e., data structures and processing functions) at a certain point in time. Also, they show that this transformation can be quite straightforward when one is studying information systems from a static perspective. However, when focusing on the dynamic perspective (i.e., incorporating a marginal transformation of a set of additional functional requirements into a set of additional instantiations of software constructs) it is easy to show that the impact of a single ‘extra’ functional requirement is not necessarily limited to the addition or modification of a single software primitive. Stated otherwise, the impact of 1 functional change can have impact *n* on the constructional side, thus showing instability.

In order for this instability to be avoided, the NS rationale would require to first decompose the complex (high-level) functional requirements into a kind of more basic requirements. The next step would then be to look for Normalized linear transformations where — at least part of — the basic *functional* requirements can be deterministically transformed into *constructional* business process patterns, thus allowing a 1

to 1 mapping. Ideally, when representing this transformation in matrix form, the transformation matrix should indeed be of a diagonalized form or at least of a Jordan normal form. Finally, this reasoning is also visually depicted in Figure 2: instead of searching for business process patterns on the functional view, NS proposes to try to decompose these functional requirements to very basic ones and subsequently considering the transformation to constructive business process patterns.

These transformations should then again be analyzed from both a static and a dynamic perspective. Ideally, these transformations should be linear and normalized. This would entail from a static perspective that the realization of functional requirements, including the possible insufficiencies, could be located in a bounded and identifiable set of constructional primitives. From a dynamic perspective, this would entail that an increase in an existing functional requirement, or the addition of a functional requirement, would have a bounded impact on the constructional view.

V. CONCLUSION AND FUTURE WORK

This paper investigated the extent to which currently available business process patterns can be applied in order to develop ‘normalized’ elements at the business process level, with the purpose to further extend the NS theory at the business level. Focusing our attention primarily on the MIT Process Handbook, three issues arose as hampering the application of this framework directly to NS elements: under-specification, lack of adherence to prescriptive design principles and an inherent top-down approach. Regarding the latter, it was noted that this aspect is strongly related to the concept of the so-called functional–constructional gap, in that lots of the existing pattern frameworks seem to provide *functional* decomposition. Therefore, they can be useful in managing domain knowledge and expertise, but are not directly transformable to *constructional* primitives at the NS level. Hence, the necessity for future identification of modular and evolvable constructional business process design patterns was called for and a NS theory based approach was proposed for studying the needed transformation between functional and constructional patterns in a dynamic context, emphasizing the need for evolvability.

A limitation of this paper is that it focused its attention into discussing only one existing framework in a rather detailed way. While we expect our proposed reasoning regarding the important, yet frequently underestimated functional–constructional gap to be applicable to many, if not most currently available pattern frameworks, some future research could then obviously investigate the extent to which our conclusions can also be applied to other existing frameworks.

Other related research at our research group will clearly be aimed at trying to find those necessary constructional business process elements. Previous research suggested Notification and Payment as potential business process design patterns, however further research and extension is definitely needed.

ACKNOWLEDGMENTS

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

REFERENCES

- [1] H. Mannaert, J. Verelst, and K. Ven, “The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability,” *Science of Computer Programming*, vol. 76, no. 12, pp. 1210–1222, 2010.
- [2] —, “Towards evolvable software architectures based on systems theoretic stability,” *Software Practice and Experience*, vol. Early View, 2011.
- [3] H. Mannaert and J. Verelst, *Normalized systems: re-creating information technology based on laws for software evolvability*. Koppa, 2009.
- [4] D. Van Nuffel, H. Mannaert, C. De Backer, and J. Verelst, “Towards a deterministic business process modeling method based on normalized systems theory,” *International Journal on Advances in Software*, vol. 3, no. 1-2, pp. 54–69, 2010.
- [5] D. Van Nuffel, “Towards designing modular and evolvable business processes,” Ph.D. dissertation, University of Antwerp, 2011.
- [6] G. M. Weinberg, *An Introduction to General Systems Thinking*. Wiley-Interscience, 1975.
- [7] L. Bertalanffy, *General Systems Theory: Foundations, Development, Applications*. New York: George Braziller, 1968.
- [8] M. Bunge, *Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems*. Boston: Reidel, 1979.
- [9] E. R. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns – Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.
- [10] V. Kodaganallur and S. Shim, “Analysis patterns: A taxonomy and its implications,” *Information Systems Management*, vol. 23, no. 3, pp. 52–61, SUM 2006.
- [11] D. C. Hay, *Data Model Patterns: Conventions of Thought*. Dorset House, 1995.
- [12] M. Fowler, *Analysis Patterns: Reusable Object Models*. Addison-Wesley Professional, 1996.
- [13] L. Silverston, *The Data Model Resource Book: v.1: A Library of Universal Data Models for All Enterprises: Vol 1*. John Wiley & Sons, 2001.
- [14] —, *The Data Model Resource Book: A Library of Universal Data Models by Industry Types: v. 2*. John Wiley & Sons, 2001.
- [15] C. Larman, *Applying UML and Patterns*. Prentice Hall PTR, 1997.
- [16] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, “Workflow patterns,” *Distributed and Parallel Databases*, vol. 14, pp. 5–51, 2003.
- [17] A. Scheer, *Business Process Engineering – Reference Models for Industrial Enterprises*. Springer-Verlag, 1998.
- [18] J. L. G. Dietz, *Enterprise Ontology: Theory and Methodology*. Springer, 2006.
- [19] W. McCarthy, “The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment,” *The Accounting Review*, vol. 57, no. 3, pp. 554–578, 1982.
- [20] P. Hruby, J. Kiehn, and C. Scheller, *Model-Driven Design Using Business Patterns*. Springer, 2006.
- [21] M. Porter, *Competitive Advantage: Creating and Sustaining Superior Performance*. Free Press, 1998.
- [22] Supply Chain Council (SCC), “Supply Chain Operations Reference Model (SCOR): Version 10.0.”
- [23] T. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C. Osborn, A. Bernstein, G. Herman, M. Klein, and E. O’Donnell, “Tools for inventing organizations: Toward a handbook of organizational processes,” *Management Science*, vol. 45, no. 3, pp. 425–443, MAR 1999.
- [24] T. Malone, K. Crowston, and G. Herman, *Organizing Business Knowledge: The MIT Process Handbook*. The MIT Press, 2003.
- [25] M. Hammer and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperCollins, 1993.
- [26] B. Pentland, C. Osborn, G. Wyner, and F. Luconi, “Useful descriptions of organizational processes: Collecting data for the process handbook,” August 1999, Unpublished Working Paper. Center for Coordination Science, MIT, Cambridge, MA.
- [27] C. Alexander, *Notes on the Synthesis of Form*. Harvard University Press, 1964, ISBN: 0674627512.
- [28] J. S. Gero and U. Kannengiesser, “The situated function-behaviour-structure framework,” *Design Studies*, vol. 25, no. 4, pp. 373–391, 2004.

Practical Experiences with Software Factory Approaches in Enterprise Software Delivery

Alan W. Brown, Ana Lopez Mancisidor, Luis Reyes Oliva
 IBM Rational
 Sta Hortensia, 26-28, Madrid, Spain
 (alanbrown, ana.lopez, luis.reyes)@es.ibm.com

Abstract—There are many pressures on software delivery organizations to produce more software faster in the context of extreme cost pressure and growing globalization of the software delivery organization. The concept of a *Software Factory* is beginning to emerge as one way to address these challenges. This paper discusses the principles of software factories for enterprise software delivery using practical examples that explore the how software delivery quality can be managed across the software supply chain. In particular, we discuss two case studies where large commercial organizations have achieved significant improvements in software quality when adopting a software factory approach. We conclude with a set of observations that highlight where this work can be usefully refined and extended.

Keywords – software engineering; software process improvement; application management; software delivery

I. INTRODUCTION

Many companies have experienced a great deal of change over the past few years due to evolution of the business environment, financial upheavals, societal changes, and technical advancement. Key to addressing these changes has been analysis of core business processes to see how they can be refined and optimized, followed by a restructuring of those business processes to better meet the new context. This business process reengineering has helped to refocus on the most compelling and valuable aspects of the business, and is a first step in readjusting investment priorities toward those business activities that are considered essential, while looking to divest those considered secondary [1].

At the same time all IT groups have been forced to lower operating costs across the organization. The direct implication is that they must not only minimize waste and inefficiency, but increase productivity and relevance to the businesses they serve.

This combination of business process restructuring and close focus on delivery efficiency have been seen in many business domains, and have resulted in techniques such as “lean manufacturing”, “supply-chain management”, and “product line engineering”. The application of these ideas in software delivery is what we refer to here as a “software factory approach” to enterprise software delivery [2, 3].

In this paper we examine this view of enterprise software delivery. We first explore the idea of the “software supply chain” and introduce the concept of the software factory. We

then detail the characteristics of the software factory approach, and illustrate those concepts using real world examples. We conclude with several key observations.

II. ELEMENTS OF A SOFTWARE FACTORY APPROACH

Analogous with changes in the industrial sector, a software factory approach to enterprise software delivery aims to reduce time to market for new products, increase flexibility and agility in component assembly, and reduce costs of production while increasing quality and end-user satisfaction. It is important to highlight several key elements of such an approach that impact enterprise software delivery.

A. Aligning business and engineering

A software factory approach to enterprise software delivery requires a well-established, multiplatform process with tooling that aligns business strategy with engineering and system deployment. Critical in building applications that meet the needs of the customer, such processes can help to identify business needs and stakeholder requirements, and drive those business goals into enterprise software delivery projects and solutions, ensuring that the final product meets the business objectives with the lowest possible cost and highest possible quality.

B. Automating processes and tasks

Automating the enterprise software delivery lifecycle can help reduce errors and improve productivity, leading to higher quality products. An integrated portfolio of tools can help teams automate specific, labor-intensive tasks—similar to the way automation is used to perform repetitive manual tasks in manufacturing. Using automation, practitioners are able to focus on creating more innovative solutions with industry-leading design and development environments that help support the delivery of high-quality, secure and scalable products. Companies that invest in automation and a more efficient means of production and delivery can experience a sizeable jump in productivity, quality, time to market and scalability.

C. Leveraging assets across the enterprise

Modern architectural and product development frameworks can be considered complex supply-chains that integrate third-party, custom, off-the-shelf and outsourced components in the overall software or system. This has led to approaches such as Service-Oriented Architecture (SOA)

frameworks, which focus on assembly of standard components to promote reuse across the enterprise, and more generally to product line engineering (PLE), where the focus is strategic reuse in developing portfolios of similar products that share many components but are differentiated by variations in features and functions [4].

The first step is to understand what assets exist and leverage them to create reusable components to extend architectural frameworks in meaningful, predictable ways.

D. Supporting integration and lean processes

Today’s enterprise software delivery teams can be highly distributed geographically. Consequently, they need flexible and agile processes with real-time collaboration, integrated across disparate platforms, roles and geographies to reap the benefits of modern software and systems frameworks. Globally distributed development can be facilitated through defined, customizable processes and best practices to support flexibility, mitigate risk with comprehensive quality management and enhance developer productivity through task and process automation.

E. Automating operational measurement and control

To help ensure predictable outcomes, the enterprise software delivery process must be governed so it can be continuously measured and improved. A fundamental aspect of this is the definition and codification of processes for developing products. These processes and best practices are corporate assets, and need to be captured in an actionable form so teams can be guided to adhere to appropriate best practices through automated workflows.

Relevant metrics should be gathered automatically at each step, including after software and systems are delivered into production. By constantly, automatically measuring the specific key value aspects of processes, these metrics can provide insight into the efficacy of existing processes and identify areas for improvement. Automated measurement and control is also critical in tightly regulated industries, such as government, aerospace, medical or financial sectors.

III. REALIZING A SOFTWARE FACTORY

Realizing a software factory requires a blueprint to organize and structure the methods and tools that deliver the capabilities to make this real.

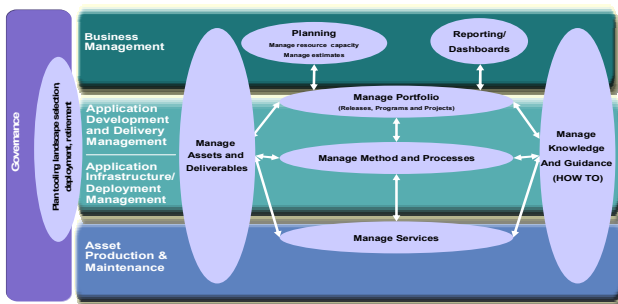


Figure 1. A Simplified Blueprint for a Software Factory.

Figure 1 illustrates a simplified software factory blueprint that we have used as the basis for several large scale enterprises. In this approach the software factory provides a collection of capabilities that support the management and delivery of enterprise software, covering 5 key areas. We briefly review each of these in turn.

A. Business management

Effective business and IT planning and portfolio management helps to streamline the business by empowering faster, better-informed decisions, and can reduce costs by prioritizing enterprise software investments to support business goals. Ultimately, proficiency in this area allows strategic intent to be converted into executable processes with measurable business results. To implement this typically requires several elements:

- **Enterprise architecture management** to help make faster, better-informed strategic and tactical decisions, prioritize enterprise software investments to support business goals, and analyze, plan and execute change with reduced risk.
- **Business process management** to help to optimize business performance by discovering, documenting, automating, and continuously improving business processes to increase efficiency and reduce costs.
- **Requirements definition and management** to minimize the number of inaccurate, incomplete, and omitted requirements. This helps teams collaborate effectively with stakeholders, reduce rework, accelerate time to market, and respond better to change.

B. Asset production and maintenance

Knowledge management and reuse best practices allow organizations to discover and leverage existing data and assets. With an understanding of the key assets, it is possible to enforce policies and best practices, manage model dependencies and even trace assets to versioned artifacts.

It is important to determine what assets exist by providing the ability to search and select across multiple asset repositories and data warehouses, relate assets to one another and leverage existing assets for reuse. Such solutions can also help administrators enforce policies and best practices, manage model dependencies and trace assets to versioned artifacts, creating a link between systems, sub-systems, code, requirements, test cases and delivered solutions. Finally, teams create new assets, transforming code into standardized artifacts such as Web or Business Process Execution Language (BPEL) services that can be used as components for building value-added applications.

C. Application development and delivery management

Smart product design and delivery optimization requires collaboration across teams to deliver quality software and systems. In addition, applying lean processes with disciplined teams in focused “centers of excellence” ensures flexibility and facilitates globally distributed enterprise software delivery. Collaborative services, automation and measurement feedback throughout the software development

lifecycle are essential to achieve levels of productivity and consistency beyond those accomplished using traditional, craft-oriented software development tools.

This requires capabilities that can help teams to collaborate across the lifecycle and automate routine tasks. Key capabilities include:

- **Change and release management.** Improving quality and productivity by effectively unifying distributed teams by managing change processes from requirements gathering through to deployment.
- **Quality management.** Advancing quality across the entire software delivery lifecycle from requirements, design, development, quality assurance, security, and compliance to deployment.
- **Architecture management.** Software development tools for design, development, and delivery that support modeling and coding activities in appropriate high-level languages, supported with a range of analysis capabilities for maintaining the architectural quality of the delivered solution.

D. Application infrastructure and deployment management

A modern application infrastructure allows organizations to cost effectively build, deploy and manage applications and products for varying business needs. Integrating service delivery across organizational boundaries and all stages of the lifecycle helps to improve time-to-market and reduce cost and risk while providing the visibility, control and automation needed to deliver a dynamic infrastructure that adapts to changing business requirements. These solutions provide capabilities to help organizations develop a robust application infrastructure, including capabilities for:

- **Product deployment.** Offering services to automatically deploy, track, and manage applications across the lifecycle.
- **Application delivery.** A set of technologies that support system build and deployment across mainframe and distributed environments.
- **Connectivity and application integration.** Services that foster collaboration, insight and cost effective re-use of data and knowledge across the organization.

E. Governance

Automated capabilities to monitor operational environments and provide feedback to the software and systems delivery processes are critical in a modern approach. Iterative improvement across the entire lifecycle ensures timely problem resolution and ensures flexibility to adapt to change in today’s business environment. These solutions for operations provide capabilities to help organizations develop a robust set of practices for automating operational monitoring and measurement. These solutions can help in:

- Application health monitoring.
- Performance management.
- Security and compliance.
- Service management.
- Performance optimization.
- Monitoring and measurement.

IV. RATIONAL JAZZ: AN INTEGRATED SOFTWARE FACTORY PLATFORM

A common collaborative platform is critical for effectively introducing a software factory approach. A collaborative development platform automates and simplifies the challenges of enterprise software delivery from project management, to the ability to leverage innovation, to the visibility and access of the development and delivery teams across the distributed supply chain.

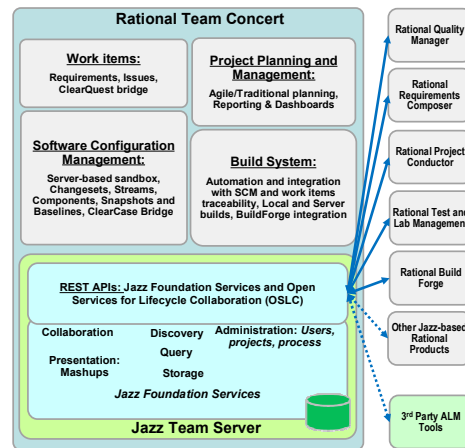


Figure 2. The Rational Jazz Platform.

IBM Rational’s approach to provide a collaborative platform for software factories is the Jazz technology – a set of integrated capabilities to unify all stakeholders in the software supply-chain, a basis for governance and management of standardized delivery processes, and the glue that enables visibility and transparency across the complete software delivery process [5].

As illustrated in Figure 2, the Rational Jazz platform consists of a set of capabilities that deliver the services necessary for a software factory. There are 2 major parts to the solution. The first is the Jazz Team Server comprising core capabilities for integrating and collaborating across teams. Access to these capabilities is via the Open Services for Lifecycle Collaboration (OSLC) interfaces. The second is the Rational Team Concert solution that embeds the Jazz Team Server as the basis for delivering core services for work item management, project planning and management, source code management, and build management. Any software factory solution built on this technology customizes and extends this platform through the addition of specific capabilities in areas such as quality management, requirements management, and so on.

The Rational Jazz Platform has been used in several different kinds of scenarios. In particular, were companies are moving toward a software factory approach, this platform offers the core capabilities on which to build and deliver the essential characteristics of a software factory: collaboration, automation, and visibility.

V. EXAMPLES

We provide 2 examples¹ of organizations that have implemented a software factory approach, realized via the Rational Jazz platform.

A. Subcontractor management at ABC Bank

1) Challenges

ABC Bank is a large worldwide financial institution with a diverse, widely distributed IT organization. In an attempt to efficiently manage rapid growth at ABC Bank, they have substantially focused on subcontracting major parts of their software development and delivery to large software centers in Latin America, Europe, and Asia. This approach was aimed at reducing fixed IT costs and increasing flexibility to adapt to customer demand. The growth of this subcontracting model is challenging due to:

- **Lack of governance to control project progress.** Across the organization, different teams were using different governance mechanisms, not connected, with progress measured during informal weekly meetings.
- **Poor communication due to different time zones, location, cultural and political differences.** Not all team members were fluent in English language and due to different time zones many discussions were inconclusive, or had to be postponed for days.
- **Inadequate planning and change management procedures.** Projects were subcontracted using a fixed-price model, and there was little flexibility to negotiate changes or modify initial planning.
- **Mismatches between user expectations and the real outcomes.** End users were not involved in requirement analysis or reviews and there were many rejected requests and conflicts across the main stakeholders.
- **Poor infrastructure for remote access and lack of a common asset repository.** Distributed teams were not notified when new versions of the common architecture framework were released and a lot of rework had to be done to adapt these changes at the last minute.
- **Unclear information sharing and privacy rules.** Integrating components from different providers raised many poorly addressed privacy and security issues.

2) Approach

To recover from this situation, ABC Bank directly focused on supply-chain management issues as part of a wider software factory approach to software delivery. They changed their development processes and infrastructure, and implemented a common development environment based on Rational Jazz platform to mitigate hidden costs and issues. In their first wave of changes they focused on:

- **Organization changes** by creating a new Software Factories Project Office in charge of negotiating and managing subcontracted projects;
- **Common infrastructure** based on a central repository, accessed from external locations using standard Internet

¹ Although the examples are real, we use fictitious name for reasons of privacy.

connection protocols. This central repository is used to share and integrate information across the teams.

- **Governance Dashboards** producing metrics and reports that measure progress of individuals, teams, and software factories to assess their performance. This central governance dashboard was updated automatically with project information in real time, allowing the enterprise to keep external developments under control and to reduce meeting and travel costs.

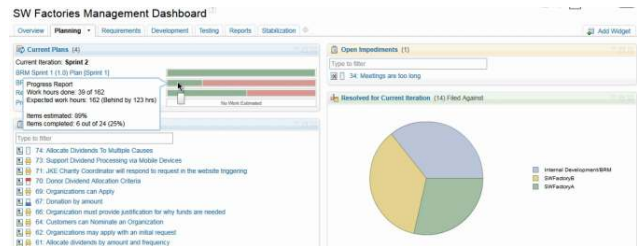


Figure 3. Software Factories Governance Dashboard.

- **Planning and change management processes** were adapted to augment traditional waterfall software development processes with iterative, agile techniques, to enable faster response to changing demands.
- **Confidentiality** was addressed by identifying every critical private data element that the company had, isolating it from subcontractor access, and explicitly granting permissions for common assets to be shared with subcontractors via a central register of shared artifacts (e.g., common architecture components).

3) Results

Implementing these changes was not easy and caused many political and technical conflicts inside ABC Bank, and across the supply-chain. However, as a result of this transformation the enterprise was able to adapt to this new software delivery model, and is starting to benefit from the reduction of fixed costs and increased flexibility into their development activities across their suppliers.

Thanks to the governance dashboard they are now able to measure status and progress of each supplier, penalizing or terminating contracts of those with less efficient delivery.

B. Testing Factory Services at XYZ

1) Challenges

XYZ is a European software services and consulting company, specializing in software development and testing services. One of the company's most important concerns is consistency and quality of delivered services, and a repeatable lean approach to software delivery. To that end, XYZ has standardized many key practices, and has obtained a CMMI Level 3 certification [6]. XYZ's engineering and quality assurance (QA) culture encourages agile practices along the full software lifecycle, making it compatible with the rules and constraints associated with CMMI.

As part of their global strategy, XYZ offers expertise and services in software testing, and provides a catalog of services to their customers, including:

- Services to assure the quality of the work-products generated in each phase of the software development lifecycle, reducing defects across phases;
- Playing the role of facilitator between the different actors in the QA process to objectively assess quality practices and review project milestones;
- Advising on quality processes and practices, including assessing deliverables, and designing appropriate test and support processes to increase quality.

A software factory approach to XYZ is important because it encourages early project involvement of testers and other quality-focused roles, and makes quality management throughout the project lifecycle a high priority.

2) Approach

To realize these needs, XYZ decided to set up several large European delivery centres, and to adopt a software factory approach to deliver its quality-focused services. It refers to these as a “software test factories”.

The basis for their software test factories is an infrastructure supporting software configuration management, build management and continuous integration practices, and agile project management. These core capabilities are fully integrated with specific testing services to manage the test plans, execute tests and assess test coverage against the project requirements.

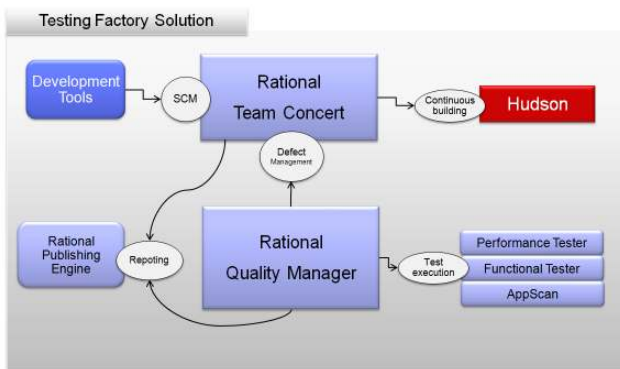


Figure 4. Testing Factory Solution Architecture.

A streamlined software factory infrastructure for QA is the cornerstone of the XYZ solution. This solution, augmented with additional tools for developers or test engineers in their day-to-day work, offers the integrated capabilities that ensure XYZ delivers quality solutions to its clients. As illustrated in Figure 4, the testing factory services were automated by providing a suite of integrated tools covering several process areas of CMMI.

In addition, the need for supporting geographically distributed teams and roles (PMO, Test Manager, Test engineers, etc.) was critical. A series of customized processes were designed that resulted in:

- **Templates** to specify test cases and test scripts to define automated or manual tests;
- **Automation** of building and continuous integration process, regression testing or other quality standards such as those imposed by CMMI;
- **Visual dashboards** to monitor the state of the projects, delayed activities, open vs. closed defects, metrics of coverage, productivity of the team, etc.

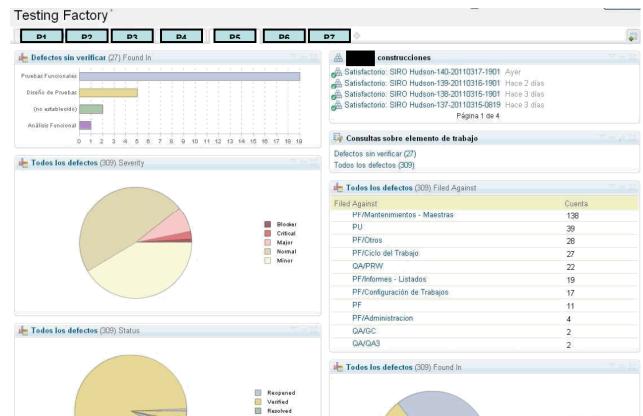


Figure 5. A Dashboard from XYZ’s Software Test Factory.

3) Results

The software factory approach to testing and quality is a critical part of XYZ’s strategy. As a result of this initiative, XYZ has successfully increased the efficiency of their processes, improving the productivity of their key testing factory in Salamanca, Spain. This solution is now being introduced at test factories across the XYZ organization.

VI. OBSERVATIONS

A. Agile Software Factories and CMMI

Is it possible to make agile methods and process maturity compatible? In a software factory context, this goal is not only possible, but mandatory. In all enterprise organizations there are policies and rules which enforce achievement of certain maturity levels. At the same time, the agile paradigm is an emerging necessity to address the challenges of flexibility in software development and delivery.

For many organizations, significant investment has been made to improve the maturity of the key management processes, with the CMMI as a focus for much of that effort. To achieve many operational goals of a software factory, CMMI can be very valuable in areas such as:

- Reducing operational costs;
- Improving the quality of the service;
- Managing capacity and resources efficiently;
- Industrialize the software lifecycle through reuse.

In contrast, the agile methods [7] (primarily oriented to optimize software development teams) are very useful for helping to ground the more abstract ‘process areas’ into a

more concrete and concise practices which will be used by the development and testing teams in:

- Delivering working software on a frequent basis;
- Promoting whole team planning and face-to-face collaboration;
- Reacting to frequent changes and reprioritizations with rapid fact-based decision making.

At first glance, these agile practices do not align well with more traditional high maturity approaches such as CMMI. However, in our experiences we can adapt these ideas in the context of a software factory model to balance the need for repeatability and governance essential for CMMI with the pragmatic needs expressed in agile approaches. Achieving this balance is critical in enterprise organizations such as ABC Bank and XYZ. A software factory approach provides a framework to realize this.

B. *Software Factories in the Cloud*

There is a high level of excitement about cloud computing and the promises that it brings to enterprises to reduce cost and increase flexibility of service delivery [8]. Many organizations are already involved in pilots, or are actively using cloud technologies and cloud-based services.

Initially, we have seen many traditional enterprise software solutions ported to a cloud platform, and included in platform images that can be uploaded to a cloud infrastructure. This is an important starting point for enterprise system use on the cloud. However, it is very limited in terms of many of the important usage scenarios for cloud technology. There is less understanding of which new enterprise software capabilities, services, and approaches will be needed in much more complex scenarios. For example, we already are seeing interesting scenarios that are raising new challenges for enterprise software delivery organizations:

- Several teams are deploying business application onto a public cloud infrastructure for access by clients around the world. How do those teams collaborate to share information to ensure that they do place sensitive data on the public infrastructure? What coordination is given to the teams to ensure the management of shared images is handled effectively?
- Multiple System Integrators and specialist vendors must deliver different parts of key enterprise solutions as part of a software supply chain that must be integrated to be delivered into production. How can the cloud be used as the delivery platform to coordinate and govern delivery and integration of these components?

These, and many more such scenarios, are stretching conventional processes, skills and technologies for enterprise software delivery. Software delivery organizations are actively working on new deployment approaches that provide the additional governance, visibility, and control that is demanded in such situations.

C. *Metrics and measures for Software Factories*

Although there are different development standards to measure in-house development, there is little standardization

in evaluating supply-chains and software factories. Standard approaches such as function point analysis and defect density can be applied, but in practice they appear inadequate.

With more complex supply-chain delivery models becoming more common, we need metrics that help us address different questions: Which software factory is more productive? How many defects are still opened? Which software factory is delayed in their deliverables? These and many other measures need to be defined and an automatic mechanism to collect these metrics must be implemented to help compare results across external providers in real time.

VII. SUMMARY

A fast-paced evolution is taking place in the context of very dramatic shifts in how IT organizations view the value they bring to their varied stakeholders, the services they deliver to clients, and the way they invest to achieve their goals. As a result, the last few years has seen a significant change in the way enterprise systems are developed, delivered, and maintained. By introducing a software factory view to enterprise software delivery, organizations can focus attention on the software supply chain, address inefficiencies in software delivery, and gain greater control and visibility into the delivery process.

In this paper we have examined the key principles that underlie this kind of software factory thinking to enterprise software delivery, and provided 2 real-world examples to illustrate how solutions can be introduced that provide value to the IT organization. The technology underpinning such an approach is critical. We briefly discussed one example technology approach based on Rational Jazz platform, and illustrated its primary characteristics as the basis for a software factory.

Much further work remains. We have made a number of observations on critical areas requiring additional work. Over the coming years we expect to see significant progress in these, and in several other key areas.

REFERENCES

- [1] J. Jeston and J. Nelis, "Business Process Management, Second Edition: Practical Guidelines to Successful Implementations", Butterworth-Heinemann, 2008.
- [2] M. Poppendieck and T. Poppendieck, "Lean Software Development: An agile toolkit", Addison Wesley, 2003.
- [3] M. Hotle and S. Landry, "Application Delivery and Support Organizational Archetypes: The Software Factory", Gartner Research Report G00167531, May 2009.
- [4] P. Clements and L. Northrop, "Software Product Lines: Patterns and Practices", 3rd Edition, Addison Wesley, 2001.
- [5] M. Goethe et al., "Collaborative Application Lifecycle Management with IBM Rational Products", IBM Redbook, December 2008.
- [6] "CMMi for Development, Version 1.3", CMU/SEI-2010-TR-033, November 2010.
- [7] R.C. Martin, "Agile Software Development: Principles, patterns, and practices", Prentice Hall, 2002.
- [8] G. Reese, "Cloud Application Architectures: Building Applications and Infrastructures in the Cloud", O'Reilly Press, 2009.

A “Future-Proof” Postgraduate Software Engineering Programme: Maintainability Issues

J Paul Gibson and Jean-Luc Raffy
Le département Logiciels-Réseaux (LOR)
Telecom & Management SudParis (TMSP)
9 rue Charles Fourier, 91011, France
Email: gibson.paul@it-sudparis.eu

Abstract—We report on the development of a software engineering programme for Masters students. Maintainability of educational programmes is critical: there is a large initial investment in developing quality programmes and we must ensure that these programmes are “future proof”. Consequently, we followed a traditional software engineering life-cycle process to develop a programme that would meet the current needs of industry, whilst also being easy to maintain with respect to future changes in industrial requirements. We show how the programme has gone through a number of refinement steps — where we have iterated through the life cycle of requirements engineering (with “client” industries), high-level design (establishment of a foundational educational architecture), implementation (by lecturers), testing (through establishment of evaluation and feedback mechanisms) and maintenance (throw updates to curriculum and course content). To conclude, we propose treating educational programmes as software, and demonstrate advantages in applying software engineering techniques for development and maintenance.

Keywords—Teaching, Education, Curriculum, Software Engineering

I. INTRODUCTION

Our institute is focused on educational programmes for telecommunications engineers. Software is playing an increasingly important role in telecommunications systems; and a strategic decision was taken, a number of years ago, to introduce a postgraduate (Masters) programme which specialised in software engineering. We have been motivated by the observations of Curran[3] and Parnas[11] concerning the need to distinguish computer science from software engineering, and the goal of making software engineering a true engineering discipline [16].

Despite following guidelines in the development of software engineering curriculae [17], [7], the programme has failed to attract the number of students that are required to make it feasible to run in the long-term. We have a capacity for teaching around 20 students (for each of the 2 years of the programme), but in the last four years we have not had more than 6 students in each year.

As a result, we have made continuous changes to the programme in order to attract more students. In particular, the programme has gradually become more specialised; moving from:

- a *software engineering* stream as part of a general information technology (IT) Masters, to
- a stand-alone Masters programme *software engineering for smart devices*, to
- its most recent incarnation as a more specialised Masters *software engineering and ambient intelligence*.

The changes have required much work, and during the process of evolving and maintaining our software engineering programme (in its different forms) we have identified the need for a process to aid us in improving our work, and the quality of the programme. As software engineers, we realised that much of our academic programme could be considered analogous to software and so we should be able to apply software engineering development principles, methods and techniques to the development of our programme. This paper reports on this insight and summarises the advantages of working within such an analogy. The results should be of interest to all developers of academic programmes, and not just those teaching software engineers.

This is work in progress: the ideas need further discussion and, in our opinion, the software engineering community is the best placed to be able to provide useful feedback. We hope in some way to be contributing to the “Push to Make Software Engineering Respectable” [12].

The remainder of the paper is structured as follows. In section II we motivate our work by introducing the premise that developing an educational programme is like developing software. In section III we give an overview of the problems in developing a software engineering programme and propose that one must consider the life-cycle of the programme analogous to the software life-cycle. Sections IV, V and VI examine the main life-cycle stages: requirements, design and implementation. Section VII reports on the different evaluation techniques that can be applied during the different stages in the life-cycle. Section VIII illustrates, by reviewing the changes made between three iterations, how the evaluation can feedback into the programme development and maintainance. Section IX concludes with some remarks about future work.

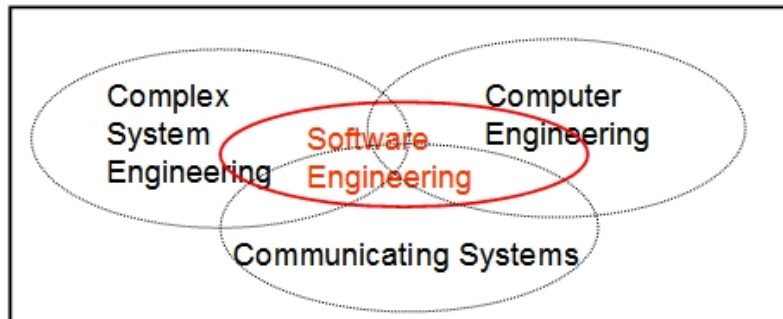


Figure 1. Software Engineering Domain of Knowledge

II. DEVELOPING AN EDUCATIONAL PROGRAMME IS LIKE DEVELOPING SOFTWARE

Most, if not all, educational programmes can be specified in a bottom-up fashion — listing a set of core components that need to be taught. Then, the traditional approach to programme development is to assign core components to programme modules and to identify dependencies between modules so that the temporal ordering of modules in the programme respects the dependency relationship. Modules are then, in general, taught and assessed independently. Irrespective of whether this is likely to produce a *good* academic programme — we would argue that it is not — it has the major disadvantage that the programme is difficult to maintain: small changes to the component requirements can have a major impact on the implementation of the programme. Further, small changes to the implementation environment (the lecturers, etc.) may necessitate major restructuring of the programme.

Most academic programmes evolve. Unfortunately, as years pass by the links between the different versions of the programme get lost. Further, documentation of the changes made (why and how) is usually very poor. As a consequence, after a number of years there is a lack of coherence between what one is trying to teach and how it is being taught.

These type of problems are very familiar to software engineers. Managing the evolution of an academic programme, like managing a software system, has two fundamental, complementary aspects:

- a continual improvement in the understanding of the problem domain through continual analysis [13]
- an iterative life-cycle of evaluation, feedback and change [1]

Where a problem domain is well-understood and academic programmes have been well-established for a number of years then there is probably no need for a maintenance (evolution) process. However, this is clearly not the case for software engineering. The discipline has been moving so fast that many subjects that are common to recent curricula are not even mentioned in the body of knowledge from 12

years ago [2].

III. DEVELOPING A SOFTWARE EDUCATION PROGRAMME: THE DOMAIN AND THE LIFE-CYCLE

A. The software engineering domain

In Figure 1 we see how the discipline of Software Engineering cannot and should not be separated from other disciplines. This figure illustrates our particular structured understanding of what we expect our students to know about:

- Software Engineering — this is the core knowledge that all our graduates must have mastered
- Complex System Engineering — software, in general, does not exist in isolation. Most software engineering problems arise because of complex interactions in and between the software and the environment in which it exists.
- Computer Engineering — software executes on a physical machine, and we expect our students to understand how such a machine operates
- Communicating Systems — more and more software systems involve communications over different types of network. As a telecommunications school we expect our students to understand how such networks operate.

This is the view of software engineering that we believe is unlikely to change in the short to medium term (it is consistent with views on software engineering that are ten years old [17]). It is also a view that best matches our institutes' expertise (in teaching and research). Later in the paper we will introduce: the scientific and mathematical foundations upon which these four engineering disciplines are constructed, and the specific state-of-the-art techniques and tools for engineering software that have been developed out of (and interaction between) these domains.

B. The Programme Life-Cycle

As work in progress, we chose to develop our programme following the simplest, best understood, waterfall life-cycle model [15]. Such a model can be defined to different levels

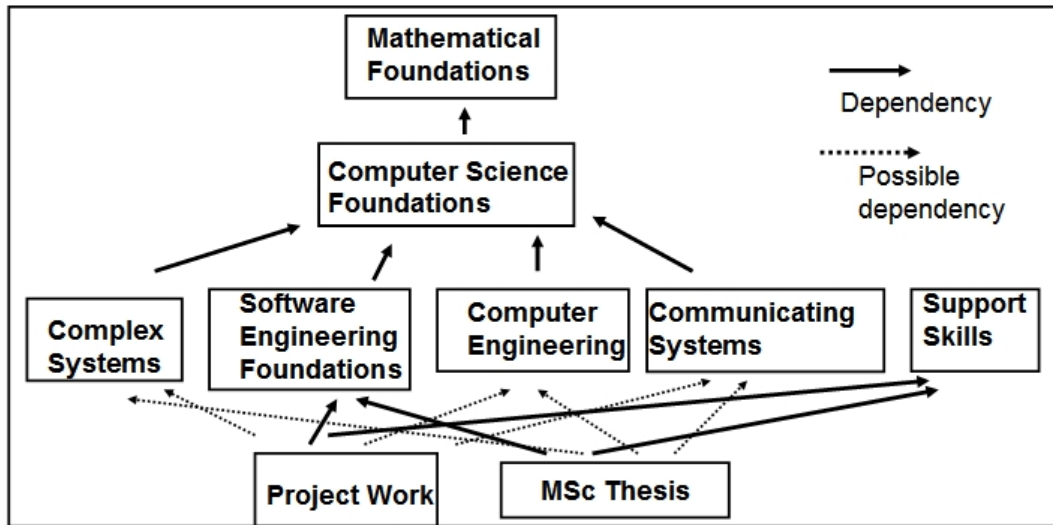


Figure 2. Architecture for Software Engineering Programme

of detail, for our initial research we chose to consider a life cycle with 3 fundamental steps — requirements, design and implementation — and feedback between each of the steps.

IV. REQUIREMENTS

Our first step is to identify the requirements of our academic programme. As with complex, software systems, these requirements must attempt to meet the needs of a number of different actors:

- the students — want to improve their prospects of employment and enjoy their education;
- the lecturers — want to teach in their area of expertise, work with good quality, well-motivated students, and help promote and further their research through teaching;
- the university administration — want to attract large number of students, graduate large number of students, and minimize costs;
- the government — want their investment in universities to be coherent and worthwhile;
- industries — want graduates that match their current needs (quantity and quality);
- research institutes — want to attract postgraduates into research careers, etc.

As programme developers we are very aware of the different compromises that exist in meeting these requirements. As such, it is critical that all interested parties are involved in the construction, evaluation and evolution of our academic programmes.

The requirements, listed above, are not specific to software engineering. We regard these in more detail in the following section - where we map specific requirements to a high-level programme design (architecture).

V. PROGRAMME DESIGN FOR MEETING HIGH-LEVEL REQUIREMENTS

In Figure 2 we represent our high-level requirements for a software engineering project:

- Our four engineering domains depend on common computer science foundations [11].
- Computer science foundations depend on mathematical foundations [5].
- All students will require support skills [14] in order to work on industrial projects and write a thesis.
- The project work [6] and thesis must demonstrate mastery of software engineering foundations, and may also depend on understanding of complex systems, computer engineering and communication systems.

The key to evolving and maintaining our programme is that these abstract components of our high-level architecture (and their interdependencies) will not change.

VI. PROGRAMME IMPLEMENTATION

A. Modules: The Programme Components

In Figure 3 we see how each of these abstract components is to be implemented:

- The mathematical and computer science foundations will be taught as individual modules
- The software engineering foundations will be taught as a number of inter-related modules
- The support skills — including innovation [4] — are not specific to software engineering
- The remaining modules address the 3 domains that overlap with software engineering - where we choose to address these domains through a software engineering perspective

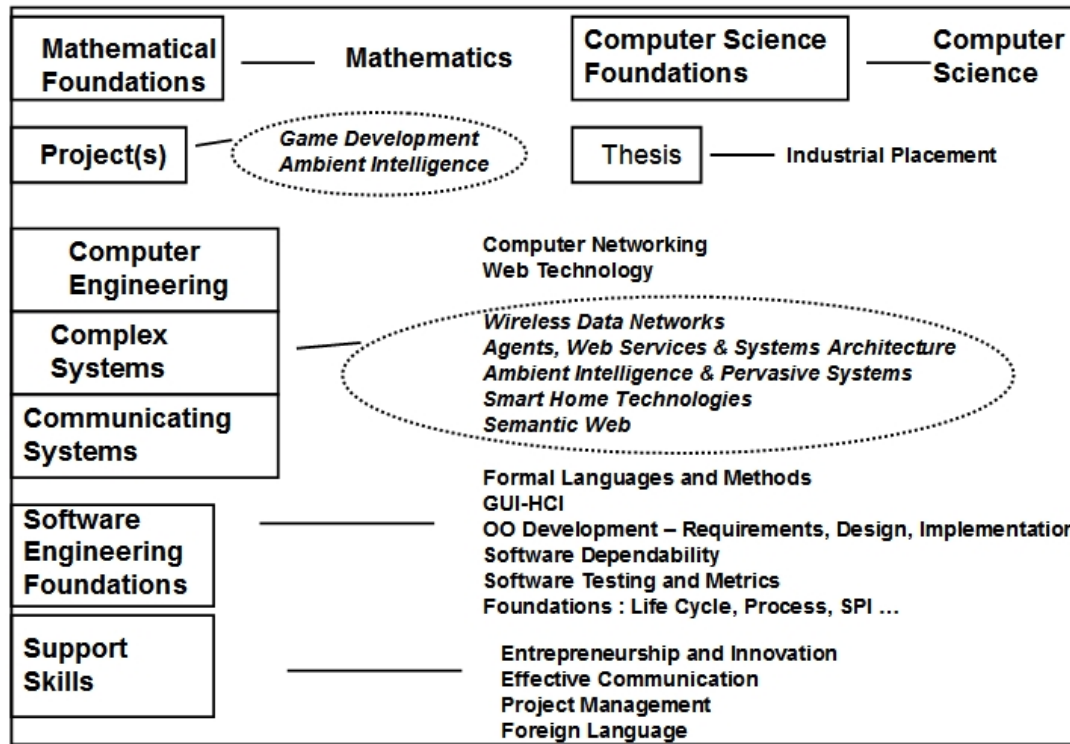


Figure 3. Implementation of Software Engineering Programme

- The project work is dependent on the the student’s knowledge in these other domains
- The thesis must be written concerning an industrial placement.

From the diagram we have also circled the modules that may change over time. (The others will always be in the programme, although internal details of each of these modules may change.) The circled modules are those which are currently being taught in the latest version of the programme (*Software Engineering and Ambient Intelligence*).

B. The Approach: PBL for Software Engineering

The previous subsection examined what we are going to teach. It should also be mentioned that we have taken some decisions as to how the modules should be taught:

- Foundational mathematics and computer science will only be taught if it is used in the engineering modules. Where possible, all material in these foundational modules will be linked to the software engineering modules.
- All software engineering modules will be taught using a problem-based-learning (PBL) approach [8]. Emphasis will be on rigour and formality, and mathematical modelling [10]
- The PBL will draw from real-world problems taken from industries that can be expected to hire our graduates.

- Modules will be coherently connected by sharing common problems.
- The ethical side of software engineering will be emphasised and the recurrent problem of plagiarism explicitly addressed [9].

VII. PROGRAMME EVALUATION

A. Accreditation

Our institute is a member of a group of schools whose Masters programmes go through an independent review (mostly by other academics) in order for them to be accredited. This accreditation is critical for attracting students as it is intended to be a good indicator of a quality programme. Further, students on accredited programmes may benefit from additional funding.

During accreditation feedback focuses on programme content. Comparisons are made with other programmes and curriculum guidelines from around the world. It is only in the current year that our programme has achieved accreditation. Many of the changes made in order to achieve accreditation were superficial in nature.

B. Industrial Feedback

A co-director for the programme is directly involved in collaboration with local industry in order to establish a *Pole de competence for complex system engineering*. Industries

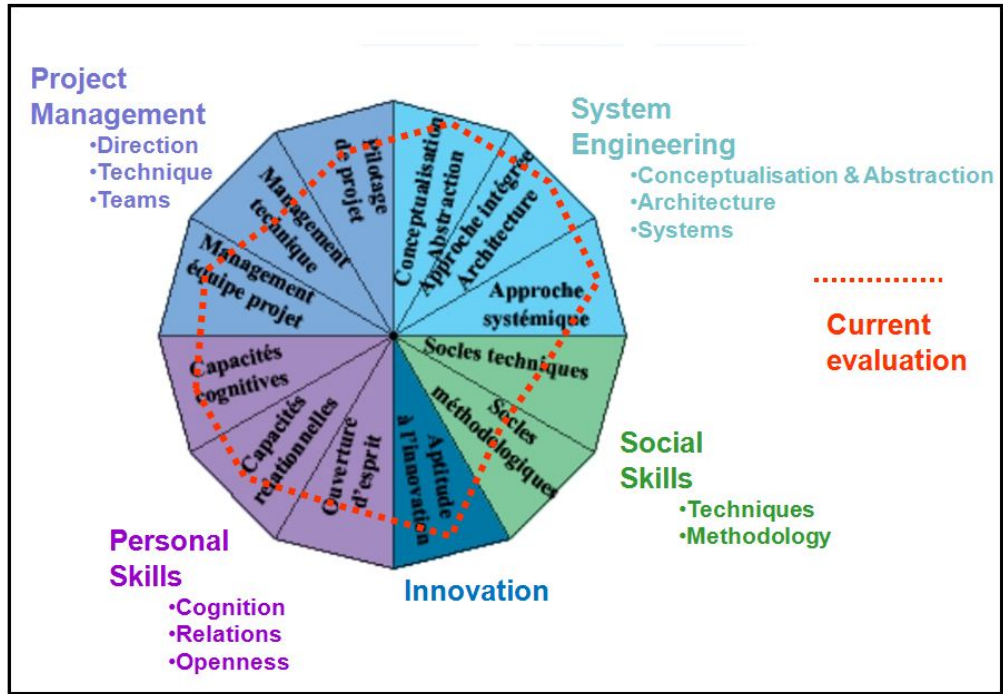


Figure 4. Industrial Evaluation Criteria

have identified a need for more (and better) software engineering students who can work in the area of complex systems (across many different industrial sectors).

In Figure 4 we see the list of competencies that these industries have identified as being core to their requirements. It should be noted that software engineering is not explicit in their criteria: in their *System Engineering* criteria they expect the engineers to be specialised in a relevant discipline (like software engineering) whilst also having generic engineering skills.

Our most recent evaluation is sketched in the diagram. This shows that we have some progress to make in all criteria, but we are weakest in *social skills*. We note that this evaluation is based on static analysis of our programme description (design and implementation).

C. Students — Quantity and Quality

Our programme will have little value if we cannot attract more students. However, we must not compromise quality for quantity. Perhaps the weakest element of our evaluation is that which we could get from students. Students are encouraged to completed feedback questionnaires concerning all aspects of the programme. With such small numbers of students, statistically significant analysis is not feasible. Rather, we focus on open questions and freeform discussions with students. Two main issues have arisen:

- Coherency between modules needs to be better addressed, and

- Standards of evaluation are not consistent between modules

The students suggest that increasing the number of common problems between modules will address the first issue. The second issue is more difficult to address — the students believe that the main difference is between the foundational modules (which are perceived to be difficult) and the technology modules (which are perceived to be not so difficult). These perceived differences have been validated through analysis of students’ results.

We plan to keep in contact with students after they graduate. However, we have no formal procedures in place: the students rest in contact through personal communication. This requires further work, on our behalf.

VIII. PROGRAMME EVOLUTION: THREE ITERATIONS

In 3 years, our programme has gone through 3 iterations. (In the fourth year we stabilised the programme in order to better evaluate it against the educational and industrial requirements.)

A. Software Engineering (Information Technology)

In this iteration, software engineering was taught as a specialist stream in a more general masters programme. Our initial evaluation identified weaknesses in this programme that could only be addressed by teaching a dedicated software engineering postgraduate programme:

- Core software engineering material was not being taught in enough detail.
- The relationship and dependencies between core material was not adequately addressed.
- Core mathematics and CS material was presented in a way that was not specific to software engineering

B. Software Engineering and Smart Devices

In this iteration we restructured our programme in order to better focus on core software engineering. For project and practical work we focused on *Smart Devices* (as this was a leading area of development in industry). Feedback from students and industry led us to prototype teaching material on developing games that would exploit the functionality of such devices. Evaluation of this programme identified weaknesses that needed to be addressed:

- We needed a clearer separation of core and non-core material.
- We needed to demonstrate the general utility of our core software engineering material by addressing more than one area in our project/practical work.
- We needed more emphasis on support skills.

This resulted in the programme as currently illustrated in Figure 3

C. Software Engineering and Ambient Intelligence

The most recent iteration has yet to be fully evaluated. We have had positive feedback from the accreditation process and from the industrial partners. However, we continue to fail to address our main weakness — there are only nine students registered for the first two years of the programme.

IX. CONCLUSIONS: REMARKS AND FUTURE WORK

We have proposed treating educational programmes as software, and demonstrated advantages in applying software engineering techniques for their development and maintenance.

Current and future work involves examining re-use of material across and between programmes; and improving evaluation processes (particularly improving feedback from students).

Our major challenge is not in knowing what to teach, or knowing how to teach; it is in having a reasonable number of students to teach. Perhaps we should add a “marketing” module to our curriculum?

REFERENCES

- [1] B Boehm. A spiral model of software development and enhancement. *SIGSOFT Software Engineering Notes*, 11:14–24, August 1986.
- [2] Pierre Bourque, Robert Dupuis, Alain Abran, James W. Moore, and Leonard Tripp. The guide to the software engineering body of knowledge. *IEEE Software*, 16:35–44, November 1999.
- [3] W. S. Curran. Teaching software engineering in the computer science curriculum. *SIGCSE Bulletin*, 35(4):72–75, 2003.
- [4] Peter J. Denning and Andrew McGettrick. Recentering computer science. *Communications of the ACM*, 48(11):15–19, 2005.
- [5] Keith Devlin. Viewpoint: the real reason why software engineers need math. *Communications of the ACM*, 44:21–22, October 2001.
- [6] Alan Dutson, Robert H. Todd, Spencer P. Magleby, and Carl D. Sorensen. A review of literature on teaching engineering design through project-oriented capstone courses. *Journal of Engineering Education*, 86:17–28, 1997.
- [7] Gary A. Ford and Norman E. Gibbs. A master of software engineering curriculum: Recommendations from the software engineering institute. *Computer*, 22:59–71, September 1989.
- [8] J. Paul Gibson. Weaving a formal methods education with problem-based learning. In T. Margaria and B. Steffen, editors, *3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science (CCIS)*, pages 460–472, Porto Sani, Greece, October 2008. Springer-Verlag, Berlin Heidelberg.
- [9] J. Paul Gibson. Software reuse and plagiarism: A code of practice. In *14th ACM SIGCSE Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2009)*, pages 55–59, Paris, France, July 2009. ACM.
- [10] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Sculpturing Event-B models with Rodin: “holes and lumps” in teaching refinement through problem-based learning. In *From Research to Teaching Formal Methods - The B Method (TFM B'2009)*, pages 7–21, Nantes, France, 2009. APCB.
- [11] David Lorge Parnas. Software engineering programmes are not computer science programmes. *Annals of Software Engineering*, 6:19–37, 1998.
- [12] Gilda Pour, Martin L. Griss, and Michael J. Lutz. The push to make software engineering respectable. *IEEE Computer*, 33(5):35–43, 2000.
- [13] Rubén Prieto-Díaz. Domain analysis: an introduction. *SIGSOFT Software Engineering Notes*, 15:47–54, April 1990.
- [14] S.H. Pulko and S. Parikh. Teaching soft skills to engineers. *International Journal of Electrical Engineering Education*, 40(11):243–254, 2003.
- [15] W. W. Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.
- [16] Mary Shaw. Prospects for an engineering discipline of software. *IEEE Software*, 7:15–24, November 1990.
- [17] Mary Shaw. Software engineering education: a roadmap. In *International Conference on Software Engineering — Future of Software Engineering Track*, pages 371–380, 2000.

Using Software Engineering Principles to Develop a Web-Based Application

Cynthia Y. Lester

Department of Computer Science
Tuskegee University
Tuskegee, Alabama, USA
cylester@mytu.tuskegee.edu

Abstract – In the United States, the software development industry is about a \$220 billion industry. Therefore, the need to produce accessible, reliable and trustworthy software that is within budget and that also meets the demands of the heterogeneous user can sometimes become an overwhelming task by organizations. Since its inception in the late 1960s, software engineering has used software processes as systematic approaches that lead to the development of software applications. However, with the introduction of the Internet and the World Wide Web, there have been changes to the way that software is produced. The aim of this paper is to present the results of an inquiry that introduced to undergraduate software engineering students an approach to developing a web-based application. Traditionally, specialized web engineering courses are offered at the graduate level or as an elective course in an undergraduate curriculum. The paper presents a semester-long project that combines some traditional software engineering processes with web engineering processes. The results from the study suggest that introducing a hybrid approach inclusive of traditional software processes and web engineering techniques can be done successfully at the undergraduate level but not without certain challenges.

Keywords – *software development; software engineering; web-based application; web engineering.*

I. INTRODUCTION

Software engineering is defined as a discipline that is concerned with all aspects of software production from the early stages of inception and specification to the maintenance of the system when it has gone into use. It has often been seen as the “cradle-to-grave” approach for producing reliable, cost-efficient software that is delivered in a timely manner, under given budget constraints, that meets the client’s needs.

The concept of software engineering was first introduced in 1968 at the NATO Science Engineering conference held in Garmisch, Germany, to discuss the ominous “software crisis [1].” The software crisis was a result of informal development practices used to meet the needs of a rapidly changing hardware industry. Software applications were often noted as being unreliable, complex, expensive and sometimes delivered years after the deadline.

Since the inception of software engineering, tremendous strides have been made to develop effective strategies to deliver reliable, cost-efficient software.

Yet, with the advent of the World Wide Web, the topic of how to deliver trustworthy, cost-efficient web applications has become one of increasing importance. Software applications no longer run on a local machine and are accessed by only those who are part of the organization. Now, users demand that software be accessible wherever they are which brings a whole new notion to the delivery of reliable, cost-efficient software. Consequently, the role of software engineering is changing to meet the demands of the heterogeneous user.

The aim of this paper is to present the results from a semester-long project in which software engineering concepts were modified to develop a web-based application. Typically undergraduate students enroll in a general software engineering course which is part of the required curriculum. This course can be taught from many different perspectives. However, to gain specific knowledge in web engineering, students must often enroll in a separate course, if one is offered. However, a survey of various undergraduate curricula did not find many web engineering courses at the undergraduate level. Consequently, if educators want to introduce to the next generation of technologists these concepts, a traditional software engineering course serves as the best vehicle.

The paper begins by presenting several time-honored software development methodologies discussed in a traditionally taught software engineering course. This discussion provides the impetus for an introduction to web engineering concepts. Additionally, the paper presents a semester-long project in which students were engaged which focused on the practical implementation of software engineering concepts for a web-based application. Lastly, challenges and future work are discussed.

II. TRADITIONAL SOFTWARE DEVELOPMENT METHODOLOGIES

Since its inception, there have been many methodologies that have emerged that lead to the production of a software product. The most fundamental activities that are common among all software processes include [1]:

- *Software specification* – engineers and customers define the software and its constraints
- *Software development* – the software is design and programmed
- *Software validation* – the software is checked to ensure that it meets the customer requirements
- *Software evolution* – the software changes to meet the changing needs of the customer

Typically, students are introduced to these activities in the undergraduate computer science curriculum through a software engineering course. This course is often a survey course which exposes students to a variety of life cycle models. The course is frequently taught from a systems approach which places an emphasis on creating requirements and then developing a system to meet the requirements. In the traditional view of software development, requirements are seen as the contract between the organization developing the system and the organization needing the system [2].

A traditional view of software development is the waterfall method. The waterfall method was the first published software development process and forms the basis for many life cycles. It was noted as a great step forward in software development [3]. The method has stages that cascade from one to the other, giving it the “waterfall” name. Figure 1 is an example of the waterfall life cycle [4].

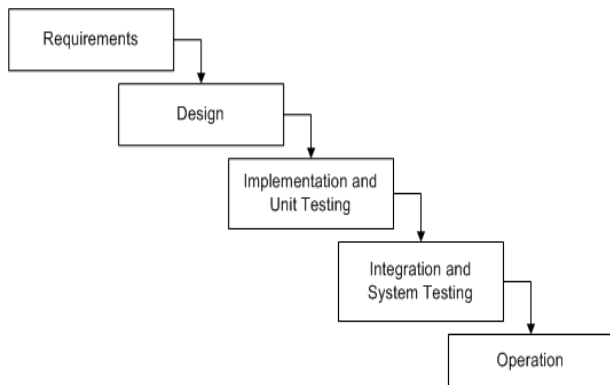


Figure 1. Waterfall model

It has been noted that the method might work satisfactorily if design requirements could be addressed prior to design creation and if the design were perfect prior to implementation [3]. Consequently, one of the main disadvantages of this model is that requirements may change accordingly to meet the needs of the customer and the change is difficult to incorporate into the life cycle. As a result of this shortcoming, additional life cycles emerged which allowed for a more iterative approach to development.

Software prototyping is based on the idea where a version or sample of the software is developed to test requirements and design feasibility [1]. Figure 2 is a

modified version of Sommerville’s process of prototype development [1]. A reason why this particular model was introduced by the author to the undergraduate software engineering students is because it serves as a precursor to web-based application development. More specifically, software prototyping provides an effective way to gain understanding of requirements, provides early testing of system design, and reduces challenges during implementation. It has been stated that the advantages of software prototyping over its predecessor include that it accommodates change easier, it allows users to see how well the system can be integrated into current work activities, and it reveals errors and oversights early on in the process [1].

However, this approach to software development is not without some concerns. For example, the prototype may not be used in the same manner as the final system, the skill level of the testers of the prototype may differ from the users of the system, and because prototypes lack full functionality rapid changes may be made without proper documentation [1]. Consequently, these concerns provide an impetus to review the spiral model and agile methods, which are presented in the next.

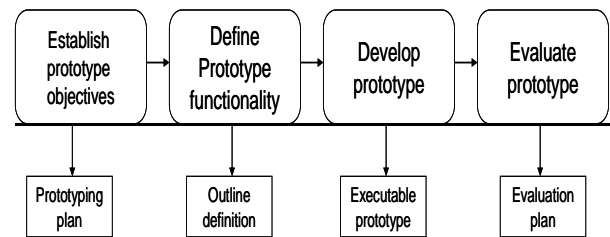


Figure 2. Prototype development

The spiral development model is also an example of an iterative process model that represents the software process as a set of interleaved activities that allows activities to be evaluated repeatedly. The model was presented by Barry Boehm in his 1988 paper entitled *A Spiral Model of Software Development and Enhancement* [5]. The spiral model is shown in Figure 3 [1]. The spiral model differs from the waterfall model in one very distinct way because it promotes prototyping; and, it differs from the waterfall and incremental development method because it takes into consideration that something may go wrong which is exercised through risk analysis.

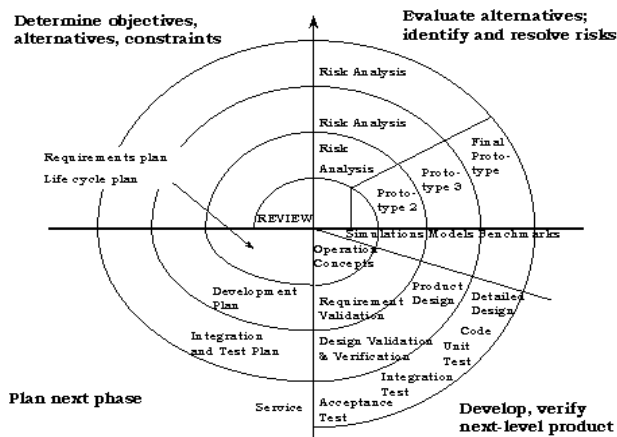


Figure 3. Spiral model

It is noted that this life cycle provides more flexibility than its more traditional predecessors. Further, this method produces a preliminary design. This phase of the life cycle was added specifically in order to identify and resolve all the possible risks in the project development. Therefore, if risks indicate any kind of uncertainty in requirements, prototyping may be used to proceed in order to determine a possible solution.

However, in these approaches, as with many of the other approaches to software development that are taught in traditional software engineering courses, a true focus on the software is mostly absent from the process. Hence, the increasingly important need to include a discussion of how to deliver working software to customers quickly. The next section explores agile methods and its life cycle.

III. AGILE METHODS

In an effort to address the dissatisfaction that the heavy-weight approaches to software engineering brought to small and medium-sized businesses and their system development, in the 1990s, a new approach was introduced termed, “agile methods.” Agile processes are stated to be a family of software development methodologies in which software is produced in short releases and iterations, allowing for greater change to occur during the design [6]. A typical iteration or sprint is anywhere from two to four weeks, but can vary. The agile methods allow for software development teams to focus on the software rather than the design and documentation [1]. The following list is stated to depict agile methods [1], [6]:

- *Short releases and iterations* - allow the work to be divided, thereby releasing the software to the customer as soon as possible and as often as possible
- *Incremental design* – the design is not completed initially, but is improved upon when more knowledge is acquired throughout the process

- *User involvement* – there is a high level of involvement with the user who provides continuous feedback
- *Minimal documentation* – source code is well documented and well-structured
- *Informal communication* – communication is maintained but not through formal documents
- *Change* – presume that the system will evolve and find a way to work with changing requirements and environments

More specifically, the agile manifesto states [1]:
“We are uncovering better ways of developing software by doing it and helping others to do it.

*Through this work we have come to value:
 Individuals and interaction over processes and tools
 Working software over comprehensive documentation
 Customer collaboration over contract negotiation
 Responding to change over following a plan
 That is, while there is value in the items on the right, we value the items on the left more.”*

While agile methods are considered as light-weight processes as compared to the traditional software processes, they too are not without some controversy. For example, it is sometimes difficult to keep the customer involved after software delivery; there may be resistance to change and tool integration; as well as teaming issues. Therefore, in an effort to address these concerns as it relates to developing web-based applications, new models in web engineering emerged.

IV. WEB ENGINEERING

The World Wide Web can be described as a multimedia environment that allows documents to be seamlessly linked over the Internet [7]. It was developed by Tim Berners-Lee with help from Robert Cailliau, both researchers at the European Laboratory for Particle Physics (CERN) [8]. The basic idea was that documents stored on computer that were linked by a network could be accessed by an authorized individual using the network. This idea however, relied on two types of software, a web server and a web browser. The web server stores the documents and “serves” them to other computers who desire access to the documents [7]. The web browser allows user to request and view the documents [7]. These ideas became common place in the 1990s and provide the foundation of today’s Web and its many uses.

In 1990, it was reported that there were less than 50 million users of the Internet in the U.S. However, by 2008 the U.S. reported approximately 230,630,000 Internet users [9]. Therefore, it stands to reason that with more users and more advanced systems, the user population of today’s technology would be more technically savvy than those user groups of yesteryear. However, the average user is now less likely to understand the systems of today as compared to the users of a decade ago. Consequently, the designers and developers of these applications must ensure that the

software is designed with the three “use” words in mind so that the user experience is successful. Hence, the application must be useful, usable, and used [2].

A. Web Engineering as a Multi-disciplinary Field

Web engineering is an emerging multi-disciplinary field that is concerned with the development of web-based applications and systems [11]. As stated by Ginige and Murugesan, the premise of web engineering is a proactive approach taken to successfully manage the diversity and complexity of web application development and to avoid potential failures [11]. Consequently, web engineering encompasses not only the technical aspects of software engineering and its traditional software processes, but also the business-related area of project management, and the humanistic side of computer science, human-computer interaction. Figure 4 is a representation of the many disciplines that provide the foundation for web engineering [12].

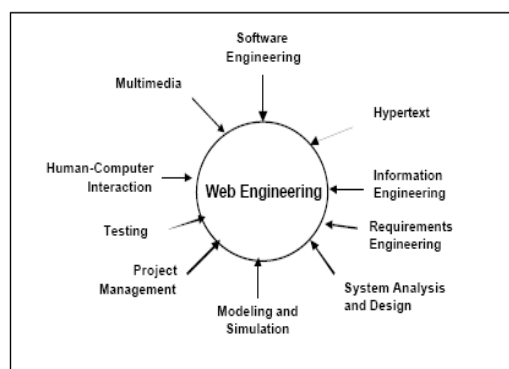


Figure 4. Web engineering

B. Web Engineering Activities

Since web engineering deals with all aspects of web-based system development, it too has many different activities akin to software engineering. These activities begin with specification, and continue with development, validation and evolution. However, specific web engineering activities include [12], [13]:

- Requirements engineering for web applications
- Techniques and methodologies for modeling web applications
- Design of functionality and interaction
- Implementation using a language for web-based applications
- Performance evaluation including verification and validation
- Operation and maintenance

Even more specific to web engineering activities are those that focus on the interaction between the application

and the user. Activities that focus on the humanistic side of web development include [12]:

- Human and cultural aspects
- User involvement and feedback
- End-user application development
- Education and training
- Team and staff development

The next section presents a semester-long project that introduces to undergraduate software engineering students an approach to developing a web-based application. The project combines the concepts of traditional software processes with web engineering.

V. DEVELOPING A WEB-BASED APPLICATION

It was the anticipation of the author that through the hands-on experience of developing a web-based application, students would gain an understanding of the software engineering process, various process models and how they could be manipulated and combined to develop a web-based application. It was also the anticipation of the author that students would understand that the fundamental ideas of software engineering are still relevant in the development of today’s software applications as well as those web-based applications.

A. The Project

The semester-long project selected for the fall 2010 semester was to develop a web-based application that could be used during the academic advising process by students and faculty in a medium-sized computer science department. The application was to replace a paper-based method used by students and faculty. The department has approximately 100 undergraduate majors which are advised by roughly seven faculty members. Student advisees are paired according to last name with a faculty advisor.

Currently, the department uses a paper-based method on which students and faculty record by hand the student’s courses, credit hours, grade earned, and semester in which a course is taken. The recording of this information is either done during the University’s registration period or during the academic advising timeframe which happens twice within the academic year, once during the fall semester and another in the spring semester. During the meeting, the student advisee discusses with the faculty advisor the progress made toward the completion of the computer science degree. At the meeting the student advisee and the faculty advisor review the paper. The paper is called a *Provisional Sheet*. The *Provisional Sheet* is updated by hand by the student and the faculty member. At the completion of the meeting, each has a copy of the updated *Provisional Sheet*.

However, problems arise if the *Provisional Sheet* is lost; if updates are made on one sheet and not the other; or if a grade, course number, or course credit is written incorrectly.

At the end of a student's matriculation, the updated *Provisional Sheet* which should now contain all the courses, grades, and credit hours in which the student was enrolled is given to the Office of the Registrar for graduation preparation. It is, at this time, if an error has occurred that it is identified. While both the student and the faculty member have sufficient time to correct an error that might have happened, the paper-based method does not truly permit for a check-and-balances procedure.

Therefore the focus of the semester-long project was to develop a web-based application that would alleviate some of the difficulties as seen with the paper-based *Provisional Sheet*. The goal of the project was to design the *Provisional Sheet* so that it could be completed and updated on-line by authorized users only, and be cross-checked with the University's student information system.

B. Learning Outcomes

Students were part of a team which was expected to meet with the customer (or representative) so that each phase of the process could be implemented. The team was also expected to produce a deliverable by the set deadline for each phase of the process and to also deliver it and make presentations to the customer (or representative).

The learning outcomes of the semester-long project included that after the completion of the project students would:

- Have a working knowledge of software engineering principles
- Understand how software engineering principles could be applied to the semester-long project
- Identify activities and implement strategies that were germane to the development of a web-based application
- Work effectively and efficiently in a team environment to produce the semester-long project

C. Project Requirements

Students were given basic requirements from the instructor for the web-based application; however, the majority of the requirements were gathered from stakeholders. Since the project was a web-based application, students had to consider basic requirements found in traditional software process models, as well as those that are part of the web engineering process.

D. Project Deliverables

Each item that the student team submitted was considered a deliverable. The project had four deliverables which were the requirements document, design document, implementation, and the test plan. The following is an overview of the project deliverables (i.e., models for web-based applications, language for web-based applications).

1) *Requirements Document*. The first document students were required to submit was the requirements document.

The requirements document was considered the official statement of what the students would implement. It included both the stakeholder requirements for the software application, which students named the *Computer Science Provisional System (CSPS)*, and a detailed specification of system requirements.

To capture the requirements students engaged in a modified version of the requirements engineering process as presented by Sommerville [1] and by Kappel et al. [13]. During the requirements engineering process, students met with stakeholders who included faculty members and students in the computer science department. Additionally students met with staff members in the Office of the Registrar as well as identified faculty and students in other academic units on campus in order to complete the elicitation and analysis phase of the requirements engineering process.

The document was meant to get the students actively involved in the planning and development of the application. Consequently, after the completion of the requirements document, students had an idea of the system architecture, functional and non-functional requirements, external interface specifications, how the application would be accessed, and by whom. Moreover, because this was to be a web-based application and not a traditional desktop application, students were charged with identifying varying levels of risk which included defining in the requirements document the method to secure student information and how change was to be accommodated.

2) *Design Document*. The design document was meant to be an in-depth description of the system design. The design showed how data flowed between system components and the trust relationships between components. Since the application was a web-based application, both the system and security requirements were described and explained how they would be implemented. Further since the *CSPS* would contain personal and confidential information, the design document elaborated on the requirements document risk analysis of critical characteristics of information that was presented in a module of the course designed by Lester on software security [14].

The team was required to use one of the decomposition strategies discussed in the course. The design document was required to have an introduction, an overview of the design strategy chosen, and the diagrams, charts, and/or details required as part of the decomposition strategy chosen. Additionally, the design document was to be based on one of the architectural design patterns which were discussed in class. The team chose the Model-View-Controller (MVC) pattern because of the stated advantages: team members had varying technical skill levels, MVC separates design concerns, and there is the likelihood of less code duplication [15].

3) *Implementation*. Students were required to implement the project based on the requirements and design documents. To implement the project students chose the PHP scripting language for use in a Linux environment. The student team chose this language and the environment based on familiarity and accessibility to the MySQL server database.

4) *Testing*. Students were required to develop a test plan which required them to perform development testing and end-user testing. The test plan was based on Sommerville's structure of a software test plan for large and complex systems but modified to be less formal and represent the smaller nature of the *Computer Science Provisional System* [1]. The modified version of the software test plan included: the testing process, test case design, hardware and software requirements and constraints.

VI. CONCLUSION

The aim of this paper was to present the results of an inquiry that introduced to undergraduate software engineering students an approach to developing a web-based application. The paper discusses traditional software engineering principles typically introduced in undergraduate software engineering courses which provides a case for introducing in these same courses some of the topics found in web engineering.

The study revealed that it is difficult to implement the entire software development life cycle. While students have a very good understanding of the requirements engineering process and developed a well-planned requirements document and design document, the implementation and testing phases proved to be challenging. The student team indicated that with the fixed timeframe of a sixteen week semester, it was difficult to fully implement the phases of the life cycle. Also, because students were under the impression that developing a web-based application would be similar to developing a web page or web site with which they had previous experience, they underestimated the time needed to produce the required documentation. They also underestimated the understanding of client/server network technology and the time needed to conduct accessibility and usability testing. Consequently, based on these observations, future work for the author is to re-design the semester-long from a heavy-weight process which is plan-driven to a more light-weight process focuses more on the software (i.e., the use of agile methods with a focus on extreme programming).

In conclusion, as the users of today's Web demand more from their web applications, it is the responsibility of educators to train the technologists of tomorrow to meet those demands. Consequently, the traditional software engineering practices that rely on well established development processes must also embrace change in order to continue to produce reliable, trustworthy software applications specifically developed for the Web.

ACKNOWLEDGMENTS

The author thanks the students enrolled in the CSCI 430-Software Engineering course, fall 2010 semester.

REFERENCES

- [1] I. Sommerville. (2011). *Software Engineering 9th Ed.* Addison Wesley, 13:978-0-13-703515-1, Boston, MA.
- [2] C. Angelov, R.V.N. Melnik, and J. Buur. (2003). The synergistic integration of mathematics, software engineering, and user-centered design: exploring new trends in education. *Future Generation Computer Systems*. Vol. 19, 299 – 1307.
- [3] B. K. Jayaswal and P.C. Patton (2007). Design for trustworthy software: Tools, techniques for developing robust software. Prentice Hall, 0-13-187250-8, Upper Saddle River, NJ.
- [4] Codebetter.com <http://codebetter.com/blogs/leonardlewallen/downloads/waterfallModel.gif>. (Accessed on October 10, 2009).
- [5] B. Boehm. (1988). A Spiral Model of Software Development and Enhancement. *IEEE Computer* 21, 5, 61-72.
- [6] F. Tsui and O. Karam. (2011). *Essentials of Software Engineering 2nd Ed.* Jones and Bartlett Publishers, 13:978-0-7637-8634-5.
- [7] D. Reed (2008). A Balanced Introduction to Computer Science 2nd Ed. Pearson Prentice Hall, 13:978-0-13-601381-5.
- [8] A Short History of the Web, Text of a speech delivered at the launching of the European branch of the W3 Consortium Paris, 2 November 1995. http://www.netvalley.com/archives/mirrors/robert_cailliau_speech.htm (Accessed May 31, 2011).
- [9] Internet users as percentage population. http://www.geohive.com/charts/ec_internet1.aspx (Accessed December 20, 2010).
- [10] A. Dix, J. Finlay, G.B. Abowd, and R. Beale. (2004). *Human-Computer Interaction*. Prentice Hall, Boston, MA.
- [11] A. Ginige and S. Murugesan. (2001). Web Engineering: A Methodology for Developing Scalable, Maintainable Web Applications. *Cutter IT Journal*. Vol. 14, No. 7, 24-35.
- [12] S. Murugesan, Y. Deshpande, S. Hansen, and A. Ginige. Web Engineering: A New Discipline for Development of Web-based Systems. (2001). *Proceedings of Web Engineering*, pp.3-13.
- [13] G. Kappel, B. Pröll, S. Reich, and W. Retschitzegger (eds), (2006). *Web Engineering - The Discipline of Systematic Development of Web Applications*. John Wiley & Sons.
- [14] C. Lester and F. Jamerson. "Incorporating software security into an undergraduate software engineering course," in *Proc. of the Third International Conference Emerging Security, Information Systems and Technology*, 2009, pp. 161-166.
- [15] Designing Enterprise Applications with J2EE Platform 2nd ed, http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html (Accessed August 4, 2011).

How to Think about Customer Value in Requirements Engineering

Xinwei Zhang^{1,2}, Guillaume Auriol^{1,2}, Claude Baron^{1,2}, Vikas Shukla^{1,2}

¹ CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse Cedex 4, France

² Université de Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France
{xwzhang, guillaume.auriol, claude.baron}@insa-toulouse.fr and vshukla@laas.fr

Abstract—Value is important for customer decisions and software design decisions. Understanding customer needs using value-focused thinking contributing to connecting customer needs and customer values and finally developing an approach of value-based Requirements Engineering. The main question of such approach is: how customer value can be reasonably quantified or measured? The ideas underlying our research are to qualify and quantify customer values on basis of the input of initial customer statements by introducing a set of techniques, e.g. multiple attributes preference theory and means-ends objectives network. In this paper, we give a preview on our proposed approach of qualitative and quantitative thinking that will enable value measurable and help make rational decision-makings.

Keywords-customer values, requirements engineering, multiple attributes preference theory, weights

I. INTRODUCTION

It is attractive to develop and provide software products that have high levels of value to customers, and that conform to the customer needs. Then there may be certain intrinsic relationships between customer needs and customer values. There is a substantive opportunity to clarify understanding of customer values and their relationships with customer needs, which finally contributes to an approach of value-based Requirements Engineering (RE). However, many discussions on RE are about deriving software requirements from customer needs and are value-neutral in nature [1][2]. Some discussions on value in RE literature are too subjective and qualitative to be quantified reasonably [3][4]. It is also easy to confusing value with weightings or rankings of needs or requirements [5][6] that are part of elements to quantify value.

Values are what customers *fundamentally* care about in decision-making [7]. There are lots of discussions about the concept and definition about “value”, and no consensus has been achieved. We think of value in a broad sense, including preference under certainty (value in a narrow sense) and preference under uncertainty (utility). All customer statements, such as needs and expectations, function and performance requirements, constraints, goals, indicate value. But some are important to customers because they are fundamentally important to customers themselves (value) while the others are means to influence the achievement of value. Researches in decision analysis have made distinctions between value models and consequences models

[7]. Value model incorporates the value or value tradeoffs and risk tolerances to evaluate consequences. Consequence models, such as performance models, model the influence relationships between design parameters to software performance.

Thus, discussions about customer values should be separated from software design solutions. The distinction is similar to the separation of *what* to do from *how* to do, or separation of *world* from *machine* in Jackson’s term [1]. We believe that understanding customer values is not so trivial and should be explored firstly in depth and width before used for software design, although iterations between customer values and means implementing customer values are always necessary.

In our approach, a set of techniques is introduced to clarify the understanding of customer needs and values. As customer statements are always expressed in different levels and granularities and some statements are even too vague to be appropriately understood, means-ends objectives network and fundamental objectives hierarchy are utilized to structure them reasonably. Then customer needs are quantified in term of value with multiple attributes preference theory. With these techniques and theory, it is then possible, for example, to model customer value and evaluate value contribution of various software solutions.

The rest of paper is structured as follows. Sections 2 gives an introduction to value-focused thinking approach. In Section 3, two techniques are introduced to structure the initially identified customer statements and to identify real customer needs. In Section 4, preference theory is utilized to construct value model with discussions about its implications for relevant problems. Section 5 outlines other related work. Finally, a conclusion is made with an outlook.

II. VALUE-FOCUSED THINKING APPROACH

Traditionally, general problem-solving approach is the alternative-focused thinking as the decision maker first focuses on alternatives, then on evaluating criteria. Value-focused thinking approach is different and proactive in nature. It first focuses on value and later on alternatives that might achieve it.

Qualitative and quantitative thinking of value is implied in value-focused thinking as shown in Fig. 1. It provides methodological basis of thinking about customer value in requirements engineering stage.

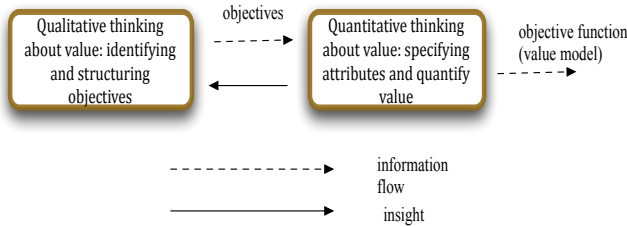


Figure 2. Qualitative and quantitative thinking of value [7]

III. UNDERSTANDING CUSTOMER NEEDS USING QUALITATIVE THINKING ABOUT VALUE

The initially elicited customer statements are usually in different levels and granularities, e.g. maximize security, access of database, sharing of information and maximize usability. “Access of database” influences negatively the security of software and is a possible means to influence the ends “maximum security”. It also is a part of “sharing of information” as others, such as “access other staffs’ files”, are also include as parts. “Sharing of information” then influences positively the usability of software and acts as one possible means of the ends “maximize usability”.

The intrinsic abstractions underlying these statements are means-ends and part-whole relationships, respectively. It is then useful to introduce some techniques encompassing these abstractions for structuring customer statements. Means-ends objectives network is utilized to trace statements in different levels and identify real customer needs hidden in statements. Fundamental objectives hierarchy is used to organize and expand the understanding of the real customer needs in different granularities. These two abstractions are also used to structure the intent specification of the software system [8].

The structuring process firstly performs means-ends analysis with customers on the initially identified customer statements. Typically, two kinds of questions are asked. One kind of questions, for example, “why this customer statement is important?” is asked to identify the ends of the statements. Then it may be always possible to ask the question of why, and may possibly arrive at statements in higher levels that may not be under current control and is not desired. An appropriate ending point to ask why question is when customers have identified the statements that are important because it is essential and important to customers in the decision context. We than obtain real customer needs in the same level. Another kind of questions, for example, “how this customer statement can be better achieved?” is asked to identify possible means to implement the customer statement, and then the first kind of questions is asked to pursue the real customer needs of these means. It is really a creatively thinking process to identify all possible means and customer needs for further exploration.

When a set of sufficiently complete customer needs in the same level has been identified, it is then necessary to clarify the understanding of each customer need in depth.

There are also two kinds of questions to be asked towards customers. One is to ask such question as “what do you mean by that customer need?” It helps to identify the parts of current need. Then we may also have the problem of when to end questioning. An appropriate ending point is when suitable attributes can be selected to measure the needs in the leaves in the hierarchy. On the other side, it is also possible to ask such question as “the customer need is a part of what?” It then traces the part to the whole. After questioning, a hierarchy of customer needs with part-whole relationships is established.

It is straightforward to find that these two techniques also support top-down and bottom-up reasoning that conforms to the usual style of human problem solving. So it is cognitively attractive for qualitative understanding of customer needs and values. Intuitively, it is similar to goal-oriented RE in structuring different levels of goals and sub-goals [9]. However, more careful examination is given to identify real customer needs and to represent them in a hierarchy with collectively complete and mutually exhaustive relationships, which facilitates to verify underlying independence conditions between them.

IV. UNDERSTANDING CUSTOMER NEEDS USING QUANTITATIVE THINKING ABOUT VALUE

The bridge that connects qualification and quantification is the selection of attribute to measure the degree to which the customer need is met. However, it is a missing element in analytic hierarchy process (AHP) and quality function deployment (QFD). Then weights of customer needs are possibly assigned independently of the attributes and their range information. But according to classic utility theory, it makes no sense to say that for example in context of selecting the best suitable software to buy, minimum software cost is important than maximum performance, or vise versa. It all depends on how much you talk about cost and performance, respectively, and where you start. It is meaningful to say that cost is important than performance only when the range of change in cost from the starting point of attribute cost is more important than the range of change in performance from the starting point of attributes performance. Three kinds of attributes are usually used for measuring with different pros and cons. The procedure and criteria to be satisfied to select desired attributes are extensively discussed in [7][10][11].

After specifying attributes, it is then necessary to check the independence conditions between the set of selected attributes. There are usually several kinds of independence conditions in concert with three major function forms [12]. When additive independence is satisfied, that is, the preference order for lotteries depends only on their marginal possibility distribution, additive function form

$$u(x_1, \dots, x_M) = \sum_{i=1}^M k_i u_i(x_i) \quad (1)$$

exists, where u_i is a single attribute utility function over attributes X_i , and the k_i are scaling constants subjecting to $\sum_{i=1}^M k_i = 1, k_i \geq 0, M \geq 2$.

This function form is similar to typically additive linear function form to calculate customer satisfaction in QFD [13][14],

$$S = \sum_{i=1}^M d_i s_i \quad (2)$$

where S is overall customer satisfaction, d_i is degree of importance of the i th customer need, and s_i is degree of attainment of the j th engineering characteristic.

However, there are at least two obvious distinctions as follows:

1) k_i in equation (1) is a relative weight of attribute X_i or corresponding need. It is determined by making value trade-offs between attributes. To assess k_i , at least M equations with k_i 's ($i=1, \dots, M$) as unknowns should be found and solved while it is necessary to identify a pair of two consequences $C_1=(x_1, x_2, \dots, x_M)$ and $C^*=(x_1^*, x_2^*, \dots, x_M^*)$ that are indifferent to customers to construct one equation. d_i , however, is usually determined by a direct weighting methods and then a normalization process without considering attributes information. Some representative methods in this kind are Analytic Hierarchy Process, 9-point direct-rating scale.

2) u_i is a single attributes utility function over attribute X_i . It can be an increasing, decreasing or non-monotonic utility function and be of concave, linear or convex form. For example, one customer is of risk aversion over cost of buying software, the corresponding function form is a decreasing utility function with concave shape. When customer is of risk neutrality, the function form is linear and consistent with typically used function form as equation (2). In this situation, every unit of achievement of attribute has the same effect on customer satisfaction. This formalization of single attribute utility function is similar to the discussion on KANO model that distinguish three categories of customer needs with distinctive customer satisfaction relationships, respectively. In KANO model, however, there is no assessment of mathematical function between customer satisfaction and different levels of achievement of customer need and no consideration of risk attitudes toward uncertainty of attribute attainment.

It is easy to find that even in this simple function form distinctions can be made. So it is necessary to rigorously test the underlying relationships between the attributes and to verify whether certain independence conditions are satisfied.

By introducing multiple attributes preference theory, there are at least following extra benefits:

1) Design features (means) are important because of their implications to implementation of real customer needs. Then weights of the means should be derived from weights of customer needs multiplied by their contribution to the achievement of customer needs. An opportunity exists to

model the relationships and function between means and achievement of customer needs,

2) Weights of needs are subjective and may be imprecise. It is then useful to do sensitivity analysis based on the available weighting information on the constructed value model,

3) Customers' group preferences can be reasonably derived from individual customer preferences by formalizing customer preferences using preference theory. It is then also possible to discuss the fairness between customers, and

4) It is also possible to improve the release planning problems by selecting a set of requirements to be implemented to maximize customer values. The *value* in cost-value approach and the *benefit* in benefit-cost ratio are then should be rectified by introducing attributes and their possibly non-linear utility functions.

V. RELATED WORK

Researches on value in RE are far to be satisfying. The concepts, sources and dimensions of value are usually discussed in literatures without a widely acceptable definition or formulation [3][4]. Some possibly limited quantification about value focus only on weighting or prioritize requirements and resource allocation [5][6]. Several approaches are usually used in the process, e.g. AHP, QFD and cost-value approach. However, these approaches, are all controversial in their validity to model customer preferences under uncertainty. Especially, weighting and relative customer satisfaction calculation in QFD is subject to certain strong set of preference independence assumptions. It is not appropriate to use them directly without verifying preference independence assumptions among attributes.

A recent proposition that is relevant to value-based RE is value-based software engineering. It proposes a framework on the basis of "4+1" theories [15]. Four of five theories in the framework: utility theory, decision theory, dependency theory (causality) and theory W (group decision making) perform almost the same works as decision analysis does. We think it is also interesting to adopt these theories to RE stage.

Multiple attributes preference theory is in the core of modeling customer values in decision analysis, and it is especially useful when customers have multiple, conflicting needs and when there may be uncertainty in the software performance and cost. Integrating these techniques from decision analysis provides a fertile field to be explored to model customer value reasonably and effectively.

VI. CONCLUSIONS AND FUTURE WORKS

We have presented the approach to understand customer needs using value-focused thinking. Customer values then become an explicit construct that can be modeled qualitatively and quantitatively. Weighting of customer needs and requirements are also discussed to enable a

reasonable way of assigning. These give important implications to value-based RE.

However, it is found that sometimes there is difficulty to make judgments whether a statement is a means or a part of another statement, although they are obviously different in concept. Some extra researches are needed to explore the point. And the proposed approach adds cognitive and modeling burden to customers and engineers, and is time-consuming. It is expected that some reasonable simplifications or approximations can be made according to the actual application contexts, making it more practical and applicable.

We are currently preparing its applications in the RE stage of aircraft system development to test its validity. A work package “requirements establishment and value generation” is initiated and some test cases have been collected. Further results about the approach and its practical applications will be reported.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement n°234344.

REFERENCES

- [1] M. Jackson, “The world and the machine,” Proc. the 17th International Conference on Software Engineering, ACM Press, Seattle, Washington, United States, 1995.
- [2] M. Jackson, “Problem frames: analyzing and structuring software development problems,” Addison-Wesley Professional, 2001.
- [3] R. Proynova, B. Paech, A. Wicht, and T. Wetter, “Use of personal values in requirements engineering –a research preview,” Proc. FEFSQ 2011, LNCS 6182, pp. 17-22. Springer, Heidelberg.
- [4] M. Kauppinen, J. Savolainen, L. Lehtola, M. Komssi, H. Tohonen, and A. Davis, “From feature development to customer value creation,” Proc. the 17th International Requirements Engineering Conference, IEEE Computer Society, Atlanta, United States, 2009.
- [5] J. Karlsson and K. Ryan, “Prioritizing requirements using a cost-value approach,” IEEE Software, vol. 14, pp. 67-74, 1997.
- [6] J. Karlsson, S. Olsson, and K. Ryan, “Improved practical support for large-scale requirements prioritizing,” Requirements Engineering, vol. 2, pp. 51-60, 1997.
- [7] R. L. Keeney, “Value-focused thinking,” Harvard University Press, Cambridge, MA, 1992.
- [8] N. Leveson, “Intent specifications: an approach to building human-centered specifications,” IEEE Trans. on Software Engineering, vol. 26, pp. 15-35, 2000.
- [9] A.V. Lamsweerde, “Requirements engineering: from craft to discipline, Proc. the 16th ACM Sigsoft Intl. Symposium on the Foundations of Software Engineering, ACM press, Atlanta , 2008.
- [10] R. L. Keeney, “Common mistakes in making value trade-off,” Operation Research, vol. 50, pp. 935-945, 2002.
- [11] R. L. Keeney, “Selecting attributes to measure the achievement of objectives,” Operation Research, vol. 53, pp. 1-11, 2005.
- [12] R. L. Keene and H. Raiffa, “Decisions with multiple objectives: preferences and value trade-offs,” Cambridge University Press, 1993.
- [13] I. V. de Poel, “Methodological problems in QFD and directions for future development,” Research in Engineering Design, vol. 18, pp. 21-36, 2007.
- [14] J. R. Hauser, and D. Clausing, “The house of quality,” Harvard Business Review, vol. 66, pp. 63-74, 1988.
- [15] S. Biffel, A. Aurum, B. Boehm, H. Erdogmus, and P. Grunbacher, “Value-based software engineering,” Springer, New York, 2006.

Migrating Functional Requirements in SSUCD Use Cases to a More Formal Representation

Mohamed El-Attar

Information and Computer Science Department
King Fahd University of Petroleum and Minerals
P.O. 5066, Al Dhahran 31261, Kingdom of Saudi Arabia
melattar@kfupm.edu.sa

James Miller

STEAM Laboratory
Department of Electrical and Computer Engineering
University of Alberta, Edmonton, Alberta, Canada
jm@ece.ualberta.ca

Abstract- Use case modeling is a popular technique to elicit and model functional requirements of a software development project. In a use case driven development methodology, use cases are used as a basis to guide the development of UML design models. In this paper, we provide a model transformation approach to transform use cases descriptions written in a nearly unstructured form to a more formal representation. A more formal representation, which is machine-readable, can be used to systematically generate other UML design models, in particular UML activity diagrams. The main advantage of using this model transformation approach is to avoid potential errors introduced by modelers if they were to develop the UML design models while depending solely on their skill and experience. The proposed model transformation approach is applied to a library system to demonstrate its applicability and to validate its correctness and effectiveness.

Keywords – Use Cases; SSUCD; SUCD; Model Transformation.

I. INTRODUCTION

Use case diagrams [3, 6] have become the de-facto modeling tool to elicit and model functional requirements for object-oriented software development projects. In a use case driven development methodology, the use case model is developed at the analysis phase used to drive the development of other UML (Unified Modeling Language) [12] design artifacts at the design phase. This process is far from straightforward since naturally there is a gap between the analysis and design phases. If the development of UML design artifacts based on use case models is dependent solely on human skill, experience and judgment, then there will be a great risk of developing design artifacts that have a design view which is inconsistent with the analytical view as presented by the use case model. As a result, system architects may construct a design that provides different functionality than that required (i.e., developing the ‘wrong’ system), leading to costly reworks and schedule overruns, in addition to the intangible cost of unsatisfied customers.

Model transformation provides a more rigorous approach towards developing UML design artifacts based on use case models. Model transformation greatly reduces the human factor during the development process thus increasing the likelihood of developing a system that satisfies its prescribed functional requirements. To this end, this paper presents a model transformation approach that transforms use cases written in a form named SSUCD (Simple Structured Use Case Descriptions) [2] to another more formal representation named SUCD (Structured Use Case Descriptions) [5]. SUCD is a

language first introduced in [5] that is used to structure use case descriptions by embedding enough structure within the use case descriptions to facilitate the transformation of workflows in use case descriptions into UML activity diagrams. Use cases are ideally written by business analysts. In [1], an experiment was conducted which revealed that the language SUCD was too difficult to be used by its potential users (business analysts). The experiment indicated that when using SUCD, the majority of defects detected in the models developed were due to syntax errors resulting from using of the SUCD language. Consequently, the authors of the SUCD language developed a simplified version of SUCD, which is SSUCD [2]. SSUCD was intentionally designed to be accessible to business analysts. The usability of SSUCD was empirically evaluated in [1]. The results of the experiment indicate that users of SSUCD develop higher quality use case models. SSUCD was also intentionally designed to help business analysts develop use case models that are consistent to combat the issue of developing inconsistent use case models when not utilizing any structure.

The remainder of the paper is organized as follows: Section 2 briefly outlines the related work and provides an introduction to the SSUCD and SUCD languages. The proposed model transformation technique is detailed in Section 3. In Section 4, a library system case study is used to evaluate the correctness and effectiveness of the proposed transformation technique. Finally, Section 5 concludes and discusses future work.

II. BACKGROUND AND RELATED WORK

There exist two tools that automate the generation of activity diagrams from use case models such as “Catalyze Suite” [8] and “TopTeam Analyst” [9]. Both tools produce diagrams similar to UML activity diagrams, which their developers refer to as ‘Flow diagrams’. However, such tools and methods depend on the utilization of use case descriptions with no structure, meaning that the source use case descriptions used are vulnerable to inconsistencies. If inconsistent use case models are used as a source to generate UML activity diagrams, therefore these inconsistencies will propagate onwards to the UML activity diagrams, which in turn will propagate to the implementation source code where the cost of fixing such inconsistency error escalates significantly. Therefore, tools that generate UML activity diagrams should be geared towards using the SSUCD language to ensure that the source use case models used are consistent. This approach presented in this paper uses use cases written in the SSUCD form to contribute towards the

overall goal of generating UML activity diagrams that provide a consistent and correct view of the system's functional requirements.

III. A BRIEF BACKGROUND TO SSUCD AND SUCD

The model transformation approach proposed in this paper depends on using SSUCD use case descriptions as input and produced SUCD use case descriptions as output. As a prelude to outlining the model transformation mapping rules and algorithms shown in Section 3, it is necessary to briefly introduce the SSUCD and SUCD languages; the components used in the model transformation. To this end, this section provides a brief introduction to SSUCD and SUCD using a use case description of a system outlined in [5]. The use case is concerned with the functionality of borrowing a book from a library. SSUCD and SUCD use cases do not mandate any particular template to be used. SSUCD and SUCD use cases however require a minimal set of fields to be present in a use case description. The fields required are the (a) Use Case Name section, (b) the Associated Actors section, (c) the Description section, and (d) the Extension Points section. SUCD use cases, being more formal than SSUCD, do contain further subsections within some sections of its template. For example, in the Extension Points section, SUCD use cases case outline Public Extension Points and Private Extension Points. A detailed description of the SSUCD and SUCD languages are out of the scope of this paper. For detailed descriptions of the SSUCD and SUCD languages as well as their formal syntax, our interested readers are referred to [2] and [5], respectively. However, to illustrate the difference between using both languages, Figures 1 and 2 show the textual description of the "Borrow Book" use case using SSUCD and SUCD, respectively.

Postconditions:
The number of borrowed books in the member's record is increased by one

Extension Points:
Balance overdue

Fig. 1. The description of the Borrow Book use case described in SSUCD

Use Case Name:
Borrow Book

Brief Description:
This use case is initiated by a Member to allow that member to borrow a book. A Librarian is then involved to carry out the transaction.

Preconditions:
The book must exist

Basic Flow:

```
{BEGIN Use Case}

{BEGIN bring book to borrow}
• Member -> Brings the book he/she would like to borrow
• PERFORM Retrieve book information (2)
• Member -> Provides library card
• Librarian -> Scans member's card
{END bring book to borrow}

{BEGIN authenticate librarian}
• INCLUDE Authenticate Librarian (1)
{END authenticate librarian}

{BEGIN scan book}
• Librarian -> Scan's book's barcode
RESUME {update member's record} {update book's status} (5)
{END scan book}

{BEGIN update member's record}
• Librarian -> Updates the Member's record with the newly borrowed book
RESUME {END}
{END update member's record}

{BEGIN update book's status}
• SYSTEM -> Changes the book's status in the database to 'Borrowed'
{END update book's status}

{END Use Case}
```

Alternative Flows:
FLOW Basic Flow (3)
AT {scan book} (4)
• Librarian -> Scans the book's barcode
IF barcode cannot be scanned
{BEGIN enter barcode manually}
• Librarian -> Enters the book's barcode number manually

Use Case Name:
Borrow Book

Brief Description:
This use case is initiated by a Member to allow that member to borrow a book. A Librarian is then involved to carry out the transaction.

Preconditions:
The book must exist

Basic Flow:
The use case begins when a member brings a book they would like to borrow. Information about the book is then retrieved from the database by entering the book's name or barcode. The member then provides their library card for the librarian to scan. The librarian needs to authenticate first before scanning the book's barcode. The librarian then updates the member's record with the newly borrowed book. The book's status is then changed in the database and set as 'Borrowed'.

Alternative Flow:
When the librarian scans the book's barcode, if the barcode cannot be scanned, then the book's barcode is entered manually.


```
{END enter barcode manually}
CONTINUE {update member's record} {update book's status}
Subflows:

SUBFLOW Retrieve book information
{BEGIN enter and retrieve book information}
• Librarian -> enters the book's name or barcode
• SYSTEM -> retrieve the given book's information from
database
{END enter and retrieve book information}

Postconditions:
The number of borrowed books in the member's record is
increased by one

PUBLIC EXTENSION POINT
Balance overdue
```

high-level metamodells for SUCD and SSUCD are shown in Figures 3 and 4, respectively.

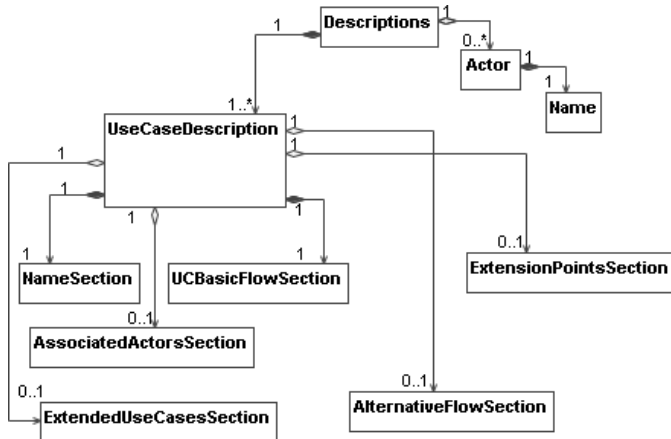


Fig. 3. The high-level components of the SUCD metamodel.

As shown in Figures 3 and 4, the SSUCD and SUCD metamodels formats the use case descriptions into several sections represented as objects. Each section describes a certain important aspect of the use case. For example, "AssociatedActorsSection" object is used to describe the list of actors associated with the given use case. It can be shown that the SSUCD metamodel is a simplified version of SUCD as the metamodel of SSUCD is a subset of the SUCD metamodel. ANTLR generated a compiler that can parse SSUCD use case descriptions. The compilation process results in the generation of an object model that represents the given SSUCD use cases.

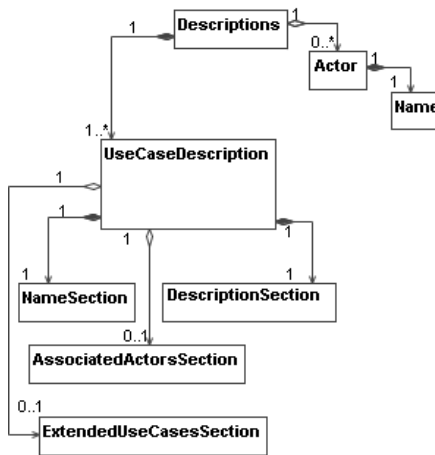


Fig. 4. The high-level components of the SSUCD metamodel.

Fig. 2. The description of the Borrow Book use case described in SUCD

It can be easily deduced from Figures 1 and 2 that SUCD use case descriptions contain far more structure than SSUCD use cases. This is the chief motivation behind this work. Due to the complexity of this transformation problem, if the transformation is performed manually then there will be a great risk of developing SUCD use cases that are inconsistent with their corresponding source SSUCD use cases.

IV. TRANSFORMING USE CASES FROM SSUCD TO SUCD

This section describes the preparation activities requisite for the transformation process to take place. In this section, we also present the model transformation rules and algorithms. Automation support is important to ensure the syntactical correctness of the models used and created and to ensure the speed and accuracy of the application of the transformation process and therefore the transformation rules and algorithms were coded using two popular tools within the model transformation research community.

A. Generating the Metamodels of SSUCD and SUCD

A model transformation process uses a source model to produce a target model based on the transformation rules and algorithm. As a prerequisite to the execution of the transformation, the source model needs to conform to a metamodel. The conformance of the source model to a metamodel ensures the syntactical correctness of the source model. Similarly, the target model also needs to conform to metamodel to ensure the syntactical correctness of the produced model. The derivation of the metamodels for both source and target models were reverse engineered from the EBNF rules for both SSUCD and SUCD, respectively. The EBNF rules for SSUCD and SUCD are defined in [2] and [5], respectively. The metamodels were reverse engineering using a tool named ANTLR (ANother Tool for Language Recognition) [7]. ANTLR is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages [7]. The

Transformation Mapping Rules and Algorithms

The transformation and mapping rules and algorithms prescribe the process through which a source model is transformed into a target model. In the scope of model-driven engineering, the transformation mapping rules and algorithms are defined in terms of model, which in turn must conform to a metamodel. The ATL (Atlas Transformation Language) metamodel was chosen as the metamodel for model transformation problem considered in this paper. ATL was selected since it provides two methods to describe

transformation rules: (a) using “matching rules” (declarative programming); and (b) using “called rules” (imperative programming). For the transformation problem at hand it was necessary to use both types of programming methods provided by ATL as complex transformation algorithms can be too difficult to program declaratively only [10]. Figures 5→8 outline the mapping rules and algorithms of the proposed model transformation technique per description section.

Use Case Name Section	
SSUCD	SUCD
<p>The “Use Case Name” section starts with the label “Use Case Name:”</p> <p>MAPPING1: Every use case in the model must have a name and therefore this section must exist in every use case description</p>	<p>The “Use Case Name Section” starts with the label “Use Case Name:”</p> <p>MAPPING1: Every use case in the model must have a name and therefore this must exist in every use case description.</p>
<p>If the use case is abstract then this section is followed by the keyword “ABSTRACT”</p> <p>Use case name as-is in free flow NL</p> <p>MAPPING2: Use case name must be unique in the entire model. No two use cases can have the same name.</p>	<p>If the use case is abstract, this section then is followed by the keyword “ABSTRACT”</p> <p>Use case name as-is in free flow NL</p> <p>MAPPING2: Use case name must be unique in the entire model. No two use cases can have the same name.</p>
<p>Transformation Rule: The use case names must be exactly the same in both SSUCD and SUCD.</p>	
<p>If the use case is implementing an abstract use case, the keyword “IMPLEMENTS” is shown followed by the name of the abstract use case. Any additional abstract use cases which the given use case implements, is stated by using a comma followed by the name of the other abstract use cases. For example: IMPLEMENTS UseCaseA, UseCaseB, UseCaseC</p> <p>MAPPING3: Use cases that are implemented must exist in the target model.</p> <p>MAPPING4: Use cases that are implemented must be abstract. In other words, they should have the keyword “ABSTRACT” in their “Use Case Name” section.</p>	<p>If the use case is implementing an abstract use case, the keyword “IMPLEMENTS” is shown followed by the name of the abstract use case. Any additional abstract use cases which the given use case implements, is stated by using a comma followed by the name of the other abstract use cases. For example: IMPLEMENTS UseCaseA, UseCaseB, UseCaseC</p> <p>MAPPING3: Use cases that are implemented must exist in the target model.</p> <p>MAPPING4: Use cases that are implemented must be abstract. In other words, they should have the keyword “ABSTRACT” in their “Use Case Name” section.</p>

Transformation Rule: The names of the implemented use cases in both SSUCD and SUCD must match. Both SSUCD and SUCD must include the keyword ABSTRACT.

<p>If the use case is specializing a concrete use case, the keyword “SPECIALIZES” is shown followed by the name of the concrete use case. Any additional concrete use cases which the given use case specializes, is stated by using a comma followed by the name of the other concrete use cases. For example: SPECIALIZES UseCaseA, UseCaseB, UseCaseC</p> <p>MAPPING5: Use cases that are specialized must exist in the target model.</p> <p>MAPPING6: Use cases that are specialized must NOT be abstract. In other words, they should NOT have the keyword “ABSTRACT” in their “Use Case Name” section.</p>	<p>If the use case is specializing a concrete use case, the keyword “SPECIALIZES” is shown followed by the name of the concrete use case. Any additional concrete use cases which the given use case specializes, is stated by using a comma followed by the name of the other concrete use cases. For example: SPECIALIZES UseCaseA, UseCaseB, UseCaseC</p> <p>MAPPING5: Use cases that are specialized must exist in the target model.</p> <p>MAPPING6: Use cases that are specialized must NOT be abstract. In other words, they should NOT have the keyword “ABSTRACT” in their “Use Case Name” section.</p>
--	--

Transformation Rule: The names of the parent use cases in both SSUCD and SUCD must match. Both SSUCD and SUCD must NOT include the keyword ABSTRACT.

Fig. 5. Transforming the “Name Section”

Figure 5 outlines the mapping rules for the “Name Section”. The purpose of “Name Section” is mainly to specify the name of the use case. The “Name Section” is also used to specify if the use case is abstract or concrete. Moreover, the “Name Section” is also used to specify if the use case is *generalizing* or *specializing* another use case. The transformation process of the “Name Section” can be fully automated since this section is very similar in the SSUCD and SUCD forms.

Associated Actors Section	
SSUCD	SUCD
<p>The “Associated Actors” section starts with label “Associated Actors:”</p> <p>MAPPING7: If the use case does not have any actors associated with it then this section is removed entirely.</p>	<p>The “Associated Actors” section starts with label “Associated Actors:”</p> <p>MAPPING7: If the use case does not have any actors associated with it then this section is removed entirely.</p>

Any associated actors are then listed (comma separated) in a new line as such: ActorA, ActorB, ActorC. MAPPING8: Actors listed in this section must exist in the model. In other words, there must be actor descriptions with the stated actor names in the "Actor Name" section.	Any associated actors are then listed (comma separated) in a new line as such: ActorA, ActorB, ActorC. MAPPING8: Actors listed in this section must exist in the model. In other words, there must be actor descriptions with the stated actor names in the "Actor Name" section.
Transformation Rule: The names of the actors listed in both SSUCD and SUCD must match.	

Fig. 6. Transforming the "Associated Actors Section"

Figure 5 outlines the mapping rules for the "Associated Actors Section". The purpose of "Associated Actors Section" is mainly to specify the names of any actors involved with the use case. Once again the "Associated Actors Sections" of the SSUCD and SUCD forms are very similar hence the transformation is straightforward and fully automated.

Description Section	
SSUCD	SUCD
The "Description" section starts with label "Description:" MAPPING9: Every use case must have a description. Therefore, every use case must have a "Description" section.	The "Description" section starts with label "Description:" MAPPING10: Every use case must have a description. Therefore, every use case must have a "Description" section.
The "Description" section in SSUCD is populated with the free flow Natural Language. The only structure involved in this section is the use of the "INCLUDE" keyword. The "INCLUDE" keyword is used to indicated other use cases which the given use case includes. The "INCLUDE" keyword is embedded within the free flow text. It is used by showing the keyword "INCLUDE" followed by two angled brackets (< >). The name of the included use case is stated between the angled brackets. For example:	The "Description" section then must have a "Basic Flow:" label.
Description: Free-flow text, free-flow text.... INCLUDE <UseCaseA> free-flow text, free-flow text... MAPPING10: The name of stated inclusion use case (the included use case) must exist in the mo	The heart of "Description" section basically consists of "headers" which contain "actions". An "action" basically consists of a bullet point, followed by the name of the actor performing the action then followed by an arrow → then followed by the action description written in natural language. For example: • Librarian → Enter member's

Transformation Rule: The conversion of this section will be semi-automated. There are only two rules to consider when converting this section. First, the actor names used in SUCD must be listed in the "Associated Actors" section of SSUCD (apart from the SYSTEM actor). Secondly, the "include" statement in SSUCD use cases stated as such INCLUDE <UseCaseA> must be mentioned at least one once in SUCD and stated as such: • INCLUDE <UseCaseA>.
--

Fig 7. Transforming the "Description Section"

Figure 7 outlines the mapping rules for the "Description Section". The purpose of "Description Section" is mainly to describe the behavior of the use case. In the SSUCD form, the description is provided in an unstructured natural language form. The only exception being the specification of an included use case where the keyword INCLUDE is used to specify the inclusion use case. Meanwhile, in the SUCD form the description section is far more structured. Each statement is specified individually along with the actor that is responsible for performing the actor. If the performer is the system itself, then the keyword SYSTEM is used. Hence, the transformation process of the "Description Section" cannot be fully automated and it requires human cognition to partition the description in the SSUCD form to bullet points in the SUCD form.

Extension Points Section	
SSUCD	SUCD
The "Extension Points" section starts with label "Extension Points:" MAPPING11: If the use case does not have any public extension points this section is removed entirely.	Public Extension Points Section: The "Public Extension Points" section starts with label "Public Extension Points:" MAPPING11: If the use case does not have any public extension points this section is removed entirely.
The name of the extension points are then listed while separated with commas as follows:	The name of the extension points are then listed while separated with commas as follows:
Extension Points: <EP1>, <EP2>, <EP3>...	Public Extension Points: <EP1>, <EP2>, <EP3>...
Transformation Rule: The names of the publication extension points listed in both SSUCD and SUCD must match.	
For an extension use case, it states the it extends another use case using the following structure: Extended Use Cases: Base UC Name: <UseCaseA> Extension Point: <EP_Name> IF <Condition> MAPPING12: The name of	For an extension use case, it states the it extends another use case as well as it states the extension behavior using the following structure: PUBLIC EXTENSION POINT BEHAVIOR EXTENDING {UseCaseA : EP_Name} MAPPING12: The name of stated use case being

<i>stated use case being extended must exist.</i>	<i>extended must exist in the model.</i>
MAPPING13: <i>The name of the stated extension point must be listed in the "Extension Points" section of the extended use case.</i>	MAPPING13: <i>The name of the stated extension point must be listed in the "Extension Points" section of the extended use case.</i>
Transformation Rule: The use case name stated in SSUCD as Base UC Name: <UseCaseA>, must be the same as that stated in SUCD as EXTENDING {UseCaseA: ...etc. The extension point name stated in SSUCD as Extension Point: <EP_Name>, must be the same as that stated in SUCD as EXTENDING {UseCaseA : EP_Name}.	

Fig 8. Transforming the "Extension Points Section"

Figure 8 outlines the mapping rules for the "Extension Points Section". The purpose of "Extension Points Section" is to state the extension points of a use case. The transformation process of the "Extension Points Section" may also be fully automated.

V. LIBRARY SYSTEM CASE STUDY

The library system discussed in this section was previously presented in [5], which the research work that introduced SUCD. This library system was specifically to evaluate the correctness of the proposed model transformation technique as the work presented in [5] outlines a set of use cases and how they are transformed into UML activity diagrams. In order to perform the evaluation, the SUCD use cases were rewritten as SSUCD use cases by extracting only the information required by the SSUCD structure. The entire set SUCD use cases and their corresponding SSUCD use cases used in this case study are available in [5]. For illustrative purposes, an example of a SUCD use case presented in [5] and its reverse-engineering SSUCD version are shown in Figures 1 and 2 (see Section 2), respectively.

A. Applying the Model Transformation

Using the reserve engineering SSUCD use cases as the source, the textual descriptions were analyzed by ANTLR to generate a representative object models that conform to the metamodel previously produced (see Section 3.1). The generated object models were used as input by ATL to apply the model transformation algorithms and mapping rules previously encoded. ATL then generates a set of object models that represent the SUCD equivalent of the SSUCD use cases used as input. As encoded in ATL, the generated object models representing the SUCD use cases are set to conform to the target metamodel (see Figure 3). A simple tool was used to read the generated object models representing the SUCD use cases to produce the text files presenting the SUCD use cases in a textual form.

B. Verifying the Correctness of the Produced SUCD Use Case Descriptions

The correctness of the produced SUCD use case descriptions was verified through two distinct means. The first approach involved the use of the *Diff* tool to check for differences between the produced SUCD use cases and the SUCD use cases already shown in [5]. Although some minor differences were found in the layout (white spaces and empty lines), the textual content of the use case descriptions were confirmed to be the same.

The second approach used to verify the correctness is to verify that the generated SUCD use case descriptions can be used as a source to generate representative UML activity diagrams using the approach presented in [5] to produce UML activity diagrams that match the UML activity diagrams shown in [5]. The generated UML activity diagrams along with the UML activity diagrams already shown in [5] were used as input by a tool named *UMLDiff* [11]. *UMLDiff* is an automated UML-aware structure differences algorithm which uses as input two object-oriented models then produces a report of the design evolution of the software system in the form of a change tree [11]. For given object models (representing the UML activity diagrams), the *UMLDiff* tool did not report any differences between the UML activity diagrams.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented an approach that helps bridge the gap between the analysis and design phases in a use case-driven development process. The contribution of this paper is a model transformation technique that is almost fully automated, which can be used to transform use case descriptions written in the SSUCD form to use cases written in the SUCD form which may then be used to generate other types of UML design artifacts. The model transformation technique helps eliminate the human factor and thus eliminating human injected errors that may result from perform the transformation completely manually. The proposed approach was applied to use cases of a library system already presented in the literature. The correctness of the proposed technique was verified by differencing the textual descriptions of the generated SUCD use case descriptions with the SUCD use case descriptions presented in [5]. The second approach involved the use of a popular tool in the model differencing research community, named *UMLDiff*, to compare the UML activity diagrams produced with the generated SUCD use case descriptions against the UML activity diagrams already presented in [Seattle]. Both verification approaches indicate the correctness and the effectiveness of the proposed technique.

Future work will be directed towards extending the SSUCD and SUCD languages to allow for the specification of functional security requirements. The model transformation technique will also need to be enhanced to facilitate the transformation of the extended SSUCD and SUCD languages.

ACKNOWLEDGEMENTS

The author would like to acknowledge the support provided by the Deanship of Scientific Research (DSR) at King Fahd University of Petroleum & Minerals (KFUPM) for funding this work through project No. JF100008.

REFERENCES

- [1] M. El-Attar and J. Miller, A Subject-Based Empirical Evaluation of SSUCD's Performance in Reducing Inconsistencies in Use Case Models, *Journal of Empirical Software Engineering*, vol.14, no. 5, pp. 477-512, (2009).
- [2] M. El-Attar and J. Miller, Producing Robust Use Case Diagrams via Reverse Engineering of Use Case Descriptions, *Journal of Software and Systems Modeling*, vol. 7, no. 1, pp. 67-83 (2008).
- [3] I. Jacobson, M. Ericsson, and A. Jacobson, *The Object Advantage*. ACM Press, 1995.
- [4] M. El-Attar and J. Miller, A User-Centered Approach to Modeling BPEL Business Processes Using SUCD Use Cases. *Journal of Software Development and Theory, Practice and Experimentation*, vol. 1, no. 1, pp. 59-76 (2007)
- [5] M. El-Attar and J. Miller, AGADUC: Towards a More Precise Presentation of Functional Requirements in Use Case Models, 4th ACIS International Conference on Software Engineering, Research, Management and Applications, Seattle, Washington, USA. pp.346-353, (2006).
- [6] Object Management Group, UML Superstructure Specification (2005). <http://www.omg.org/docs/formal/05-07-04.pdf>, Version 2.0 formal/05-07-04. Accessed March 2011.
- [7] T. Parr, *The Definitive ANTLR Reference: Building Domain-Specific Languages (Pragmatic Programmers)*. Pragmatic Bookshelf, 2007.
- [8] SteelTrace. "Catalyze Suite". Available [Online] www.steeltrace.com. Last Accessed March 2011.
- [9] TechnoSolutions. "Top Team Analsyt". Available [Online] http://www.technosolutions.com/topteam_requirements_management.html. Last Accessed March 2011.
- [10] The Eclipse Foundation. ATL – A Model Transformation Technology. Available Online at (<http://www.eclipse.org/atl/>). Last accessed March 2011.
- [11] Z. Xing and E. Strou, UMLDiff: An Algorithm for Object-Oriented Design Differencing, *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pp. 54-65, 2005.
- [12] OMG 2003, "UML Superstructure Specification", Object Management Group, <http://www.omg.org/docs/ptc/03-08-02.pdf>, 2003.

KM-SORE: Knowledge Management for Service Oriented Requirements Engineering

Muneera Bano, Naveed Ikram
 Department of Software Engineering
 International Islamic University Islamabad, Pakistan
muneera@iiu.edu.pk, naveed.ikram@iiu.edu.pk

Abstract—Service-oriented Software Engineering is a new style for creating software using reusable services which are available over the web. The biggest challenge in this process is to discover and select the appropriate services that match system requirements. Currently, none of the proposed approach has been accepted by research community as a standard. There is very little empirical work available that addresses requirements engineering in service oriented paradigm. The aim of this study is to propose a framework for requirements engineering in SOSE. The framework is based on a new idea, that integrating Knowledge Management in Service Oriented development would improve requirement engineering phase as it does for traditional software engineering. The framework is developed in the light of the issues and challenges identified by published literature and the feedback of practitioners and researchers working on service oriented projects.

Keywords- Service Oriented Software Engineering (SOSE); Requirement Engineering (RE); Knowledge Management (KM).

I. INTRODUCTION

Reusability of software is a major concern for software engineers. In current market conditions meeting deadlines and producing quality software is vital. In these conditions recoding what has already been coded in a good quality is wasting your time. Component Based Software Development (CBSD) was the result of the efforts by research community in software engineering for providing methods and techniques for effective, faster and economical software development by ensuring the reuse of existing software modules. Along with the benefits these solutions posed some new challenges for developers. CBSD though proved promising for software reuse and maintainability but it still faces issues like heterogeneity of platforms and protocols, and difficulty of locating required components and selecting them against system requirements [1]. Another effort to overcome these issues is the new paradigm of Service Oriented Software Engineering [2], which is a new architectural style for building applications that support loose coupling among web services. The basic building block of software in SOSE is a web service, which is accessible via internet. Web service is a ready to use software and can be accessed via interface or API over internet by using XML standard messaging format of Web Service Definition Language (WSDL). The service provider publishes specification of service in central repository Universal Description, Discovery and Integration (UDDI), which is explored by service requester. When a service is selected requester and provider make Service Level

Agreement (SLA). For the last few years the number of services on the web has increased exponentially. Discovering appropriate web service which satisfies the requirements of the requester has become a challenge. In SOSE services are in ready to use state so the focus in this case is on identifying the services that accurately or at least appropriately fulfil system requirements. Requirements Engineering (RE) for SOSE can have different traditional development activities such as modelling, specification, and analysis but RE processes are carried out in different way [3]. The RE revolves around making a match between ready to use software components and user requirements and the result should be a compromise agreed upon by all stakeholders.

In this paper, we propose a framework for requirements engineering in SOSE. The framework is formulated on the results obtained from published literature and opinions of practitioners and researchers working on service-oriented projects.

Section II describes motivation for our research. Section III is prior related work done on SOSE and the gap analysis of it. Section IV gives details of proposed solution and Section V describes the proposed framework KM-SORE. Section VI outlines future work followed by references and appendix.

II. MOTIVATION

The additional task the requirement engineer has to perform in SOSE is to gain knowledge by exploring existing services with the aim of matchmaking between requirements and available services [4]. Service-oriented Software Engineering (SOSE) is a new field and is not fully mature yet. Though, recently a lot of interest has been shown by both industry and academia researchers [2][5][6][7][8][9], but there are still problems and challenges in this field [10][11]. Broadly, these challenges can fall into four categories that relate to main four phases of SOSE [12];

A. Specification (Planning) Issues

By specification issues we mean, the problems that are faced when we want to know the requirements for system, and during planning of acquiring these requirements and making them complete.

B. Discovery Issues

This category deals with searching for the services that actually meet the functional and non functional requirements.

C. Composition Issues

Services are selected based on their individual functionalities. Next, we need to see if they will work properly in a workflow by making composition that satisfies system requirements.

D. Management Issues

If a requirement is changed, or a new version of service is launched, or the service becomes unavailable due to any reason, re-composition and re-deployment of system is required.

The issues identified in literature against above mentioned SOSE phases are summarized in the following list, and details can be found in [12];

- *Web Service Discovery*
 - *Matching user requirements and available service*
 - *Automated and Dynamic Service Discovery*
 - *Iterative Discovery Process*
 - *Completing requirements with discovery*
 - *High Level Language/Tool Support*
- *Innovation and creativity in RE*
- *Requirement change and evolution*
- *Semantic gaps in specification*
- *Knowledge Management in service-oriented SDLC*
- *Non functional Requirements gathering and assessment*
- *Web service Dependency discovery*
- *Platform dependence in selected web services*
- *Lack of standard RE process for SOSE*

With these challenges and issues on one side and the promises proposed for systems developed in service-oriented paradigm (cost effective, reduce time effort, reusability, agility, platform independence, loose coupling etc.) there is a need of systematic investigation of the real nature of these problems and what solution can be proposed to overcome them. In Systematic Review conducted to explore SOSE challenges [13] it was found that total 8 challenges of RE were worked upon by researchers in year 2007 and 2008. The empirical work is not sufficient in this field. There is a need of new RE process [3], which should consider only the service-oriented paradigm of software development life cycle [1][14]. There has been no standard accepted so far for RE process in this domain [10][11]. This motivates further exploration and research in this emerging area.

III. RELATED PRIOR WORK

SOSE is currently under focus of research community from different perspectives. There have been many methods, techniques and tools proposed by different mega projects and research teams. They include; SeCSE [6], SODIUM [8][15], SENSORIA [7], IBM SOA [9][16], MICROSOFT [17], SOAF [18], SDLM [19], and work other researchers [20] [3]. These approaches are proposing solutions in their own ways and are not sharing any common ground [10][11].

That is not going to be helpful in providing any unified approach in future as it happened with RUP and UML [10]. To provide unification there should be more standards, models and patterns proposed for this new field [10]. Some of the proposed approaches still lack validation from industrial feedback. According to the systematic review of Qing Gu et al [13], empirical work related to requirement engineering in SOSE is not sufficient. There is a need for further empirical work in this area [10], with real life projects to provide feedback for improvements in current methods and practices and also to enrich the knowledge in SOA domain and open further research directions.

Table IV in Appendix section shows comparison and gap analysis of existing technologies with respect to the phases of SOSE, along with associated issues reported in literature. Table V in Appendix section shows comparison for gap analysis of work done to the issues identified in section "Motivation".

IV. PROPOSED SOLUTION

Knowledge Management has been proved helpful in core activities of traditional software engineering [21][22][23]. Considering SOSE as a sub field of software engineering, we have deduced that knowledge management process if integrated into all activities of SOSE, would improve the RE process and would be helpful in tackling the issues of RE in SOSE. Not equivocally, but the idea has been supported by some of the researchers in different ways. In [24], KMP is considered important to accomplish tasks of Business Process Management (BPM) in Service-oriented Architecture (SOA). In [25], the authors have highlighted the need for novel approach for sharing service knowledge and application specific information. Knowledge is required to build trust among distributed parties on heterogeneous platforms, when we do automated composition [26]. Knowledge management can improve cooperative work among services [27][28]. XML messaging data if managed can provide information regarding web service dependency by the calls one service make to the others [29].

The philosophy of service orientation is built on the idea of software reusability and agility of the process. The designers and developers must know what solutions are available in order to develop right system, with correct process. The required knowledge is about available services, previous decisions and their results, constraints of using any tool/technique/method, service interdependencies etc.

If we summarize the whole discussion, we can conclude,

- *SOSE is a shift of paradigm from SE*
- *Traditional RE cannot be applied to SOSE*
- *SOSE is facing challenges in RE*
- *KM has been proved promising in SE*
- *KM can improve RE in SOSE*

Therefore, we propose that if Knowledge Management is used in Service-oriented Software Engineering it would

help in overcoming most of the issues of Requirements Engineering. We formulate the proposed framework on the basis of issues and challenges of RE in SOSE highlighted in published literature and then conducting a survey to get the opinion of practitioners and researchers working on service-oriented projects about the issues and impact of KM on SOSE. The first part was to conduct a literature survey, and the second part was a questionnaire-based web survey.

A. STEP1: Literature Survey

The aim of performing literature survey was to extract the list of issues and challenges of requirements engineering in SOSE that has been reported in published literature. The results from this phase are published [12]. The factors identified from this phase are listed in motivation section.

B. STEP2: Survey

The purpose of conducting the survey was to validate the list of issues of RE in SOSE, extracted from published literature, from practitioners working on service-oriented system development and to get their opinion on using KM in SOSE. The population comprised of those people who have worked on service-oriented projects either as technical team member or as a researcher. The instrument for survey was questionnaire based on the identified factors from issues. The items in questionnaire used Likert scale of five levels to measure agreement level. We administered the survey on web and sent the link through email to invite the practitioners around the world. The duration for the survey was from 16th December 2010 to 23rd January 2011. A total of 117 responses were received from all around the world in this duration with almost 5.2% response rate. 20% of the responses were received from USA and 17% from India. 60% of the respondents have experience in relevant area between 4 to 9 years. 42% of the respondents are SOA architects. Out of 117 respondents 100 had experience as a practitioner and 77 had worked as a researcher in SOSE. After analysis 8 responses were rejected for providing incomplete information. And out of 109 responses, 9 were only researchers, 32 were only practitioners and 68 had experience both as a researcher and practitioner. The ranking of the factors was analyzed by grouping respondents into above mentioned three categories; only Practitioners (P), only Researchers (R) and experience of both practitioner and researcher (P+R). The ranking in Table I has been calculated for the factors on the basis of average agreement level they have achieved in their question items. The ranking provides an interesting overview of what is important for each group of respondents as they all have a different ranking for the measurement factors. The difference in opinion is mostly because SOSE is a new field, and a shift of paradigm from traditional software engineering. Most of its concepts are not fully mature yet and they are not fully understood and appreciated by designers and developers, resulting in a poor implementation of the SOSE concepts. This according to the

respondents is one of the reasons for the resulting issues besides other. Knowledge Management in Service-oriented Software Engineering has got highest agreement level in overall ranking in above table.

Measurement Factors	Agreement Percentage (Stronly agree+agree)			
	All (109)	Only P (32)	Only R (9)	P+R (68)
Knowledge Management in Service-oriented Software Engineering	76	78	60	76
Matching user requirements and available service	75	73	73	77
Iterative Discovery Process	74	60	89	80
Semantic gaps in specifications	53	63	63	54
Automated and Dynamic Service Discovery	57	60	59	56
High Level Language Support	35	37	41	32
Eliciting requirements through service discovery	72	72	89	71
Service Testing	61	75	55	35
Requirement change and evolution	48	50	33	49
Innovation and creativity in RE	67	78	66	62
Non functional Requirements gathering and assessment	59	59	67	59
Lack of standard RE process for SOS	54	41	44	62

TABLE I. RANKING OF ISSUES AND CHALLENGES FROM SURVEY

It has been reported in literature that KM (Knowledge Management) improves the software development process. In traditional software engineering, knowledge management helps in; Decreasing development time and cost, and increasing quality, Making better decisions, Understanding the domain, Communication, Acquiring knowledge about new technologies, Accessing domain knowledge, Sharing knowledge about local policies and practices, Capturing knowledge and knowing who knows what, Collaborating and sharing knowledge [21]. In core SE activities KM can support in; Document management, Competence management, Expert identification, Software reuse (making developers aware of existing software contents/components) [21]. Similarly, if knowledge management is applied in SOSE, along with above mentioned benefits, it would help to increase understanding of the engineers and would address the issues that arise due to misunderstanding of this style of solution making. Overall the survey results have indicated that KM would have a good impact on SOSE life cycle. According to comments of respondents it would help to solve following issues of SOSE; *matchmaking between requirements and services, iterative discovery process, decision making, semantic gap, eliciting requirements through service discovery, re-composition, automating the discovery process.*

Naturally, SOSE would face fewer challenges in integrating KM then tradition software development. The central repository UDDI contains all information about specification of available services. The conversion method of requirements into formal queries, search process,

retrieved results etc. could all be codified and stored. Such that, if we face a problem the previously stored knowledge can be retrieved and analyzed. Integrating creativity and innovation in requirements engineering for SOSE would help in making new ways for solutions and maintaining a KM along with the process would help in not repeating the mistakes. KM would definitely take cost and efforts for its implementation as apprehended by respondents, but it is like an investment for improvement in the process where the benefits become visible with time. The respondents had a concern that whether there would be a need of some specific to field KMP for SOSE. Current KMPs for SE were not proposed considering service-oriented paradigm. How a KM for traditional SE can be adapted for SOSE, is yet an area of exploration. According to the systematic literature review conducted by Bjørnson and Dingsøyr on knowledge management in software engineering [22], the main emphasize of SE has been so far on technocratic category of KM in Earl's taxonomy for Knowledge management strategies [30]. The KM process for SOSE will mainly cover technocratic and organizational sub category from behavioral of Earl's Taxonomy for KM.

V. FRAMEWORK KM-SORE

The framework has integrated KM with SOSE life cycle in the light of findings from literature and survey. After analyzing the issues of requirements engineering in SOSE and results from the survey we found that the issues are somewhat interlinked because the phases of SOSE work in iterations. RE in SOSE is not a discrete activity but is related to the Discovery and Composition phase as well. There for any problem cannot be described as belonging to one phase. The issues of RE are also overlapping into other phases as well. All the phases of SOSE depend on each other for their functioning. Table II shows their overlapping in different phases of SOSE [12].

The framework proposes to integrate KM in all the phases of SOSE. These phases are interrelated and work in iteration for successful composition. J. Ward and A. Aurum [31] have given a refined list of KM process activities from literature. They have given list of seven activities. On highest level of abstraction three activities are required for KMP; Knowledge Creation (KC), Knowledge Storage (KS), Knowledge Retrieval (KR). Integration of KM in SOSE will require Knowledge creation, storage and retrieval in all four main phases. Table III summarize the overall idea of this integration.

In a common storage space, the codified knowledge from all phases will be stored to make it available for all phases, ultimately storing the information in organization's central knowledge base. Figure 1 shows this integration graphically. The framework provides an overall view of how KM would be integrated in SOSE phases. Exactly what strategy and tools have to be selected for knowledge management, is to be decided by the organization. The literature has guidance available on how

to select an appropriate strategy [32] [33] [34] and software tool [35] [36].

SOSE Phase	Related Issues
Planning	Matching user requirements and available service Following iterative Discovery Process Completing requirements with iterative discovery Semantic gaps in specification Non functional Requirements gathering and assessment
Discovery	Matching user requirements and available service Following automated and dynamic Service Discovery Following iterative Discovery Process Completing requirements with iterative discovery Semantic gaps in specification Non functional Requirements gathering and assessment
Composition	Semantic gaps in specification Web service Dependency discovery
Management	Dealing with requirement change and evolution

TABLE II. ISSUES AND THEIR RELATION TO PHASES OF SOSE

SOSE PHASE	KMP ACTIVITY		Related issues
Planning	KC	Initial composition design	Matching user requirements and available service Ease of iterative Discovery Process Completing requirements with iterative discovery Semantic gaps in specification Non functional Requirements gathering and assessment
	KS	Codifying created knowledge	
	KR	User requirements, results of discovery	
Discovery	KC	Queries and results of discovery	Matching user requirements and available service Ease of automated and dynamic Service Discovery Ease of iterative Discovery Process Completing requirements with iterative discovery Semantic gaps in specification Non functional Requirements gathering and assessment
	KS	Codifying created knowledge	
	KR	Initial composition design, user requirements	
Composition	KC	Workflow for composition	Semantic gaps in specification Web service Dependency discovery
	KS	Codifying created knowledge	
	KR	Results of discovery, user requirements	
Management	KC	Changes in workflow of composition and their reasons	Ease of requirement change and evolution
	KS	Codifying created knowledge	
	KR	User requirements, workflow of previous composition, results from previous discovery	

TABLE III. KNOWLEDGE MANAGEMENT IN SOSE

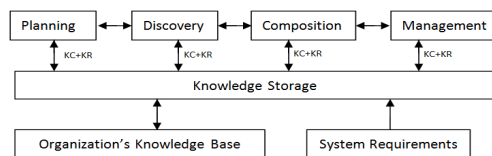


Figure 1. KM Activities integrated with SOSE phases

VI. CONCLUSION AND FUTURE WORKS

According to the results of our survey, KM has been acknowledged by practitioners to improve the issues and challenges of RE in SOSE. Currently, the framework has its foundations on the basis of results obtained from literature and survey. Our next task is to apply the framework in experimental setup to see the results it would produce. We will be conducting an experiment on two projects of SOSE one with proposed framework and one without it. The results from both projects will be compared. It will be an observational field experiment, where we will be evaluating the checklist of issues of RE in SOSE in both projects to see any difference in the results. Improvement will be assessed based on the checklist of issues of RE in SOSE, mainly based on time and ease of performing the task along with the phases of SOSE. The data collection units during observations will be: Accuracy rate of discovery results, Time of decision for selection of query, Composition success/failure rate, Time for accommodating service change request.

REFERENCES

- [1] H. P. Breivold and M. Larsson, "Component-Based and Service-Oriented Software Engineering: Key Concepts and Principles," in *33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, 2007, pp. 13-20.
- [2] W. T. Tsai, "Service-oriented system engineering: a new paradigm," in *IEEE international workshop on service-oriented system engineering (SOSE)*, Beijing, 2005, pp. 3-8.
- [3] W. T. Tsai, Z. Jin, P. Wang, and B. Wu, "Requirement engineering in service-oriented system engineering," in *proceedings of International Workshop on Service-Oriented System Engineering*, pp. 661-668.
- [4] G. Spanoudakis, A. Zisman, and A. Kozlenkov, "A service discovery framework for service centric systems," in *Proceedings of the IEEE International Conference on Services Computing (SCC'05)*, 2005, pp. 251-259.
- [5] A. Arsanjani, "Service-oriented modeling and architecture," *IBM developer works*, 2004.
- [6] S. Crew, *Service Centric System Engineering—EU/IST Integrated Project*. 2004.
- [7] M. Wirsing and M. Hözl, *Rigorous Software Engineering for Service-Oriented Systems—Results of the Sensoria project on Software Engineering for Service-Oriented Computing*. Springer, 2010.
- [8] S. Topouzidou, "SODIUM, Service-Oriented Development In a Unified framework," *Final report ISTFP6-004559*. <http://www.atc.gr/sodium>.
- [9] A. Arsanjani, L. J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, "S3: A service-oriented reference architecture," *IT professional*, vol. 9, no. 3, pp. 10-17, 2007.
- [10] A. Kontogogos and P. Avgeriou, "An Overview of Software Engineering Approaches to Service Oriented Architectures in Various Fields," in *18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, 2009. WETICE '09. 2009, pp. 254-259.
- [11] E. Ramollari, D. Dranidis, and A. J. H. Simons, "A survey of service oriented development methodologies," in *The 2nd European Young Researchers Workshop on Service Oriented Computing*, 2007.
- [12] M. Bano and N. Ikram, "Issues and Challenges of Requirement Engineering in Service Oriented Software Development," in *Fifth International Conference on Software Engineering Advances (ICSEA)*, 2010, 2010, pp. 64-69.
- [13] Q. Gu and P. Lago, "Exploring service-oriented system engineering challenges: a systematic literature review," *Service Oriented Computing and Applications*, vol. 3, no. 3, pp. 171-188, 2009.
- [14] A. Brown, S. Johnston, and K. Kelly, "Using service-oriented architecture and component-based development to build web service applications," *interactions*, vol. 1, pp. 2-4.
- [15] S. Topouzidou, *SODIUM, Service-Oriented Development In a Unified framework, Final report ISTFP 6-004559*. .
- [16] A. Arsanjani, "How to identify, specify, and realize services for your SOA," *IBM Corporation, February*, 2005.
- [17] A. Sehmi and B. Schwegler, "Service-oriented modeling for connected systems-part 1," *The Architecture Journal*, vol. 7, pp. 33-41, 2006.
- [18] A. Erradi, S. Anand, and N. Kulkarni, "SOAF: An architectural framework for service definition and realization," in *SCC'06. IEEE International Conference on Services Computing*, 2006, pp. 151-158.
- [19] M. P. Papazoglou and W. J. Van Den Heuvel, "Service-oriented design and development methodology," *International Journal of Web Engineering and Technology*, vol. 2, no. 4, pp. 412-442, 2006.
- [20] W. T. Tsai, "Service-oriented system engineering: a new paradigm," in *IEEE international workshop on service-oriented system engineering (SOSE)*, Beijing, 2005, pp. 3-8.
- [21] I. Rus and M. Lindvall, "Knowledge management in software engineering," *IEEE software*, pp. 26-38, 2002.
- [22] F. O. Bjørnson and T. Dingsøyr, "Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used," *Information and Software Technology*, vol. 50, no. 11, pp. 1055-1068, 2008.
- [23] A. Aurum, F. Daneshgar, and J. Ward, "Investigating Knowledge Management practices in software development organisations—An Australian experience," *Information and Software Technology*, vol. 50, no. 6, pp. 511-533, 2008.
- [24] Jianmei Guo, Yinglin Wang, Xijuan Liu, and Hongxia Tong, "A Service-oriented Integration Model of Knowledge and Business Processes," in *IEEE International Conference on Networking, Sensing and Control. ICNSC*, 2008, pp. 1674-1679.
- [25] D. Huang, Y. Yang, and J. Calmet, "A Knowledge-based Security Policy Framework for Business Process Management," in *Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce*, 2006, pp. 154-160.
- [26] G. Koumoutsos and K. Thramboulidis, "A knowledge-based framework for complex, proactive and service-oriented e-negotiation systems," *Electronic Commerce Research*, vol. 9, no. 4, pp. 317-349, 2009.
- [27] H. Kojima, K. Funaki, and T. Inoue, "Web Service Systems for Cooperative Work Support in Knowledge Creation Processes," *Human Interface and the Management of Information. Designing Information Environments*, pp. 94-103.
- [28] Jiang Jichen and Gao Ming, "A process-driven content-oriented integration framework for knowledge management systems," in *SOLI '09. IEEE/INFORMS International Conference on Service Operations, Logistics and Informatics*, 2009, pp. 213-218.
- [29] S. Basu, F. Casati, and F. Daniel, "Web service dependency discovery tool for SOA management," in *IEEE International Conference on Services Computing, 2007. SCC 2007*, 2007, pp. 684-685.
- [30] M. Earl, "Knowledge management strategies: toward a taxonomy," *Journal of Management Information Systems*, vol. 18, no. 1, pp. 215-233, 2001.
- [31] J. Ward and A. Aurum, "Knowledge management in software engineering—describing the process," in *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, 2004, pp. 137-146.
- [32] K. Haggie and J. Kingston, "Choosing your knowledge management strategy," *Journal of Knowledge Management Practice*, vol. 4, pp. 1-23, 2003.

[33] J. Robertson, "Developing a knowledge management strategy," *Knowledge Management Column*, 2004.

[34] B. Choi and H. Lee, "Knowledge management strategy and its link to knowledge creation process," *Expert Systems with applications*, vol. 23, no. 3, pp. 173-187, 2002.

[35] M. Lindvall, I. Rus, and S. S. Sinha, "Software systems support for knowledge management," *Journal of Knowledge Management*, vol. 7, no. 5, pp. 137-150, 2003.

[36] M. Lindvall, I. Rus, and S. S. Sinha, "Technology support for knowledge management," *Advances in Learning Software Organizations*, pp. 94-103, 2003.

APPENDIX

Related Prior Work	Specification	Discovery	Composition	Management	Issues [10][11]
SeCSE (Analysis and Design)	Tools and methods for testing service specification and quality Early Service Discovery (ESD) to complete requirements	Supporting framework for runtime service discovery, and search engine	QoS aware service composition in order to solve re planning problem Architecture-based Service Discovery (ASD)	Self healing service composition, requirement monitoring Run Time Service Discovery (RTSD)	Lack of Industrial case studies Unable to handle heterogeneous service composition Covers analysis and design of SDLC only Only workflow technique is used (not supporting semantic web techniques)
SODIUM (Complete Composition Life Cycle)		XML based Query language (USQL) search engine for USQL, Languages for Unified Discovery and Composition	Heterogeneous service composition USCL (Unified Service Composition Language) A Methodology for Service Composition VSCL (Visual Service Composition Language)		The project is focused mainly on providing methods/tools for composition of heterogeneous web services. There focus is different and is not from requirement engineer's perspective.
SENSORIA (Complete SDLC)	UML Profile for Service-Oriented Systems(UML4SOA) Ontology for SOAs, SENSORIA Reference Modelling Language (SRML) Prototype Language for Business Policies		Mathematical models for simulation and verification of service composition		Not evaluated by researchers yet.
IBM		Service identification method, Service classification and categorization method	Service realization method		Covers analysis and design of SDLC only SOMA lacks openly available detailed description of the methodology, which makes it difficult to further analyze its capabilities.
SOAF	Information elicitation	Service identification and service definition	Service realization	roadmap and planning (management)	It has given the framework for whole life cycle about what to do but details on guidelines of how to do are missing. Lack of tool support.
Papazoglou et al Framework SDLM	Planning and Analysis (Using methods of BPM, RUP, CBD)		Service construction, testing, provisioning, deployment	Monitoring	The proposed approach does not provide enough guidelines for explicit consideration of service model artifacts.

TABLE IV. COMPARISON AND GAP ANALYSIS OF EXISTING TECHNIQUES WITH PHASES SOSE AND ASSOCIATED ISSUES

Issues from Literature	SeCSE	SENSORIA	SODIUM	IBM	SOAF	SDLM
Matching user requirements and available service	Yes	No	No	Yes	Yes	Yes
Automated and Dynamic Service Discovery	Yes	No	No	No	No	No
Iterative Discovery Process	Yes	No	No	No	No	No
Completing requirements with discovery	Yes	No	No	No	No	No
High Level Language Support	Yes	Yes	Yes	No	No	Yes
Innovation and creativity in RE	Yes	No	No	No	No	No
Requirement change and evolution	Yes	Yes	No	Yes	Yes	Yes
Semantic gaps in specification	No	Yes	No	No	Yes	Yes
Knowledge Management in service-oriented SDLC	No	No	No	No	No	No
Non functional Requirements gathering and assessment	Yes	No	No	Yes	No	No
Web service Dependency discovery	No	No	No	No	No	No

TABLE V. COMPARISON AND GAP ANALYSIS OF EXISTING TECHNIQUES TO THE ISSUES OF RE IN SOSE REPORTED IN LITERATURE

Brainstorming as a Route to Improving Software Processes

Celestina Bianco

Systelab Technologies

Barcelona, Spain

email: celestina.bianco@systelabsw.com

Abstract-This contribution shows how a “guided brainstorming” process facilitates the applicability of an assessment and process improvement model (such as for example SPiCE or CMMi) to Companies that develop Critical Software applications. For a Manufacturer that deals with time-to-market and customer satisfaction, as well as with regulations and laws, both efficiency and efficacy are mandatory for regulatory and commercial purposes, and become strictly related. A Process Improvement is often necessary for a company that develops and distributes critical software in order to establish a Quality System, which is a set of established procedures with measurable and auditable output. To be effective these procedures must be or become a natural part of the daily activities. To provide an efficient way to approach and implement the improvement plan, we suggest guided brainstorm sessions, which are a good interactive opportunity for team and formalization building. Although the concept of brainstorming is not novel in organizations, it is generally a first approach to discussions that are later formalized. The use of this technique for definition of critical requirements or regulated activities is not common. For the definition of Quality Systems, a top-down approach (definition, training, roll-out, audit) is generally used whereas brainstorming can be considered bottom-up approach, from experience to formalization.

Keywords-improvement; participation; experience; motivation.

I. INTRODUCTION

This contribution shows how a “guided brainstorming” process facilitates the applicability of an assessment and process improvement model (such as SPiCE or CMMi) to Companies that develop Critical Software applications. It comes after a study for the definition of a simplified Spice model for Small Companies and a study about the application of Spice to Medical Devices.

Critical Software manufacturing companies are generally of limited dimension, or small / medium departments inside large Companies. We define as critical, the software that controls apparatus or activities that may imply direct benefits or risks for life, security, and the environment. In applications such as Medical Devices (our specific sector), the software has generally to provide and handle the overall protection to guarantee safety and liability of the system.

The Manufacturer deals with time-to-market and customer satisfaction, as well as with regulations and laws. The risks derived from the product characteristics and from the processes applied to design control have to be

considered, in addition to typical project and commercial risks.

In this context, both efficiency and efficacy are mandatory for regulatory and commercial purposes, and become strictly related.

A Process Improvement plan, tailored and tuned to the needs of the Company, is the main route to reach the trade-off of the efficacy and efficiency required.

What is an efficient and effective way to approach and implement the improvement plan for a Small Company? We suggest that guided brainstorm sessions are a good interactive opportunity for team and formalization building.

The paper has the following structure. Section II presents maturity levels of processes following standard models, and develops brainstorming methods. Section III exposes potential evolution of topics under survey.

II. PROCESSES FOLLOWING STANDARDS MODELS AND USE OF BRAINSTORMING METHODS

From SPICE Level 2 to Level 3, from an informal process to an established process, the advantages of using a standard reference model for Process Improvement are multiple:

First of all, the model is ready and has been proved in a number of other companies. This may not be feasible as it is, as the Improvement Plan may need to be implemented with limited resources, often in parallel with the daily tasks assigned. This implies that small companies need a specific approach to make ROI of Process Improvement interesting for the business.

Process Improvement requests for a company that develops and distributes critical software [2][3][4] have a main outcome, namely, the definition and control of a Quality System [5][6], which is a set of established procedures with measurable and auditable output. Translated in SPICE concepts [1], it means that Process Improvement has capability level 3 as a target.

SPICE capability level 2 “certifies” that a controlled common process is followed within a Company, with activities that are carried out in a common way by all team and project members, with similar outcomes. When a Company reaches SPICE capability level 3, those processes that are managed and controlled have been formalized and established within the Company.

To move from level two to the following levels, the steps to an effective and efficient improvement for a Small Company can be summarised as:

- Select the set of basic processes that are needed for Design Control and ancillary mandatory activities
- Define the processes
- Define the support for processes (instructions, templates, checklists, etc.)

A. Define your Quality System

The set of processes to be established is the minimum set that covers the Project Life Cycle phases, including control of design, validation, configuration, risk management, and all others specific to the sector, required to comply with customer requirements and with regulations.

In a Company that is active in a critical sector, most of those processes are implicitly defined. If a common approach is applied through the projects the Company is generally at a capability level 2. It has to move one step higher by establishing the processes mentioned above.

1) Guided Brainstorming

To acquire process formalization and description for roll-out, training and control, we suggest using guided brainstorming sessions.

Guided brainstorming goes beyond the purpose of allowing “everyone to say his own opinion freely about a subject”. Applied as a working tool, it is more structured:

It is a guided meeting attended by people involved with the subject, often experts themselves.

Each attendee replies with 3 to 5 answers or suggestions to each of the questions specified in the agenda of the meeting. Such as for example: “list which are, in your opinion, the 3 main characteristics of xxx”.

Attendees get 10 to 15 minutes to prepare and write the answer separately.

The answers are then presented and explained.

They are rated and ranked for popularity (i.e. given a higher rating if more attendees gave same or similar answers then ranked by the resulting score).

The first arguments in the list, the highest rated, are then refined and considered in detail. The result of the discussion is recorded.

For example, if the question was “Which are the 3 main activities of process X?”, the three most rated proposed actions are detailed in terms of responsibility, flow, inputs, outputs, etc.

A guided brainstorm is called with the purpose of describing and building a process or sub-process or supporting material for the required phases, with the contribution of all the attendees. The moderator of the meeting has prepared a minimum set of items that the list of answers HAS to contain, and will guide the participants to include them. The approach is based on the belief that the process is implemented and implicitly defined in the majority of the parts, and the people involved are the best candidates to describe, analyze and formalize it. The moderator will help to fill in the gaps.

A step forward in a small company, equally relevant after the identification and definition of processes, is to define

templates and practical instructions which will easily and cost-effectively carry out the tasks. Templates and instructions can also be the result of brainstorm sessions, generally with a smaller number of participants.

2) Advantages

What are the advantages of the technique?

In the belief that in the Company processes are implemented, in order to reach a higher capability level and compliance with regulation, they must be formalized in procedures, and so, becoming repeatable and controllable.

Brainstorm meetings are used to gather the practices, share and formalize with the advantage that:

- All are protagonists and motivated
- The base is a company culture already in place
- Sharing is not imposing
- Changes and up-grading of the practices, if needed, are easily accepted
- Roll-out of the process will not find resistance as it is accepted “by definition” by people that contributed to build it
- It is also an economical saving, as you don’t have to call in experts to define processes and train on them

3) Drawbacks

This technique is not a panacea; it presents some drawbacks. Warnings to be considered are:

- As it is based on the use of existing knowledge and experience, it can be effective only if the activities to formalize are already part of the culture and daily experience.
- Instead, if one wants to apply the technique to introduce new concepts, some extra technique and expertise is required. The outcomes of the brainstorm have to be reviewed and revisited after a pilot experience, often happening for processes such as risk management and metric collection. In these cases brainstorming becomes a good training technique.
- The strength of this approach is its usage and sharing of experience. It is also a weakness as the quality system generated is recognized by the authors but does not have the “authority” of a recognized standard for their colleagues (no man is a prophet in his own land!). Mapping to the standard used as a reference to guide the brainstorm will be useful.
- There is a time cost involved in the preparation of the meetings and formalization of the results, which is partially avoided by the use external experts. On the other side the ROI is high in terms of training and roll-out.
- The outcome is not accepted without time limit and has to be periodically updated even if it may still be valid because:
- People change and so does the approach
- People need to be protagonists and new people need to be part of a revision to accept their own

procedures as did the team defining the original system.

Based on the analysis of the advantages and draw backs, and on the experience within our Company and companies with which we have worked using the method, we believe that the approach is appropriate and of high ROI in companies with real internal practical know how on the activities to be organized and formalized in processes. Most of the draw-backs have simple mitigations and the others are in fact the correct stimulus for continuous improvement and action plans.

Once the Company reaches the level of compliance with regulation, continuous process improvement will focus on improving efficiency. Regulatory compliance is required to stay in the business.

The above brings us to the conclusion that through the initial scope of the technique, small companies, and with expansion larger ones, gain the advantages of being people and company-culture centred.

If the company is entering a new sector or business and the participants in brainstorming sessions do not have specific previous experience to share and formalize, the technique can still give good results. Advantages such as motivation, participation and buy-in are still relevant. In this case it will be necessary to acquire the help of a moderator who is expert in the sector of the application, able to guide and focus the questions and answers for the purpose of getting compliance with the requirements.

B. Examples

1) Example 1 – Brainstorming to define SW Design Process

a) Rules:

- Each participant receives two questions then is requested to give 1 to 5 replies to each question
- There will be 15 min. for individual answering
- The answers will be collected in a round table session. Rating of an answer is equal to the number of people giving the same one
- The 5 most rated answers will be discussed to generate a draft procedure
- *The total time of the meeting will be two hours*
- *The moderator will distribute minutes*

b) Questions:

- list 5 major steps in a controlled Software Development
- Process list 5 major activities to support and keep a project under control

TABLE I. ANSWER QUESTION 1

What	Rating
Requirements	IIII
Design	IIII
Project Planning	III

Risk Analysis	III
Integration and Test	III
Test Planning	II
Test Protocol definition	I
Release control	I

TABLE II. ANSWER QUESTION 2

What	Rating
Risk Management	IIII
Release Planning	III
Defect Management	III
Change Management	III
Configuration Control	II

A draft SW Design Procedure was defined based on the first table, defining tasks for each major step, responsibility and templates, based on current practices and suggestions for improvement. The second table defines the supporting activities and required metrics. Later on, parts of the SW design flow were detailed in dedicated procedures (for example requirement management, testing etc.)

2) Example 2 – Brainstorming to define SW Testing Process

a) Rules : same rules

b) Questions:

- list 5 major SW Testing Tasks
- list 5 mandatory sections of a SW Test Plan
- list 5 mandatory sections of a SW Test Case

TABLE III. ANSWER QUESTION 1

What	Rating
Test Run	IIII
Design Tests	II
Planning Test	II
Test Report	II
Defect Management	II
Coverage Metrics	--

TABLE IV. ANSWER QUESTION 2

What	Rating
Features to be tested	III
Responsibility	II
Strategy	II
Test Environment	II
Version description	--
Pass / Fail criteria	--
Defect Metrics	--

As the participants DO NOT have experience in organized formal testing, the third questions were even harder to reply to. The moderator enters the meeting with a pre-prepared list of answers expected and leads the group to understand which of the answers are **out of scope**, and which **are missing**. The tables of answers are completed and

corrected and on this basis the brainstorm continues as shown in the previous example.

C. Lessons learnt

The method has been applied initially in our Company, when the Software department contained less than 20 people, and the testing team was starting with a group of 5, including myself.

The Company had defined a Quality System based on international standards that applied to another division, working for Space missions. The Quality System was not tailored to the new department, and too complicated for a small team. On the other side, the people in the new department had good experience of Software engineering. Use of the brainstorming technique allowed us to define in a few months the nucleus of the Quality System that later became the base for the current one. The Quality System developed on this nucleus applies to a Company currently able to sell in international markets and to comply with different regulations and standards.

The resistance in following procedures common to many Software Engineers has been overcome by the fact that everyone was in some way the author of the procedure, and because they were aware of the requirements beneath each process.

The method was then applied in other companies which either wanted to gain more efficiency or comply with regulation. The two most successful examples involved the definition of the basic software design processes in a Company developing economically critical software, and the creation of the team and processes for Software verification and validation in a Company developing Medical Devices. In both Companies the Software team was unstructured and parts of the projects were assigned to consultants.

Of major relevance for the first company was the formalization of the flow, in order to describe it to the external partners and monitor the progress and quality.

For the second it was mandatory to define the process of testing to be able to check internal and external parts and to demonstrate to Regulatory Bodies and customers the quality of the system and the compliance with regulations.

Both companies were aware that an efficient process had to be simple. They matched their goal defining a Quality System based on their experience, with the help and review of an expert.

III. EVOLUTION OF THE METHOD FOR EVOLUTION OF THE QUALITY SYSTEM

In our Company the roll-out of the first Quality System defined was relatively simple with low resistance, as described above but in five years difficulties appeared.

The processes defined initially had been progressively updated to adapt them to the new standards, practices, projects. At that point the main problem was not the adequacy of the process, it was that the resistance to apply it had increased.

The people who had defined the initial nucleus are no longer employed in the company, or have been assigned to other roles.

Two years ago we decided to revisit all the defined processes and add new ones. We used approximately the same technique in a slightly different format, adapted to the current dimension of the company and to the variety of services. Brainstorming was used in larger groups to gather needs and experience and details then discussed in small working groups.

We defined a set of procedures and activities similar to the original ones, produced by the new owners of the processes.

REFERENCES

- [1] ISO/IEC FDIS 15504-2 Software engineering — Process assessment — Part 2: Performing an Assessment, Part 5: An exemplar Process Assessment Model
- [2] FDA 21 CFR 820 - Quality System Regulation US Food and Drug Administration, 1997
- [3] FDA General principles of software validation. US Food and Drug Administration, 2002
- [4] Medical Devices - Directive 2007/47/EC on Medical Devices, European Parliament and Council, 2007
- [5] ISO 9000:2008 Quality management systems - Requirements
- [6] ISO 13485:2003 Medical devices - Quality management systems - Requirements for regulatory purposes

Web-Based Focus Groups for Requirements Elicitation

Carla Farinha
IST / Opensoft, S.A.
Lisboa, Portugal
carla.farinha@opensoft.pt

Miguel Mira da Silva
IST / INOV
Lisboa, Portugal
mms@ist.utl.pt

Abstract—Requirements determine how an Information System should operate. Requirements Engineering errors become failure reasons of the Information System. In this paper we propose a Web-based focus group method to overcome many problems of the requirements elicitation activity. Collaboration is promoted in discussions involving all stakeholders to achieve consensual decisions. The proposal was evaluated with two real-world experiments the results of which reveal the potential of the method. The successful implementation of the proposed method can avoid many limitations of the requirements elicitation traditional methods, such as misunderstandings between stakeholders and analysts.

Keywords: *Requirements Elicitation; Focus Groups; Collaboration Tools*

I. INTRODUCTION

Requirements are the heart of Information Systems Development since the earliest days of computing [1] because they determine how the system will operate [2], [3], [4]. Nevertheless, errors on the elicitation activity still represent major causes for the failure of these systems [5].

Requirements elicitation is influenced by many factors, including contextual, human, economic, and educational factors. It is also constrained by issues such as the specific process and project, the difficulties in communication and understanding between stakeholders and analysts, the quality of identified requirements, stakeholders' conflicts, and the experience and practice of the analyst [4]. Moreover, stakeholders should identify real needs but they may not know what they need and analysts may not understand business concepts [6], [7], [8]. Errors made at this activity cost around 80-100 times more if discovered at the implementation stage [1] and are very hard to fix [9].

The social nature of Requirements Elicitation has been leading to the usage of social sciences approaches [4], including ethnography [10], [11] interviews [3], [12] or group work [13-17].

We propose a web-based Focus Groups method to overcome major problems of Requirements Elicitation [15]. Requirements are discussed between all stakeholders that want to contribute with ideas. The goal is to find a global overview of real needs and to negotiate incoming conflicts. Finally, identified needs are resumed in a report according to their relevance. Our proposal was evaluated with Action Research with a real problematic situation, developing skills of organizational members to overcome that situation, and adding scientific knowledge [18, 19].

We present recent trends of requirements elicitation methods and relate collaboration tools in Section 2. Section 3 describes our proposal and research, including our experiments. Finally, Section 4 discusses concluding remarks and future research work.

II. REQUIREMENTS ELICITATION

Requirements Elicitation is a critical activity of the Requirements Engineering process [4, 20] for many reasons. First, the activity relies on a complex and error-prone communication between stakeholders and analysts. Second, stakeholders are not always clear about what they want or need. Finally, analysts may not understand the business concepts [1, 7, 8].

The communication nature of the Requirements Elicitation activity and its social context is incontestable [6, 8, 12]. As such, methods for this activity are deriving from social sciences' methods [4].

A. *Methods from Social Sciences*

Zowghi and Coulin [4] surveyed aspects of techniques, approaches and tools for requirements elicitation and aggregated them in 8 groups that cover the whole spectrum. The groups that actively involve stakeholders are ethnography, interview and group work, considered as alternative to each other.

Ethnography is the observation of people in their natural environment [10, 11]. Crabtree, Nichols, O'Brien, Rouncefield and Twidale [11] studied this method and reveal limitations, including risk of incorrect interpretation [10], impossibility of identifying new requirements [20] or difficulty of generalizing results. Sommerville [21] says that ethnography is incomplete, being useful as a complement.

Interviewing is an informal interaction where analysts gather requirements asking questions about the system in use and the system to be [4, 9, 21]. Davey and Cope [3] studied interviews as best practice for requirements elicitation but they admit that more research is needed about the nature of conversations in the field to bring successful results. Goguen and Linde [12] evaluated techniques for eliciting requirements, including interviews, and concluded that this method is limited by the stimulus-response interaction and by the need of participants to share basic concepts and methods. Sommerville [21] states that interviewing is unsuitable to identify organizational requirements and constraints, but could be used as complementary method.

Group work, such as brainstorming, joint application development, creativity workshops or focus groups, gathers stakeholders to collaborate reaching solutions about an identified problematic situation. Typical limitations of group work are dominant participants, biased opinions, high logistic costs and gathering stakeholders [4, 22].

Concluding, ethnography is incomplete on covering the fundamental types of activities of requirements elicitation and has relevant limitations. As for group work and interviews, the group work have many advantages since they can obtain a more complete overview of the system, richer information and resolution of conflicts through stakeholders' discussions rather than with individual interviews [4].

B. Group Work Methods

There are many group work methods, including brainstorming and workshops, which includes JAD, creativity workshops and focus groups.

Brainstorming joins stakeholders in informal discussions to rapidly generate ideas without focusing on any one. It is used to develop the preliminary mission statement for the project but not to explore requirements [4].

Joint Application Development (JAD) discussions focus needs of business and users rather than technical issues to make decisions. JAD differs from brainstorms since main goals of the system are already established before the discussion [4]. Davidson [14] studied 3 organizations using JAD and concluded that, although there are improvements in systems development, JAD is difficult to sustain in practice. Coughlan [2] also presented studies of JAD in practice, demonstrating that JAD forces a rigid user-designer interaction, is weak at acquiring knowledge and complicated to use in practice.

Creativity workshops encourage creative thinking to discover and invent system requirements [23]. Maiden and Robertson [23] used creativity workshops to discover stakeholder and system requirements, concluding that the overall process was successful but not all of the workshop sessions.

Focus Groups are discussion groups facilitated by a specialist that follows a guide to orientate the discussion around key questions [24, 25]. The preparation of the method requires defining groups (size and composition) and procedures (number of sessions and moderator guide). This method differs from other group-based methods because of the group special characteristics: homogeneous and focused on key topics to collect inductive and natural information [24]. Engelbrektsson, Yesil and Karlsson [17] studied methodological considerations with focus groups, concluding that an efficient choice of participants and mediating tools are also important to enhance the requirements elicitation activity. Farinha and Mira da Silva [16] applied Focus Groups in real-world experiments to better elicit requirements of information systems, concluding that stakeholders discuss different perspectives about the system as a whole and collaborate to formalize the requirements according to their needs but dominant users and analysis costs were serious limitations. Kasirun and Salim [26] evaluated a requirement elicitation tool based on

a forum that employed Focus Groups, demonstrating that a web-based tool supports shared involvement and eases requirements elicitation but further research is needed to prove results.

Many researchers have been studying requirements elicitation problem using social sciences' methods, particularly group work. However, the problem still exists and much more empirical work is needed [2, 3].

C. Collaboration Tools to Elicit Requirements

The intense communication of requirements elicitation [7, 22], demands a high level of collaboration [27]. However, it is difficult to gather stakeholders at the same time and place [1, 9, 28]. This is why collaboration tools have been applied to requirements elicitation [4, 27].

Collaboration tools allow the cooperation of all stakeholders in several phases of the software process, including in requirements engineering. Choosing one of the wide range of collaboration tools demands defining types of tasks to accomplish, making an inventory of the existing software and hardware infrastructures and knowing the experience and capabilities of the team [27].

Herbsleb [29] studied a desired global development considering coordination as a key. He considered challenges in several areas, including eliciting and communicating requirements, concluding that is needed a systematic understanding of what drives the need to coordinate and effective mechanisms for bringing.

Whitehead [27] studied goals of collaboration in software engineering and existing collaboration tools, particularly web-based tools. He realized that there is no integrated web-based environment to cover the entire development lifecycle. He also concluded that is important to understand the collaborative nature of software engineering combined with low costs of high capabilities of communication platforms to improve collaboration in the creation of software artifacts.

III. PROPOSAL

In this paper we propose to address requirements elicitation problems, including difficulties on gathering stakeholders, misunderstandings between stakeholders and analysts, quality of identified requirements and stakeholders' conflicts. In order to do so, our key concerns were allowing an asynchronous and distributed communication; avoiding interpretation of results by analysts; inviting all stakeholders to contribute with their ideas; and obtaining agreement of all stakeholders about the identified needs.

To accomplish these challenges, we propose an effective requirements elicitation method, based on a collaboration tool that integrates the Focus Group method. Before using the method, boundaries of the system and key discussion topics must be defined by project managers. Finally, a report with the results must be delivered to project managers.

Although many proposals of collaboration tools with focus groups exist, we introduce distinctive features. One of the most important features is opening the discussion to all stakeholders, allowing the input of all necessary parts of the problem. We also added a voting system, which is not usually applied to focus groups. We compared a focus group

with anonymous contributions versus a focus group with identified contributions. Finally, we evaluated our proposal in a real environment.

In order to be successful, we assume that all stakeholders are interested in the Information System and want to contribute with ideas. We also assume that questions to discuss are defined by client project managers that know the limits of their desired solution. Finally, we believe that stakeholders actually take advantage of the asynchronous communication, which allows thinking on the problem, understanding other perspectives and generate new ideas to contribute again.

Our requirement elicitation method demands defining limits and goals with project managers conduct the focus group and resume results. We propose the following steps:

- 1) *Define scope and questions*
- 2) *Prepare the focus group in a collaboration tool configuring desired properties (anonymity, navigation, etc.)*
- 3) *Conduct the focus group*
- 4) *Report results*

A. Define scope and questions

This step intends to define limits of the desired solution and key questions to address in the focus group. Analysts have to meet with project managers to define these limits and the questions they find important to discuss.

B. Prepare the focus group

The focus group must be configured in a collaboration tool according to desired and adequate features. For example, shall the participants be identified or anonymous? Is there a navigation rule to answer the questions? How shall participants give their contributions?

In this experiment, we compared two approaches:

- 1) *Comment-oriented forum*, based on comments.
- 2) *Vote-oriented forum*, based on comments and votes.

The comment-oriented forum was configured in a self-hosted blogging tool. Each page had a question where stakeholders could comment. Following the orientations of a regular focus group moderator guide, these pages had a sequential order. Participants were advised to navigate and comment according to that order in the first time but they could freely navigate and comment any page. Opening and ending questions were excluded because they wouldn't make sense in an online method. Anonymity was integrated in this forum so that stakeholders could freely express opinions.

The vote-oriented forum was configured in a question & answer tool. Only key questions from a regular focus group were initially introduced to reduce the extensiveness of the time spent to answer. Participants could comment questions and others' ideas vote on posted ideas and introduce new questions. No navigation recommendations were provided. Anonymity wasn't integrated in this approach.

C. Conduct the focus group

The administration of the forums had to moderate the discussions. This moderation required to follow comments

and, if needed, delete comments, encourage discussions or probe for more information.

When conflicts of interests rise and it is impossible to satisfy different identified needs, participants shall be advised to resolve the disagreement. Whether they all agree with a solution becomes irrelevant after all because the most voted need is chosen.

D. Report Results

In order to avoid interpretation of results, analysts shall list identified needs and organize in a descending order. This order must be according to the number of positive comments asking for the need or to the positive votes.

IV. EVALUATION

Our proposal was evaluated on an enterprise of technological solutions, mainly e-government solutions. There are 60 employees that are from 23 to 40 years old and, the majority, are IT savvy.

The desired Information System was an in-house Information System to manage activities. The enterprise had an old solution that was outdated and not aligned with current needs. The modules they wanted to improve were time reporting, project management and financials.

Both forums defined in subsection B of the proposal were evaluated for a period of time. The comment-oriented forum was applied to discuss the time reporting module while the vote-oriented forum was applied to discuss the other two modules. Table I resumes the features of these forums.

TABLE I. FEATURES OF THE FORUMS

Forum	Comment-Oriented	Vote-Oriented	
Module Feature	Time Reporting	Project Management	Financials
Participation	Comments	Comments and Votes	Comments and Votes
Period	20 days	20 days	20 days
Initial Questions	8	3	3
Sequential Navigation	Yes	No	No
Anonymity	Yes	No	No

All employees have to report time and, as such, stakeholders of the time reporting module are all of the employees. As with project management and financials modules, only project managers and directors have to coordinate these activities. As a result, only the 14 employees that are project managers and directors were invited to discuss these two modules.

V. RESULTS

The main results are presented in Table II. The comment-oriented forum obtained around 10.25 comments per discussion topic and 15 identified needs. Anonymous comments made it impossible to measure the average comments per user and no vote system was integrated. Some

participants posted figures and graphics as examples of their perspectives.

The vote-oriented forum obtained, per discussion topic, around 18 comments in the project management module discussion and 11.30 comments in the financials module discussion. No other discussion topics were introduced by the participants. The average comments per user were 4.79 in the project management module discussion and 1.36 in the financials module discussion. The vote system verified around 7.14 votes per user. Finally, 33 needs were identified for the project management module and 11 needs for the financials module.

TABLE II. MEASURED INDICATORS

Forum Module Characteristic	Comment-Oriented	Vote-Oriented	
	Time Reporting	Project Management	Financials
Comments per topic	10.25	18	11.30
Comments per user	-	4.79	1.36
Votes per user	-	7.14	
Identified Needs	15	33	11

The number of comments during the discussion period was also measured. Figure 1 shows that the comment-oriented forum had a regular participation over time. However, the vote-oriented forum had a higher participation on the last days of the discussion, particularly in the last three days.

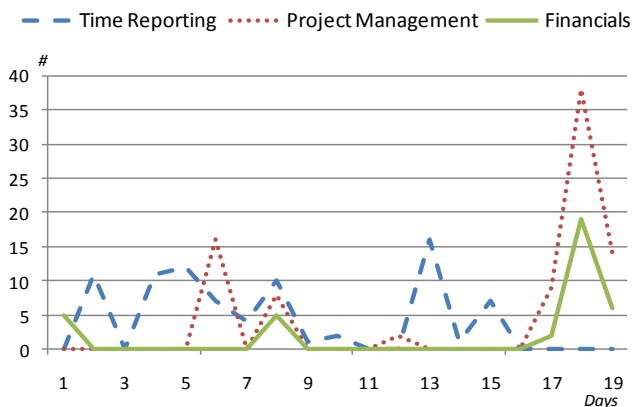


Figure 1. Participation during the Discussion Period.

The resolution of conflicts was performed by participants themselves. When conflicts of interests rose and it was impossible to satisfy different identified needs, participants were advised to resolve the disagreement. The discussion with all stakeholders allowed them to understand the other perspective. More support and justifications were given to one of the perspectives and the most approved was finally agreed.

After closing the discussion period, employees were asked about their participation, including reasons that lead them to participate or not. The received feedback involved 30% of the participants invited to the comment-oriented forum and 50% of the participants invited to the vote-oriented forum. The results are presented in Table III.

TABLE III. FEEDBACK

Type	Feedback
Nonparticipation Reasons	No ideas, lack of time to participate and recent employees with no experience
Positive Aspects	Extensive to all employees, anonymity, simple participation rules, structured discussion and open comments about discussion topics
Improvements	Present just key discussion topics, give rewards to participants, support the vote system to avoid repeating ideas, suppress suggested answers, no anonymity to avoid unreasonable censures

No transcriptions of the discussion were needed since it was already written. Identified needs were resumed in a descending order and included in a report delivered to project managers. Priorities were defined according to the number of references in the comment-oriented forum and to the number of votes in the vote-oriented forum

VI. LEARNING

Stakeholders' feedback confirms that this initiative was considered useful so that they could express their opinions.

Results show a higher participation rate in the vote-oriented forum although this forum involved only 14 stakeholders. Votes also counted as participations, which may explain this result. Typically, people avoid spending time on writing or exposing detailed ideas and the participants' final feedback for the voting system highlights this fact. It is easier to vote than to write comments.

Although we verified a higher participation rate in the vote-oriented forum, we consider that both forums had a fair participation rate. The contributions were rich, since they were always justified and, some included uploads of hand-made illustrations to explain and substantiate the participant's idea. As such, the comments were not only quantitatively but also qualitatively meaningful.

The asynchronous communication aspect of this tool allowed participants to express perspectives along time whenever they could. Also, the online tool allowed users to contribute with their ideas from wherever they were.

We also verified that users of both forums did not answer to all of the discussion topics. The topics with high discussion rates were key discussion topics in the comment-oriented forum and difficulties questions to judge the existing modules in the vote-oriented forum. This not only reveals that stakeholders prefer to directly answer key questions, but also that people are critical since there were more criticisms than comments about positive aspects or new ideas. Criticisms are also useful to understand what is wrong and should be improved.

The identified needs were mostly technical requests. For example, users wrote that the existing modules are too slow, that some buttons should be available, that certain

information is needed or that a particular navigation would be better than the existing one. This can easily be explained by the participants' profiles: almost all participants are computer engineers and, as such, they understand the problems behind the existing modules and tend to express technical opinions. This is clear since the comments of less technical users were also less technical.

Anonymity promoted free answers in the one hand, but it lead to less participation and conflicts in the other hand. Anonymity brought unexpected criticisms that otherwise could not have been revealed. These criticisms also brought conflicting perspectives that were discussed, always reaching consensus. However, the forum without anonymity had more comments per user indicating a commitment sense of identified users. The lack of conflicts may mean that users agreed to each other or that they did not feel free to disagree. Some participants believe that anonymity allows free expression of ideas without fear of consequences while others think that anonymity brings unreasonable criticisms.

Figure 1 shows that the comment-oriented forum had more regular participation than the vote-oriented one, which had more participation at the end of the period. This fact may be explained because votes also count as participation and at the end of the period there were more comments to vote. Actually, this voting system helps users not to repeat ideas and to prioritize needs. It is also possible to vote when comments are available, incrementing the participation rate.

Most important, suggested needs were integrated in a new version of the time reporting module and the overall feedback of stakeholders was positive. Moreover, they use the module more frequently, with fewer problems and without spending much time. These results prove that our method helps overcoming limitations, such as difficulties in communication and understanding between stakeholders and analysts, quality of identified requirements, stakeholders' conflicts, and the experience and practice of the analyst. Also, this method overcomes well know problems of group work methods, including dominant users with no limitations of time for each participant; gathering stakeholders at the same time and place with an online tool; generalization of results with the involvement of all stakeholders; and simplification of the analysis since no transcriptions or interpretations are needed.

VII. CONCLUSION

Stakeholders approved the initiative of freely expressing ideas in a simple and structured online forum. They prefer the voting system than repeating others' ideas and the discussion of key questions than starting with trivial and irrelevant discussion topics. Anonymity on the one hand helps users to make more censorious comments but, on the other hand, seems to take out the participation commitment and to discourage the disagreement of ideas. As such, some participants would prefer anonymity while others think that comments should be identified to avoid unreasonable and thoughtless criticism. We conclude that this proposal allowed quickly eliciting needs from all stakeholders.

After implementing the suggested needs for the time reporting module, the results prove that the method was a

success. Stakeholders are pleased with the new system and report their time more frequently, without mistakes or complaints. As such, we can conclude that the proposed method was a success.

Our method seems more effective over existing methods with our new features. The most significant feature is opening the discussion to all stakeholders. With this features, we wanted to ensure that inputs of important requirements would not be forgotten. Also, our proposal added a voting system and compared the anonymity aspect in two different approaches. This is also an unusual configuration of collaborative tools, particularly those who integrate focus groups as an elicitation method. These features bring effectiveness to the method by allowing a richer overview of the system with inputs of everyone, ordered needs with priorities given by participants and free expression with the anonymous approach or commitment sense with the approach with identified participants.

The major concerns we wanted to address, allowing an asynchronous and distributed communication; avoiding interpretation of results by analysts; inviting all stakeholders to contribute with their ideas; and obtaining agreement of all stakeholders about the identified needs, were met. As such, the problems we identified to resolve were actually addressed.

This paper proves that a collaboration tool based on focus groups allows eliciting requirements more quickly than the regular focus group of our previous experiment. This fact is easily understood since our previous effort required weeks to schedule a meeting with key stakeholders at the same time and place and more weeks to analyze results, transcribing the sessions and examining the information.

This paper also proves that a collaboration tool overcomes many problems of regular focus groups and elicitation methods. For example, communication between stakeholders and analysts is eased with this tool. Quality of identified requirements is higher since discussion provides richer information. Stakeholders' conflicts may be resolved in discussion with other participants. Dominant users are no longer a problem since time spent with the contribution of a participant does not steal time of another participant. Gathering stakeholders at the same time and place is no longer a problem as well.

Note that the demonstrated improvements are not meant to apply in general domains but in the maintenance/evolution of an existing Information System.

More research work is needed to confirm these results in other projects. The feedback of stakeholders also suggests other aspects to include in a future research. First, rewards to participants should be given to encourage participation. Second, only key discussion topics should be initially provided but allowing to add new discussion topics. Third, the voting system should always be present. Fourth, suggested answers should be removed so that users do not feel biased to answer.

ACKNOWLEDGMENT

We would like to thank the participants in the presented experiment as well as the managers of the organization which made the experiment possible.

REFERENCES

- [1] D. Avison and G. Fitzgerald, *Information systems development: methodologies, techniques and tools*. New York, NY: McGraw-Hill, 2006.
- [2] J. Coughlan and R. Marcredie, "Effective communication in requirements elicitation: a comparison of methodologies," *Requirements Engineering*, vol. 7, pp. 47-60, 2002.
- [3] B. Davey and C. Cope, "Requirements elicitation - what's missing?," in *Issues in Informing Science and Information Technology*, vol. 5, 2008, pp. 543-551
- [4] D. Zowghi and C. Coulin, "Requirements Elicitation A Survey of Techniques, Approaches and Tools," in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds. Berlin: Springer, 2005.
- [5] T. S. Group, "The Chaos Report," 2009.
- [6] A. Al-Rawas and S. Easterbrook, "Communication problems in requirements engineering: a field study," in *First Westminster Conference on Professional Awareness in Software Engineering* London, 1996.
- [7] J. F. M. Burg, *Linguistic instruments in requirements engineering*. Amsterdam: IOS Press, 1997.
- [8] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in *Conference on The Future of Software Engineering 2000*, 2000, pp. 35-46.
- [9] R. Hossenlopp and K. B. Hass, *Unearthing business requirements: elicitation tools and techniques*. Vienna: Management Concepts, 2007.
- [10] A. Crabtree, "Ethnography in participatory design," in *1998 Participatory Design Conference* Seattle, 1998, pp. 93-105.
- [11] A. Crabtree, D. M. Nichols, J. O'Brien, M. Rouncefield, and M. B. Twidale, "Ethnomethodologically-informed ethnography and information system design," *Journal of the American Society for Information Science*, vol. 51, pp. 666-682, 2000.
- [12] J. A. Goguen and C. Linde, "Techniques for Requirements Elicitation," in *Requirements Engineering*, S. Fickas and A. Finkelstein, Eds.: IEEE Computer Society, 1993, pp. 152-164.
- [13] W. S. Davis and D. C. Yen, "Joint application design (JAD)," in *The Information System Consultant's Handbook*: CRC Press, 1998.
- [14] E. J. Davidson, "Joint Application Design (JAD) in practice," *Journal of Systems & Software*, vol. 45, pp. 215-223, 1999.
- [15] E. W. Duggan and C. S. Thachenkary, "Supporting the JAD facilitator with the nominal group technique," *Journal of Organizational and End User Computing*, 2004.
- [16] C. Farinha and M. M. d. Silva, "Focus Groups for eliciting requirements in information systems development," in *14th UK Academy for Information Systems*, 2009, p. 20.
- [17] P. Engelbrektsson, Ö. Yesil, and I. C. M. Karlsson, "Eliciting customer requirements in Focus Group interviews: can efficiency be increased?," in *7th International Product Development Management Conference* Belgium, 2000.
- [18] J. Bhattacharjya and J. Venable, "An Action Research Approach to Strategic Information Systems Planning in a Non-profit Organization," in *3rd International Conference on Qualitative Research in IT and IT in Qualitative Research* Brisbane, Australia, 2006.
- [19] D. Coghlan and T. Brannick, *Doing Action Research in your own organization*, 3 ed. Los Angeles: SAGE Publications Ltd, 2009.
- [20] M. Sadiq, M. Shahid, and S. Ahmad, "Adding thread during software requirements elicitation and prioritization," *International Journal of Computer Applications*, vol. 1, 2010.
- [21] I. Sommerville, *Software engineering*, 6 ed. Harlow, England: Pearson Education, 2001.
- [22] J. L. Maté and A. Silva, *Requirements engineering for sociotechnical systems*. Hershey, USA: Idea Group Inc (IGI), 2005.
- [23] N. Maiden and S. Robertson, "Integrating creativity into requirements processes: experiences with an air traffic management system," in *13th IEEE International Conference on Requirements Engineering* Paris, 2005, pp. 105-114.
- [24] R. A. Krueger and M. A. Casey, *Focus groups: a practical guide for applied research*. California: SAGE, 2000.
- [25] J. Kitzinger, "The methodology of Focus Groups: the importance of interaction between research participants," *Sociology of Health & Illness*, vol. 16, pp. 103-121, 1994.
- [26] Z. M. Kasirun and S. S. Salim, "Supporting collaborative requirements elicitation using Focus Group discussion," *International Journal of Software Engineering and its Applications*, vol. 3, 2009.
- [27] J. Whitehead, "Collaboration in software engineering: a roadmap," in *Future of Software Engineering, 2007. FOSE '07*, Minneapolis, 2007.
- [28] D. Apshvalka, D. Donina, and M. Kirikova, "Understanding the Problems of Requirements Elicitation Process - A Human Perspective," in *Information systems development: challenges in practice, theory, and education*, vol. 1, C. Barry, K. Conboy, M. Lang, G. Wojtowski, and W. Wojtowski, Eds. New York, NY: Springer, 2008, p. 600.
- [29] J. D. Herbsleb, "Global software engineering: the future of socio-technical coordination," in *Future of Software Engineering, 2007. FOSE '07*, Minneapolis, 2007, pp. 188-198.

Mapping Architectural Concepts to SysML Profile for Product Line Architecture Modeling

Shahliza Abd Halim, Mohd Zulkifli Mohd Zaki, Noraini Ibrahim, Dayang N. A. Jawawi and Safaai Deris

Software Engineering Department
Faculty of Computer Science and Information Systems,
Universiti Teknologi Malaysia,
81310 Skudai, Johor, Malaysia.

shahliza@utm.my, zulkiflizaki@utm.my, noraini_ib@utm.my, dayang@utm.my, safaai@utm.my

Abstract— There are different proposals for modelling Product Line Architecture. To model the Product Line Architecture (PLA), the most important elements are the explicit treatment of its commonality and variability representation. This paper concentrates on the use of Architecture Description Language (ADL) and its integration with object oriented modeling, for the representation of architecture in order to model PLA architecture construct and variability construct effectively. Consequently, the possibility of integration which involves the mapping between the xADL and SysML a UML2 based profile extension to enable the profile to be incorporated to an existing UML commercial tool was investigated. The result of the mapping is proposed extension to SysML profile. The profile is then applied to a case study of Autonomous Mobile Robot Product Line. Based on the case study evaluation, the profile has shown a significant improvement to the existing SysML for modelling PLA.

Keywords-product line architecture (PLA); autonomous mobile robot (AMR),architecture description language, xADL

I. INTRODUCTION

Product line architecture (PLA) is the first artifacts which realise the requirements for the Software Product Line (SPL) and also is the abstraction for detail design. PLA modelling differs from other single system architecture modelling where PLA should be able to express commonality and variability explicitly in its architecture. Thus, in addition for modelling the basic architectural structure, PLA has to model variability information such as optional and variants structure and the rule in choosing between different variants and optional structures as it will affect the derivation of product specific architecture.

Therefore, it is essential to model architecture in a formal manner which ensures a better tool support and also a comprehensive architecture description. A consistent, complete and correct architecture description is by using Architecture Description Language (ADL) [1]. Nevertheless ADL is reported to not integrate well with software development methodology and tools [2]. Another paradigm for representing architecture is with UML which has been used as an architecture modelling language and also a de facto modelling language used in the industry, even so there

are arguments concerning its modelling notations inadequacy for representing architecture [3, 4].

Integrating both languages, ADL and UML can be considered as having a synergistic relationships where the combination enables a precise and explicit architecture description and at the same time having a wider usage among UML users in commercial tool. Among the proposed integration approaches are from [2, 5]. This paper concentrates on how to map architecture concept from xADL to one of UML profile, SysML for an explicit representation of architectural and variability construct for modelling PLA.

SysML is a profile targeted for system engineering where the strength of SysML compared to UML 2.0 is based on its new addition of requirements and parametric diagram as well as its additional constructs in architecture modelling. xADL is chosen due to its specialised schema targeted for product line architecture description [6].

The remainder of this paper is organised as follows: In Section II, the paper discusses the problem background which motivates the focus of this paper. Section III discusses on the methodology of the mapping and the profile proposed based on the mapping. Section IV demonstrates the applicability of the proposed profile in modeling Autonomous Mobile Robot (AMR) Product Line case study. Section V discusses on the feasibility of the results. Lastly, Section VI discusses on the conclusion and recommendation for future work.

II. BACKGROUND

There are two extension mechanisms in customising UML metamodel. First class extension mechanism is by adding or removing metaclasses in Metamodel Object Facility (MOF) or can be referred as heavyweight extension. Another mechanism is by using profile which does not allow any modification of existing metamodel other than by adjusting the metamodel with constructs suitable for the particular domain, platform or method [7]. Profile extension is also known as lightweight extension. Thus, the latter option is opted as SysML is already an established profile for modelling System Engineering applications. Thus, by extending the profile, the best aspect of SysML can be leveraged while lowering the learning curve. Furthermore, the profile extension is conformed to standard UML hence can also be supported by existing UML tool.

The first notions of modelling architecture by using UML in a formal manner by mapping it to ADL were done in [3, 8] Nonetheless, both researchers do not concentrate on UML 2.0. Even so, they both highlight a useful consideration to be acknowledged in the mapping. Garlan, Cheng and Kompanek [8] discuss the pros and cons of using different metaclasses for representing architectural concepts. Medvidovic and Rosenblum [2] describe a more detailed mapping between C2 and Wright architecture style and UML which involved two strategies of either using UML "as is" or by using stereotypes with restrictions by means of OCL. The moves towards mapping of UML 2.0 with ADL were done in [9-11]. Only one of the researchers proposed on using UML 2.0 in order to model variability in architecture. Nonetheless, the concentration of Choi [11] is on representing variability in PLA behavior through connector . There is also a research done by Maga and Jazdi [12] on extending variability in SysML profile. Although the researchers have proposed a variety of variants in their extension, however they do not specifically address the PLA and its constructs in the extension. This research concentrates on filling in the gap in the PLA modelling by concentrating on a formal architectural and variability construct based on xADL and its mapping to SysML for profile extension.

III. MAPPING STRATEGY FROM ARCHITECTURE DESCRIPTION TO SYSML

The mapping strategy is basically divided into three steps: Mapping basic architecture construct; Mapping variability construct and mapping the constrain. However, the third strategy is not the focus for this paper hence is not being elaborated. The mapping steps are as shown in Figure 1. Each step is described in detail as follows:

A. Mapping Basic Architectural Construct

The corresponding elements of the mapping are between the *Structure and Types* schema in xADL which can be mapped into two corresponding packages in SysML, *Blocks* package and *Ports&Flows* package. The first mapping is between component from xADL and block from SysML. Block is equivalent to component where block is an extension of class metaclass in SysML. The same notion is also used by Medvidovic and Rosenblum [3] where their component is an extension of UML class metaclass instead of extending from component metaclass. In the case of mapping between Connector, Interface, Link, Point and Group from xADL *Structure and Types* schema, roughly all the elements are also present in SysML *Ports&Flow* package. Nonetheless, construct such as connector is not explicitly specified. Instead it can only be identified when a relationships between two roles in SysML are specified. Therefore, stereotypes were explicitly added for both connector and role. Signature schema in xADL construct is also added as stereotype extending an interface metaclass.

B. Mapping Variability Construct

The schema in xADL has to be in an equivalent metamodel form before the mapping to SysML can be done.

Thus, for Variants package and Options package in xADL which do not have a corresponding matching in SysML, a package called Variability package is extended from the original SysML profile to support variability as shown in Figure 1. The added stereotype is extended from the class metaclass.

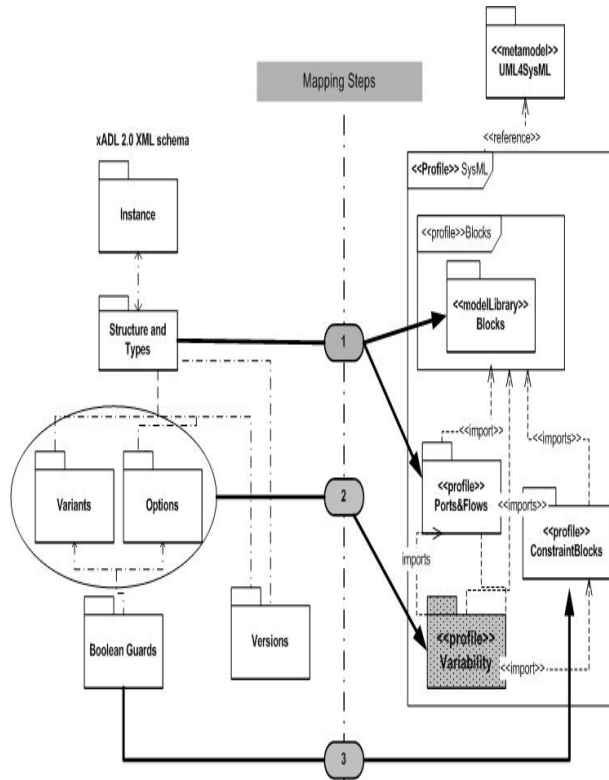


Figure 1. Mapping Strategy

C. Mapping results

Based on the specified strategy, the proposed profile for representing architecture based on the mapping is as shown in Figure 2. The profile is divided into three sections, the metaclass section which consists of UML classes reused in SysML known as UML4SysML. The architectural construct section which shows the extension of stereotype classes shaded in grey. The variability construct section which shows the extension of stereotype to represent variability, variants and option can be applied to the architecture construct since both the variability constructs extends from class metaclass. Another variability construct, representing guard in xADL schema is added as a stereotype extension from ConstrainBlock stereotype.

IV. CASE STUDY

In order to validate the applicability of the extended modelling in SysML, the extended model was applied to product line of Autonomous Mobile Robots (AMR). The product line consists of five different but similar applications of AMR. Four of the AMR are AMR for research, AMR for teaching, i-wheelchair and intelligent scooter based on the research collaboration done at Embedded Real Time and

Software Engineering Research Lab (ERetSEL), Universiti Teknologi Malaysia. The fifth AMR is the parking assistant based on the work of Polzer, Kowalewski and Botterweck [13]. The five AMR product line (AMRPL) are as shown in Figure 3.

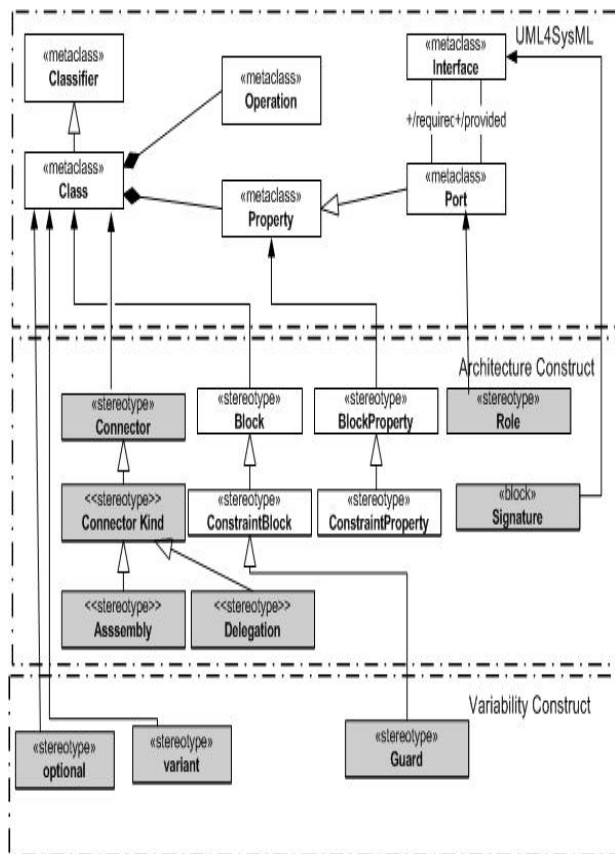


Figure 2. xADLUMLProfile

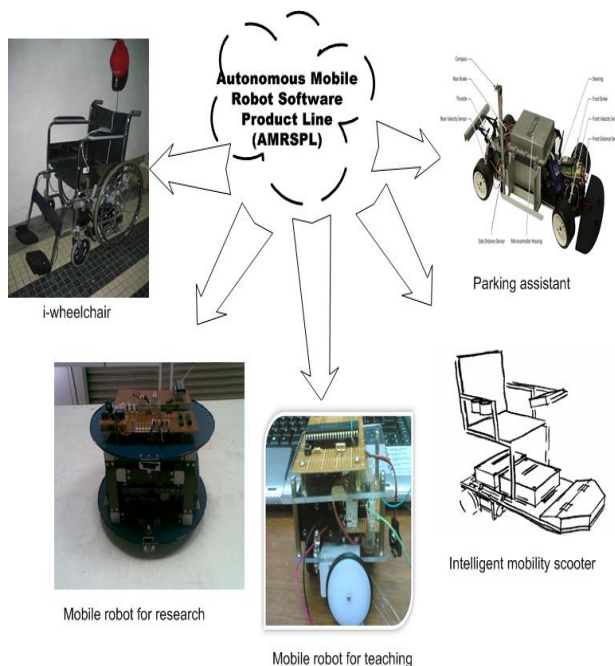


Figure 3. AMR Product Line (AMRPL)

In order to identify the commonality and variability of the AMRPL requirements, approach by Abd Halim, Jawawi and Safaai [14] is used. However, in order to simplify this paper, the common and variable function is represented in use case diagram as shown in Figure 4.

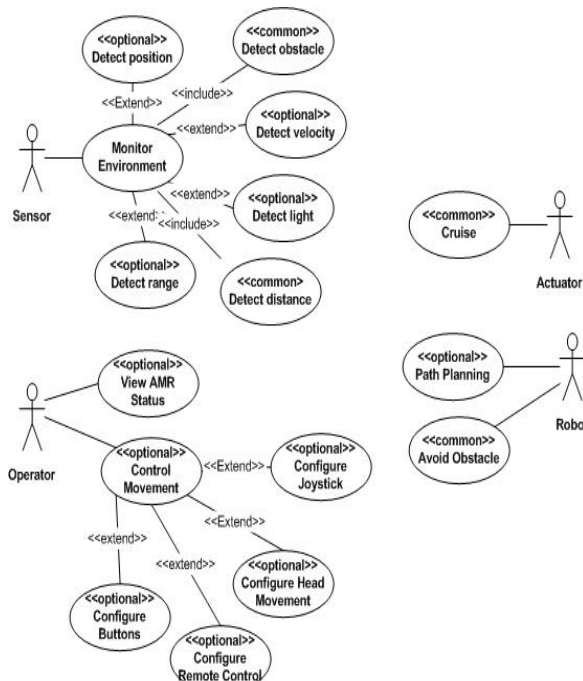


Figure 4. AMRPL Use Case

The proposed profile is used in modelling general architecture, the block definition diagram (bdd) in SysML. The bdd diagram shows the structure of the components in the form of *noncomposite* relationships which will explicitly shows the common and variable blocks involved in the system. Other than *noncomposite* relationships, another relationships that can be shown in bdd is the *whole-part* relationships [15]. However, in PLA, a noncomposite relationship is more suitable as the block can either be selected or not selected based on its variability and commonality and it would not affect the block which it related to. The *whole-part* relationships is not chosen as it will affect the relationships between blocks which is not being selected for composition.

Furthermore, the profile will then be used for modelling specific architecture, which is shown in SysML internal block diagram (ibd). In this diagram the component which are modelled as having a *part* component in the bdd will be elaborated further, which will explicitly show the variability in the connectors and the internal components relationships. Both bdd and ibd diagram can be referred at Figure 5 and Figure 6 respectively.

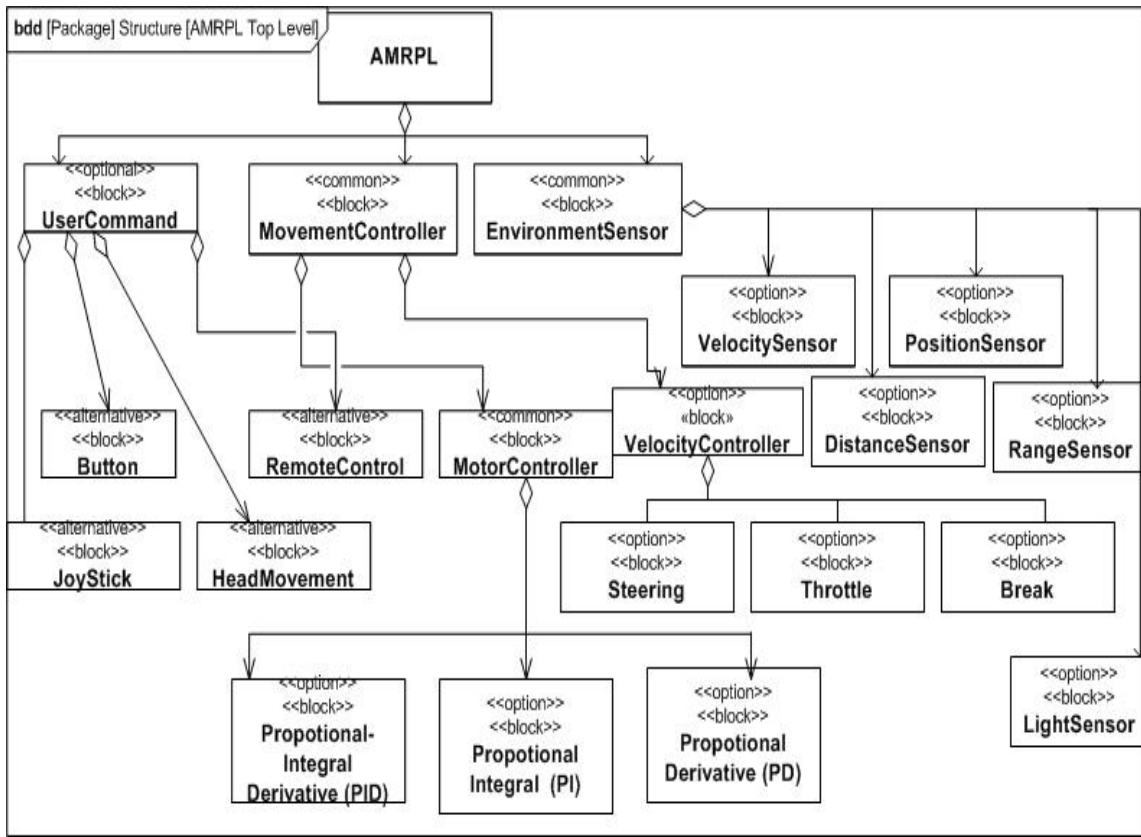


Figure 5. Block Definition Diagram for AMRPL

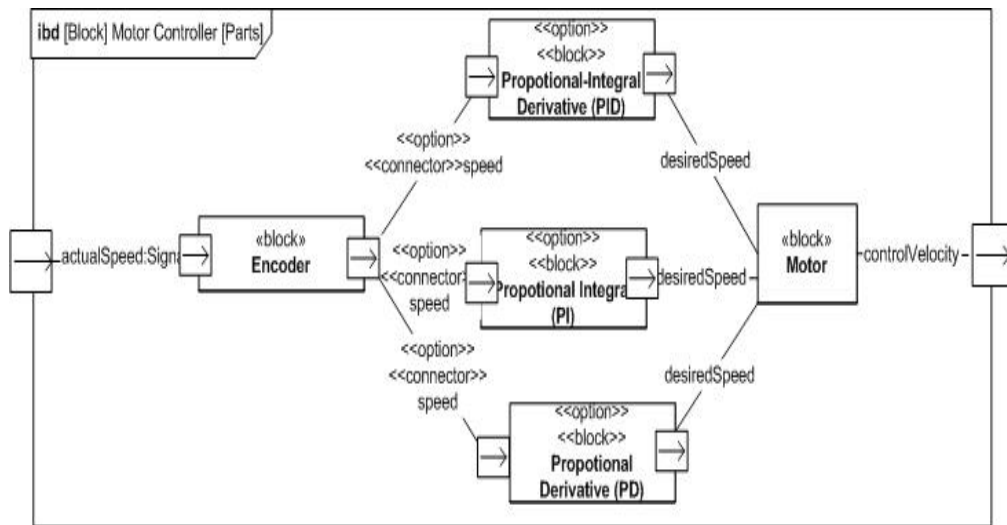


Figure 6. Internal Block Diagram for Motor Controller Part

V. DISCUSSION

From the case study, PLA modelling can be seen in different granularity based on the SysML bdd and ibd diagram. In both diagrams, the proposed xADL UML profile is used to augment the diagram with commonality and variability stereotypes and also with a more concrete architectural construct. From the case study, bdd diagram in Figure 5 shows common and optional stereotype in the AMRPL such as Motorcontroller block which have the option of using PID, PI or PD as a controller. The Motorcontroller blocks and its parts is further refined in ibd diagram in Figure 6 where the common and variable connectors are clearly shown by the stereotypes. Nonetheless, there are few constructs in the profile that is not being applied to the case study such as the use of delegation and assembly and the use of signature for representing interface. However, the case study did reflect a significant potential in modelling PLA in both bdd and ibd diagram thus helps in the understanding of the PLA for an easier product specific derivation for AMRPL.

According to our experience with this case study, the profile can explicitly show the commonality and variability in the AMRPL. The mapping from xADL to SysML profile further helps in formalising the architectural concepts of the modelling. Nonetheless, rule is essential to ensure consistency between the model elements. Therefore, OCL rule should be added in the profile to constraint metaclasses between the bdd and ibd diagram. The constraint will determine the consistency between the different views of the block diagram. Consequently, it is essential to understand how the rule in xADL Guard can be translated into OCL rule for the purpose.

VI. CONCLUSION

This paper concentrates on how to map architecture concept from xADL to one of UML profile, SysML for an explicit representation of architectural and variability construct for modeling PLA. From the case study, it did show a noteworthy contribution in modelling PLA in terms of its blocks and its connector and also in terms of the granularity of the modelling. A more extensive case study should be done in the future in order to fully validate the proposed profile. Furthermore, rules to infuse consistency of the metaclasses and its instance should be further explored. It is hoped from the explicit modelling of PLA commonality and variability can further help reuser to derive a product specific application in the application engineering phase.

ACKNOWLEDGMENT

This research is fully funded by the Research University Grant (RUG) from the Universiti Teknologi Malaysia (UTM) and Ministry of Higher Education (MOHE) under

Cost Center No.Q.J130000.7128.03J23. Our profound appreciation also goes to ERetSEL lab members for their continuous support in the working of this paper.

REFERENCES

- [1] Taylor, R.N., N. Medvidovic, and E.M. Dashofy, *Software Architecture: Foundations, Theory and Practice*. John Wiley & Sons, Inc.) 2009.
- [2] Kandé, M.M. and A. Strohmeier. Towards a UML profile for software architecture descriptions. 2000: In Springer-Verlag, pp. 513-527.
- [3] Medvidovic, N., et al., Modeling software architectures in the Unified Modeling Language. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2002. vol 11, pp. 2-57.
- [4] Medvidovic, N., E.M. Dashofy, and R.N. Taylor, Moving architectural description from under the technology lamppost. *Information and Software Technology*, 2007. vol 49, pp. 12-31.
- [5] Cheng, S.W. and D. Garlan. Mapping Architectural Concepts to UML-RT. in *Proceedings of the Parallel and Distributed Processing Techniques and Applications Conference*. 2001.
- [6] Dashofy, E.M., A. Hoek, and R.N. Taylor, A comprehensive approach for the development of modular software architecture description languages. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2005. 14(2): pp. 199-245.
- [7] SysML. *OMG SysML Specification v. 1.0*. Retrieved from <http://www.sysml.org>. 2006. Last accessed on 21.7.2011.
- [8] David, G., C. Shang-Wen, and J.K. Andrew, Reconciling the needs of architectural description with object-modeling notations. *Sci. Comput. Program.*, 2002. 44(1): pp. 23-49.
- [9] Goulão, M. and F.B. e Abreu, Bridging the gap between Acme and UML 2.0 for CBD. *SAVCBS 2003 Specification and Verification of Component-Based Systems*, 2003: pp. 75-79.
- [10] Meister, J., R. Reussner, and M. Rohde. Applying patterns to develop a product line architecture for statistical analysis software. in *Software Architecture*, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference, 2004.
- [11] Choi, Y., et al. An approach to extension of UML 2.0 for representing variabilities. *Fourth Annual ACIS International Conference*, IEEE, 2005, pp. 258-261.
- [12] Maga, C.R. and N. Jazdi, Survey, Approach and Examples of Modeling Variants in Industrial Automation. *Journal of Control Engineering and Applied Informatics*, vol.13, pp. 54-61.
- [13] Polzer, A., S. Kowalewski, and G. Botterweck. Applying Software Product Line Techniques in Model-based Embedded Systems Engineering. in *MOMPES 2009*, Vancouver, Canada, pp. 2-10.
- [14] Halim, S.A., D.N.A. Jawawi, and S. Deris. Requirements Identification and Representation in Software Product Line. In *Asia Pacific Software Engineering Conference (APSEC'09)*, IEEE, Pulau Pinang, Malaysia, pp. 340-346.
- [15] Friedenthal, S., A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. (The OMG Press), Morgan Kaufmann, 2008.

Exploring Architecture Design Alternatives for Global Software Product Line Engineering

Bedir Tekinerdogan

Department of Computer Engineering
Bilkent University
Ankara, Turkey
e-mail: bedir@cs.bilkent.edu.tr

Semih Cetin, Ferhat Savci

Cybersoft Information Technologies, Ata Plaza 3/3,
34758,
Atasehir, Istanbul, Turkey
e-mail: {semih.cetin, ferhat.savci}@cs.com.tr

Abstract — Current trends in software engineering show that large software projects have to operate with teams which are working in different locations. An analysis of current global software engineering literature shows that the focus has been basically on single system development. Yet, very often organizations do not aim to develop a single product but a *product line* for a particular market segment. Unfortunately, the notion of global software development has not been explicitly addressed in product line engineering. We introduce and define the notion of *global software product line engineering (GSPLE)* to integrate global software engineering paradigm with the software product line engineering paradigm. Based on an analysis of architectural approaches in both paradigms we define the space of the different software architecture design alternatives for GSPLE. We illustrate the architecture design alternatives using examples of an industrial context.

Keywords-Product Line Engineering; Global Software Development; Business Strategies

I. INTRODUCTION

Current trends in software engineering show that large software projects have to operate with teams that are working in different locations. The reason behind this globalization of software development stems from clear business goals such as reducing cost of development, solving local IT skills shortage, and supporting outsourcing and offshoring [1]. There is ample reason that these factors will be even stronger in the future, and as such we will face a further globalization of software development [8]. To cope with these problems the concept of global software engineering (GSE) is introduced [9]. GSE is a relatively new concept in software development that can be considered as the coordinated activity of software development that is not localized and central but geographically distributed.

An analysis of current global software engineering literature shows that the focus has been basically on single system development. Yet, very often organizations do not aim to develop a single product but a *product line*. A product line is defined as a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are

developed from a common set of core assets in a prescribed way [11]. Despite earlier software reuse approaches, software product line engineering (SPLE) aims to provide pro-active, pre-planned reuse at a large granularity to develop applications from a core asset base. The key motivation for adopting a product line engineering process is to develop products more efficiently, get them to the market faster to stay competitive and produce with higher quality [14]. In alignment with these goals different software product line engineering approaches have been proposed [5][11].

Unfortunately, the notion of global software development has not been explicitly addressed in product line engineering. On the other hand, an analysis of the current product line engineering approaches shows that global software development is not explicitly addressed. We can observe valuable knowledge on defining organization structures for product lines [4][11] but these do not explicitly consider the concern of globalization of the product line engineering process. To apply systematic, anticipated reuse for global software development we believe that global software development will substantially benefit from software product line engineering. In parallel, similar to single system development in which teams might be spread over different locations [3], it is also expected that product line engineering projects might operate with teams which are working in different locations. The reason for this globalization of product line engineering will also be based on the general motivations for global software development.

In this paper, we introduce and define the notion of *global software product line engineering (GSPLE)* to integrate global software engineering paradigm with the software product line engineering paradigm. The motivation for GSPLE stems from the industrial context of Cybersoft, a leading company in global software development in Turkey. The efforts to define the architecture for GSPLE have shown that the integration of SPLE and GSE can be done in multiple different ways. Based on an analysis of architectural approaches in both paradigms and our experiences we define the space of the different software architecture design alternatives for GSPLE. We illustrate the architecture design alternatives using examples of an industrial context.

The remainder of the paper is organized as follows. In Section II we briefly introduce a conceptual model for GSE, followed by an analysis to software product line engineering in section III. Section IV discusses the stakeholder analysis for GSPLE. Section V describes the strategies for integrating SPLE with GSE. Section VI discusses the design alternatives. Section VII provides the related work and section VIII concludes the paper.

II. CONCEPTUAL MODEL FOR GSE

GSE is a software development approach that can be considered as the coordinated activity of software development that is not localized and central but geographically distributed. Overall we can identify four important key concerns in GSE:

Development - the software development activities typically using a software development process. This includes activities such as requirements analysis, design, implementation and testing. Each product development site will address typically a subset of these activities.

Communication – communication mechanisms within and across sites. Typically the different sites need to adopt a common communication protocol.

Coordination – coordination of the activities within and across sites to develop the software according to the requirements. Coordination will be necessary to align the workflows and schedules of the different sites. An important goal could be to optimize the development using appropriate coordination mechanisms.

Control – systematic control mechanisms for analyzing, monitoring and guiding the development activities. This does not only include controlling whether the functional requirements are performed but also which and to what extent quality requirements are addressed.

Each of these concerns and the way they are allocated in the GSE environment will have a direct impact on the architecture. In principle, we assume that each of these concerns can be mapped to a separate implementation unit, or *layer*. Based on this assumption we have defined the conceptual layered model for GSE system as defined in Figure 1.

Here we have depicted GSE system as consisting of a structure with separate activity layers that depend on each other. The layering is defined based on conceptual relations. Activities in the *development layer* are coordinated by the *coordination layer*. The coordination of the activities will be controlled by functionality in the *control layer*. Finally, the development, coordination and control layers will require suitable communication mechanisms which are provided by the *communication layer*. In Figure 1, we have provided communication layer as a sidebar indicating that all layers will use this layer. Alternatively, a separate specific communication mechanism could be provided for each layer.

Based on this layered view of GSE system we need to decide how to allocate each layer to different nodes in the

GSE environment. In the following sections we will define the different concrete deployment alternatives for GSE systems based on this model.

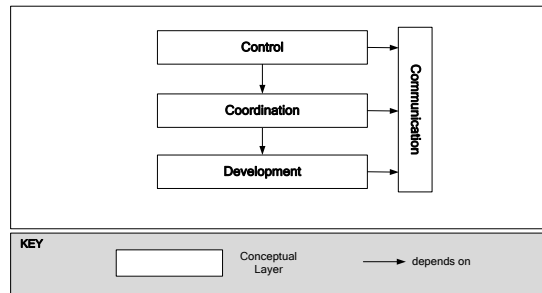


Figure 1. Layered View of GSE system with four key concerns

III. SOFTWARE PRODUCT LINE ENGINEERING

Global software development can be focused on single software development or *product line engineering* [5]. Although different product line engineering processes have been proposed they share the same concepts of *domain engineering*, in which a reusable platform and product line architecture is developed, and *application engineering*, in which the results of the domain engineering process are used to develop the product members.

In general the adopted product line engineering approach has not been directly considered for global software engineering. Integration of both paradigms would in principle mean to define and align the common product line engineering process to a given GSE software architecture.

Since each unit can be considered as a separate, independent unit, the GSE system can be also set up as a *production line*. The concept of *production line* is defined in the industrial engineering and denotes a set of sequential operations established in a factory whereby materials are put through a refining process to produce an end-product; or components are assembled to make a finished article. Although the notion of software *product line engineering* is quite popular this does not seem to be the case for software *production line engineering*. Nevertheless, we think that this is important for GSE. In principle, the development units in GSE can also be considered as separate domain specific entities that aim to develop particular intermediate products, and likewise a production line can be set up.

IV. DESIGN SPACE FOR GSPLE ARCHITECTURE

It appears that we can combine the three different concepts of Global Software Engineering, Software Production Line and Software Product Line Engineering in different ways. We depict the different possibilities in Table 1. The names of the alternatives indicate whether the development is local (L) or global (G), whether production line (Pn) is applied or a conventional approach is used (C), and whether the focus is on product line (Pl) or single-system development (S). As such, the first four alternatives define the case of local software development in which the

development units are co-located. The last four alternatives define the alternatives for global software development.

To denote the integration of global software engineering with product line engineering we define the notion of *global software product line engineering*. GSPLE spans the last two rows of Table 1 (GCPI and GPnPI). GSPLE can be considered as a special form of product line engineering process in which the development teams are not collocated but distributed as it is defined by the GSE paradigm. The integration of both paradigms might be based on practical necessity but in parallel will also combine the benefits of both product line engineering and global software development. From a reuse perspective we could state that GSPLE even further broadens reuse by also reusing development teams and not only artefacts. In the following we describe each alternative and provide the architectural template and an example.

TABLE 1. DESCRIPTION OF PRODUCT LINE INTEGRATION ALTERNATIVES WITH GLOBAL SOFTWARE DEVELOPMENT

Strategy	Description
LCS	Software development at a single site without product and production lines.
LPnS	Software development at a single site with production line but not focused on product variability management
LCPI	Software development at a single site focused on product variability management without production line
LPnPI	Software development at a single site with production line and focused on product variability management
GCS	Software development at multiple sites without product and production lines
GPnS	Software development at multiple sites with production line but not focused on product variability management
GCPI	Software development at multiple sites focused on product variability management without production line
GPnPI	Software development at multiple sites with production line and focused on product variability management

A. Local Single System Development

Local Single System Development is the traditional way of software development located at a single site. In the following sections we will also introduce product line and production line engineering for GSE, but for now we assume that a single system is developed at a single site. The deployment view for GSE system for this case is shown in Figure 2. Note that the four layers/concerns are mapped to a single deployment node. From a theoretical perspective we could consider local system development as a special case, the simplest one, of global software development.

Example:

John Doe Software Co. develops an accounting system accustomed for Non-Exist Tech. Ltd.. The accounting system is developed at a single site using a traditional, non-product line engineering, development approach.

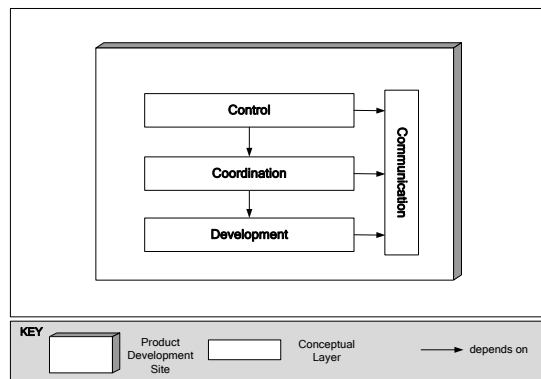


Figure 2. Local Single System Development

B. Local Single System Development with Production Line

We could define a software product line engineering as an application of the Pipes and Filters pattern [2]. Hereby the filters define processing units, whereas the pipes define the mechanism for distribution and communication. A conceptual model of software product line engineering is given in Figure 3. In principle a number of filters, i.e. production units can be defined which can be linked in different ways to each other. However, the key design principle for having independent filters as defined in the Pipes and Filters pattern also seem to apply for the software production line engineering process. This is to say that each production unit can be (largely) seen as a separate, black box unit that can accept input, process this and provide it to the output. In principle, the production units are not aware of each other.

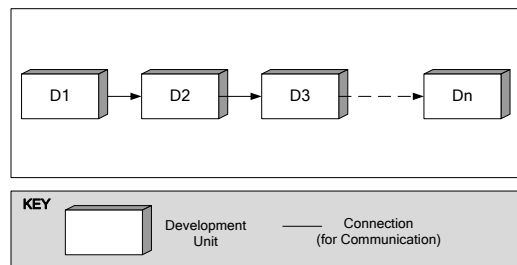


Figure 3. Software Production Line Engineering Process defined using the Pipes and Filter Pattern

Figure 4 shows the deployment view when we apply production line engineering to single-site single system development. Here the Pipes and Filters pattern has been applied to the development process units within a single site. These could be typically the applied workflows of the software development process. In Figure 4, we assume that we apply a centralized control and coordination mechanism.

However, these could also be equally distributed leading to a distributed coordination and control system of the development process.

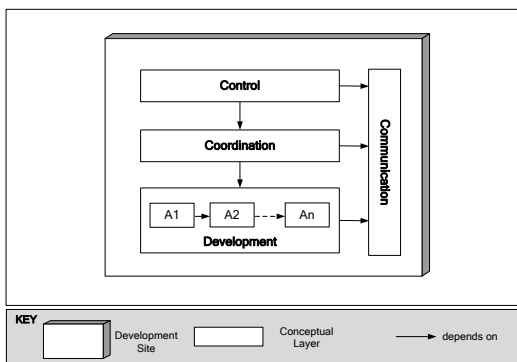


Figure 4. Local Single System Development with Production Line

Example:

John Doe Software Co. has a custom software production line based on SpringSource [12], which is used to develop an accounting system accustomed for Non-Exist Tech. The company intentionally employed the production line to reuse infrastructural modules such as logging, content management, object to relational mapping, etc.

C. Local Software Product Line Development

A product line is defined as a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [5]. The key motivation for adopting a product line engineering process is to develop products more efficiently, get them to the market faster to stay competitive and produce with higher quality. In alignment with these goals different software product line engineering approaches have been proposed. These approaches seem to share the same concepts of domain engineering, in which a reusable platform and product line architecture is developed, and application engineering, in which the results of the domain engineering process are used to develop the product members [5][9].

Example:

John Doe Software Co. develops accounting products for different customers like Non-Exist Tech. by reusing the assets and managing the variability of these assets specific to accounting domain. The company developed the product by using conventional techniques, but not based on a production line infrastructure.

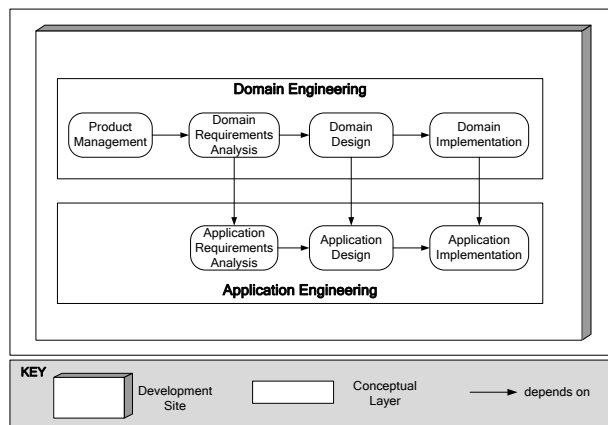


Figure 5. Local Single Product Line Development

D. Local Product Line Development with Production Line

A product line development can be realized on a production line platform. Hereby multiple variant products are developed based on set of sequential production units whereby components are assembled to make a finished article. Similar to the case for single system development with production line we could apply here the Pipes and Filters pattern.

Figure 6 shows an example of a local product line development with production line. Hereby, we have chosen for centralized control and coordination of the product line engineering activities (domain engineering and application engineering).

It appears that we could also have different interpretations and applications of local product line development with production line. For example, we could also apply production line engineering only for domain engineering, or only for application engineering.

Example:

John Doe Software Co. develops accounting products for different customers like Non-Exist Tech. by reusing the assets and managing the variability of these assets specific to accounting domain. The company developed the product by using its custom production line based on SpringSource. In this case, both the business domain specific assets and infrastructural modules are reused.

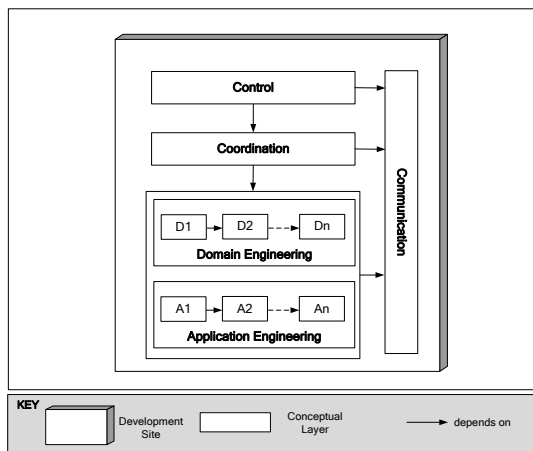


Figure 6. Local Product Line Development using a Production Line for both Domain Engineering and Application Engineering with Centralized Control and Coordination

E. Global Software Development with Single System Development

This section and the following three sections focus on defining the architecture design alternatives for GSE system in particular. We first consider GSE for single system development. We have defined the GSE with single system development alternative in Figure 7. Here the development of a single product is distributed over multiple sites (denoted by multiplicity 1..*).

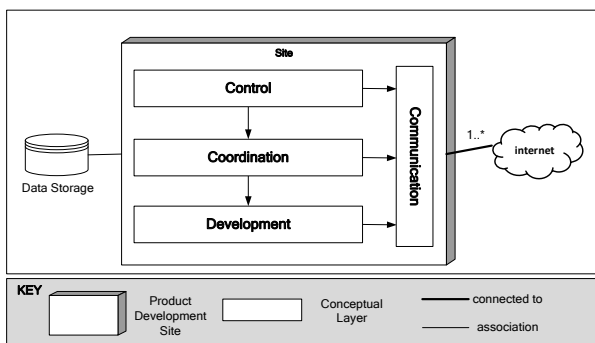


Figure 7. Global Software Development Single System Development

However, again we can observe here several sub-alternatives. These are defined basically due to the different application of the coordination and control mechanisms. In particular we can distinguish among the following alternatives as defined in Table 2.

TABLE 2. POSSIBLE ALTERNATIVE CONFIGURATIONS FOR CONTROL AND COORDINATION CONCERNS IN GSE

Alternative	Control	Coordination
1	Central	Central
2	Central	Distributed
3	Distributed	Central
4	Distributed	Distributed

A selection of one of the four alternatives will result in a refinement of the architecture in Figure 7. For example, Figure 8 shows the alternative with a central control and coordination, whereby development is distributed. Figure 9 defines an alternative with distributed control and central coordination. Of course not all the possible deployment alternatives might make sense. These should be validated from the requirements in practice.

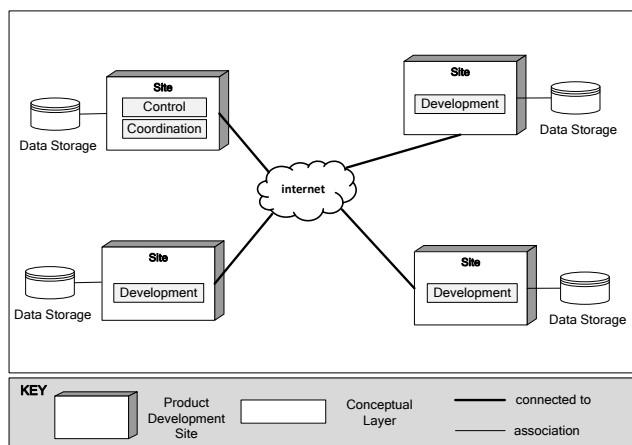


Figure 8. Deployment View of GSE with Single System Development using Central Control and Central Coordination

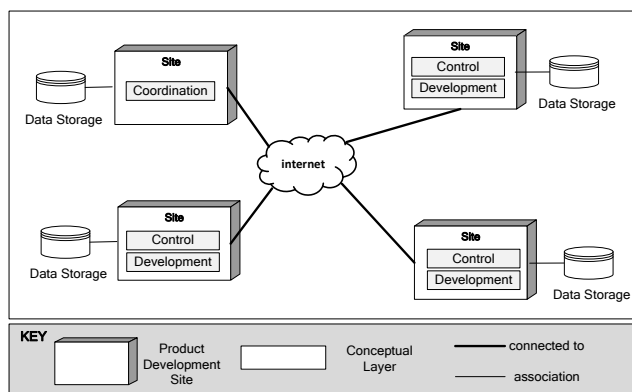


Figure 9. Deployment View of GSE with Single System Development using Distributed Control and Central Coordination

Example:

John Doe Software Co. distributes the development of an accounting system accustomed for Non-Exist Tech. Ltd. to different units all over the world. The company employed classical processes and approaches without having reuse insight for assets and infrastructural modules.

F. Global Single Software Development with Production Line

Figure 10 shows the case for global single software development with production line. Since GSE is used, the architecture will consist of multiple sites. The focus is on the development of a single system and as such the domain

engineering process is missing. However, the development is based on the production line paradigm.

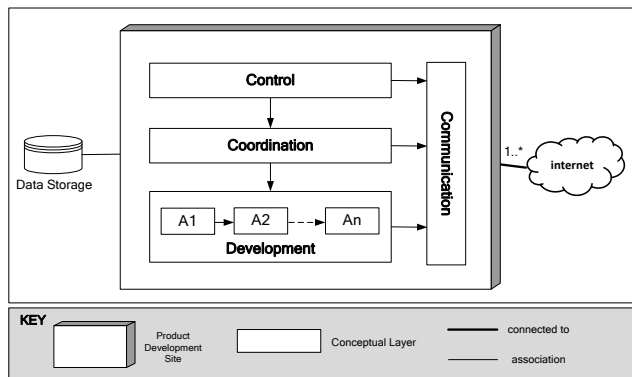


Figure 10. Global Software Development Single System Development with Production Line

Example

John Doe Software Co. has a custom distributed software production line based on SpringSource, which is installed at its business units all over the world to develop an accounting system accustomed for Non-Exist Tech. The company intentionally employed the production line to reuse infrastructural modules such as logging, content management, object to relational mapping, etc. within all its units.

G. Global Software Development for Product Line Engineering

Figure 11 represents the most difficult case for designing GSE system. It focuses on distributed development for a product line, in which the concept of production line is adopted.

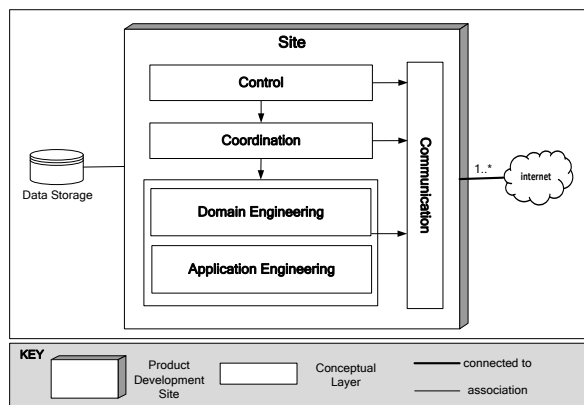


Figure 11. Global Software Development for Product Line Engineering

Example

John Doe Software Co. develops accounting products for different customers like Non-Exist Tech. by reusing the assets and managing the variability of these assets specific to accounting domain. The company distributed the development efforts of the product to different business

units all over the world by using classical techniques, but not based on a production line infrastructure.

H. Global Software Development for Product Line Engineering with Production Line Engineering

Figure 12 represents the most difficult case for designing GSE system. It focuses on distributed development for a product line, in which the concept of production line is adopted.

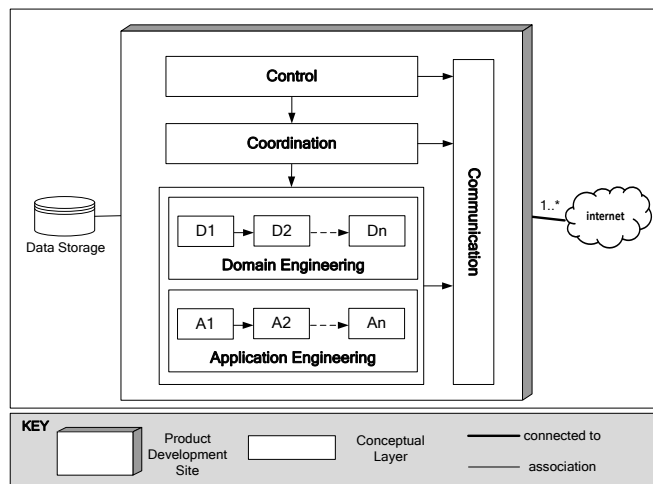


Figure 12. Global Software Development for Product Line Engineering with Production Line Engineering

Example:

John Doe Software Co. develops accounting products for different customers like Non-Exist Tech. by both reusing the assets and managing the variability of these assets specific to accounting domain. The company developed the product by using its custom distributed production line based on SpringSource, which can centrally control and monitor the whole development items and deliverables precisely. The production line based product variability management allows the reuse of business domain specific assets and infrastructural modules in a distributed way.

V. RELATED WORK

Notably, architecting in GSE has not been widely addressed. The key research focus in the GSE community seems to have been in particular related to tackling the problems related to communication, coordination and control concerns. Clerk et al. [4] report on the use of so-called *architectural rules* to tackle the GSE concerns. *Architectural rules* are defined as “principles and statements about the software architecture that must be complied with throughout the organization”. They have defined four challenges in GSE: time difference and geographical distance, culture, team communication and collaboration, and work distribution. For each of these challenges they list possible solutions and describe to what extent these solutions can be expressed as architectural rules. The work

of Clerk et al. aims to shed light on *what* kind of architectural rules are necessary to guide the GSE. We consider our work complementary to this work. In our work the design actions that relate to the expected answers of questions are defined as design actions.

In a position paper of Siemens, Paulish [10] provides some guidelines about how to develop a product line using a centralized product line management team and distributed component development teams. For this, the author proposes to decompose the large-scale requirements into a well-structured set of software components that can be developed in parallel among globally distributed development teams. Likewise it is aimed to develop the product line using global software engineering practices. Further it is recommended to keep small teams that use agile processes and which are controlled by a central organization. Further, the author describes some best practices for formal requirements engineering and architecture design to develop the software components that will make up the product line. Using the approach it is aimed to reduce the time-to-market and increase productivity. The architecture as proposed by Paulish is one of the alternatives that we have defined in Table 1. In fact, Paulish focuses more on the overall process for supporting product line engineering using global software engineering. In our approach we have focused on the architectural design of global software product line engineering. We believe that both approaches are complementary to each other.

A common practice is to model and document different architectural views for describing the architecture according to the stakeholders' concerns [6][9]. An architectural view is a representation of a set of system elements and relations associated with them to support a particular concern. Having multiple views helps to separate the concerns and as such support the modeling, understanding, communication and analysis of the software architecture for different stakeholders. Architectural views conform to viewpoints that represent the conventions for constructing and using a view. An architectural framework organizes and structures the proposed architectural viewpoints. Different architectural frameworks have been proposed in the literature. Examples of architectural frameworks include the Kruchten's 4+1 view model [9], the Siemens Four View Model and the Views and Beyond approach (V&B)[6]. In our work we have defined the architecture that represents the deployment view of the system. This view appeared to be one of the most useful views since it is able to depict the multi-site character of GSE. However, we could easily consider other views such as decomposition view or uses view. We consider this as part of our future work.

VI. CONCLUSIONS

We have defined the notion of *global software product line engineering* that considers the application of product line engineering in a global development environment. Our study shows that we can in essence identify 8 possible

integration alternatives of product line engineering with global software engineering. We have made a distinction between two global software product line engineering approaches: (1) GSPLE without production line and (2) GSPLE with production line. Obviously the latter GSPLE approach is the most difficult alternative but on the other hand will also lead to enhanced reuse.

The goal of this work was primarily to shed light on the challenges related to the architecture design of GSE system. The alternatives that we have shown can be used as templates for GSE architect to derive the architect for a particular project. Further, we consider this work as an initial step towards integrating product line with global software engineering. Our future work will focus on enhancing the concepts that we have discussed in this paper and applying this within an industrial context of Cybersoft.

REFERENCES

- [1] R.D. Battin, R. Crocker, J. Kreidler, K. Subramanian. Leveraging Resources in Global Software Development. IEEE Software, 18(2), p. 70-77, Mar/Apr, 2001.
- [2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture Volume 1 - A System of Patterns, Wiley, 1996.
- [3] E. Carmel and R. Agarwal. Tactical Approaches for Alleviating Distance in Global Software Development. IEEE Software, March/April, p. 22-29, 2001.
- [4] V. Clerc, P. Lago, H. van Vliet. *Global Software Development: Are Architectural Rules the Answer?* In: Proc. of the 2nd International Conference on Global Software Engineering, pp. 225-234. IEEE Computer Society Press, Los Alamitos, 2007.
- [5] P. Clements, L. Northrop. Software Product Lines: Practices and Patterns. Boston, MA: Addison-Wesley, 2002.
- [6] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. Documenting Software Architectures: Views and Beyond. Second Edition. Addison-Wesley, 2010.
- [7] J.D. Herbsleb. Global Software Engineering: The Future of Socio-technical Coordination. International Conference on Software Engineering. p. 188-198, 2007.
- [8] J.D. Herbsleb and D. Moitra. Global Software Development. IEEE Software, March/April, p. 16- 20, 2001.
- [9] P. Kruchten. The 4+1 View Model of Architecture. IEEE Software, 12(6):42-50, 1995.
- [10] D. Paulish, Product Line Engineering for Global Development, Siemens AG, pp. 1-6, 2005.
- [11] K. Pohl, G. Böckle, F. van der Linden. Software Product Line Engineering – Foundations, Principles, and Techniques, Springer, 2005.
- [12] SpringSource Tool Suite. <http://www.springsource.com/products/sts>
- [13] T. Stahl, M. Voelter. Model-Driven Software Development, Addison-Wesley, 2006.
- [14] K. Schmid, M. Verlage. The Economic Impact of Product Line Adoption and Evolution. IEEE Software, Vol. 19, No. 4, July/August 2002, 50-57.
- [15] B. Sengupta, S. Chandra, V. Sinha. A research agenda for distributed software development, In Proceedings of the 28th international conference on Software engineering, pp. 731-740, 2006.
- [16] J. Whitehead, *Collaboration in Software Engineering: A Roadmap*, In FOSE '07: 2007 Future of Software Engineering, pp. 214-225, 2007.
- [17] J.A. Zachman. A Framework for Information Systems Architecture. IBM Systems Journal, Vol. 26. No 3, pp. 276-292, 1987

Towards CMMI-compliant MDD Software Processes

Alexandre M. L. de Vasconcelos
 Centro de Informática
 Universidade Federal de Pernambuco
 Av. Jornalista Anibal Fernandes s/n, 50740-560
 Cidade Universitária, Recife-PE, Brazil
 e-mail: amlv@cin.ufpe.br

Giovanni Giachetti, Beatriz Marín, Oscar Pastor
 Centro de Investigación en Métodos de
 Producción de Software - PROS
 Universitat Politècnica de València
 Camino de Vera s/n, 46022, Valencia, Spain
 e-mail: {ggiachetti, bmarin, opastor}@pros.upv.es

Abstract — In the last years, Model-Driven Development (MDD) approaches have taken an important role in the quality improvement of software products. These approaches perform the automatic compilation of high-abstraction models to generate the final application code. In this way, MDD approaches aim at reducing development costs as well as increasing productivity, portability, interoperability, and ease of software evolution; i.e., achieving higher product quality. A major obstacle for MDD approaches to be massively adopted by industry is their lack of alignment to well-defined quality models for software processes. We advocate that performing a compliance analysis, based on a software process quality model, is the first step to deal with this obstacle. In this paper, we analyze the degree of compliance of an industrially applied MDD approach with the CMMI-DEV quality model. In particular, we determine those characteristics that meet the technical solution process area of CMMI-DEV and identify improvement opportunities to obtain a proper alignment of the MDD approach with this model.

Keywords – MDD; OO-Method; CMMI; Software Process Quality; Feature-based Analysis.

I. INTRODUCTION

Developing high quality software has been a continuous concern in the Software Engineering community. To achieve this goal, several software development approaches have emerged. In this context, the Model Driven Development (MDD) approach [1][3][4] has become subject of current research. The main idea behind MDD is the automatic generation of code from models through successive transformation of higher abstraction's level models (problem domain) into more concrete models (solution domain).

The MDD paradigm advocates that the initial software development and the implementation of future changes are all made in the model. In this way, MDD allows lower development costs, and higher productivity, portability, interoperability, and ease of software evolution [5]; i.e., higher software quality.

In parallel to the research on MDD and to the gradual adoption of this approach, many software development organizations are strongly seeking improvement and/or assessment of their software processes on the basis of quality models [6][7]. Such organizations aim to improve the efficiency of their processes and the quality of the products developed by the enactment of these processes as well as to

meet market and stakeholders needs. Hence, given the importance of software process quality models, MDD approaches must be compliant with these models to be widely used by software development organizations. Since this challenge has not been properly addressed by any MDD approach yet, further research into this direction is necessary. Thus, we propose the following research question: “*Is it possible to design a MDD process that fully complies with a well-defined software process quality model?*”

We advocate that MDD approaches must be analyzed with regard to software process quality models as a first step towards answering this research question. In this paper, we analyze the compliance of a specific MDD approach, named OO-Method [8], with the Technical Solution (TS) Process Area (PA – a cluster of related practices that, when implemented collectively, satisfies a set of goals for making improvements in an area) of the CMMI-DEV (Capability Maturity Model Integration for Development) software process quality model [9]. This analysis is performed by using an assessment method based on SCAMPI (Standard CMMI Appraisal Method for Process Improvement) [10]. As consequences of this analysis, certain weaknesses in the OO-Method approach have been indicated and adjustments have been proposed to make this MDD approach fully compliant with TS. Thus, the OO-Method approach can be integrated into a complete CMMI-based software process.

OO-Method and CMMI (across the entire paper CMMI and CMMI-DEV are being used as synonyms) have been chosen for the analysis because the former is a MDD approach that has been successfully applied in the software industry [11] and the latter is the most frequently adopted software process quality model [6][7]. TS has been chosen to be analyzed because the main objectives of this PA are the design and the implementation of information system's requirements, which are also the main objectives of MDD.

The contribution of this paper is twofold. First, practitioners can benefit from the analysis by adapting it to detect weaknesses on other MDD approaches in relation to a software process quality model. As a result, they can decide whether adopt a specific approach (although having to modify it) or discard it and adopt another one. Second, this type of analysis can be useful in academia for identifying room for improvement in existing or new MDD approaches, and therefore, for discovering further research areas. By using this analysis as reference, other MDD approaches can

be analyzed and improvements can be proposed for them so that their industrial acceptance could increase.

The rest of this paper is organized as follows. Section II presents background and related work. Section III describes the SCAMPI-based assessment method which was used in the compliance analysis. Section IV presents the compliance analysis of OO-Method regarding TS. Finally, Section V summarizes some conclusions and proposes further work.

II. BACKGROUND

In this section, we provide a brief explanation of OO-Method, CMMI-DEV, and some works related to this paper.

A. OO-Method

OO-Method [8] is an object-oriented method for conceptual modeling and automatic code generation that is supported by the industrial tool *Olivanova* [12]. It provides a precise UML-like notation, which is used to specify a *Conceptual Schema* that describes a system at the problem space level. The development process suggested by OO-Method has two phases (Fig. 1): *Development of a conceptual schema* and *Generation of a software product*.

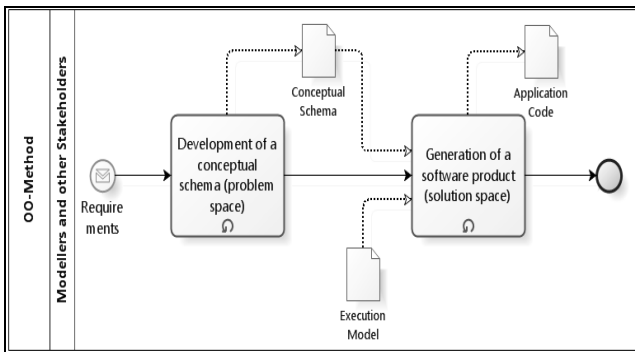


Figure 1. Phases and artifacts of the OO-Method MDD approach

The first phase consists of eliciting and representing the essential properties of the information system under study, thereby creating the corresponding conceptual schema. In the second phase, a precise execution model, conformed by a set of compilation patterns, indicates the correspondences between the conceptual schema and the pieces of code in a target implementation platform. Thus, the application code is automatically generated for an input conceptual schema.

B. CMMI-DEV

CMMI-DEV [10] is a guide to implement a continuous process improvement for developing products and services. For accomplishing this task, it provides two representations: **Continuous**, which assesses the capability level of individual Process Areas (PAs) that are selected based on the organization’s business goals; and **Staged**, which assesses the maturity level of a whole development process.

The compliance analysis presented in this paper focuses on the continuous representation, since only one PA is analyzed. In this representation, capability levels have goals and practices (decomposition of goals) of two types: **1)** Specific Goals (SGs) and Specific Practices (SPs), which are

applied only to a particular PA; and **2)** Generic Goals (GGs) and Generic Practices (GPs), which are applied equally to all PAs. From the assessment of practices and goals, it is possible to classify the capability level of a PA on a scale from 0 to 3 (unlike the previous versions of CMMI, the continuous representation no longer has capability levels 4 and 5). OO-Method will be assessed against level 1 because, according to CMMI-DEV, this level corresponds to the basis for improvement initiatives in a specific PA.

C. Related Work

Several authors have discussed the compliance of CMMI or CMM (the ancestor of CMMI) in relation to traditional or agile software development processes (e.g., [13][14][15], [16]). However, results obtained from these works are not completely useful to MDD approaches, which have characteristics that differentiate them from those other approaches [1][3][4][8]. For example, MDD approaches are mainly focused on: **1)** modeling rather than coding; **2)** implementing changes directly in the model rather than in the code; **3)** maintaining the model updated; **4)** synchronizing the model and the code; **5)** automatically verifying properties in the model; and **6)** automatically generating the complete code from models rather than using the model as a guide for manual code programming, or as a post-mortem code documentation.

In addition, at least one of the following problems can be found in the works related to those other software development approaches: **1)** non-use of the SCAMPI grades of satisfaction; **2)** lack of explicit/objective criteria for attributing grades; **3)** analysis based only on the activity descriptions, without requiring documental evidences or requiring only some of the evidences; **4)** analysis not in the same depth level as SCAMPI; **5)** lack of details about the rationale behind the analysis; or **6)** non-provision of solutions to fill in the gaps found in the analysis.

Although there are some specific works related to the compliance of MDD approaches with CMMI or CMM, they fail to deal with this issue properly. The works do not explain in detail how an approach complies with the quality model, where the approach should be adjusted for compliance, and whether/where the approach conflicts with the quality model requirements. The most relevant related works found are described as follows.

An engineering and management software process to support the achievement of CMM level 3 is proposed in [17]. The process uses MDA (a standard for MDD) [18] in the context of system families and CMM. However, an explicit mapping between the process and Key Process Areas (KPAs – a CMM concept that is equivalent to a PA of CMMI) is not presented, neither satisfaction grades are assigned.

The impacts in the software process and the main concerns to be dealt with when using MDD for the implementation of the CMM’s KPAs are discussed in [19]. However, the discussion is presented in a high abstraction level, without providing any explicit compliance mapping neither the attribution of grades to KPAs.

The MDD Maturity Model, which establishes five capability levels towards the progressive adoption of MDD

within an organization, is proposed in [20]. The authors argue that this model is compliant and complementary to CMMI staged. However, they do not present evidences to justify this affirmation.

The compliance between a software development process (based on the formal specification language CSP-CASL) with the PAs of requirements management, product integration, requirements development, technical solution, verification, and validation of CMMI is analyzed in [21]. The analysis concluded that, in general, it is possible to conciliate the development based on formal specifications with the requirements of CMMI. Grades are assigned for the respective SPs and SGs of each analyzed PA. However, the grades are not based on the SCAMPI criteria and evidences are not presented to justify all of the assigned grades.

The Model-Driven (software) Development Process (MDDP) is presented in [22]. It covers the software development stages from business processes through system requirements, analysis, and design models into test scripts and code. The authors argue that it can be used to comply with every level of CMMI staged. But, a grading scheme and an explicit mapping identifying the documental evidences that give support to this compliance are not presented.

In [23], a case study on the use of MDD to support the implementation of processes for the following PAs of CMMI is described: requirements management, technical solution, product integration, verification, and validation. The study concluded that the use of MDD helps, but is not sufficient to satisfy all requirements of those PAs. However, evidences to justify this conclusion are not presented.

Unlike those works, we present a detailed compliance mapping which uses a SCAMPI-based grading method. The mapping was produced with the help of experts on the MDD approach and a CMMI consultant/member of assessment teams [1]. These facts reinforce the mapping validity.

III. A SCAMPI-BASED ASSESSMENT METHOD

SCAMPI [10] is a method to objectively assess the development process of an organization according to the requirements of respective PAs of CMMI. It deals with the consolidation of evidences (e.g., presentations, documents and interviews) related to the execution of the process in actual projects. The evidences are used, by an assessment team, to support the attribution of grades to practices, goals and, finally, to the evaluated PAs.

Although SCAMPI-based analyses are usually performed using artifacts from actual projects, we defined an assessment method based on existing publications on OO-Method [8][11][12][24] in order to obtain results independent from any organizational context, and draw conclusions without influences from the environment in which the approach is used. Therefore, these results and conclusions can be generalized to any organization that uses OO-Method or similar MDD approaches.

Assessment based on publications can be seen as a feature-based analysis performed as part of a major evaluation scenario [25]. For instance, an organization that follows or plans to follow CMMI might analyze the possibility of adopting a MDD approach as part of its

development process; in this scenario, the organization can perform a feature-based analysis on each candidate MDD approach, and perform a preliminary selection (a subset of the candidate approaches) based on the analysis' results. Then, the selected approaches can be used on pilot projects to attest their effectiveness and to decide about the adoption.

The proposed SCAMPI-based assessment method uses the following types of evidences for the compliance analysis:

- Affirmations (*AFs*): statements described in the process that confirm or support implementation (or lack of implementation) of a practice as well as information obtained from experts in the approach.
- Artifacts (*ARs*): tangible evidences, mentioned in the process description, that are indicative of the work being performed and represent either the primary outputs of a model practice or a consequence of implementing a model practice.

The assessment is performed on a bottom-up way, from the practices up to the goals. Hence, for characterizing the level of implementation of a Specific Practice (SP) or Generic Practice (GP), the following grades are used:

- Fully Implemented (FI): *ARs* are present and judged to be adequate for demonstrating the practice implementation. No weaknesses are found.
- Largely Implemented (LI): *ARs* are present and judged to be adequate for demonstrating the practice implementation. However, one or more weaknesses are found.
- Partially Implemented (PI): some or all data required is absent or judged to be inadequate, some data provided (if exist) suggest that aspects of the practice are implemented, and one or more weaknesses are found; or the data supplied to the assessment team present conflicts, i.e., certain data indicate that the practice is implemented and other data indicate the practice is not implemented, and also, one or more weaknesses are noted.
- Not Implemented (NI): some or all data required is absent or judged to be inadequate, data supplied (if exist) do not support the conclusion that the practice is implemented, and one or more weaknesses are noted.

Based on the grades defined for a practice, each Specific Goal (SG) or Generic Goal (GG) is graded as:

- Satisfied: if and only if all corresponding practices are graded as either LI or FI, and the aggregation of weaknesses associated with the goal does not have a significant impact on the goal achievement; or,
- Unsatisfied: if at least one of the corresponding practices has a grade different from LI or FI.

Based on the grades defined for the goals (SGs and GGs), the capability level of a PA is defined. For instance, the capability level 1 has to satisfy the associated GG (hereafter called GG 1). Furthermore, GG 1 is "Satisfied" if all the SGs associated to the PA are graded as "Satisfied".

For capability levels higher than 1, a PA must satisfy the GG associated to the current level as well as all the GPs associated to the lower levels. The evaluation of the current GG is performed by applying the grading method defined previously to all the GPs associated to the GG (for instance, a process has capability level 2 for a specific PA if it satisfies all the GPs associated to the GG of capability level 2 and also satisfies GG 1). The application of this SCAMPI-based assessment method to OO-Method is shown in next section.

IV. COMPLIANCE ANALYSIS OF OO-METHOD WITH THE TECHNICAL SOLUTION PROCESS AREA

The purpose of TS is providing guidance for design, development and implementation of the given product requirements [9]. It focuses on evaluating and selecting a solution, developing a detailed design of the solution, and implementing the design as a product or product component.

The compliance analysis presented in this section uses an instance of the assessment method described in Section III. Based on publications about OO-Method, the CMMI expert carried out the analysis playing the role of an assessor. Then, experts on the MDD approach reviewed the analysis to identify possible flaws or misinterpretations. After that, discussions on the results were carried out by the experts (on CMMI and on the MDD approach) to validate the analysis.

The following subsections detail the results of the compliance analysis of OO-Method regarding the capability level 1 of TS process area. For each SG, the purpose of each corresponding SP is presented, the practice is mapped to *AFs* and *ARs*, and the SP is graded. After grading all the SPs of a SG, each SG is graded. Finally, a summary of the results is presented, where the whole PA is graded. Improvement suggestions are also discussed.

A. SG 1 Select Product Component Solutions

This specific goal includes SP1.1 and SP1.2.

SP 1.1 Develop Alternative Solutions and Selection Criteria

This SP identifies and analyzes alternative solutions to enable the selection of a balanced solution in terms of cost, schedule, performance, and risk. Selection criteria typically address costs (e.g., time, people, money), benefits (e.g., product performance, capability, effectiveness), and risks (e.g., technical, cost, schedule).

AFs: The application's architecture is defined based on a three-layer architectural pattern and restrictions on the selected technological platform, implementation language(s), and persistence service(s). Then, the conceptual model compilation is parameterized by the chosen architecture and the application's source code can be automatically generated [8].

ARs: Conceptual Schema, Execution Model, Application Code.

Grade: PI

OO-Method does not explicitly analyze alternative solutions prior to code generation (the architecture is usually defined by the application's developers jointly with the clients); neither defines criteria for the architecture selection.

However, if the source code generated is not adjusted to the quality requirements of a particular application, it can be regenerated for alternative platforms [8].

SP 1.2 Select Product Component Solutions

This SP selects the product component solutions based on selection criteria. Lower level requirements are generated from the selected architecture and used to develop product component designs. Interfaces among product components are described. The description of the solutions and the rationale for selection are documented.

AFs: Components identified during the phase "Development of a conceptual schema" are allocated to the architecture layers [8]. A documentation manager [26], which is part of the suite of tools to support OO-Method, automatically generates documentation from the conceptual schema, describing each component and its interface with other components.

ARs: Conceptual Schema, Execution Model, Application Code, Generated documentation, Documentation manager tool.

Grade: LI

OO-Method does not have an explicit artifact for documenting the selection decisions that are related to product component solutions, neither their rationale.

Conclusion

SP1.1 is graded as PI and SP1.2 is graded as LI. Therefore, SG 1 is graded as "Unsatisfied".

B. SG 2 Develop the Design

This specific goal includes SP2.1, SP2.2, SP2.3, and SP2.4.

SP 2.1 Design the Product or Product Components

This SP designs the product or its components in two phases, which can overlap in execution: preliminary (abstract) and detailed (concrete) design.

AFs: During the phase "Development of a conceptual schema", an abstract architecture is defined. A concrete architecture is defined during the phase "Generation of a software product", according to an execution model driven by restrictions on the selected technological platform, implementation language(s), and persistence service(s) [8].

ARs: Conceptual Schema, Execution Model, Application Code.

Grade: FI

No weak points have been identified.

SP 2.2 Establish a Technical Data Package

This SP records the design in a technical data package (a collection of items providing the developer a description of the product or product components) created during preliminary design.

AFs: A documentation manager is responsible for documenting the architecture definition from the conceptual schema created, and a repository

manager is responsible for creating and administrating a model library (including access control and management of model versions) [26].

ARs: Generated documentation, Data repository, Documentation manager tool, Repository manager tool, Conceptual Schema, Execution Model, Application Code.

Grade: FI

No weak points have been identified.

SP 2.3 Design Interfaces Using Criteria

This SP designs product component interfaces using established criteria (e.g., critical parameters that should be defined, or at least investigated, to ascertain their applicability).

AFs: OO-Method defines criteria for validating the Conceptual Schema in terms of correctness and completeness. The specification samples presented in part II of [8] illustrate the use of these criteria. Additional criteria for evaluating consistency, correctness and completeness are defined in [24].

ARs: Conceptual Schema, Validation criteria, Execution Model, Application Code.

Grade: FI

No weak points have been identified.

SP 2.4 Perform “Make, Buy, or Reuse” Analysis

This SP evaluates whether the product components should be developed, purchased, or reused based on established criteria.

AFs: Not identified.

ARs: Not identified.

Grade: NI

OO-Method does explicitly mention this analysis in its process; neither establishes criteria for performing it. However, the approach makes it possible to integrate developed components with pre-existing components and/or systems (legacy code).

Conclusion

SPs 2.1, 2.2 and 2.3 are graded as FI, and SP 2.4 is graded as NI. Therefore, SG 2 is graded as “Unsatisfied”.

C. SG 3 Implement the Product Design

This specific goal includes SP3.1, SP3.2, and SP3.3.

SP 3.1 Implement the Design

This SP implements the design of the product components and includes the allocation, refinement, and verification of each product component.

AFs: Once architecture is chosen, its code can be automatically generated. Prior to the source code generation, it is possible to have a conceptual model validation [8].

ARs: Conceptual Schema, Validation criteria, Execution Model, Application Code.

Grade: FI

No weak points have been identified.

SP 3.2 Develop Product Support Documentation

This SP develops and maintains documentation that will be used to install, operate, and maintain the product.

AFs: The documentation manager [26] automatically generates support and end-user documentation from the conceptual schema created.

ARs: Generated documentation, Documentation manager tool, Conceptual Schema, Application Code.

Grade: FI

No weak points have been identified.

Conclusion

SP 3.1 is graded as FI and SP 3.2 is graded as FI. Therefore, SG 3 is graded as “Satisfied”.

D. Summary of Assessment and Improvement Suggestions

As Table I summarizes, the OO-Method development process has capability level 0 with regard to TS.

TABLE I. OVERALL RESULTS OF OO-METHOD ASSESSMENT

Goals and practices of TS	Grades
SG 1 Select Product Component Solutions	Unsatisfied.
SG 2 Develop the Design	Unsatisfied.
SG 3 Implement the Product Design	Satisfied

In spite of this negative result, several convergence points have been pointed out, and most of the weak points found are easy to solve with the improvement suggestions described in Table II. In general, the improvements are simple adjustments in the development process, mainly related to explicit documentation of evidences.

TABLE II. IMPROVEMENT SUGGESTIONS FOR OO-METHOD

Improvements	Affected SPs
Extension of the development process to include an explicit analysis of alternative solutions prior to the code generation, as well as the explicit creation of a document defining criteria for the architecture selection.	SP 1.1
Explicit documentation of selection decisions (related to product component solutions) and their rationale.	SP 1.2
Explicit definition of criteria to perform “make, buy or reuse” analysis and the creation of an activity to explicitly perform this analysis prior to the code generation.	SP 2.4

Thus, by tailoring OO-Method with these improvements, it is possible to turn the grade of all SGs of TS into “Satisfied”. As a result, the development process of OO-Method can reach the capability level 1 for this PA. However, in order to confirm the effectiveness of the changes, the improvements should be implemented in a new version of OO-Method and the modified approach should be used in actual projects.

V. CONCLUSIONS AND FUTURE WORK

Even though this work has presented an analysis for CMMI and OO-Method, its purpose is to emphasize the need of analyzing the compliance of MDD approaches in relation to any software process quality model and to show how it can be addressed. Hence, the analysis can be adapted to other quality models [6][7] and to other MDD approaches [27].

In this paper, the compliance of OO-Method in relation to TS process area was analyzed. As a result, it was detected that this MDD approach does not sufficiently implement certain SPs. Hence, improvements were proposed aiming to fully implement the SPs for which weaknesses were found.

Some of the problems found in the analysis of OO-Method are also common to other approaches [27]. For instance, the lack of “make, buy, or reuse” analysis is a problem found in most of the approaches. Moreover, some of the features that were satisfied for OO-Method (e.g., the full code and document generations) are not presented in all of the other approaches [27].

This work is a starting point for several future works. We plan to perform a SCAMPI-based analysis of other MDD approaches for assessment and comparison; an assessment of the OO-Method development process applied to real projects must be addressed and an evaluation of the proposed improvements must be performed; the compliance with other PAs and capability levels of CMMI-DEV will be analyzed against OO-Method; and a systematic literature review [28] will be conducted to verify the existence of other works related to the compliance of MDD and software process quality models. All of these works are part of a research agenda towards the development of a complete MDD-based software process compliant with CMMI.

ACKNOWLEDGMENT

This work has been developed with the support of the Brazilian Research Agency CAPES, under the grant #BEX3229/10-6, and the Spanish Government, under the projects ORCA (PROMETEO/2009/15) and PROS-REQ (TIN2010-19130-C02-02).

REFERENCES

- [1] A. Vasconcelos, “Curriculum Vitae”, in Portuguese, <http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K4787134E7>, Last access 2011/07/18
- [2] B. Selic, “The Pragmatics of Model-Driven Development,” *IEEE Software*, vol. 20(5), 2003, pp. 19-25
- [3] D. Schmidt, “Model-Driven Engineering,” *IEEE Computer*, vol. 39(2), 2006, pp. 25-31
- [4] M. Völter, “Model-Driven Software Development – Technology, Engineering, Management,” Wiley, 2007
- [5] P. Mohagheghi and V. Dehlen, “Where is the Proof? – A Review of Experiences from Applying MDE in Industry,” *LNCS*, Springer, 2008, vol. 5095, pp. 432-443
- [6] F. J. Pino, F. García, and M. Piattini, “Software process improvement in small and medium software enterprises: a systematic review,” *Software Quality Journal*, vol. 16(2), 2008, pp. 237-261
- [7] M. Unterkalmsteiner, T. Gorschek, A. Islam, C. Cheng, R. Permadi, and R. Feldt, “Evaluation and Measurement of Software Process Improvement - A Systematic Literature Review,” *IEEE-TSE*, 2011
- [8] O. Pastor and J. C. Molina, *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*, 1st edn. Springer, New York, 2007
- [9] SEI, “CMMI for Development, Version 1.3,” CMU/SEI-2010-TR-033, <http://www.sei.cmu.edu>, 2010, Last access 2011/07/18
- [10] SEI, “Standard CMMI Appraisal Method for Process Improvement (SCAMPI) A, Version 1.3: Method Definition Document”, CMU/SEI-2011-HB-001, 2011, <http://www.sei.cmu.edu>, 2010, Last access 2011/07/18
- [11] PROS, Model Driven Development and Automatic Code Generation, <http://www.pros.upv.es/index.php/en/lineas/69-lineadmm>, Last access 2011/07/18
- [12] O. Pastor, J. C. Molina, and E. Iborra, “Automated production of fully functional applications with OlivaNova Model Execution”, *ERCIM News*, n° 57, 2004
- [13] L. V. Manzoni and R. T. Price, “Identifying Extensions Required by RUP to Comply with CMM Levels 2 and 3,” *IEEE TSE*, vol. 29(2), 2003, pp. 181-192
- [14] C. Santana, C. Gusmão, A. Rouiller, and A. Vasconcelos, “Achieving Software Quality Certifications through Agile Software Development,” *Internat. Journal of Advanced Manufacturing Systems*, vol. 11(1), 2008, pp. 1-6
- [15] J. Smith, “Reaching CMM Level 2 and 3 with the Rational Unified Process – White Paper,” <http://www.wthreex.com/rup/portugues/papers/pdf/rupcmm.pdf>, 2000, Last access 2011/07/18
- [16] J. Diaz, J. Garbajosa, and J. A. Calvo-Manzano, “Mapping CMMI Level 2 to Scrum Practices: An Experience Report,” *Proc. EuroSPI 2009*, CCIS, Springer, 2009, vol. 42, pp. 93-104
- [17] ESI, “Model-driven Architecture inSTRumentation, Enhancement and Refinement,” IST-2001-34600, MASTER-2003-D3.2-V1.0-PUBLIC, 2003
- [18] OMG, “Model Driven Architecture (MDA) Guide Version 1.0.1,” <http://www.omg.org/mda/>, 2003, Last access 2011/07/18
- [19] R. Steinhau, et al., “Guidelines for the Application of MDA and the Technologies covered by it,” Deliverable 3.2, MODA-TEL Consortium, IST-2001-37785, Interactive Objects Software GmbH, 2003
- [20] E. Rios, T. Bozheva, A. Bediaga, and N. Guilloreau, “MDD Maturity Model: A Roadmap for Introducing Model-Driven Development. Model Driven Architecture – Foundations and Applications,” *LNCS*, Springer, 2006, vol. 4066, pp. 78-89
- [21] S. Mishra and B. Schlingloff, “Compliance of CMMI Process Area with Specification Based Development,” *Proc. VI Internat. Conf. on Softw. Engineering Research, Management and Applications*, IEEE Computer Society, 2008, pp. 77-84
- [22] Crag Systems, The Model-Driven Development Process, http://www.cragssystems.co.uk/development_process, 2008, Last access 2011/07/18
- [23] S. Fricker, “Introducing Model-Driven Development for CMMI Engineering Process Areas,” *SEPG 2006*, <http://www.secc.org.eg/sepg%202006/ingredients/Indexes/aut horindex.html#f>, 2006, Last access 2011/07/18
- [24] B. Marín, G. Giachetti O. Pastor, and A. Abran, “A Quality Model for Conceptual Models of MDD Environments” *Advances in Software Engineering*, (Special Issue: New Generation of Software Metrics), 2010
- [25] B. Kitchenham, “DESMET: A Method for Evaluating Software Engineering Methods and Tools,” TR96-09, Department of Computer Science, University of Keele, 1996
- [26] CARE-Technologies Web Page, <http://www.care-t.com>, 2011/07/18
- [27] J. Estefan, “Survey of Model-Based Systems Engineering Methodologies – Rev. B,” Technical Report, INCOSE MBSE Initiative Focus Group, 2008
- [28] B. Kitchenham and S. Charters, “Guidelines for performing Systematic Literature Reviews in Software Engineering Version 2.3,” Keele University and Durham University Joint Technical Report EBSE-2007-01, 2007

From Boolean Relations to Control Software

Federico Mari, Igor Melatti, Ivano Salvo, and Enrico Tronci

Department of Computer Science

Sapienza University of Rome

Via Salaria 113, 00198 Rome, Italy

Email: {mari,melatti,salvo,tronci}@di.uniroma1.it

Abstract—Many software as well digital hardware automatic synthesis methods define the set of implementations meeting the given system specifications with a boolean relation K . In such a context a fundamental step in the software (hardware) synthesis process is finding effective solutions to the functional equation defined by K . This entails finding a (set of) boolean function(s) F (typically represented using OBDDs, *Ordered Binary Decision Diagrams*) such that: 1) for all x for which K is satisfiable, $K(x, F(x)) = 1$ holds; 2) the implementation of F is efficient with respect to given implementation parameters such as code size or execution time. While this problem has been widely studied in digital hardware synthesis, little has been done in a software synthesis context. Unfortunately the approaches developed for hardware synthesis cannot be directly used in a software context. This motivates investigation of effective methods to solve the above problem when F has to be implemented with software. In this paper, we present an algorithm that, from an OBDD representation for K , generates a C code implementation for F that has the same size as the OBDD for F and a worst case execution time linear in nr , being $n = |x|$ the number of input arguments for functions in F and r the number of functions in F .

Keywords-Control Software Synthesis; Embedded Systems; Model Checking

I. INTRODUCTION

Many software as well digital hardware automatic synthesis methods define the set of implementations meeting the given system specifications with a boolean relation K . Such relation typically takes as input (the n -bits encoding of) a state x of the system and (the r -bits encoding of) a proposed action to be performed u , and returns *true* (i.e., 1) iff the system specifications are met when performing action u in state x . In such a context a fundamental step in the software (hardware) synthesis process is finding effective solutions to the functional equation defined by K , i.e., $K(x, u) = 1$. This entails finding a tuple of boolean functions $F = \langle f_1, \dots, f_r \rangle$ (typically represented using OBDDs, *Ordered Binary Decision Diagrams* [1]) s.t. 1) for all x for which K is satisfiable (i.e., it enables at least one action), $K(x, F(x)) = 1$ holds, and 2) the implementation of F is efficient with respect to given implementation parameters such as code size or execution time.

While this problem has been widely studied in digital hardware synthesis [2][3], little has been done in a software synthesis context. This is not surprising since software

synthesis from formal specifications is still in its infancy. Unfortunately the approaches developed for hardware synthesis cannot be directly used in a software context. In fact, synthesis methods targeting a hardware implementation typically aim at minimizing the number of digital gates and of hierarchy levels. Since in the same hierarchy level gates output computation is *parallel*, the hardware implementation WCET (*Worst Case Execution Time*) is given by the number of levels. On the other hand, a software implementation will have to *sequentially* compute the gates outputs. This implies that the software implementation WCET is the number of gates used, while a synthesis method targeting a software implementation may obtain a better WCET. This motivates investigation of effective methods to solve the above problem when F has to be implemented with software.

In this paper we present an algorithm that, from an OBDD representation for K , effectively generates a C code implementation for K that has the same size as the OBDD for F and a WCET linear in linear in nr , being $n = |x|$ the size of states encoding and $r = |u|$ the size of actions encoding. This allows us to synthesize correct-by-construction *control software*, provided that K is provably correct w.r.t. initial formal specifications. This is the case of [4], where an algorithm to synthesize K starting from the formal specification of a Discrete-Time Linear Hybrid System (*DTLHS* in the following) is presented. Thus this methodology allows a correct-by-construction control software to be synthesized, starting from formal specifications for DTLHSs.

Note that the problem of solving the functional equation $K(x, F(x)) = 1$ w.r.t. F is trivially decidable, since there are finitely many F . However, trying to explicitly enumerate all F requires time $\Omega(2^{r2^n})$ (being n the number of bits encoding state x and r the number of bits encoding state u). By using OBDD-based computations, our algorithm complexity is $O(r2^n)$ in the worst case. However, in many interesting cases OBDD sizes and computations are much lower than the theoretical worst case (e.g., in Model Checking applications, see [5]).

Furthermore, once the OBDD representation for F has been computed, a trivial implementation of F could use a look-up table in RAM. While this solution would yield a better WCET, it would imply a $\Omega(r2^n)$ RAM usage. Unfortunately, implementations for F in real-world cases are

typically implemented on microcontrollers (this is the case, e.g., for *embedded systems*). Since microcontrollers usually have a small RAM, the look-up table based solution is not feasible in many interesting cases. The approach we present here will rely on OBDDs compression to overcome such obstruction.

Moreover, $F : \mathbb{B}^n \rightarrow \mathbb{B}^r$ is composed by r boolean functions, thus it is represented by r OBDDs. Such OBDDs typically share nodes among them. If a trivial implementation of F in C code is used, i.e., each OBDD is translated as a stand-alone C function, OBDDs nodes sharing will not be exploited. In our approach, we also exploit nodes sharing, thus the control software we generate fully takes advantage of OBDDs compression.

Finally, we present experimental results showing effectiveness of the proposed algorithm. As an example, in less than 1 second and within 70 MB of RAM we are able to synthesize the control software for a function K of 24 boolean variables, divided in $n = 20$ state variables and $r = 4$ action variables, represented by a OBDD with about 4×10^4 nodes. Such K represents the set of correct implementations for a real-world system, namely a multi-input buck DC/DC converter [6], obtained as described in [4]. The control software we synthesize in such a case has about 1.2×10^4 lines of code, while a control software not taking into account OBDDs nodes sharing would have had about 1.5×10^4 lines of code. Thus, we obtain a 24% gain towards a trivial implementation.

This paper is organized as follows. In Section III we give the basic notions to understand our approach. In Section IV we formally define the problem we want to solve. In Section V we give definition and main properties of COBDDs (i.e., *Complemented edges OBDDs*), on which our approach is based. Section VI describes the algorithms our approach consists of. Finally, Section VII presents experimental results showing effectiveness of the proposed approach.

II. RELATED WORK

Synthesis of boolean functions F satisfying a given boolean relation K in a way s.t. $K(x, F(x)) = 1$ is also addressed in [2]. However, [2] targets a hardware setting, whereas we are interested in a software implementation for F . Due to structural differences between hardware and software based implementations (see the discussion in Section I), the method in [2] is not directly applicable here. An OBDD-based method for synthesis of boolean (reversible) functions is presented in [3] (see also citations thereof). Again, the method in [3] targets a hardware implementation, thus it is not applicable here.

In [4], an algorithm is presented which, starting from formal specifications of a DTLHS, synthesizes a correct-by-construction boolean relation K , and then a correct-by-construction control software implementation for K . However, in [4] the implementation of K is not described in

detail. Furthermore, the implementation synthesis described in [4] has not the same size of the OBDD for F , i.e., it does not exploit OBDD node sharing.

In [7], an algorithm is presented which computes boolean functions F satisfying a given boolean relation K in a way s.t. $K(x, F(x)) = 1$. This approach is very similar to ours. However [7] does not generate the C code control software and it does not exploit OBDD node sharing.

Therefore, to the best of our knowledge this is the first time that an algorithm synthesizing correct-by-construction control software starting from a boolean relation (with the characteristics given in Section I) is presented.

III. BASIC DEFINITIONS

In the following, we denote with $\mathbb{B} = \{0, 1\}$ the boolean domain, where 0 stands for *false* and 1 for *true*. We will denote boolean functions $f : \mathbb{B}^n \rightarrow \mathbb{B}$ with boolean expressions on boolean variables involving $+$ (logical OR), \cdot (logical AND, usually omitted thus $xy = x \cdot y$), $\bar{}$ (logical complementation) and \oplus (logical XOR). We will also denote vectors of boolean variables in boldface, e.g., $\mathbf{x} = \langle x_1, \dots, x_n \rangle$. Moreover, we also denote with $f|_{x_i=g}(\mathbf{x})$ the boolean function $f(x_1, \dots, x_{i-1}, g(\mathbf{x}), x_{i+1}, \dots, x_n)$ and with $\exists x_i f(\mathbf{x})$ the boolean function $f|_{x_i=0}(\mathbf{x}) + f|_{x_i=1}(\mathbf{x})$.

Finally, we denote with $[n]$ the set $\{1, \dots, n\}$.

1) *Most General Optimal Controllers: A Labeled Transition System (LTS)* is a tuple $\mathcal{S} = (S, A, T)$ where S is a finite set of states, A is a finite set of actions, and T is the (possibly non-deterministic) transition relation of \mathcal{S} . A controller for an LTS \mathcal{S} is a function $K : S \times A \rightarrow \mathbb{B}$ enabling actions in a given state. We denote with $\text{Dom}(K)$ the set of states for which a control action is enabled. An LTS control problem is a triple $\mathcal{P} = (S, I, G)$, where \mathcal{S} is an LTS and $I, G \subseteq S$. A controller K for \mathcal{S} is a strong solution to \mathcal{P} iff it drives each initial state $s \in I$ in a goal state $t \in G$, notwithstanding nondeterminism of \mathcal{S} . A strong solution K^* to \mathcal{P} is optimal iff it minimizes path lengths. An optimal strong solution K^* to \mathcal{P} is the most general optimal controller (we call such solution an *mgo*) iff in each state it enables all actions enabled by other optimal controllers. For more formal definitions of such concepts, see [8].

Efficient algorithms to compute mgos starting from suitable (nondeterministic) LTSs have been proposed in the literature (e.g., see [9]). Once an mgo K has been computed, solving and implementing the functional equation $K(\mathbf{x}, \mathbf{u}) = 1$ allows a correct-by-construction control software to be synthesized.

2) *OBDD Representation for Boolean Functions: A Binary Decision Diagram (BDD)* R is a rooted directed acyclic graph (DAG) with the following properties. Each R node v is labeled either with a boolean variable $\text{var}(v)$ (internal node) or with a boolean constant $\text{val}(v) \in \mathbb{B}$ (terminal node). Each R internal node v has exactly two children, labeled with $\text{high}(v)$ and $\text{low}(v)$. Let x_1, \dots, x_n be the boolean

variables labeling R internal nodes. Each terminal node v represents $f_v(\mathbf{x}) = \text{val}(v)$. Each internal node v represents $f_v(\mathbf{x}) = x_i f_{\text{high}(v)}(\mathbf{x}) + \bar{x}_i f_{\text{low}(v)}(\mathbf{x})$, being $x_i = \text{var}(v)$. An *Ordered BDD* (OBDD) is a BDD where, on each path from the root to a terminal node, the variables labeling each internal node must follow the same ordering.

IV. SOLVING A BOOLEAN FUNCTIONAL EQUATION

Let $K(x_1, \dots, x_n, u_1, \dots, u_r)$ be the mgo for a given control problem $\mathcal{P} = (\mathcal{S}, I, G)$. We want to solve the *boolean functional equation* $K(\mathbf{x}, \mathbf{u}) = 1$ w.r.t. variables \mathbf{u} , that is we want to obtain boolean functions f_1, \dots, f_r s.t. $K(\mathbf{x}, f_1(\mathbf{x}), \dots, f_r(\mathbf{x})) = K|_{u_1=f_1(\mathbf{x}), \dots, u_r=f_r(\mathbf{x})}(\mathbf{x}, \mathbf{u}) = 1$. This problem may be solved in different ways, depending on the *target implementation* (hardware or software) for functions f_i . In both cases, it is crucial to be able to bound the WCET (*Worst Case Execution Time*) of the obtained controller. In fact, controllers must work in an endless closed loop with the system \mathcal{S} (*plant*) they control. This implies that, every T seconds (*sampling time*), the controller has to decide the actions to be sent to \mathcal{S} . Thus, in order for the entire system (plant + control software) to properly work, the controller WCET upper bound must be at most T .

In [2], f_1, \dots, f_r are generated in order to optimize a *hardware* implementation. In this paper, we focus on software implementations for f_i (*control software*). As it is discussed in Section I, simply translating an hardware implementation into a software implementation would result in a too high WCET. Thus, a method directly targeting software is needed. An easy solution would be to set up, for a given state \mathbf{x} , a SAT problem instance $\mathcal{C} = C_{K1}, \dots, C_{Kt}, c_1, \dots, c_n$, where $C_{K1} \wedge \dots \wedge C_{Kt}$ is equisatisfiable to K and each clause c_i is either x_i (if x_i is 1) or \bar{x}_i (otherwise). Then \mathcal{C} may be solved using a SAT solver, and the values assigned to \mathbf{u} in the computed satisfying assignment may be returned as the action to be taken. However, it would be hard to estimate a WCET for such an implementation. The method we propose in this paper overcomes such obstructions by achieving a WCET proportional to rn .

V. OBDDS WITH COMPLEMENTED EDGES

In this section, we introduce OBDDs with complemented edges (COBDDs, Definition 1), which were first presented in [10][11]. Intuitively, they are OBDDs where else edges (i.e., edges of type $(v, \text{low}(v))$) may be complemented. Then edges (i.e., edges of type $(v, \text{high}(v))$) complementation is not allowed to retain canonicity. Edge complementation usually reduce resources usage, both in terms of CPU and memory.

Definition 1. An *OBDD with complemented edges* (COBDD in the following) is a tuple $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ with the following properties: i) $\mathcal{V} = \{x_1, \dots, x_n\}$ is a finite set of *ordered* boolean variables; ii) V is a

finite set of *nodes*; iii) $\mathbf{1} \in V$ is the *terminal* node of ρ , corresponding to the boolean constant 1 (non-terminal nodes are called *internal*); iv) for each internal node v , $\text{var}(v) < \text{var}(\text{high}(v))$ and $\text{var}(v) < \text{var}(\text{low}(v))$; v) $\text{var}, \text{low}, \text{high}, \text{flip}$ are functions defined on internal nodes, namely: $\text{var} : V \setminus \{\mathbf{1}\} \rightarrow \mathcal{V}$ assigns to each internal node a boolean variable in \mathcal{V} , $\text{high}[\text{low}] : V \setminus \{\mathbf{1}\} \rightarrow V$ assigns to each internal node v a *high child* [*low child*] (or *true child* [*else child*]), representing the case in which $\text{var}(v) = 1$ [$\text{var}(v) = 0$], $\text{flip} : V \setminus \{\mathbf{1}\} \rightarrow \mathbb{B}$ assigns to each internal node v a boolean value; namely, if $\text{flip}(v) = 1$ then the else child has to be complemented, otherwise it is regular (i.e., non-complemented).

COBDDs associated multigraphs: We associate to a COBDD $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ a labeled directed multigraph $G^{(\rho)} = (V, E)$ s.t. V is the same set of nodes of ρ and there is an edge $(v, w) \in E$ iff w is a child of v . Moreover, each edge $e \in E$ has a type $\text{type}(e)$, indicating if e is a then, a regular else, or a complemented else edge. Figure 1 shows an example of a COBDD depicted via its associated multigraph, where edges are directed downwards. Moreover, in Figure 1 then edges are solid lines, regular else edges are dashed lines and complemented else edges are dotted lines.

The graph associated to a given COBDD $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ may be seen as a forest with multiple rooted multigraphs. In order to select one root vertex and thus one rooted multigraph, we define the *COBDD restricted to* $v \in V$ as the COBDD $\rho_v = (\mathcal{V}, V_v, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ s.t. $V_v = \{w \in V \mid \text{there exists a path from } v \text{ to } w \text{ in } G^{(\rho)}\}$ (note that $v \in V_v$).

Reduced COBDDs: Two COBDDs are *isomorphic* iff there exists a mapping from nodes to nodes preserving attributes $\text{var}, \text{flip}, \text{high}$ and low . A COBDD is called *reduced* iff it contains no vertex v with $\text{low}(v) = \text{high}(v) \wedge \text{flip}(v) = 0$, nor does it contains distinct vertices v and v' such that ρ_v and $\rho_{v'}$ are isomorphic. Note that, differently from OBDDs, it is possible that $\text{high}(v) = \text{low}(v)$ for some $v \in V$, provided that $\text{flip}(v) = 1$ (e.g., see nodes 0xf and 0xe in Figure 1). In the following, we assume all our COBDDs to be reduced.

COBDDs properties: For a given COBDD $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ the following properties follow from definitions given above: i) $G^{(\rho)}$ is a rooted directed acyclic (multi)graph (DAG); ii) each path in $G^{(\rho)}$ starting from an internal node ends in $\mathbf{1}$; iii) let v_1, \dots, v_k be a path in $G^{(\rho)}$, then $\text{var}(v_1) < \dots < \text{var}(v_k)$.

A. Semantics of a COBDD

In Definition 2, we define the semantics $\llbracket \cdot \rrbracket$ of each node v of a given COBDD ρ as the boolean function represented by v , given the parity b of complemented edges seen on the path from a root to v .

Definition 2. Let $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD. The semantics of the terminal node $\mathbf{1}$ w.r.t. a flipping bit b is a boolean function defined as $\llbracket \mathbf{1}, b \rrbracket_\rho := \bar{b}$. The semantics of an internal node $v \in V$ w.r.t. a flipping bit b is a boolean function defined as $\llbracket v, b \rrbracket_\rho := x_i \llbracket \text{high}(v), b \rrbracket_\rho + \bar{x}_i \llbracket \text{low}(v), b \oplus \text{flip}(v) \rrbracket_\rho$, being $x_i = \text{var}(v)$. When ρ is understood, we will write $\llbracket \cdot \rrbracket$ instead of $\llbracket \cdot \rrbracket_\rho$.

Example 1. Let ρ be the COBDD depicted in Figure 1. If we pick node $0xe$ we have $\llbracket 0xe, b \rrbracket = x_2 \llbracket \mathbf{1}, b \rrbracket + \bar{x}_2 \llbracket \mathbf{1}, b \oplus \mathbf{1} \rrbracket = x_2 \bar{b} + \bar{x}_2 b = x_2 \oplus b$.

Theor. 1 states that COBDDs are a canonical representation for boolean functions (see [10][11]).

Theorem 1. Let $f : \mathbb{B}^n \rightarrow \mathbb{B}$ be a boolean function. Then there exist a COBDD $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$, a node $v \in V$ and a flipping bit $b \in \mathbb{B}$ s.t. $\llbracket v, b \rrbracket = f(x)$. Moreover, let $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD, let $v_1, v_2 \in V$ be nodes and $b_1, b_2 \in \mathbb{B}$ be flipping bits. Then $\llbracket v_1, b_1 \rrbracket = \llbracket v_2, b_2 \rrbracket$ iff $v_1 = v_2 \wedge b_1 = b_2$.

VI. SYNTHESIS OF C CODE FROM A COBDD

Let $K(x_1, \dots, x_n, u_1, \dots, u_r)$ be the mgo for a given control problem. Let $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD s.t. there exist $v \in V$, $b \in \mathbb{B}$ s.t. $\llbracket v, b \rrbracket = K(x_1, \dots, x_n, u_1, \dots, u_r)$. Thus, $\mathcal{V} = \mathcal{X} \cup \mathcal{U} = \{x_1, \dots, x_n\} \cup \{u_1, \dots, u_r\}$ (we denote with \cup the disjoint union operator, thus $\mathcal{X} \cap \mathcal{U} = \emptyset$). We will call variables $x_i \in \mathcal{X}$ as *state variables* and variables $u_j \in \mathcal{U}$ as *action variables*. More in-depth details may be found in [8].

A. Synthesis Algorithm: Overview

Our method *Synthesize* takes as input ρ , v and b s.t. $\llbracket v, b \rrbracket = K(x, u)$. Then, it returns as output a C function `void K(int *x, int *u)` with the following property: if, before a call to `K`, $\forall i \ x[i-1] = x_i$ holds (array indexes in C language begin from 0) with $x \in \text{Dom}(K)$, and after the call to `K`, $\forall i \ u[i-1] = u_i$ holds, then $K(x, u) = 1$. Moreover, the WCET of function `K` is $O(nr)$.

Note that our method *Synthesize* provides an effective implementation of the mgo K , i.e., a C function which takes as input the current state of the LTS and outputs the action to be taken. Thus, `K` is indeed a control software.

Function *Synthesize* is organized in two phases. First, starting from ρ , v and b (thus from $K(x, u)$), we generate COBDD nodes v_1, \dots, v_r and flipping bits b_1, \dots, b_r for boolean functions f_1, \dots, f_r s.t. each $f_i = \llbracket v_i, b_i \rrbracket$ takes as input the state bit vector x and computes the i -th bit u_i of an output action bit vector u , where $K(x, u) = 1$, provided that $x \in \text{Dom}(K)$. This computation is carried out in function *SolveFunctionalEq*. Second, f_1, \dots, f_r are translated inside function `void K(int *x, int *u)`. This step is performed by maintaining the structure of the COBDD nodes representing f_1, \dots, f_r . This allows us to

exploit COBDD node sharing in the generated software. This phase is performed by function *GenerateCCode*.

Thus function *Synthesize* is organized as in Algorithm 1. Correctness for function *Synthesize* is stated in Theor. 2.

Algorithm 1 Translating COBDDs to a C function

Require: COBDD ρ , node v , boolean b

Ensure: *Synthesize*(ρ, v, b):

- 1: $\langle v_1, b_1, \dots, v_r, b_r \rangle \leftarrow \text{SolveFunctionalEq}(\rho, v, b)$
 - 2: *GenerateCCode*($\rho, v_1, b_1, \dots, v_r, b_r$)
-

B. Synthesis Algorithm: Solving a Functional Equation

In this phase, starting from ρ , v and b (thus from $\llbracket v, b \rrbracket = K(x, u)$), we compute functions f_1, \dots, f_r s.t. for all $x \in \text{Dom}(K)$, $K(x, f_1(x), \dots, f_r(x)) = 1$.

To this aim, we follow an approach similar to the one presented in [7]. Namely, we compute f_i using f_1, \dots, f_{i-1} , in the following way: $f_i(x) = \exists u_{i+1}, \dots, u_n \ K(x, f_1(x), \dots, f_{i-1}(x), 1, u_{i+1}, \dots, u_n)$. Thus, function *SolveFunctionalEq*(ρ, v, b) computes and returns $\langle v_1, b_1, \dots, v_r, b_r \rangle$ s.t. for all $i \in [r]$, $\llbracket v_i, b_i \rrbracket = f_i(x)$.

C. Synthesis Algorithm: Generating C Code

In this phase, starting from COBDD nodes v_1, \dots, v_r and flipping bits b_1, \dots, b_r for functions f_1, \dots, f_r generated in the first phase, we generate two C functions: i) `void K(int *x, int *u)`, which is the required output function for our method *Synthesize*; ii) `int K_bits(int *x, int action)`, which is an auxiliary function called by `K`. A call to `K_bits(x, i)` returns $f_i(x)$, being $x[j-1] = x_j$ for all $j \in [n]$. This phase is detailed in Algs. 2 (function *GenerateCCode*) and 3 (function *Translate*).

Given inputs $\rho, v_1, b_1, \dots, v_r, b_r$ (output by *SolveFunctionalEq*), Algs. 2 and 3 work as follows. First, function `int K_bits(int *x, int action)` is generated. If $x[j-1] = x_j$ for all $j \in [n]$, the call `K_bits(x, i)` has to return $f_i(x)$. In order to do this, `K_bits(x, i)` traverses the graph $G^{(\rho_{v_i})}$ by taking, in each node v , the then edge if $x[j-1] = 1$ (with j s.t. $\text{var}(v) = x_j$) and the else edge otherwise. When node $\mathbf{1}$ is reached, then $\mathbf{1}$ is returned iff the integer sum $c + b_i$ is even, being c the number of complemented else edges traversed. Parity of $c + b_i$ is maintained by initializing a C variable `ret_b` to \bar{b}_i , then complementing `ret_b` when a complemented else edge is traversed, and finally returning `ret_b`.

Thus, Algs. 2 and 3 generate `K_bits` in order to obtain the above described behavior. Namely, for all v_i output by the first phase (function *SolveFunctionalEq*), *GenerateCCode* calls *Translate* with parameters ρ, v_i, W , where W maintains the set of nodes already translated in C code. This results, for all such v_i , in a recursive graph traversal of $G^{(\rho_{v_i})}$ where, for each internal node $w \notin W$ which was not already translated, a C code block $B = B_1 B_2$ is generated s.t. B_1 is of the form `L_w: if (x[j-1]) goto L_h;`

(line 7 of Algorithm 3) and B_2 has one of the following forms: i) else goto L_l ; (if $\text{flip}(w) = 0$, line 9 of Algorithm 3) or ii) else {ret_b = !ret_b; goto L_l ; } (otherwise, line 8 of Algorithm 3). For the terminal node, the block L_l : return ret_b; is generated. Note that maintaining the set of already translated nodes W allows us to fully exploit COBDDs nodes sharing.

Algorithm 2 Generating C functions

Require: COBDD ρ , v_1, \dots, v_r , boolean values b_1, \dots, b_r

Ensure: $\text{GenerateCCode}(\rho, v_1, b_1, \dots, v_r, b_r)$:

```

1: print "int K_bits(int *x, int action) {
  int ret_b; switch(action) {"
2: for all  $i \in [r]$  do
3:   print "case ",  $i - 1$ , ": ret_b = ",  $\bar{b}_i$ , ";
     goto L_";  $v_i$ , ";"
4: print "}" /* end of the switch block */
5:  $W \leftarrow \emptyset$ 
6: for all do  $i \in [r]$   $W \leftarrow \text{Translate}(\rho, v_i, W)$  done
7: print "}" K(int*x, int*u) {int
  i; for(i=0; i<r; i++) u[i]=K_bits(x, i);}
```

Algorithm 3 COBDD nodes translation

Require: COBDD ρ , node v , nodes set W

Ensure: $\text{Translate}(\rho, v, W)$:

```

1: if  $v \in W$  then return  $W$ 
2:  $W \leftarrow W \cup \{v\}$ , print "L_";  $v$ , ";"
3: if  $v = 1$  then
4:   print "return ret_b;";
5: else
6:   let  $i$  be s.t.  $\text{var}(v) = x_i$ 
7:   print "if(x[",  $i - 1$ , "]=1)goto L_"; high( $v$ )
8:   if flip( $v$ ) then print "else {ret_b = !ret_b;
     goto L_"; low( $v$ ), ";"
9:   else print "else goto L_"; low( $v$ )
10:   $W \leftarrow \text{Translate}(\rho, \text{high}(v), W)$ 
11:   $W \leftarrow \text{Translate}(\rho, \text{low}(v), W)$ 
12: return  $W$ 
```

Algorithm Correctness: Correctness of our approach, i.e., of function Synthesize in Algorithm 1, is stated by Th. 2 (for the proof, see [8]).

Theorem 2. Let $\rho = (\mathcal{V}, V, \mathbf{1}, \text{var}, \text{low}, \text{high}, \text{flip})$ be a COBDD with $\mathcal{V} = \mathcal{X} \cup \mathcal{U}$, $v \in V$ be a node, $b \in \mathbb{B}$ be a boolean. Let $\llbracket v, b \rrbracket = K(\mathbf{x}, \mathbf{u})$. Then function $\text{Synthesize}(\rho, v, b)$ generates a C function $\text{void K}(\text{int } *x, \text{int } *u)$ with the following property: for all $\mathbf{x} \in \text{Dom}(K)$, if before a call to $K \forall i \in [n] x[i - 1] = x_i$, and after the call to $K \forall i \in [r] u[i - 1] = u_i$, then $K(\mathbf{x}, \mathbf{u}) = 1$. Furthermore, function K has WCET $O(nr)$.

An Example of Translation: Consider the COBDD ρ shown in Figure 1. Within ρ , consider mgo $K(x_0, x_1, x_2, u_0, u_1) = \llbracket 0x17, 1 \rrbracket$. By applying SolveFunctionalEq , we obtain $f_1(x_0, x_1, x_2) = \llbracket 0x15, 1 \rrbracket$ and $f_2(x_0, x_1,$

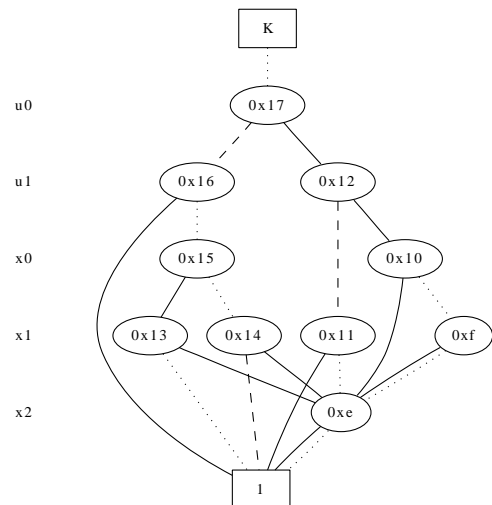


Figure 1. An mgo example

```

int K_bits(int *x, int action) { int ret_b;
  switch(action) { case 0: ret_b = 0; goto L_0x15;
                  case 1: ret_b = 0; goto L_0x10; }
  L_0x15: if (x[0] == 1) goto L_0x13;
         else { ret_b = !ret_b; goto L_0x14; }
  L_0x13: if (x[1] == 1) goto L_0xe;
         else { ret_b = !ret_b; goto L_1; }
  L_0xe:  if (x[2] == 1) goto L_1;
         else { ret_b = !ret_b; goto L_1; }
  L_0x14: if (x[1] == 1) goto L_0xe;
         else goto L_1;
  L_0x10: if (x[0] == 1) goto L_0xe;
         else { ret_b = !ret_b; goto L_0xf; }
  L_0xf:  if (x[1] == 1) goto L_0xe;
         else { ret_b = !ret_b; goto L_0xe; }
  L_1:   return ret_b; }

void K(int *x, int *u) { int i;
  for(i = 0; i < 2; i++) u[i] = K_bits(x, i); }
```

Figure 2. C code for the mgo in Figure 1 as generated by Synthesize

$x_2) = \llbracket 0x10, 1 \rrbracket$. Note that $0xe$ is shared between $G^{(\rho_{0x15})}$ and $G^{(\rho_{0x10})}$. Finally, by calling GenerateCCode (see Algorithm 2) on f_1, f_2 , we have the C code in Figure 2.

VII. EXPERIMENTAL RESULTS

We implemented our synthesis algorithm in C programming language, using the CUDD package for OBDD based computations and BLIF files to represent input OBDDs. We name the resulting tool KSS (*Kontrol Software Synthesizer*). KSS is part of a more general tool named QKS (*Quantized feedback Kontrol Synthesizer* [4]).

1) *Experimental Settings:* We present experimental results obtained by using KSS on given COBDDs ρ_1, \dots, ρ_4 s.t. for all $i \in [4]$ ρ_i represents the mgo $K_i(\mathbf{x}, \mathbf{u})$ for a buck DC/DC converter with i inputs (see [6] for a description of this system), where $n = |\mathbf{x}| = 20$ and $r_i = |\mathbf{u}| = i$. K_i is an intermediate output of the QKS tool described in [4].

For each ρ_i , we run KSS so as to compute $\text{Synthesize}(\rho_i, v_i, b_i)$ (see Algorithm 1). In the following, we will call $\langle v_{1i}, b_{1i}, \dots, v_{ii}, b_{ii} \rangle$, with $v_{ji} \in V_i, b_{ji} \in \mathbb{B}$, the out-

Table I
KSS PERFORMANCES

r	CPU	MEM	$ K $	$ F^{unsh} $	$ Sw $	%
1	2.2e-01	4.5e+07	12124	2545	2545	0.0e+00
2	4.2e-01	5.3e+07	25246	5444	4536	1.7e+01
3	5.2e-01	5.9e+07	34741	10731	8271	2.3e+01
4	6.3e-01	6.5e+07	43065	15165	11490	2.4e+01

put of function *SolveFunctionalEq*(ρ_i, v_i, b_i). Moreover, we call $f_{1i}, \dots, f_{ii} : \mathbb{B}^n \rightarrow \mathbb{B}$ the i boolean functions s.t. $\llbracket v_{ji}, b_{ji} \rrbracket = f_{ji}(\mathbf{x})$. All our experiments have been carried out on a 3.0 GHz Intel hyperthreaded Quad Core Linux PC with 8 GB of RAM.

2) *KSS Performance*: In this section we will show the performance (in terms of computation time, memory, and output size) of the algorithms discussed in Section VI. Table I show our experimental results. The i -th row in Table I corresponds to experiments running KSS so as to compute *Synthesize*(ρ_i, v_i, b_i). Columns in Table I have the following meaning. Column r shows the number of action variables $|\mathbf{u}|$ (note that $|\mathbf{x}| = 20$ on all our experiments). Column *CPU* shows the computation time of KSS (in secs). Column *MEM* shows the memory usage for KSS (in bytes). Column $|K|$ shows the number of nodes of the COBDD representation for $K_i(\mathbf{x}, \mathbf{u})$, i.e., $|V_{v_{\ell_i}}|$. Column $|F^{unsh}|$ shows the number of nodes of the COBDD representations of f_{1i}, \dots, f_{ii} , without considering nodes sharing among such COBDDs. Note that we do consider nodes sharing inside each f_{ji} separately. That is, $|F^{unsh}| = \sum_{j=1}^i |V_{v_{ji}}|$ is the size of a trivial implementation of f_{1i}, \dots, f_{ii} in which each f_{ji} is implemented by a stand-alone C function. Column $|Sw|$ shows the size of the control software generated by KSS, i.e., the number of nodes of the COBDD representations f_{1i}, \dots, f_{ii} , considering also nodes sharing among such COBDDs. That is, $|Sw| = |\cup_{j=1}^i V_{v_{ji}}|$ is the number of C code blocks generated by lines 5–6 of function *GenerateCCode* in Algorithm 2. Finally, Column % shows the gain percentage we obtain by considering node sharing among COBDD representations for f_{1i}, \dots, f_{ii} , i.e., $(1 - \frac{|Sw|}{|F^{unsh}|})100$.

From Table I we can see that, in less than 1 second and within 70 MB of RAM we are able to synthesize the control software for the multi-input buck with $r = 4$ action variables, starting from a COBDD representation of K with about 4×10^4 nodes. The control software we synthesize in such a case has about 1.2×10^4 lines of code, whilst a control software not taking into account COBDD nodes sharing would have had about 1.5×10^4 lines of code. Thus, we obtain a 24% gain towards a trivial implementation.

VIII. CONCLUSION AND FUTURE WORK

We presented an algorithm and a tool KSS implementing it which, starting from a boolean relation K representing

the set of implementations meeting the given system specifications, generates a correct-by-construction C code implementing K . This entails finding boolean functions F s.t. $K(x, F(x)) = 1$ holds, and then implement such F . WCET for the generated control software is linear linear in nr , being r the number of functions in F and $n = |x|$. KSS allows us to synthesize correct-by-construction control software, provided that K is provably correct w.r.t. initial formal specifications. This is the case in [4], thus this methodology, e.g., allows to synthesize correct-by-construction control software starting from formal specifications for DTLHSS. We have shown feasibility of our proposed approach by presenting experimental results on using it to synthesize C controllers for a buck DC-DC converter.

In order to speed-up the resulting WCET, a natural possible future research direction is to investigate how to parallelize the generated control software, as well as to improve don't-cares handling in F .

Acknowledgments: This work has received funding both from MIUR project TRAMP and the FP7/2007-2013 project ULISSE (grant agreement n°218815).

REFERENCES

- [1] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.
- [2] D. Baneres, J. Cortadella, and M. Kishinevsky, "A recursive paradigm to solve boolean relations," *IEEE Trans. Comput.*, vol. 58, pp. 512–527, April 2009.
- [3] R. Wille and R. Drechsler, "Bdd-based synthesis of reversible logic for large functions," in *DAC*, 2009, pp. 270–275.
- [4] F. Mari, I. Melatti, I. Salvo, and E. Tronci, "Synthesis of quantized feedback control software for discrete time linear hybrid systems," in *CAV*, ser. LNCS 6174, 2010, pp. 180–195.
- [5] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. The MIT Press, 1999.
- [6] F. Mari, I. Melatti, I. Salvo, and E. Tronci, "Quantized feedback control software synthesis from system level formal specifications for buck dc/dc converters," *CoRR*, vol. abs/1105.5640, 2011.
- [7] E. Tronci, "Automatic synthesis of controllers from formal specifications," in *ICFEM*. IEEE, 1998, pp. 134–143.
- [8] F. Mari, I. Melatti, I. Salvo, and E. Tronci, "From boolean functional equations to control software," *CoRR*, vol. abs/1106.0468, 2011.
- [9] A. Cimatti, M. Roveri, and P. Traverso, "Strong planning in non-deterministic domains via model checking," in *AIPS*, 1998, pp. 36–43.
- [10] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a bdd package," in *DAC*, 1990, pp. 40–45.
- [11] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient boolean function manipulation," in *DAC*, 1990, pp. 52–57.

Empirical Evidence in Software Architecture: A Systematic Literature Review Protocol

Nadia Qureshi, Naveed Ikram, Muneera Bano, Muhammad Usman

International Islamic University Islamabad, Pakistan

nadia_iiu@yahoo.com, {naveed.ikram, muneera, m.usman} [@iiu.edu.pk](mailto:iiu.edu.pk)

Abstract—Software Architecture (SA) plays important role in software development as it acts as a skeleton and the whole development revolves around it. As the SA as a discipline is maturing, large number of empirically supported studies are being reported in SA. There is a need to systematically aggregate, analyze and synthesize evidence based studies in SA. We plan to systematically investigate evidence-based SA studies to see and report state of the art in evidence based SA reported research. This paper aims at providing a brief description of systematic literature review (SLR) protocol to describe a process for synthesizing the empirically supported work in the area of SA. Protocol for this review has already been developed and its implementation is in progress. Expected outcome of this review will be state-of-the-art of empirical work in the field of software architecture, strength and effectiveness of empirical work, best practices and future research directions.

Keywords—*systematic literature review; software architecture state-of-the-art.*

I. INTRODUCTION

Software Architecture acts as a skeleton for the software development. SA needs to be created early during the software development and then the whole development process revolves around this skeleton, keeping into account the constraints and facilities implied by the software architecture. Few decades back there was nothing like architecture. The concept of software architecture was first introduced in 1968 when layering was used in program development [1], then this concept was enhanced and structure of software was emphasized [2] [3]. Increasing complexity and software quality needs urged the practitioners to opt modularity and ultimately it turned into the form which is now called software architecture. Software architecture is responsible for incorporating quality in software by accommodating quality attributes and functional requirements. Moreover, software architecture must have to accommodate the continuous changing needs so it should be flexible enough to evolve. Academia and industry both are well aware of the importance of software architecture that is why there exists lots of empirical literature on various sub areas of software architecture. But there is a need to summarize and aggregate this literature to find out actual status of the field, to identify gaps, scope for further research and quality of the work. This is the reason to undertake this systematic literature review. This document provides an outline of the protocol for this systematic literature review and it is developed based on the guidelines of Kitchenham [4].

The remaining portion of the paper consists of following sections; Section II describes motivation and background, Section III explains the outline of research methodology and Section IV concludes the paper along with future work.

II. BACKGROUND

The main motive to undertake this systematic review is to identify all empirical research related to software architecture, aggregate the empirical studies and summaries the evidence for future use. Focus of this SLR is limited to aggregate empirically supported literature i.e. literature based upon some evidence (case study, experiment, experience report and lesson learned etc such studies are also called evidence-based studies). Evidence-based literature is more valuable than literature based upon authors' personal opinion. Similar work exists in several studies where researchers summarized the available literature and pointed out future directions but the focus of those studies was not empirical evidence. The studies, those reported state-of-the-art in software architecture, did not performed qualitative and quantitative evaluation of empirical data at a time.

The concept of software architecture as a separate discipline started to emerge in 1990 [5] [6] and developed later on [7]. Since then, the key research areas of software architecture and its future directions have been identified time to time by conducting informal literature reviews and surveys [5] [8] [9]. The research paradigms used in software architecture research have been focused by identifying the types of research questions which were structured to use and the research design devised to answer those questions; state-of-the-art in software architecture with a perspective of growth in technology maturation model has also been described [10] [11]. The chronological history of the software architecture field, its innovative methods, tools, techniques, software architecture community, papers, books and conferences has already been aggregated [5].

The above mentioned studies aggregated the existing literature of software architecture and each of these studies has different concerns and varying scope. These studies were carried out as normal literature surveys without following a systematic process. None of these studies attempted to aggregate the evidence-based software architecture literature. Evaluating empirical evidence is equally important for academia and software industry, as systematically gathering and summarizing empirical evidence will help researchers in future and practitioners will also get quantified measures to make informed decisions [12]. There is much work that points towards the need to systematically gather empirical evidence in software Engineering [13] [14]; so conducting the research using a systematic and unbiased methodology is

necessary. Mapping study [15] and many systematic literature reviews exist in the field of software architecture [16] [17] [18]. They differ from our review in a way that their scope is limited to one sub-area of software architecture as opposed to our review. We focus on whole SA discipline. Moreover our review is focused on only empirically supported evidence based SA studies.

III. SYSTEMATIC LITERATURE REVIEW PROTOCOL

Systematic Literature Review is a form of secondary study and it is an established methodology used for the identification, analysis and interpretation of available literature relevant to the research question [4]. Systematic literature Review does not result in novel ideas, it is a fair and repeatable methodology for evaluation of the existing evidence [4]. There are three steps of a systematic literature review process i.e., planning, conducting and reporting [4]. This section will explain the outline of systematic literature review planning phase. Systematic review protocol is the outcome of this phase. Systematic review protocol is the detailed plan that describes the whole review procedures. It is better to develop a pre-planned protocol before conducting systematic literature review [4].

The research questions are phrased considering the overall objective of this systematic literature review so, that these questions can capture the existing empirical knowledge of software architecture field. By answering these research questions the needs for future research will be identified from existing empirical literature. Moreover the strength and validity of identified empirical literature will also be identified.

RQ1: what is the state-of-the-art in empirical studies of software architecture?

The main motive behind this research question is to find out the current state of the software architecture field in terms of existing empirical studies and to extract some future guidelines from the existing empirical literature. The data obtained as an answer of this question will be evaluated quantitatively in terms of frequency of occurrence and will depict the mature and underdeveloped areas of software architecture along with other relevant information in terms of quantity of the studies.

RQ2: what is the strength of empirical evidence reflected in empirical software architecture literature?

Objective behind this question is to find out the effectiveness and strength of empirical evidence in terms of source of evidence and methods used. Strength of empirical evidence is important for future research. The studies obtained for both of these questions will be same but the main difference is in the perspective, for this question data will be evaluated for quality of work to know what is the source of data and what study design have been used to obtain this evidence etc.

The overall Evidence based investigation is focused on the type of question given by guidelines of Kitchenham [4]. "Assessing the frequency or rate of project development factor such as the adoption of a technology or the frequency of project success or failure" And "identify and/or scope future research activities". So the questions of this SLR will

be assessing the future research scope by evaluating and aggregating the available literature.

A. Search Strategy

- Identify Major Search Terms

Search criteria used to construct major search terms is as follows

- Derive major terms from Research Questions;*
- Software architecture, empirical*
- Find alternative spellings and synonyms of major terms;*
- Software OR System*
- Architecture OR Structure OR Design*
- Empirical OR Industrial OR Case study OR Experiment OR Experience Report OR Lesson learned*
- Use Boolean Operators 'AND' and 'OR' to concatenate search terms if these operators are allowed to be used in the search database strings. Use 'OR' operator to concatenate synonyms of the search terms while use 'AND' to concatenate major search terms.*
- ((Software Architecture OR Software Structure OR Software Design OR System Architecture OR System Structure OR System Design) AND (Empirical OR Industrial OR Case Study OR Experiment OR Experience Report OR Lesson Learned))*

- Resources to be Searched: Springerlink, IEEE Explore, ACM Digital library, ScienceDirect, EI Compendex

- Search Constraints

Search is limited to published studies related to research questions. Search will be applied on conferences papers, journal articles, and workshop papers. This review will consider the work in English and since 1972 (After Parnas work on software structure and decomposition [2] [3]).

B. Publication Selection

- Inclusion Criteria

Research articles based on empirical evidence related to software architecture will be included with either professionals or students as subjects of investigation. Only one instance will be included if multiple studies report same empirical results.

- Exclusion Criteria

Editorials, prefaces, discussions, comments, summaries of tutorials, workshop brief, panels and duplicate studies will be excluded. Studies with insufficient focus on software architecture or with absence of empirical data will be excluded

- Selecting Primary Studies

Search Strings will be applied on the databases and obtained references will be archived in a Reference library. Duplicates will be removed. In the first phase titles of studies will be assessed upon inclusion exclusion criteria. In the next phase the abstracts will be reviewed and after that full text of the selected studies will be

assessed upon inclusion exclusion criteria. As the inclusion/exclusion criteria is multiphase so the results of each screening phase will be maintained in separate libraries. The papers that are not clearly relevant or irrelevant will be included or excluded in discussion meeting with secondary researcher/research supervisor.

C. Publication Quality Assessment

Quality Instrument will be used to assign quality score to the studies as a support for data analysis and synthesis. The Quality instrument consists of 5 sections; a main section contains generic checklist items applicable to all the studies while other 4 sections are specific for research design used in the study. These sections are survey, case study, experiment and experience report. These criteria are based upon SLR guidelines [4], along with revised set of items adopted from various checklists that have already been used [19] [20] [21] [22] [23]. This checklist generation was a mutual group effort and we are using it in other similar studies [27] as well. The detailed checklist is in Table I.

TABLE I. QUALITY CHECKLIST ADOPTED FROM [4] [19] [20] [21] [22] [23]

Quality Checklist	
<i>Generic</i>	
Are the aims clearly stated?	YES/NO
Are the study participants or observational units adequately described?	YES/NO/PARTIAL
Was the study design appropriate with respect to research aim?	YES/NO/PARTIAL
Are the data collection methods adequately described?	YES/NO/PARTIAL
Are the statistical methods justified by the author?	YES/NO
Is the statistical methods used to analyze the data properly described and referenced?	YES/NO
Are negative findings presented?	YES/NO/PARTIAL
Are all the study questions answered?	YES/NO
Do the researchers explain future implications?	YES/NO
<i>Survey</i>	
Was the denominator (i.e. the population size) reported?	YES/NO
Did the author justified sample size?	YES/NO
Is the sample representative of the population to which the results will generalize?	YES/NO
Have "drop outs" introduced biasness on result limitation?	YES/NO/NOT APPLICABLE
<i>Experiment</i>	
Were treatments randomly allocated?	YES/NO
If there is a control group, are participants similar to the treatment group participants in terms of variables that may affect study outcomes?	YES/NO
Could lack of blinding introduce bias?	YES/NO

Quality Checklist	
<i>Generic</i>	
Are the variables used in the study adequately measured (i.e. are the variables likely to be valid and reliable)?	YES/NO
<i>Case Study</i>	
Is case study context defined?	YES/NO
Are sufficient raw data presented to provide understanding of the case?	YES/NO
Is the case study based on theory and linked to existing literature?	YES/NO
Are ethical issues addressed properly (personal intentions, integrity issues, consent, review board approval)?	YES/NO
Is a clear Chain of evidence established from observations to conclusions?	YES/NO/PARTIAL
<i>Experience Report</i>	
Is the focus of study reported?	YES/NO
Does the author report personal observation?	YES/NO
Is there a link between data, interpretation and conclusion?	YES/NO/PARTIAL
Does the study report multiple experiences?	YES/NO

Some of the checklist items will be graded on yes/no and few with partially. Scores will also be assigned according the grades, 1 for yes, 0 for No and 0.5 for partially. The total sum of the scores will be used for the quality assessment of studies.

D. Data Extraction Strategy

Data-Extraction will be performed by using extraction forms. Each paper selected for data extraction will be assigned a unique ID. A general form will obtain generic data about the studies like title of the study, author(s) name, year of publication, journal/conference name etc. Then the data extraction form will extract data specifically relevant to research questions. The data will be extracted with the help of a classification scheme. This classification scheme is adapted from [24]. The extracted data for research questions is as:

For RQ1 data extraction form will extract following information:

- Software Architecture area (Software Architecture design, Software Architecture Documentation and Specification, Software Architecture Analysis, Software Architecture Evolution, Software Architecture Knowledge Management etc)
- Research output (New Tool/Technique/Process, Modification of Tool/Technique/Process, Usage Experience of Tool/Technique/Process, Software Architecture issues and Challenges).
- Subjects of investigation (Academia, Industry, Mixed)
- Country (involved in research)
- Conference/ Journal
- Year of Publication

For RQ2 the extracted information is as follows:

- Type of evidence (case study, experiment, experience report etc)
- Data collection method (interview, questionnaire etc)
- Type of research: Data about type of research will be extracted based upon an existing classification of research [25] [26] i.e. validation research, evaluation research, solution proposal, philosophical papers, opinion papers and experience papers.

E. Data Synthesis Strategy

Data Extracted from selected literature will be analyzed using quantitative and qualitative synthesis methods. The classification scheme used in data extraction will help here to separate the concerns and categories. Relationships among various categories of data will also be pointed out with multiple perspectives. After depicting data in quantitative summaries a thorough qualitative analysis of the data will also be performed to evaluate the strengths of the literature and to draw certain patterns. The expected outcomes will contain information like:

- Publication chronology of included studies
- Distribution of included studies in publication channels along with most cited studies
- List of best practices
- Percentage of studies for different research and evidence types
- Analysis of evidence type versus participant type
- Analysis of evidence type versus SA area type.
- Analysis of evidence type versus type of research
- And more complex analysis comprising more than two parameters.

The quantitative information will be depicted in the form of Bar graphs, Bubble plots etc. A thorough Qualitative analysis of the claims, future directions, recommendations and personal reflections will be performed to draw certain research patterns, future direction and existing gaps.

We are conducting similar studies in other disciplines as well like requirements engineering [27] and using almost the same extraction and synthesis strategies in all studies.

IV. CONCLUSION AND FUTURE WORK

Software Architecture (SA) is maturing into a discipline and has now a long history of research and development. It has its own workshops, conferences and special issues in journals. Large number of empirically supported studies has been published in SA. There lacks a study which presents state of the art of empirically supported work in overall SA discipline. This paper presents the plan for conducting such study i.e. a systematic literature review to present state of the art of empirically supported evidence based SA work. The study will help SA practitioners and researchers to find out mature practices and techniques, patterns/trends in research, gaps and future directions where more emphasis should be placed. The implementation of this systematic literature review protocol is under progress. Search strings returned 5617 results. Screening of the studies upon titles and

abstracts is complete. At present data extraction and quality ranking procedure is under progress.

TABLE II. PUBLICATIONS OBTAINED FROM VARIOUS PUBLICATION CHANNELS.

S. NO.	Publication distribution		
	Publication Channel	No. of studies	%age of studies
1	IEEE	1930	34.35%
2	ACM	1761	31.35%
3	ScienceDirect	259	4.61%
4	SpringerLink	308	5.5%
5	EI Compendex	1359	24.19%
	Sum	5617	100%

REFERENCES

- [1] E. W. Dijkstra, "The structure of the 'THE'-multiprogramming system," in *Proceedings of the first ACM symposium on Operating System Principles*, 1967, pp. 10.1-10.6.
- [2] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053-1058, 1972.
- [3] D. L. Parnas, "Information distribution aspects of design methodology," *Methods*, vol. 4, no. 5, pp. 6-7.
- [4] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," *Engineering*, vol. 2, no. EBSE 2007-001, 2007.
- [5] P. Kruchten, H. Obbink, and J. Stafford, "The past, present, and future for software architecture," *Software, IEEE*, vol. 23, no. 2, pp. 22-30, 2006.
- [6] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40-52, 1992.
- [7] D. E. Perry, "State of the Art: Software Architecture," in *International Conference on Software Engineering*, 1997, vol. 19, pp. 590-591.
- [8] D. Garlan, "Research directions in software architecture," *ACM Computing Surveys (CSUR)*, vol. 27, no. 2, pp. 257-261, 1995.
- [9] Y. Chen, X. Li, L. Yi, D. Liu, L. Tang, and H. Yang, "A ten-year survey of software architecture," in *IEEE International Conference on Software Engineering and Service Sciences (ICSESS)*, 2010, pp. 729-733.
- [10] M. Shaw, "The coming-of-age of software architecture research," in *Proceedings of the 23rd international conference on Software engineering*, 2001, p. 656.
- [11] M. Shaw and P. Clements, "The golden age of software architecture," *Software, IEEE*, vol. 23, no. 2, pp. 31-39, 2006.
- [12] D. Falessi, M. A. Babar, G. Cantone, and P. Kruchten, "Applying empirical software engineering to software architecture: challenges and lessons learned," *Empirical Software Engineering*, vol. 15, no. 3, pp. 250-276, 2010.
- [13] T. Dyba, B. A. Kitchenham, and M. Jorgensen, "Evidence-based software engineering for practitioners," *Software, IEEE*, vol. 22, no. 1, pp. 58-65, 2005.
- [14] B. J. Oates, "Widening the scope of evidence gathering in software engineering," in *11th International Workshop on Software Technology and Engineering Practice*, 2003, pp.59-64.

- [15] E. Y. Nakagawa, D. Feitosa, and K. R. Felizardo, "Using systematic mapping to explore software architecture knowledge," in *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, 2010, pp. 29-36.^h
- [16] R. Farenhorst and R. C. de Boer, "Knowledge management in software architecture: State of the art," *Software Architecture Knowledge Management: Theory and Practice*. Springer, Under submission.
- [17] B. J. Williams and J. C. Carver, "Characterizing software architecture changes: A systematic review," *Information and Software Technology*, vol. 52, no. 1, pp. 31-51, 2010.
- [18] H. P. Breivold, I. Crnkovic, and M. Larsson, "A systematic review of software architecture evolution research," *Information and Software Technology*, 2011, ISSN 0950-5849, 10.1016/j.infsof.2011.06.002. (<http://www.sciencedirect.com/science/article/pii/S0950584911001376>)
- [19] B. A. Kitchenham, O. P. Brereton, D. Budgen, and Z. Li, "An Evaluation of Quality Checklist Proposals-A participant-observer case study," in *13th International Conference on Evaluation and Assessment in Software Engineering*, 2009, available online http://www.bcs.org/upload/pdf/ewic_ea09_s3paper1.pdf
- [20] B. Kitchenham et al., "Can we evaluate the quality of software engineering experiments?," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2010, pp. 1-8.
- [21] M. Host and P. Runeson, "Checklists for software engineering case study research," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 2007, pp. 479-481.
- [22] D. Budgen and C. Zhang, "Preliminary reporting guidelines for experience papers," in *Proceedings of EASE*, 2009, vol. 2009, pp. 1-10.
- [23] T. Dyba and T. Dingsoyr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology*, vol. 50, no. 9-10, pp. 833-859, 2008.
- [24] D. Šmite, C. Wohlin, T. Gorschek, and R. Feldt, "Empirical evidence in global software engineering: a systematic review," *Empirical Software Engineering*, vol. 15, no. 1, pp. 91-118, 2010.
- [25] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion," *Requirements Engineering*, vol. 11, no. 1, pp. 102-107, 2006.
- [26] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering*, 2008, pp. 71-80.
- [27] T. Ambreen, M. Usman, N. Ikram and M. Bano, "Software requirement engineering: A systematic literature review protocol", in *6th International Conference on Software Engineering Advances, 2011., in press.*

Agile Development of Interactive Software by means of User Objectives

Begoña Losada, Maite Urretavizcaya, Isabel Fernández de Castro

Dept. of Computer Languages and Systems

Faculty of Computer Engineering - University of the Basque Country
20001 San Sebastián

{b.losada, maite.urretavizcaya, isabel.fernandez}@ehu.es

Abstract—Agile methods, model-driven developments and user-centred design are three approaches widely accepted in the development of interactive software. In this paper we present InterMod, a new approach that integrates all three methods. The project planning is based on *User Objectives* and the process is organised as a series of iterations, where the work is distributed in different workgroups according to some developmental and integration activities, each one driven by models. The requirements are incrementally collected and evaluated with models based on user-centered design. To speed up this validation, we put forward the *SE-HCI model*, which enriches a human-computer interaction model with the semantics of the application and some basic characteristics of an abstract prototype. This allows gather and validate the requirements incrementally. Moreover, this iterative process speeds up the development and generates results from the project progress.

Keywords—Software Engineering; Agile method; User-Centered Design; Model-Driven Development.

I. INTRODUCTION

Currently, Agile Methods (AM) and Model-Driven Development (MDD) are the predominant approaches in Software Engineering. AM are able to develop a software product incrementally and iteratively. They get feedback from the client at each incremental delivery and, as a result, adapt the development plan accordingly. Most studies report increased code quality when agile methods are used but they also report a lack of attention to design and architectural issues [1]; moreover, it must be noted that in the area of Software Engineering quality software comes from good design.

Current trends establish design as final product models characterised by iterative and incremental development while at the same time promoting formal development along the lines of traditional or waterfall methodologies. Some authors [2] point out that a drawback of MDD is that the models are difficult to maintain, because as a project progresses changes come up and new requirements are added.

On the other hand, in the area of Human-Computer Interaction, User-Centered Design (UCD) is the dominant approach. Under UCD, the end user is involved in the process of multidisciplinary development based on iterative design and evaluation so the designer understands the user's needs and tasks [3][4]. But, as is the case with traditional or

heavyweight methodologies, with UCD all requirements must be gathered and evaluated before they are implemented [5][6][7].

To make up for the weaker aspects of these proposals, efforts are being made to integrate agile methods into both model-driven design [8][9], and into UCD [10]. However, due to the fact that a majority of software engineering development processes focus on software architecture, satisfactory integration has not yet been achieved. Therefore, we focus our efforts on integrating these three techniques and we base our methodology in user-centered models starting from requirements gathering.

The main contributions of the paper can be summarised as follows:

- c1. We propose a new approach to improve *Software Development* by applying *User Centered* and *Model-Driven Development* in an *Agile* manner.
- c2. A new integrated model, involved in a Model Driven Process, to support the project requirements, is presented: the SE-HCI model. It facilitates usability and other kinds of incremental evaluation, tested by a multidisciplinary team of developers and users, just as proposed by UCD.
- c3. Finally, we present an agile methodology organised as a series of iterations by means of *User Objectives* (UO) as a new way to promote a correct development. This iterative approach guides the incremental development of software.

Our paper is structured as follows. Section 2 outlines the primary characteristics of agile methods and how they compare to the other abovementioned approaches. Section 3 presents our proposal, situating it in the context of related work. We explain phases and development activities of our approach and its model structure, especially for the requirements model, and we show graphically a project iteration example. Finally, we draw some conclusions and outline our future work.

II. AGILE SOFTWARE DEVELOPMENT: VIRTUES AND DEFECTS RELATIVE TO OTHER APPROACHES

Agile software development establishes the following as principles [11]: Individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan.

This approach challenges waterfall or heavyweight methods in which one activity begins only when the previous one finishes and where extensive and well-founded documentation is required. The rationale behind these traditional methods is to reduce the number of corrections further on in the process and consequently reduce the cost of the project. However, in practice this type of planning fails, as it doesn't allow the changes that inevitably come up during development [12]. Because of this, versions of the agile philosophy such as eXtreme Programming (XP) [13], Scrum [14], Crystal [15], Feature Driven Development (FDD) [16], UP [17] and others, currently prevail.

A. Agile processes and user-centered design

One of the important aspects of UCD is the collaboration between users and developers in building software solutions, each one bringing their experience to bear [18]. According to Norman [19], it is first necessary to think about the needs of those who will be using the product that is being created in order to model that information, and then iteratively evaluate the product with users. Thus, the intention is to improve the product's usability such that it is easy to learn, it is easy to use, errors are reduced and users are satisfied, as defined in ISO standard 9241-11 [20].

Both proposals centre on the user/client and propose an iterative development process. These contrast with traditional architecture-based development processes, which are directed by the developers, who structure and control the users' activities.

Nevertheless, the differences between UCD and AM are great in terms of how they act and what their interests are [10]. On the one hand, the flexibility in action when faced with changes that the agile philosophy recommends is at odds with interface design prior to implementation (up-front), according to the principles of UCD. On the other hand, UCD develops a holistic product, while the agile process results in subproducts in an incremental process. And while agile methods focus on code development, UCD methods focus on the design of the interaction that users will engage in.

Finally, it must be noted that both approaches seek to satisfy the users' needs. However, in AM users are involved in checking that the functionality has been correctly implemented, while in UCD users give input regarding other aspects such as user satisfaction or efficiency of use for the whole application. UCD focuses on how end users work with the system, whereas AM is more concerned with how software should be built or how the process is managed.

B. Agile processes and model-driven development

In MDD, models serve principally as documentation and guidance for the subsequent implementation phase. Although building models is very useful in other areas of engineering, in Software Engineering there is great apathy toward building and using models. Many developers think that modelling demands the creation of excessive and extensive documentation, which ultimately is of little help when it comes time to implement and maintain the system [2]. This is because the changes that arise throughout development

make these models difficult to update. In fact, many developers skip the model redesign phases and prefer to modify the code directly.

From this point of view, we have two issues that strongly conflict in software development. On the one hand, MDD needs to maintain model consistency as changes come up during application development. That is to say, our system will be more flexible if the model that represents it is an accurate and updated abstraction of itself [21]. On the other hand, due to unforeseen changes, AM perform modifications on the implementations that are not reflected in the designs. Therefore, if the constructed model does not correspond with reality and our code was initially generated from the model, this could spell failure for the project.

III. INTERMOD, AN INTEGRATED PROPOSAL

InterMod [22] is a methodology whose aim is to help with the accurate development of interactive software. Although it is suitable for use with web design, its utility is not restricted to just that area. Our latest studies have led us to place a new focus on the methodology by integrating an agile process with the other two philosophies namely, UCD and MDD, already present in our previous work. Also a new vision of the Requirements Models together with the SE-HCI model and the User Objective, to guide the process, are included in this paper.

Our proposal is the following: organise the project as a series of iterations, just as the agile methodologies do, and distribute the work in the iterations according to different developmental activities of the *User Objectives*. A *User Objective* (UO) is a user desire e.g. "buying a t-shirt" or "reserving a meeting room in a workplace", that can be achieved by one or more user functionalities. These are defined by means of the possibilities that the end user will perform in the application interface.

The Feature-Driven Design approach (FDD) [16] also uses MDD and divides the labour into different features (e.g. "calculate the total of a sale" or "add a new customer to a customer list") to see measurable progress of the project. The functionalities implied in our UOs are always direct user's intentions, whereas the features can be user's or system's needs. In FDD, use cases obtain the features that allow the domain objects to be modelled (class diagram and the operations required in the system). However, our primary goal is not to model the domain objects but rather to model the tasks (user actions in the interface), navigation (action/reaction between the user and system) and presentation (visual aspects). We focus the development from the UCD perspective, as a new vision that obtains partially the interface before implementing the business logic. Once the objectives have been evaluated in terms of testing and usability of requirements, our proposal naturally ties in with the FDD perspective to model the domain objects.

InterMod has four main steps, i.e. the initial step *Analyse Overall Project* takes place at the project beginning, and then an iterative process with three steps follows: *Build User Objectives List*, *Plan Parallel Iteration* and *Perform Iteration Activities*.

A. InterMod steps: Activities and Models

Fig. 1 shows a scheme of the InterMod process and the models associated with it.

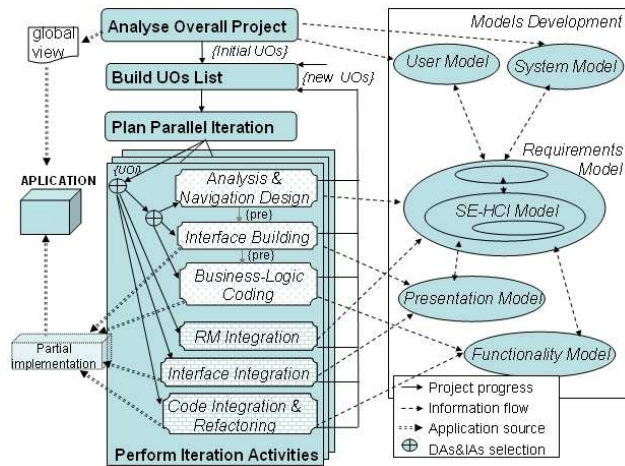


Figure 1. InterMod process and development activities

At the beginning of the project, it is necessary to analyse it as a whole in order to determine: (a) what the starting UOs are, and (b) the design decisions that will guide and give coherence to this iterative and incremental process. InterMod proposes the **Analyse Overall Project** step to achieve these challenges. The starting UOs (such as those most important or needed), together with a provisional general menu incorporating some functionalities, provide the global view of the application. And this analysis draws up the models that help to collect the defining characteristics of the system type (e.g. device type, security, window size, colour, logo, etc) and those of the user (e.g. colour preferences, font, size, some limitations as colour blindness, deafness, vision loss, etc). These characteristics are collected in the *System Model* and the *User Model* respectively. All developments in the project will inherit, supplement or extend these models in order to guide and ensure coherence throughout the entire application.

The application requirements are incrementally collected during the progressive UO List construction. Each iteration begins with a revision of the UOs list. The **Build User Objective List** step updates the list with the new UOs derived either from previous UO developments or from the new needs of the project. It is possible that a UO breaks on two or more new UOs because of its complexity; on the contrary, some UOs may be merged in one new integrated UO because of its simplicity. That is, the UOs included in the list may be modified, in the sense of agile methodologies [23], through the different evaluations undertaken by developers and users, or by the continuous meeting among members of the same and different teams.

In order to achieve a UO, different activities must be realised. The next step, **Plan Parallel Iteration**, decides for the current iteration:

a) what UOs to develop

b) what activities to make for those UOs

c) how to distribute these different activities to the workgroups (if there is more than one).

The iteration ends with the **Perform Iteration Activities** step. Each workgroup performs the activities established in its plan.

InterMod has two kinds of **Activities**: Developmental Activities and Integration Activities.

The **Developmental Activities** (DAs) associated with each UO are strongly related:

- A1. Analysis and Navigation Design
- A2. Interface Building
- A3. Business-Logic Coding.

Just as UCD recommends, before coding a relevant UO, its interface must be validated. However, unlike UCD, it is not required that the complete application interface be developed before moving to the implementation of the business logic; instead this approach stays framed in the development of one or several UO groups. That is, each UO requires the three DAs to be developed but a prerequisite relation must be done $A1 < A2 < A3$ ('<' means prerequisite). A1 has not got any prerequisite activity. A DA of a User Objective is possible to deal with if and only if the UO is in the UO list and its prerequisite is achieved.

Furthermore, to assure a correct incremental progress of the project, some **Integration Activities** (IAs) are needed:

- I1. Requirement Models (RM) Integration
- I2. Interface Integration
- I3. Code Integration & Refactoring

A restriction is necessary for controlling the correct development of an IA. Thus, it is possible to carry out an IA I_k ($K=1..3$) for a concrete UO_j ($j=0..n$) if and only if the UO_j is the fusion of two UOs belonging to the UO List and the DAs A_k of these fused UOs are already made. To ensure consistency in the final application, evaluations of the incrementally obtained products as well as heuristic and metric evaluations are included in all activities.

All iterations are guided by the same action plan that divides the work according to the activities of different UOs, in such a way that each DA will be next driven by models and all the integration processes can lead to the revision and modification of these models. Even during final integration of the software there may be revisions of all models and new UOs can be created.

The activities of **analysis and navigation design** and **RM integration** deals with the *Requirements Model* (RM), which includes the *Semantically Enriched Human-Computer Interaction (SE-HCI) model* (more detail in section III.B). In the **Interface Building** activity, the *Presentation Model* is created for a UO previously designed and evaluated, and the **Interface Integration** activity fuse together the *Presentation Model* of some UOs. The *Presentation Model* of a specific UO settles the graphical elements and others

characteristics gathered from the *Requirements Models*. There are several languages for modeling user interfaces widely used and tested, such as XIIML[24] or UIML [25], and they may be used to reflect this model. Finally, the **Business-Logic Coding** and **code Integration & Refactoring activities** deal with the *Functionality Model* that guides the implementation in a particular programming language. This model inherits the behaviour characteristics from the UO *Requirements Models* evaluated in the first activity. UML or SysML [26] are alternative languages typically used to represent this model.

B. The SE-HCI model in a Model Driven Process

We propose interactive software development based on user-centered models generated and evaluated during the project, following the Object Management Group’s Model Driven Architecture proposal [27].

For each UO the designers involved in the **Analysis & Navigation Design activity** formalise the established Requirements Models (RMs): *Task Model* and *SE-HCI Model* (see Figure 2.).

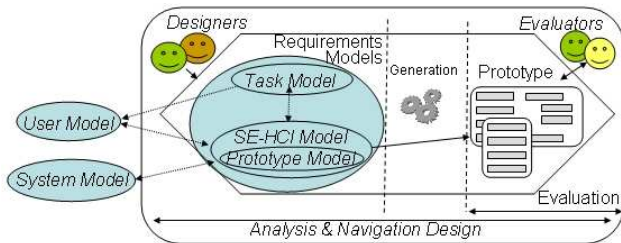


Figure 2. The SE-HCI model involved in a Model-driven process

The *Task Model*, which is a classic element in Model-Based User Interface Development [28][29][30], describes user performance in completing each task. The concept that is the basis of RM is the **Task**, which allows user performance to be captured. This concept is complemented by the **ordering** of tasks (Sequential, Indifferent, Choice, Concurrent), **iteration** which establishes whether it is compulsory to carry out a task and how many times it is necessary to do so (Unitary, Optional, Repetitive) and **hierarchy**, which correctly places the task in the complete set of tasks. That is, the *Task Model* defines the semantic aspects of the application to be associated (see Fig. 3).

The *SE-HCI Model*, which incorporates information from the *User* and *System Models*, is an abstract description constructed over the *Task Model*. The *SE-HCI Model* is the core of our proposed methodology, and it not only gathers the requirements from the *Task Model* but it also incorporates three essential aspects. The first two are behaviour aspects and the third, visual aspect (see Fig. 3):

- 1) The system direct communication with the user. The description of both the actions that users and the system can carry out at the user interface level (who performs the intervention: usr/sys), during an interactive session [28], and their possible temporal relations are here included. That is, it generates those communications in

which the system directly communicates with the user by displaying an error window or a simple message. This means that the system's operations on other elements in the application's environment, such as a database, won't be expressed in the model since they will not be involved in any direct communication with the user.

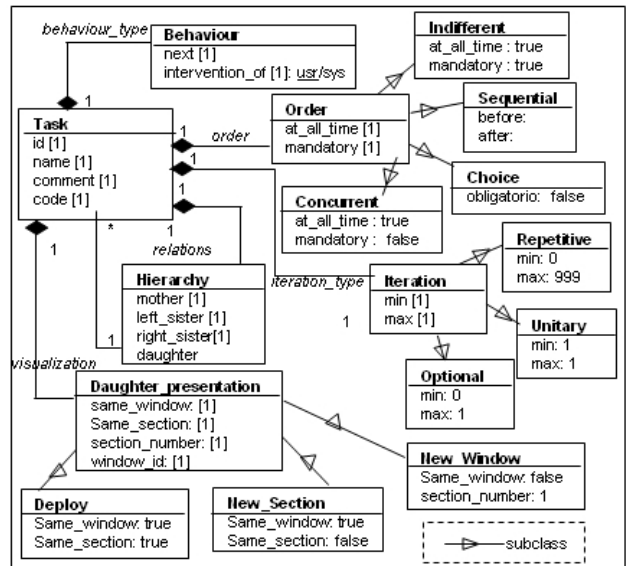


Figure 3. The Requirement Meta-Model

- 2) The descriptions of the correct interactions, taken from the *Task Model*, as well as the incorrect ones. Both types of interactions express the different application runs (next task to perform). That is, this model represents the semantics of the application through interface navigation.
- 3) The basic visual characteristics, such as colours, sections, button types, etc. The *SE-HCI* incorporates a *Prototype Model* that gathers these aspects, some of that are assumed from the *User* and *System Models*.

Different techniques can be used to implement this specification. Fig. 4 shows a graphical example of the SE-HCI for the development of a website; it has been made with a HTA technique [31] (some symbols express the characteristics of the tasks). In this case, we use a XML format to express that SE-HCI specification.

In line with user-centered designs, our proposal stresses, like Hix's model [32], the integration of the evaluation process at all stages of the lifecycle rather than just at the end as is the case in the classic cascade lifecycle. The *RMs* make it possible to quickly produce incremental prototypes by adapting the design according to the modifications prompted by both user and software developer evaluations. Similar to our proposal, Propp and his colleagues [33] start with task models in the process of developing interactive applications and they then define the navigational structure, the creation of an Abstract User Interface (AUI) that is independent of the device, and one or more Concrete User Interfaces (CUI).

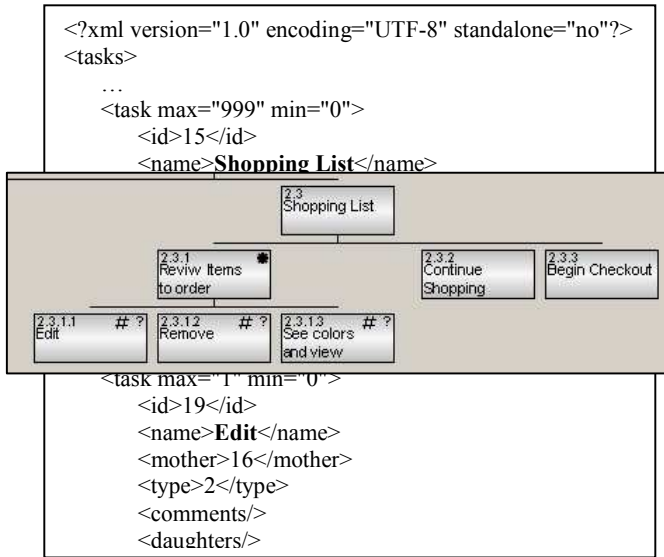


Figure 4. Snapshot of the XML description and Task Hierarchy of an application development

During the development process they perform several usability evaluations. We propose the evaluation of the requirements involved in the SE-HCI with an abstract prototype (Fig. 5) created automatically by transforming the *SE-HCI model*. From this point, the evaluation can be carried out jointly by the designers, customers and developers. According to Wiegiers [34] we think that it's hard to visualise exactly how software will behave by reading textual requirements or studying analysis models. Users are more willing to try a prototype than to read a document. Wiegiers says: "A prototype is useful for revealing and resolving ambiguity and incompleteness in the requirements."

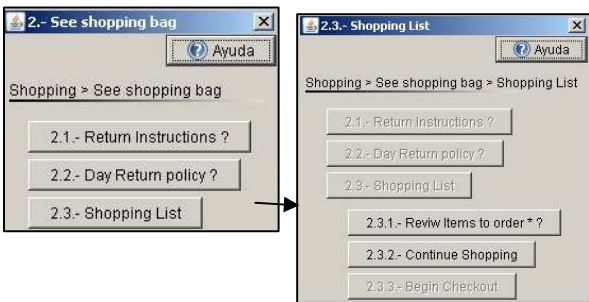


Figure 5. An Abstract Interface with simple menus and buttons

C. Iterations in InterMod. A general example

In this section we explain an iteration progress of a project. In order to facilitate and simplify the general example comprehension, we represent graphically *Activities* as shown in TABLE I. As above mentioned, each *Developmental Activity* (DA) is driven by models: A1-Requirements Models, A2- Presentation Model and A3-Functionality Model. And each *Integration Activity* (IA) is

involved in models integration: I1- Requirements Models, I2- Presentation Model and I3- Functionality Model.

TABLE I. INTERMOD ACTIVITIES

Development Activities	Graphical representation	Integration Activities
A1. Analysis & Navigation Design		I1. RM Integration
A2. Interface Building		I2. Interface Integration
A3. Bus.-Logic Coding		I3. Code Integration & Refactoring

Fig. 6 shows a snapshot of the Project Progress State and the Plan obtained for the Parallel Iteration after some iterations (iteration i).

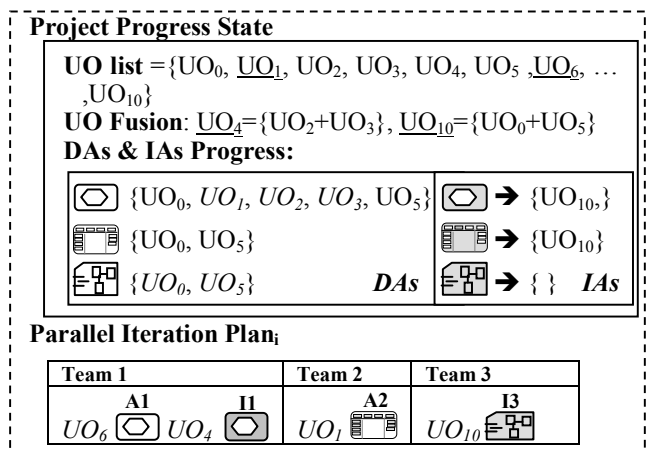


Figure 6. A Snapshot of the Project Progress with InterMod

Three aspects characterise the state of the project: the UO list, the UOs fusion list and the UOs progress according to the *Activities* (DAs or IAs) performed. This iteration begins with the **Build UO List** step to revise the UOs list. After that, the process goes on with the **Plan Parallel Iteration** step where the project members have decided:

- a) *The UOs to perform* (underlined in Fig.6 - UO list and UO Fusion),
- b) *The activities for these UOs*, which have been selected taking into account their prerequisites.
- c) *The distribution into three teams*, as follows:

- The first team takes responsibility for two activities: **A1** activity for UO₆ (in the UO list) and **I1** for UO₄. As it is shown, UO₄ is the fusion of the objectives 2 and 3 (UO Fusion in Fig. 6). The I1 Activity is possible because the progress of the project assures that both, the RMs of UO₂ and UO₃ are already validated (see A1 list in "DAs & IAs Progress" in Fig.6).
- The second team builds the interface (A2) for the UO₁ whose prerequisite is reached (UO₁ is in the A1 list).

- Meanwhile, team 3 must integrates and refactors the code referred to UO₁₀ that is composed of the objectives 0 and 5 that have been already coded.

The evolution of a UO is not predictable. In each work meeting project members will select the best UOs activities to do. Fig. 7 presents two different possible evolutions of UO₁₂ which is composed of {UO₄, UO₆}.

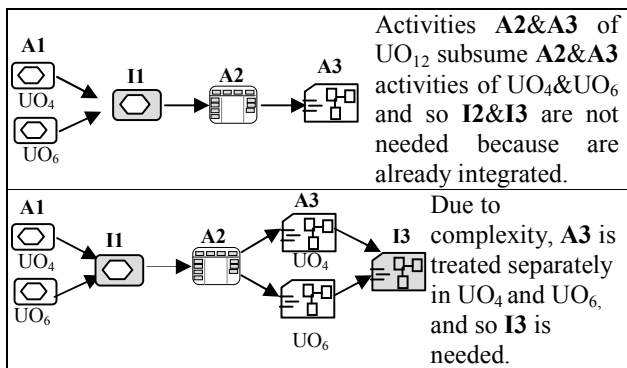


Figure 7. A Snapshot of two UO₁₂ possible evolution

When the Plan is ready, the teams go on with the activities assigned. After each iteration is completed, the process repeats:

- **Step2- Build of UO list-** All teams contribute with their work and evaluation results to the list actualization.
- **Step3- Plan Parallel Iteration-** Taking into account the prerequisites of the activities and the project needs, the distribution is carried out.
- **Step4- Perform Iteration Activities-** Each team makes their activities and the process goes again to the Step2 until the application is completed.

IV. CONCLUSION AND FUTURE WORK

In this article we presented a new vision of the InterMod methodology, a proposal integrating three philosophies: UCD, MDD and AM. From the point of view of agile methods, our work is organised in a series of iterations in which the user objectives (UO) to be dealt with are developed. This iterative process speeds up the development and generates results from the project progress. InterMod proposes some developmental and integration activities driven by models to achieve the UOs. In the first analysis, the initial user objectives are obtained and then, the different activities to achieve these UOs are distributed among the workgroups. Each iteration is open to include new user objectives, whether obtained through previous refinements or through evolution or alterations during the agile development of the application itself. The possibility to distribute the work in parallel increases the speed of resolution, although the process itself requires integration points to ensure consistency.

The SE-HCI model is the core of our proposal models architecture. It is involved in a Model Driven Process that

obtains an abstract prototype created automatically by transforming the *SE-HCI model*. This prototype allows the evaluation of the requirements and facilitates the end user's participation, as recommended by UCD and AM. Early evaluations of the requirements reduce the number of the corrections further on in the process and therefore, reduce its cost.

This process allows for the gathering and validation of the requirements incrementally. Because of this agile approach, InterMod, unlike UCD, does not require the complete development of the application interface before the implementation of the business logic, but assures usability.

The new InterMod methodology has been refined in parallel with the development of a demonstrator. A small initial set of UOs has evolved to a complex system. It has been carried out by means of UO creation, development and integration processes. This makes us think of the scalability and practicability properties of the proposed methodology. However these aspects have not been treated in this paper as a deeper work needs to be done.

We are currently working on reusing models. It should be understood in the broadest sense of the word. A UO model can be defined once in a project, but it can be reused at different points in the project. Similarly, a model developed in previous applications can be reused in a current project. Thus, a model can be converted into a pattern or a solution to a design problem. That is to say, we believe that it is important to value the possibility of creating patterns, in order to facilitate and speed up design processes.

ACKNOWLEDGMENT

This work has been partially supported by TIN2009-14380 and DFG 157/2009.

REFERENCES

- [1] Mcbreen, P., "Questioning Extreme Programming", Pearson Educ., Boston, MA, USA 2003
- [2] Ambler, S., "Debunking Modeling Myths", <http://www.ambysoft.com/onlineWritings.html> (Last Access: August 2011).
- [3] Norman, D.A. and Draper, S.W., "User-Centered System Design: New Perspectives on HCI", Lawrence Erlbaum Associates, Inc, Mahwah, NJ, USA, 1986.
- [4] Vredenburg, K., Isensee, S., and Carol Righi, C., "User-Centered Design: An Integrated Approach", Prentice Hall, 2001.
- [5] Norman, D., "Why doing user observations first is wrong". *Interactions* 13, 4, 2006, pp.50--63
- [6] Cooper, A. and Reimann, R.: About Face 2.0, "The Essentials of Interaction Design", JohnWiley & Sons, Inc., Indianapolis, Indiana, USA, 2003
- [7] Constantine, L. and Lockwood, L.: Software for Use, "A Practical Guide to the Models and Methods of Usage-Centered Design". ACM Press, Addison-Wesley Co., 1999
- [8] Robles, E., Grigera, J., and Rossi, G., " Bridging Test and Model-Driven Approaches in Web Engineering", in: Gaedke M., Grossniklaus M, Diaz O. (eds.) ICWE 2009. LNCS, vol. 5648., Springer, Heidelberg 2009, pp. 136--150

- [9] Ambler, S.W., “The object primer: agile modeling-driven development with UML 2.0.” Cambridge University Press, Cambridge 2004
- [10] Ferreira, J., “Interaction Design and Agile Development, A Real- World. Pers.”, Ph. D. 2007.
- [11] Fowler, M. and Highsmith, J., “The agile manifesto, Software Development”, 2001, pp 28--32
- [12] Highsmith and J., Cockburn, A., “Agile Software Development: The business of innovation”. Computer 34, 9, 2001, pp120—127
- [13] Beck, K., “Extreme Programming Explained-Embrace Change”. Addison-Wesley, 2000.
- [14] Schwaber, K. and Beedle, M. “Agile Software Development with Scrum”, Prentice-Hall, 2002.
- [15] Cockburn, A., “Agile Software Development”, Addison-Wesley, 2002
- [16] Palmer, S.M. and Felsing, J.M., “A practical guide to feature-driven development”. Prentice-Hall USA, 2002.
- [17] Jacobson, I., Booch, G., and Rumbaugh, J., “The Unified Software Development Process”, Addison-Wesley, 1999.
- [18] Robey, D., Welke, R., and Turk, D., “Traditional, iterative, and component-based development: A social analysis of software development paradigms”, Information Technology and Management, Volume 2, Number 1, 2001, pp53-70
- [19] Norman, D.A., “The invisible Computer”, Cambridge M.A. MIT Press, 1998.
- [20] ISO, (International Organization for Standardisation), 9241-11. Ergonomic requirements for office work with visual display terminals. Part 11: Guidance on usability, 1998.
- [21] Eric Evans, Domain-Driven Design, “Tackling complexity in the heart of software”, Addison Wesley, 2004
- [22] Losada, B., Urretavizcaya M., and Fernández-Castro, I., “The InterMod Methodology: An Interface Engineering Process linked with Software Engineering Stages”, In Macías, J.A., Granollers, T., Latorre, P. (eds). New Trends on Human-Computer Interaction: Research, Development, New Tools and Methods. Springer, 2009
- [23] Larman, C., “Agile & Iterative development: A manager’s guide”. Addison-Wesley, 2004.
- [24] eXtensible Interface Markup Language <http://www.ximl.org/> (Last Access: August 2011).
- [25] Abrams, M. and Helms, J., UIML Specification, 2002 <http://www.oasis-open.org/committees/download.php/5937/uiml-core-3.1-draft-01-20040311.pdf> (Last Access: August 2011).
- [26] Nolan, B., Brown, B., Balmelli, L., Bohn, T., and Wahli, U., “Model Driven Systems Development with Rational Products”. ibm.com/redbooks 2007
- [27] Object Management Group. Model Driven architecture. Technical report, 2003 <http://www.omg.org/mda> (Last Access: August 2011).
- [28] Paternò, F. “Model-Based Design and Evaluation of Interactive Applications”, Springer-Verlag London, 1999
- [29] Puerta, A., “A model based interface development environment”, IEEE Soft. Vol.14-4, 1997
- [30] Limbourg, Q., Vanderdonck, V., Michotte, B., and Bouillon, L., “USIXML: A Language Supporting Multi-path Development of User Interfaces”. LNCS , 3425, 2005, pp 200—220,
- [31] Annet, J. and Duncan, K.D., “Task Analysis and Training Design”, Occupational Psychology, vol. 41, 1967, pp. 211-221
- [32] Hix, D., and Hartson, H.R., “Developing User Interfaces: Ensuring Usability Through Product and Process”, John Wiley and Sons, New York NY, 1993.
- [33] Propp, S., Buchholz, G. and Forbrig, P., “Integration of Usability Evaluation and Model-based Software Development”, Journal Advances in Engineering Software. Vol. 40 Issue 12. 2009, pp 1223—1230
- [34] Wiegers, K. E., “Software Requirements”. Microsoft Press, 2003, pp 234--235

REfIS: A Stage-based Methodology for Eliciting Requirements

Felipe S. Ferraz^{1,2}, Leopoldo P. Ferreira¹, Rodrigo E. Assad^{1,2}, Renato A. G. P. França^{1,2}, Silvio Meira¹

Informatics Center
Federal University of Pernambuco
Recife, Brazil

¹{lpf,fsf3,rea,srlm@cin.ufpe.br}

²{fsf,rea,ragpf@cesar.org.br}

Abstract— Eliciting requirements is one of the most important phases in software development, which can lead the project to success or to failure. Particularly, when it comes to security requirements, the main responsables for specifying software system have a lack of knowledge at security policies and the mechanisms for achieving them. This article proposes and presents a stage-based methodology called *REfIS* that aims to guide requirements engineers through the elicitation requirement process of software system. The methodology consists of three phases: (1) dispersion of knowledge about a certain universe of study through Casual Layered Analysis (CLA), (2) creation of a future scenario using Futures Wheel and (3) extraction of requirements from the analysis of the generated scenario. Finally, this methodology will be applied and validated at the initial phase of the development of a real P2P backup System in order to extract requirements.

Keywords-eliciting requirements; methodology; innovation systems.

I. INTRODUCTION

The widely accepted concept of the innovation refers to the flow of technology and information among people, enterprises and institutions as a key to an innovative process or product. It contains and gives different and new interaction between the actors involved in the process and the process itself. It brings that actor across that new experience based on his previously knowledge.

When dealing with requirements, and users needs and expectations we may be facing a infinite universe of possibilities available for the specification of those kind of system we are also facing the infinite of the unknown, sure, we are presented to new ways, process and products and with that the task of define requirements meets challenges towards its time. This short paper, is a brief introduction to a new Stage-based methodology, called REfIS, that intends to present the definitions to systems that are not thinkable working on theirs definitions using techniques used among brain storm meetings. The methodology and the paper will introduce the use of CLA and Future Wheels combined as a approach to better understand and define innovational system requirements.

II. BACKGROUND

The main objective of this methodology is to combine CLA [1] and Future Wheels [3] techniques to propose a new approach to requirement analyses. To that, this section will

shortly introduce those techniques focusing on present them as the first and second stages of REfIS.

A. Causal Layered Analysis (CLA)

Causal Layered Analysis [1] can be regarded as a sophisticated technique to organize thoughts and views about the future. Although ones affirms that this is only a way to predict the future, Inayatullah holds the idea that it can create transformative spaces for the creation of alternative futures. Particularly, this technique is less to do with forecasting methods and more with understand the present and past to build alternative future scenarios.

This technique is composed of four layers [2]: Litany, Social causes, Worldview and Myth/Metaphors. However, each one of these layers has different proposes and focus varying from different perspective of knowing. The main idea is to conduct a deep research by moving up and down these layers. Figure 1 depicts the layers of the Causal Layered Analysis.

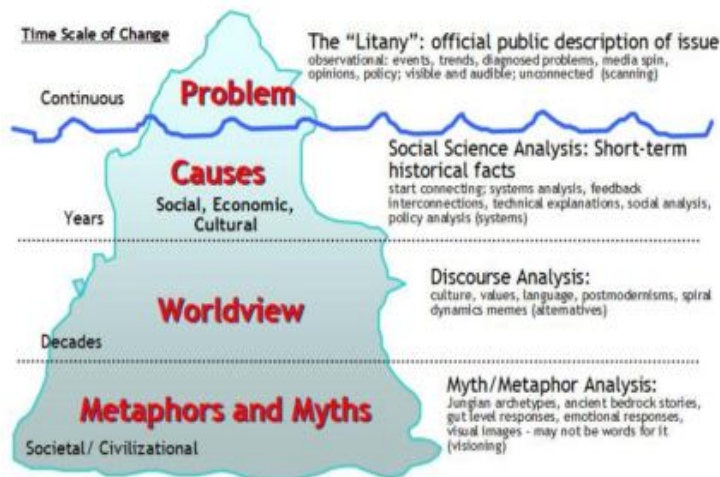


Figure 1. Layers of Causal Layered Analysis [1].

The first layer of CLA is called Litany. The participants of the meeting discuss, at this point, the public descriptions of the subject that is being analyzed. However, the view of the reality presented here are rarely questioned or used to make any suppositions about a near future.

The second layer, called Causes, is responsible for analyze the subject through the definitions found on the social sciences. At this layer, systemic events including

socials, technologic, economics, environmental, politics and historical are analyzed and its interpretations should be relied on quantitative data. Finally, the information gathered at the first layer are now questioned and explained.

The third layer is known as the Worldview. At this point of the analysis, all daily world view of the subject or event being studied should be discussed. Social-cultural factors of the population involved in the study are essentially required due to the strong impact caused by the context on the subject that is being analyzed.

Last but not least, the fourth layer, called Myth and Metaphor, is responsible for including in the discussion the myths, legends and metaphors related to the subject or event that is being analyzed owing to its influence on the beliefs of the participants as well as the society.

After all layers are analyzed and discussed, it is usually common to build a fifth layer called Future Choices where one makes statements about the events whose outcomes have not yet been observed. This forecasting process is depicted at figure 2 below.

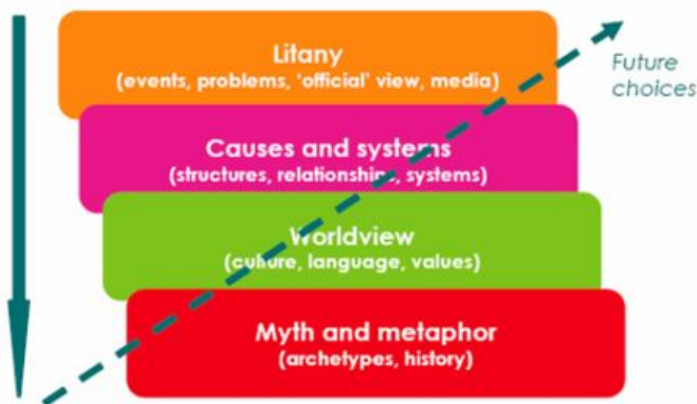


Figure 2. Building future scenarios through CLA [1].

B. Futures Wheel

Futures Wheel [3] was developed in 1971 by Jerome C. Glenn as a cross-impact analysis technique mainly used for predict impacts of future events, trends, ideas or values on a given context through a structured brainstorming process. Consequently, stakeholders are able to build relationships between events besides elicit and mitigate problems that might occur in a near future. In Glenn’s own words, the “Future wheels moves the mind from linear, hierarchical, and simplistic thinking to more network-oriented, organic and complex thinking”.

When a group (usually groups of 8 to 12 individuals) decides to brainstorm about a specific subject, it is written on piece of paper or a white-board, circled and placed in an oval at the centre by the leader of the brainstorming session or a facilitator’s guiding. After that, it is requested to the other members to say whatever comes to their mind about the item that is being shown besides raising relevant questions to the discussion. As statements are offered by the team, the leader

draws a wheel-like chart around the first item radially. At the end, the leader invites the participants to argue about the likely consequences of the new items that have just been drawn. Additionally, one can draw interconnecting lines between the primary and secondary impacts of a trend in order to establish relationship between them.

Usually, this process tends to go very quickly with the participants listing consequences with little or no evaluation. Alternatively, the wheel can be also edited in order to make it more realistic. In this approach, every statement discussed by the group has to be approved by all in order to be included in the wheel. The brainstorm session thereby tends to take more time due to the acceptance of prior criticism.

As a result of the session, the team should have developed a mind-mapping diagram that will work as a heuristic device for thinking about the future. To put it simply, the outcome of the process aims to nurture a future-conscious perspective. The final design seems like a hub with spokes radiating from it. As an example, figure 3 depicts the forecasting of the future of a videocassette recorder (VCR) device using Future Wheels technique.

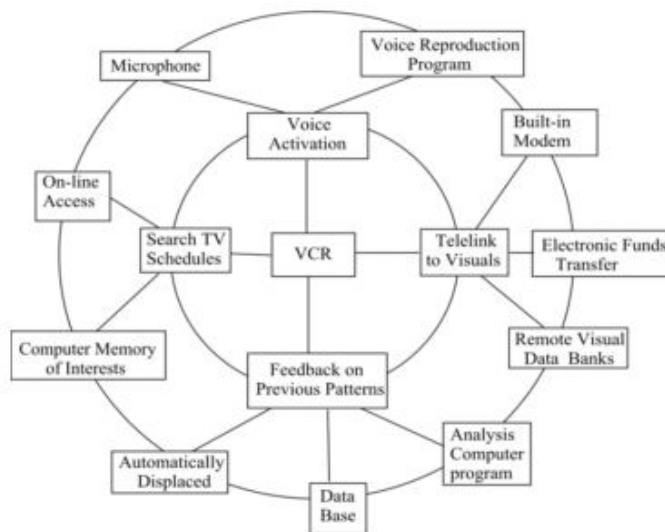


Figure 3. Example of a Future Wheel exploring the future of the VCR technology [3].

A second version of Future Wheel was also proposed by Glenn in order to consider a wider range of consequences. For example, electric engineers would naturally tend to identify technologies improvements on our VCR example and put less effort on economics or environmental consequences. Consequently, the wheel is originally divided into pre-determined sections or domains to force the team to think about the trend as broad as possible.

III. METHODOLOGY

ReFIS relies on Requirement Eliciting for Innovation Systems. Our goal is to guide stakeholders at early stages of a software development process, specially the requirement eliciting process [4] [5]. We understand that, when it comes

to Innovation Systems that claims to be built on new business models, its requirements are not trivially defined and therefore require a methodology that helps the development team to think, discuss, and analyze everything the systems should perform [6].

Trustworthy, the methodology proposed by this paper is divided into three stages; however, the first stage can be omitted if the subject to be discussed is widely known by the stakeholders.

The first stage is called the *Dispersion*. At this initial phase, the development team is invited to open a free discussion about the main subject of the software that will be soon developed. As an example, if the software concerns about a Video Conference system for facilitate company’s internal meetings, the group may choose as the central event of the discussion, actual technologies for video-conferences systems. Once the subject is chosen, a second session should be started to analyze the consequences of the implementation of this idea using Causal Layered Analysis (see section 2a). The expected outcome of this analysis is the filling of a table that contains the variables identified by each layer on CLA.

The second stage is called the *Modeling*. After the subject is widely discussed by the group, a second session is opened for modeling a scenario that describes the consequences and impacts on the software development process from the perspective of the trend analyzed on the previous stage. For modeling the scenario, we suggest the use of the second version of Future Wheels technique. Consequently, the wheel must be sectioned into domains that may have some influences on the topic discussed. As an example, we divided a generic wheel on different contexts of impacts

(educational, economical, political, etc.), as depicted on figure 4.

Finally, the group will answer the following questions for each circle in each domain:

(a) What would happen if this circle is omitted from the implementation?

(b) What requirement(s) should be implemented to prevent this from occurring?

In a future scenario, the circles can be joined into one single circle in order to define a broader requirement that cover more than one consequence.

At the end of the process, the group is expected to create a wide variety of requirements grouped by sections of interests. These requirements now can be used to compose a structured Requirement Documentation.

IV. CONCLUSIONS AND FUTURE DIRECTIONS

At this moment of our study, we are preparing a Quasi-Experiment [7] to analyze the impact on time, quality and stakeholder’s feedback of the process. This experiment will take place at the Recife Advanced Center of Research and Study (CESAR) and will be applied on a new P2P Backup [8] [9] system that is being developed by local researchers [11][12] [13]. As it is a new area of study and there is a lack of solutions using this kind of technology, stakeholders are quite unsure about what requirements should be elicited for this particular system [10]. Our main goal is to guide their requirement tasks by combining these two powerful techniques described in the past sections of this article.

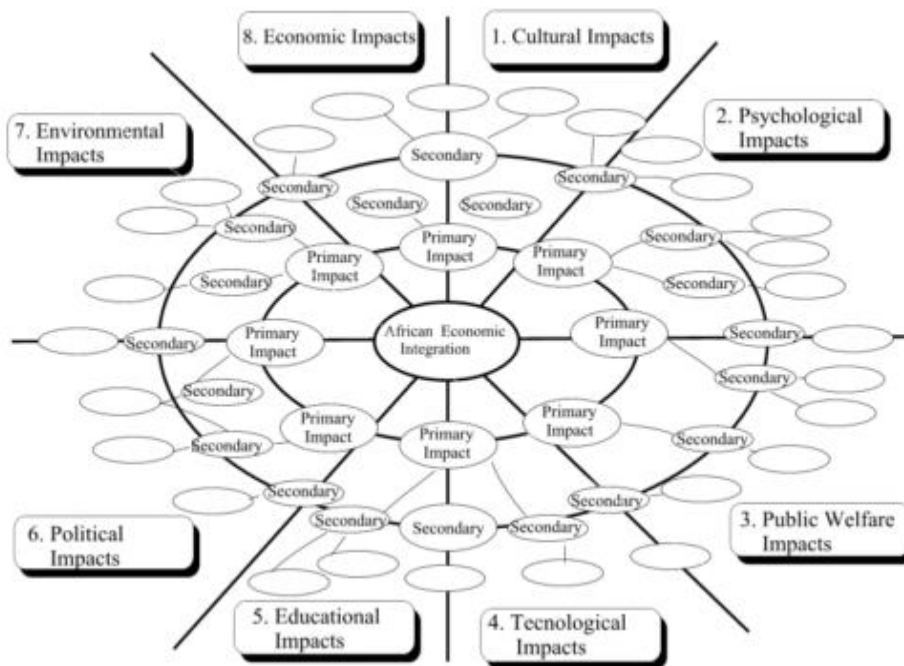


Figure 4. Example of a Future Wheel divided into sections [3]

In order to define specific goals for this experiment, we have established the following questions:

Q1 : What are the quantitative and qualitative benefits of using *Refis* from the point of the view of the stakeholders in the context of eliciting requirements in contrast of past methods used by the team?

Q2 : What strengths and weakness were identified during the technique appliance?

Q3 : What other techniques can be combined with *Refis*?

After the experiment evaluations, our academic and professional commitment is to present this work into further details to the science computer community as a new and promising methodology for eliciting requirements.

V ACKNOWLEDGMENTS

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES¹), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.

REFERENCES

- [1] S. Inayatullah, "The causal layered analysis (CLA) Reader," 2004.
- [2] S.I. This and S. Inayatullah, "Causal Layered Analysis - Deepening the future," 2005, pp. 1-22.
- [3] J.C. Glenn, "The futures wheel," Washington, DC: United Nations University.(Part of Glenn 1994a), 1994.
- [4] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," Proceedings of the Conference on the Future of Software Engineering, ACM New York, NY, USA, 2000, pp. 35-46.
- [5] G. Sindre and A.L. Opdahl, "Eliciting security requirements with misuse cases," Requirements Engineering, vol. 10, 2004, pp. 34-44.

[6] M. Hadavi, V. Hamishagi, and H. Sangchi, "Security Requirements Engineering; State of the Art and Research Challenges," Proceedings of the International MultiConference of Engineers and Computer Scientists, vol. 1, 2008, pp. 19-21.

[7] G.E.L. Cohen and G. Susan G. Jr, "Effectiveness of Self-Managing Teams : A Quasi-Experiment," Human Relations, Vol.47, No 1, 1994, pp. 13-43.

[8] C. Miller, P. Butler, A. Shah, and A.R. Butt, "PeerStripe: a p2p-based large-file storage for desktop grids," Proceedings of the 16th international symposium on High performance distributed computing, ACM, 2007, p. 222.

[9] R. Butt, T. a Johnson, and Y.C. Hu, "Kosha: A Peer-to-Peer Enhancement for the Network File System," Proceedings of the ACM/IEEE SC2004 Conference, 2004, pp. 51-51.

[10] C. Batten, K. Barr, A. Saraf, and S. Trepetin, "pStore: A secure peer-to-peer backup system," Unpublished report, MIT Laboratory for Computer Science, 2001, pp. 130-139.

[11] M. Pinheiro, R. Assad, F. Ferraz, L. Ferreira, and S. Meira . An Availability Algorithm for Backup Systems Using Secure P2P Platform. In: International Conference of Software Engineering Advances, 2010, Nice, France. ICSEA, 2010, pp. 471-478.

[12] T. Katter, R. Assad, F. Ferraz, L. Ferreira, and S. Meira. Security Quality Assurance on Web-Based Application through Security Requirements Tests: Elaboration, Execution and Automation. In: International Conference of Software Engineering Advances, 2010, Nice, France. ICSEA, 2010, pp. 272-277.

[13] F. Ferraz, R. Assad, and S. Meira . A Relating Security Requirements and Design Patterns: Working with Design Pattern to reduce Security Requirements implementation impacts. In: International Conference on Software Engineering Advances, 2009, Porto, Portugal. ICSEA, 2009, pp. 9-14.

¹INES – <http://www.ines.org.br>

A Metamodel for Representing Safety LifeCycle Development Process

Huaxi (Yulin) Zhang
IRIT, University of Toulouse
118 Route de Narbonne
31062 Toulouse Cedex 9, France
zhang@irit.fr

Brahim Hamid
IRIT, University of Toulouse
118 Route de Narbonne
31062 Toulouse Cedex 9, France
hamid@irit.fr

Damien Gouteux
IRIT, University of Toulouse
118 Route de Narbonne
31062 Toulouse Cedex 9, France
gouteux@irit.fr

Abstract—Metamodeling process supports the effort of creating flexible process models. The purpose of process models is to document and communicate processes and to enhance the reuse of processes. Thus, processes can be better taught and executed. Results of using metamodel process are an increased productivity of process engineers and an improved quality of the models they produce. However, most useful metamodels are activity-oriented, and the required concepts of safety lifecycle, such as validation, can not be easily modeled through these metamodels. In this paper, we propose a safety-oriented process metamodel to support all the requirements of safety control. As a proof of concept, we examine a process model that has several safety lifecycle requirements: the IEC 61508 safety lifecycle V-model standard.

Keywords-Safety lifecycle, Development process, Modeling, Process metamodel

I. INTRODUCTION

Over the last two decades, the need for a formally defined safety lifecycle process has emerged. This is because the inevitable requirement for better processes eventually pushed control systems to a level of complexity where sophisticated electronics and programmable systems have become the optimal solution for control and safety protection [1]. The industrial processes trend to have following characters:

- Industrial processes are becoming more and more complex.
- Increasing numbers of people and organizations are involved.
- High cost in case of an unwanted spurious process trip.
- Large consequences in case the process gets out of control.

With these emergent requirements, many safety lifecycles have been proposed by different associations, like IEC (International Electrotechnical Commission) or ISA (International Society of Automation). These safety lifecycles are adopted by different domains or enterprises with some modifications to adapt different requirements (for example, domain specific requirements). However, as the fundamental differences between traditional development process and safety lifecycle are huge, such as different kinds of safety checks and the safety relationships between these checks and phases,

to model these different safety lifecycles with traditional used process metamodel is not simple and direct. Most process metamodels such as SPEM (Software & Systems Process Engineering Metamodel), UMA (Unified Method Architecture), OPF (OPEN Process Framework), focus on modeling the process model with activity-oriented viewpoint to accommodate a large range of development processes. Furthermore, no process metamodel is rich enough or oriented to serve as the support of a safety lifecycle.

The goal of the paper is to present an ongoing work devoted to extend exiting framework with support for safety lifecycle development. That is, we propose a new safety lifecycle development processes technique in order to make easy their use in a building process of system/ software applications with safety support. The proposed vision is to use modeling techniques to obtain high level of abstractions in order to avoid the cost of building a process for each applications properties and/or for each domain. Reaching this purpose requires to get (1) a common representation of safety lifecycle process for several domains; (2) a process flexible structure; (3) guidelines for domain specific implementation of the process and (4) guidelines to guarantee the correctness of the process with regard to safety requirements. Thus, we propose a PPFS metamodel which response all these requirements which is developed under the European project TERESA, oriented to different concerns, namely safety lifecycle, pattern, repository, embedded system and non-/extra- functional properties. In this paper, we just concentrate on the aspect of safety lifecycle.

The remaining of this paper is organized as follows. Section II defines the context of the safety lifecycle and the problem definition is presented followed by a motivating example. Section III discusses the state of the art of process metamodels from the safety related viewpoint. Section IV outlines the PPFS process metamodel. Section V presents how the PPFS metamodel supports the safety lifecycle with its safety-related concepts. Section VI illustrates the PPFS metamodel by the IEC 61508 standard safety lifecycle V-model. Section VII concludes and draws future work directions.

II. PROBLEM STATEMENT

The main difficulty to overcome in the development of critical embedded systems is how to avoid the cost of building a process for properties of each application and/or for each domain. One way to obtain high level of abstraction is to make use of meta-modeling techniques. Informally, a process has several views with regard to the considered level of abstraction. This decomposition and separation of uses illuminates how to create, to specialize processes. This implies that a process is created at high level abstraction and then it will be transformed into more specific one. The common safety engineering meta-model will have to recognize the need to separate expertise on applications. As a result, individual application domains could have different safety engineering processes, for example, a domain where application engineers do not use model-driven engineering should have a more decoupled interaction with the modeling artifacts.

A. Safety Lifecycle: Definition and Concepts

The *safety lifecycle* can be defined as: an engineering process designed to achieve a risk-based level of safety with performance criteria that allow versatile technologies and optimal design solutions [2]. The risk-based levels are recognized as *system integrity level* (SIL). SIL measures the confidence which can be attributed on the fact that the integrity of the system functions conform with the requirements.

Many safety lifecycles are proposed, such as IEC 61508 [3], IEC 61511 [4], and ANSI/ISA S84.01 [5]. The differences between the safety lifecycle and normal development process are only the integration of safety related phases into process, but also the special concepts used to verify whether the safety lifecycle and the SIL requirements are correctly implemented and satisfied. Generally, there are four types of checks used to validate the safety lifecycle [3]:

- **Verification.** Confirmation by examination and provision of objective evidence that the intended functions have been correctly implemented and the requirements have been satisfied. and assurance that the safety analysis remains valid for the system as implemented.
- **Validation.** The activity of demonstrating that the safety-related system under consideration, before or after installation, meets in all respects the safety requirements specification for that safety-related system.
- **Functional safety audit.** Systematic and independent examination to determine whether the procedures specific to the functional safety requirements comply with the planned arrangements, are implemented effectively and are suitable to achieve the specified objectives.
- **Functional safety assessment.** Investigation, based on evidence, to judge the functional safety achieved by one or more E/E/PE safety-related systems, other tech-

nology safety-related systems or external risk reduction facilities.

Beyond these checks, the interaction and influences between the process phases should be considered. This means the safety relationships between checks and phases within one development process. To support these relationships, four basic flows should be modeled: control flow, retrieve flow, validation flow and verification flow. These flows represent the interactions and influences between checks and process phases. process, such as he verification flow, validation flow, etc. Thus, the same time with four types of checks, it also specify four types of flow relationships in process.

B. Motivating Example

Safety lifecycles are practiced in different domains or different enterprises with different kinds of versions [2]. The domain specific requirements lead different safety lifecycles, which are modified from the general or standard lifecycle to adapt their specific requirements. For example, the IEC (International Electrotechnical Commission) 61508 is today globally recognized and considered as the basic standard to evaluate the suppliers' products. IEC 61508 [3] recommends a V-model safety lifecycle, as shown in Fig. 1. How to define this kind of safety lifecycle model, such as IEC 61508 V-model, is raised as a problem. Thus, in this paper, we use IEC 61508 as a motivating example.

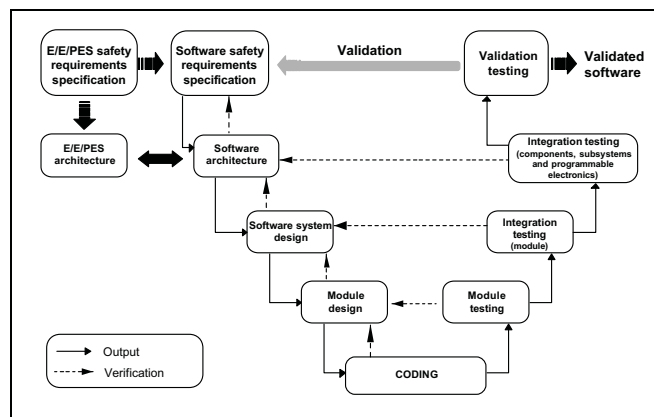


Figure 1. The V model of IEC 61508

Considering above modeling problem, we find that a safety-related metamodel, which can be applied to model these different lifecycle models, is stringently required. Thus the claim of this paper is that a safety-related process metamodel should capture the safety related process concepts to facilitate the modeling of safety-related development process. In other words, in order to model a safety lifecycle, a process metamodel should model SIL, checkpoints and different flows between checks and phases. These concepts as the minimum support and basic elements for safety-related

lifecycle, should be modeled by a process metamodel. State-of-the-art of process metamodels have been analyzed from this perspective, trying to answer the following questions:

- Do existing process metamodels support safety lifecycles?
- If so, are these metamodels can capture all required safety concepts mentioned above explicitly?

III. STATE OF THE ART OF PROCESS METAMODEL

Meta-process modeling supports the effort of creating flexible process models. The purpose of process models is to document and communicate processes and to enhance the reuse of processes. Thus, processes can be better taught and executed. Results of using meta-process models are an increased productivity of process engineers and an improved quality of the models they produce [6].

Process metamodels can be modeled from different views: activity-oriented, product-oriented and decision-oriented views [6], [7], [8]. Most process metamodels adopt the activity-oriented views, such as SPEM, UMA and OPF.

The SPEM (Software & Systems Process Engineering Metamodel) was created by the Object Management Group [9] as a *de facto*, high-level standard for processes used in object-oriented software development. The scope of SPEM is purposely limited to the minimal elements necessary to define any software and systems development process, without adding specific features for particular development domains or disciplines. The goal is to accommodate a large range of development methods and processes of different styles, cultural backgrounds, levels of formalism, lifecycle models, and communities. Thus, with SPEM, it is not easily to model all the specific concepts required by safety lifecycle.

The Unified Method Architecture (UMA) [10] has been developed within IBM¹, which is mostly used in industry to support the most important standards. The metamodel of UMA is based on SPEM, thus it has the same weakness as SPEM.

The OPEN Process Framework (OPF) is defined by OPEN [11]. Generally, it is a componentized OO development methodology underpinned by a full metamodel. The drawback of OPF is just like above twos.

Thus, we can find that these metamodels are not designed to support safety lifecycle. In some view, they permit to model the safety related concepts. With the above mentioned characters of safety lifecycle, we give a comparison between these metamodels as shown in tables I and II. Table I evaluates how these metamodels support four kinds of checks mentioned in the beginning and Table II compares these metamodels from the special required relationships of safety lifecycle. Tables I and II evaluate these metamodel from

¹UMA has been developed in a collaborative effort by the architects of the IBM Rational Unified Process (RUP).

the facility of use and the easiness of comprehension via the mentioned safety concepts. The tables use four levels to evaluate these metamodel from + to +++++. From these tables, as SPEM is a general process metamodel, we can find that it is difficult to use to model safety lifecycle. UMA and OPF are better than SPEM, however they are also not designed to orient and model safety lifecycle. We can just adjust some of their concepts to represent the safety audit and safety assessment in a more general way. For the safety relationship, all these metamodels are in same level, they do not have any specific concepts to model the safety relationship, however we can still adjust their control flow concepts to safety relationship. But the semantic information of all these safety checkpoints and relationships are difficult to reserve and illustrate in these metamodels.

Metamodel	Validation	Verification	Safety audit	Safety assessment
SPEM	+	+	+	+
UMA	++	++	++	++
OPF	++	++	+++	+++

Table I
COMPARISON OF EXISTING PROCESS METAMODELS IN CHECKPOINTS

Metamodel	Control Flow	Retrieve Flow	Validation Flow	Verification Flow
SPEM	++++	++	++	++
UMA	++++	++	++	++
OPF	++++	++	++	++

Table II
COMPARISON OF EXISTING PROCESS METAMODELS IN ASSOCIATIONS

Except above mentioned process metamodels, there are also other activity metamodels like OOSPICE [12], SMSDM [13]. Beyonds these, the other types of process metamodel such as decision based etc, do not orient to safety critical system development neither. As far as we know, the studied process metamodels unfortunately do not support safety related development process explicitly or facilitate the modeling of safety lifecycles. Beyonds these, many safety critical systems use safety instrument systems (SIS) to manage the safety lifecycle, however, these SIS do not have any process metamodel. Some works like [14] are proposed to model different standards and try to give recommendations during the application development using these standards. In conclusion, these existing metamodels are (1) not explicitly or directly describing the safety concepts as the first-classes and (2) not easily to use or comprehend, such as the different flows cannot be differentiated with each other. Thus, this analysis results in requirements for the process metamodel presented in this paper with following characteristics:

- Design with the viewpoint: safety-related.

- Support safety-related development process with its necessary required concepts: SIL, checkpoints and safety control relationships.

IV. OVERVIEW OF PPFs METAMODEL

To response the above requirements of metamodel, we propose a metamodel called PPFs. This metamodel is designed under the European project TERESA. It supports several engineering concerns, namely: safety lifecycle, pattern, repository, embedded system and non-/extra- functional properties, as shown in Fig. 2. In our works, we deal with a metamodel PPFs (Process-based Pattern Fundamental Structure), which is designed to orient several engineering concerns, namely: safety lifecycle, pattern, repository, embedded system and non-/extra- functional properties, as shown in Fig. 2. In this paper, we concentrate on its safety related concern.

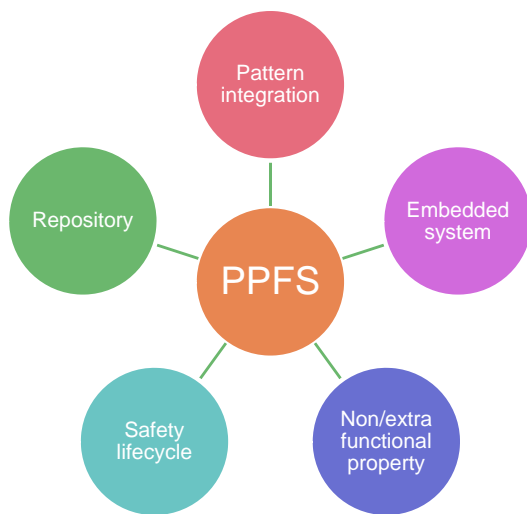


Figure 2. The characteristics of PPFs metamodel

The PPFs metamodel describes all the artifacts (and their relations) required to capture all the facets of safety-life cycle processes. It contains different packages depicted in Fig. 3 which supply different capabilities. In order to compare with other process metamodels, we give a simplified version of metamodel with necessary elements to capture safety-related concepts as shown in Fig. 4.

In this paper, we concentrate on presenting the safety-related part of the PPFs metamodel.

V. SAFETY CONCERN OF PPFs METAMODEL

In this section, the safety-related concepts in PPFs will be introduced, including SIL, the checks and safety relationships.

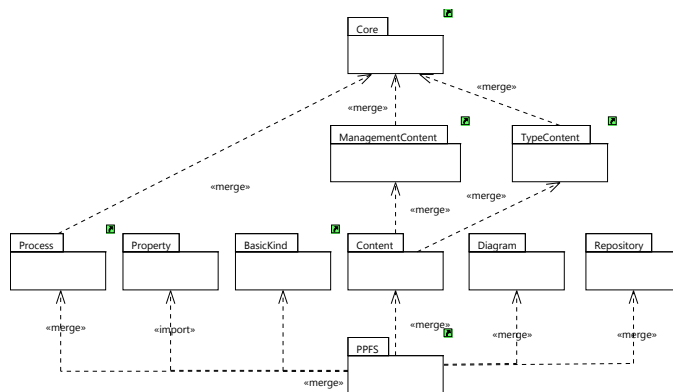


Figure 3. Structure of PPFs Metamodel

A. SIL

SIL in the PPFs metamodel is modeled as enumeration class with five levels from zero to four.

- SIL 4: the highest target and most onerous to achieve, requiring state of the art techniques (usually avoided)
- SIL 3: less onerous than SIL 4 but still requiring the use of sophisticated design techniques.
- SIL 2: requiring good design and operating practice to a level not unlike ISO 9000.
- SIL 1: the minimum level but still implying good design practice.
- SIL 0: referred to as “not-safety related” in terms of compliance.

With these five levels, the SIL attribute of process class can be set to SIL value to determine the process demand rate, which is a measure of the integrity and the stability of the process (see Fig. 4).

B. Checkpoint

Checkpoint is defined as an activity or phase which presents the safety checks in different levels of process. In other words, in the PPFs, safety checks are named checkpoints. They are used to verify whether the safety requirements are correctly implemented. To fulfill the requirements presented in Section II, we specify four kinds of checkpoints: validation, verification, safety audit and safety assessment. The structure of checkpoint and related classes is depicted in Fig. 5.

Furthermore, in order to facilitate the extension of the metamodel, the different kinds of checkpoints are defined as CheckpointKind. With this class, the checkpoint can be easily extended by different required types. The relationship is shown in Fig. 6.

In the following, we present four kinds of checkpoints predefined in the PPFs metamodel.

- 1) *Verification*: The definition of validation is a confirmation by examination and provision of objective evidence that (i) the intended functions have been correctly implemented

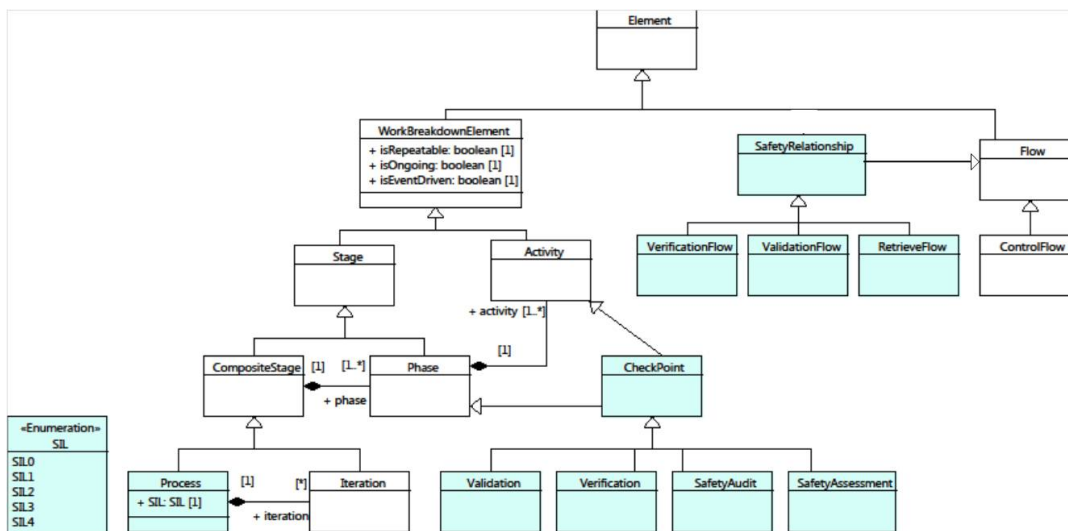


Figure 4. The structure of PPFS from safety-related viewpoint

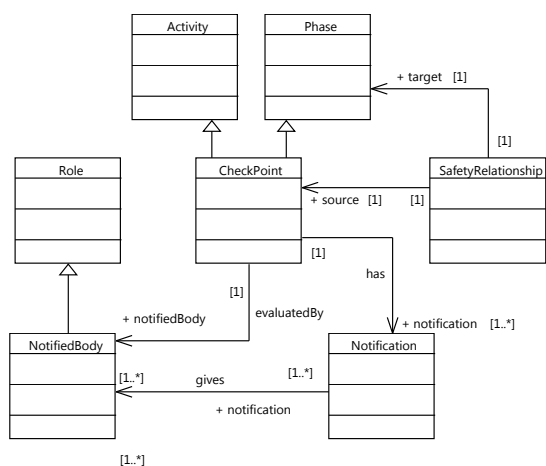


Figure 5. Structure of Checkpoint

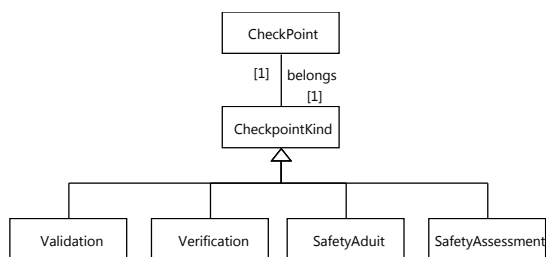


Figure 6. The types of checkpoint

and (ii) the requirements have been satisfied and (iii) assurance that the safety analysis remains valid for the system as implemented.

2) *Validation*: The activity of demonstrating that the safety-related system under consideration, before or after

installation, meets in all respects the safety requirements specification for that safety-related system.

3) *Safety audit*: Safety audit defines a systematic and independent examination to determine whether the procedures specific to the functional safety requirements comply with the planned arrangements, are implemented effectively and are suitable to achieve the specified objectives. Figure. 7 gives an example of safety audit, which serves as a checkpoint and also a phase or activity.

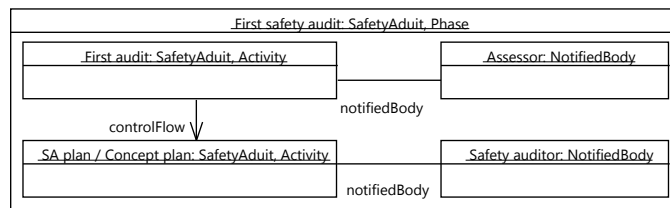


Figure 7. The example of Safety audit

4) *Safety assessment*: Safety assessment is defined as an investigation, based on evidence, to judge the functional safety achieved by one or more E/E/PE safety-related systems, other technology safety-related systems or external risk reduction facilities.

C. Safety Relationships

There are five kinds of safety relationships: internal verification, external verification, validation and retrieve flow. We precisely define different kinds of verification relationships in the PPFS metamodel.

1) *Control Flow*: is a Flow element that presents the continuation of one Work Breakdown Element to another Work Breakdown Element. The control flow presents the

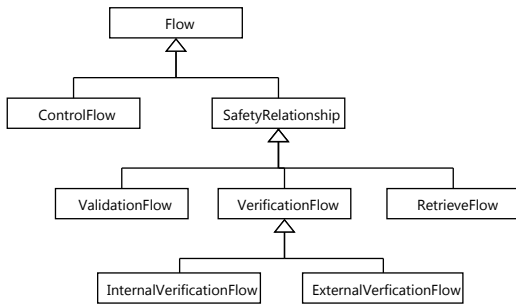


Figure 8. Structure of Flows

next work breakdown element after finishing the previous one.

2) *Internal Verification Flow*: is a Flow element that presents the internal verification relationship of one Work Breakdown Element to another Work Breakdown Element.

Internal Verification Flow represents the internal verification that we performed before start a new development phase. These actions must be carried out in order to check that the actions performed in the immediately previous phase have been done in a proper way. These actions are performed by a group independent to the design team and these actions are restricted to the left branch of the Safety Life Cycle (V-Model). These verification actions are shared between the safety audit team (Safety Auditor) and the team in charge of carried out the internal reviews.

3) *External Verification Flow*: is a Flow element that presents the external verification relationship of one Work Breakdown Element to another Work Breakdown Element.

External Verification Flow represents the normal verification performed at the right branch of Safety Life Cycle (V-Model). These actions are performed by the verification team and they start at the end of the implementation phase. The typical actions in this kind of verification are often listed below:

- Static analysis - Code coverage/Syntactic analysis
- Unit Tests
- Integration Tests
- System Tests - Validation Tests

4) *Validation Flow*: is a Flow that represents the validation relationship between two Work Breakdown Element. In safety lifecycle V-model, validation executes at the end of the implementation phase in V-model to confirm that the installed and commissioned SIFs meet the Safety Requirements Specification (SRS).

In our metamodel, although the validation concept comes from the safety lifecycle, we still make it generalization. That means validation can be concerned different perspective, not only just for safety, for example dependability validation, security validation etc.

5) *Retrieve Flow*: is a Flow that represents the retrieve relationship from checkpoints to phases or activities. The retrieve action will be proceeded when the checkpoints don't pass the examination. The process will turn back to the previous Work Breakdown Element to reexamine or redo the works. Figure 9 shows an example of retrieve flow.

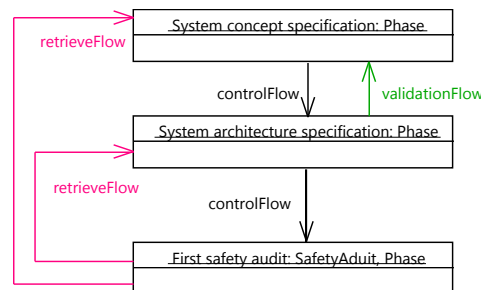


Figure 9. The example of retrieve flow

VI. AN ILLUSTRATION: IEC 61508 SAFETY LIFECYCLE

In this section, we try to illustrate the use of modeling framework by modeling IEC 61508 standard safety lifecycle V-model by the PPFS metamodel. Fig. 10 depicts the IEC 61508 V-model instantiated from the PPFS metamodel. From this illustration, we can easily demonstrate that it is more direct and precise using the PPFS metamodel to define the different safety lifecycle models. As software process covers the entire software development and contains almost all the necessary information of the development, thus it is difficult to present the entire process with all the information in one model. Normally, we use one process model to present the overall development in first level, and then decompose the process with different sub-models that correspond each phase of development. Fig. 10 is an example of the first level model of process.

VII. CONCLUSION AND FUTURE WORKS

This paper presented and illustrated our proposed PPFS metamodel from the safety-related viewpoint. Few process metamodel are rich enough or oriented to serve as the support of a safety lifecycle. Most process metamodels such as SPEM [9], UMA [10], OPF [11], focus on modeling the process model with activity-oriented viewpoint to accommodate a large range of development processes. As mentioned in Section III, a safety-oriented process metamodel is required. The PPFS metamodel fulfills all the required characteristics mentioned. It permits (1) to design process model from the safety-related viewpoint, (2) to support safety-related development process with SIL (safety integrity level), checkpoints and safety control relationships, (3) to facilitate modeling the domain specific safety lifecycle.

The PPFS metamodel presented in this paper is also illustrated by a case study of IEC 61508 standard safety-

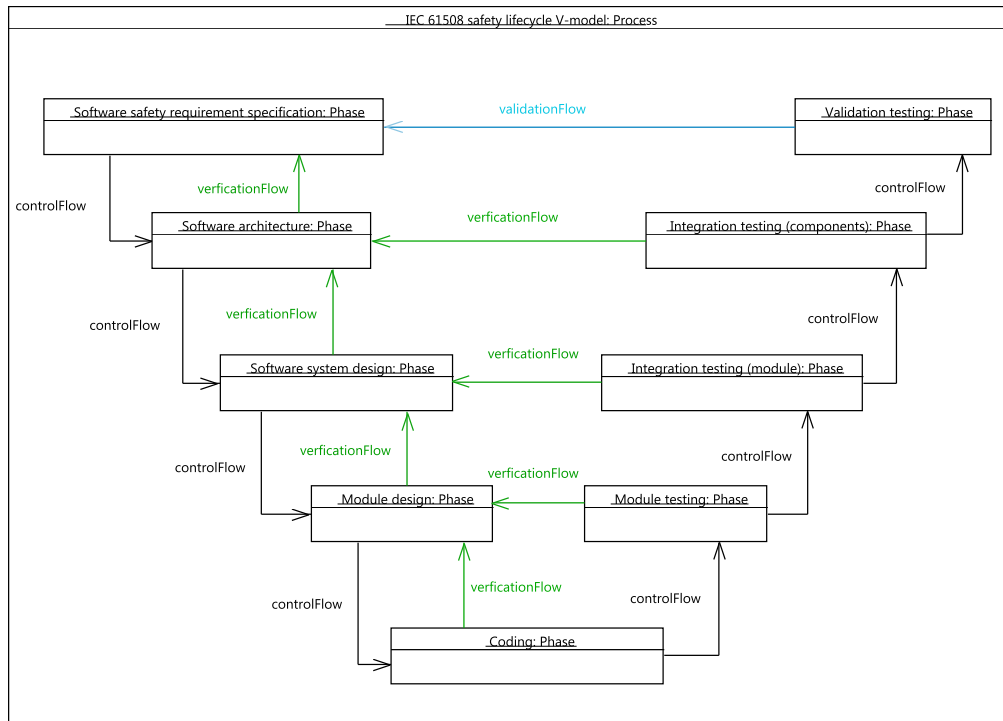


Figure 10. PPFs Metamodel instantiated by the IEC 61508 safety lifecycle.

lifecycle V-model. By this illustration, we can validate the feasibility and effectiveness of the PPFs metamodel.

As future work, we plan to extend the meta-model to refine the specifications of safety lifecycle in order to support (i) design pattern solutions, (ii) repository and (iii) extra-functional and non-functional properties.

Acknowledgements. This work is initiated in the context of SEMCO framework. It is supported by the European FP7 TERESA project and by the French FUI 7 SIRSEC project.

REFERENCES

[1] D. J. Smith and K. G. L. Simpson, *Functional Safety: A straightforward guide to applying IEC 61508 and related standards*, 2nd ed. Elsevier: Butterworth Heinemann, 2004.

[2] Exida, "Iec 61508 overview report (version 2.0)," Tech. Rep., January 2006.

[3] I. S. . IEC 61508, *Functional safety of electrical/ electronic/programmable electronic safetyrelated systems*, International Electrotechnical Commission Std., 2000.

[4] I. S. . IEC 61511, *Functional safety - Safety instrumented systems for the process industry sector*, International Electrotechnical Commission Std., 2003.

[5] A. S. S84.01, *Application of Safety Instrumented Systems for the Process Industry*, International Society for Measurement & Control Std., 1996.

[6] C. Rolland, "A comprehensive view of process engineering," in *Proceedings of the 10th International Conference on Advanced Information Systems Engineering*. London, UK: Springer-Verlag, 1998, pp. 1–24.

[7] C. Rolland, N. Prakash, and A. Benjamen, "A multi-model view of process modelling," *Requirements Engineering*, vol. 4, pp. 169–187, 1999.

[8] C. Hug, A. Front, D. Rieu, and B. Henderson-Sellers, "A method to build information systems engineering process metamodels," *J. Syst. Softw.*, vol. 82, pp. 1730–1742, October 2009.

[9] *Software & Systems Process Engineering Meta-Model Specification*, OMG, 2008.

[10] EPF. www.eclipse.org/epf.

[11] O. P. F. (OPF). <http://www.opfro.org/>.

[12] B. Henderson-Sellers and C. Gonzalez-Perez, "A comparison of four process metamodels and the creation of a new generic standard," *Information & Software Technology*, vol. 47, no. 1, pp. 49–65, 2005.

[13] *Standard Metamodel for Software Development Methodologies*, Standards Australia, 2004.

[14] L. Y. C. Cheung, P. W. H. Chung, and R. J. Dawson, *Managing process compliance*. Hershey, PA, USA: IGI Publishing, 2003, pp. 48–62. [Online]. Available: <http://portal.acm.org/citation.cfm?id=954321.954326>

On the Extensibility of Plug-ins

Vanea Chiprianov, Yvon Kermarrec
 Institut Telecom, Telecom Bretagne
 Université européenne de Bretagne
 Technopole Brest Iroise, CS 83818 29238
 Brest Cedex 3, France
 UMR CNRS 3192 Lab-STICC
 Vanea.Chiprianov@telecom-bretagne.eu

Siegfried Rouvrais
 Institut Telecom, Telecom Bretagne
 Université européenne de Bretagne
 Technopole Brest Iroise, CS 83818 29238
 Brest Cedex 3, France
 Siegfried.Rouvrais@telecom-bretagne.eu

Abstract—There are software engineering tooling problems for which the solution benefits from being encapsulated as a plug-in. Among these problems, to ensure higher leverage, there are categories for which is important that their solution is extensible. However, extending a plug-in in practice often takes a long time, requires expertise, involves hacks and produces low quality code. In this paper, we advocate that assuring early in the design that a plug-in is extensible, by providing the necessary extension points, increases its re-usability, improves its evolution, and ultimately reduces the development time of the extender plug-in. We identify categories of software engineering problems whose solutions benefit from being extensible plug-ins, and review existing approaches to extending plug-ins. Finally, we report on our experience, with some of these approaches, in extending an Eclipse plug-in for a domain specific modeling language graphical editor.

Keywords—Plug-in; extensibility; framework; software architecture; software design; design pattern; Domain Specific Language; modeling; experience report.

I. INTRODUCTION

Our knowledge for solving software engineering problems is increasingly being encapsulated in tools. These tools provide the maximum of benefits when they operate in an environment that can provide integration with existing elements such as editors, compilers, debuggers, profilers and visualizers. A major challenge is to develop tools that can span different, heterogeneous and future environments.

A software plug-in is a set of software components that adds specific capabilities to a larger software application [1]. As an auxiliary "client" module or expansion, it permits to add specific capabilities to a larger "host" software application. For example, external capabilities may be functions, services, features, or support for handling a file format. The plug-in pattern, Figure 1, from [2], presents how to design an application in order to allow its extension at runtime by dynamically loaded modules or classes. The plug-in loader is part of what is called the framework.

Well-known examples of systems based on plug-ins include web-browsers (e.g., the add-ons [3] for Firefox), graphics editing programs [4], games (plug-ins are called

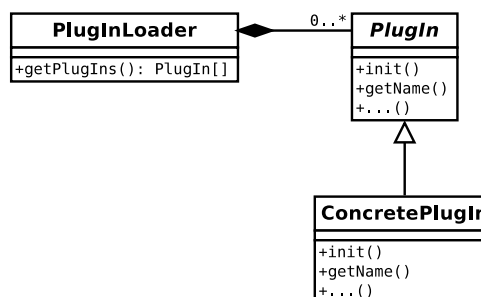


Figure 1. UML class diagram for the plug-in pattern, from [2].

mods [5]), integrated development environments (IDE) (e.g., Eclipse), tools for formal analysis and verification.

Plug-in systems are developed in order to benefit from the following advantages [6]:

- Stability of system design. New features are added through plug-ins, independent of the core functionalities of the application.
- Reduced frequency of context switches. The user remains in the same integrated environment, experiencing a feeling of continuity.
- Increased usability. The user does not need to learn to use a new environment for the system functionality.
- Re-usability of framework functionality. Basic shared functionality is provided by the framework, so liberating the plug-ins from assuring it, reducing complexity and increasing modularity and understandability [2].
- High flexibility in tool customization. The user can select exactly the plug-ins tailored to her needs.
- Interoperability. In many research communities, all tools are developed using the same framework.
- Easy extensibility [7]. New tools can be added without the need to understand the framework code. Extensibility [8] is defined as the degree of usability and safety in contexts beyond those initially intended. Extensibility includes, but is not restricted to, extensibility.

Plug-in extension may be considered as a subproblem of the customizing libraries issue. Library customization con-

sists in adding new or modifying existing pieces of code. A recent comparative study [9] surveyed most of the techniques for library customization. Despite their differences, all techniques require some sort of hole/hook point/expansion in the "host" code. The extensions have to "hook" into a main application or framework environment [10]. For this, "client" extensions must declare how they interact with the "host" and the "host" must provide interaction points. Because of this shared need of all extension approaches, we advocate that designers of the "host" extended plug-in appear as good candidates to identify and provide these extension points, for the expansion approach of their choice.

However, plug-ins present drawbacks also:

- Difficult installation of new plug-ins. Compatibility issues with already present plug-ins, versioning problems, impede users and may even provoke reliability problems if the existing application stops.
- Restriction to the chosen framework. The framework may not support adequately all the necessary functionality and/or technologies (e.g., limitation to only a certain operating system).

To improve plug-in extensibility, and based on our experience report (Section IV), we defend in this paper that extension points should be included in the very initial design of certain plug-ins (Section II) software architecture in some manner (Section III). As such, developing plug-ins from scratch or refactoring them with those extension points in mind, opens the way towards easier plug-in interoperability, extensibility, integration, installation, increases openness to several frameworks, leading to more potential uses.

II. CATEGORIES OF EXTENSIBLE PLUG-INS

It is quite safe to assume that plug-in providers would like to easily make their plug-in available in multiple frameworks. But experiments [11] on porting plug-ins to other frameworks suggest that only limited reuse is possible. Conceiving a plug-in as an extension of an adapter plug-in that takes care of framework particularities greatly increases reuse. So extension may be a great benefit or even a requirement for certain categories of plug-ins. In this section, we identify such categories of plug-ins.

One such category are tools for domain specific languages (DSLs). A DSL [12] is a language restricted to and focused on a particular domain. Implementing a DSL and its associated tools usually have as starting point an existing, more general purpose, base language and its tools. For example, SysML [13] (a modeling language for systems engineering applications) is defined as an extension of UML. Developing tools (e.g., editors, code generators) for such DSLs benefits from reusing base language tools. The base language tools are often part of a tool-set, an IDE, and implemented as plug-ins. Therefore, developing tools for DSLs based on another language often consists in extending plug-ins. The case study presented in Section IV is an example of an

editor plug-in for such a DSL. Another example of plug-in extension for domain specificity is Ginga-NCL (Nested Context Language) [14], a declarative environment for IPTV services. An NCL application itself acts as a plug-in of another parent NCL application.

Tools for coverage, profiling and the collection of different kinds of runtime information are also particularly suitable for plug-in extension. Many of them are implemented as part of an IDE, as plug-ins, to assist the developer. Also, they may present several variants that share nonetheless common functionality, and so be suitable for extension. An example is InsECTJ [15], a framework which is also a plug-in. It is a set of Eclipse plug-ins for the collection of runtime information (coverage, profiling, and data values from specific points in a program execution). It defines a core plug-in, which implements the general framework and uses an extension point to expose its functionality to specific probe inserters. A probe inserter is an instrumentation module that implements the extension point in the core plug-in. Probe inserters are bundled in a second Eclipse plug-in.

In the Web development community, there are numerous web browser plug-ins that are extended at their turn. For example, Firebug [16] is a Firefox add-on for editing, debugging, and monitoring HTML, JavaScript and other Web languages. It has a number of extensions [17] that typically come in the form of Firefox add-ons. For example, Firebug Code Coverage is a Firefox add-on and Firebug extension that adds entry function code coverage for JavaScript code.

Another category consists of tools that implement different strategies (cf. the Strategy pattern [18]) as part of a tool-chain. For example, the RDB2RDF Plugin [19] is an Eclipse plug-in that supports the standard relational database schema (RDB) to Resource Description Framework (RDF) Mapping Language (R2RML). R2RML mappings provide the ability to view existing relational data in the RDF data model, expressed in a structure and target vocabulary of the mapping author's choice. New mapping algorithms can be added by the user through the implementation of an interface.

Plug-ins that are ported to other frameworks constitute another category for which plug-in extension is beneficial. To achieve more reuse when porting plug-ins to another platform, [11] propose to construct an adapter layer, written in a language supported by the framework, and conforming to the frameworks plug-in interface. The adapter in turn communicates with the plug-in through, for example, messages or remote procedure calls. We propose to construct the adapter as an extensible plug-in, so greatly simplifying the communication between the adapter and the plug-in. An adapter extensible plug-in will also reduce the *Restriction to the chosen framework* drawback of plug-ins (cf. Section I).

The need for extensible plug-ins is a real one, as shown by the numerous examples in different categories we have identified here. And even more categories should be identified. They can then be used by designers to decide if their

plug-in belongs to one of them. If it is the case, designers know there are high chances extensibility will be needed for their plug-in. So they can create an architecture accordingly.

III. PLUG-IN EXTENSION METHODS

To implement plug-in extension, common approaches, inspired from code generation techniques, are:

- Hacks. Though undesirable, many programmers use them. Hacks add code in many places, which reduces readability, maintenance, re-usability;
- C-style preprocessor directives. Simple `#ifdef`'s can be used to include or exclude code, but readability is low and errors can be introduced easily;
- Object-Oriented Programming (OOP). Interfaces and (multiple)-inheritance provide variation/expansion points. They can be used at design time of the "host" plug-in to provide expansion points for the "client" code. Once the expansion points in the "host" are in place, the expansions consist of only adding new classes. This provides clear separation, facilitating readability, maintenance, extensibility. However, supplementary levels in inheritance hierarchies may result in loss of performance at run time;
- Feature-Oriented Programming [20]. A feature is an increment in program functionality. Feature interaction with core functionality and other features is defined in "lifters". At its essence, extending code this way is the same as OOP method overriding;
- Aspect-Oriented Programming [21]. An aspect is a supporting function, separated from the main logic. Aspects are added to main logic at various joint points. It allows adding functionality to an existing class transparently, which implies clean structuring of code. However, specifying point-cuts uniquely can be hard;
- Fragment-Oriented Program Generation. Pieces of code are combined to form a complete program. Pieces can be used by functions as regular input or output parameters. Composition of pieces is performed by plugging fragments into the holes declared in other fragments. Declaration of holes is needed.

Despite their differences, all these techniques require some sort of hole/hook point/expansion in the "host" code. Because of this shared need of all extension approaches, we advocate that designers of the "host" extended plug-in appear as good candidates to identify and provide these extension points, for the expansion approach of their choice. In this way, non-functional properties (e.g., re-usability, flexibility, extensibility) of the "host" plug-in are improved.

IV. EXPERIENCE REPORT

In this section we report on our experience with extending a plug-in for Eclipse. Eclipse is an open source, extensible IDE, but also an extensible application framework upon which software, usually as plug-ins, can be built [10]. Using

the OSGI framework to install, update or remove plug-ins on the fly, Eclipse can be easily customized. Moreover, it provides a mechanism to add features to a plug-in. In fact, there are numerous dependencies between plug-ins, some of them extending others. Eclipse allows building tools that integrate seamlessly with the environment and other tools.

A. Telecommunications Service Creation

As our research context is in the telecommunications area, we investigate domain specific models, meta-models and model transformations for service creation. We rely on a multi-layer, multi-view approach, as largely recognized in the Enterprise Architecture community. Recent efforts [22] [23] [24] of telecom operators (service providers) on defining meta-models for modeling services are indicative of the need for specific, dedicated modeling telecom languages and tools. Moreover, one of the service providers' requirements identified by [25] is to have an overall representation of service creation taking in all business, management, and technical activities. To meet these needs, we propose defining a graphical telecom DSL (with semantics implemented through code generation) as an extension of an Enterprise Architecture modeling language [26].

ArchiMate [27] is an Enterprise Architecture modeling language, a standard developed by the Open Group, with a large and growing user community. We propose defining our DSL as an ArchiMate extension. Archi [28] is a free, open source, cross-platform editor to create ArchiMate models. Archi is developed as a plug-in for Eclipse 3.6.1. To reuse existing tools for ArchiMate, we define a telecom DSL editor (Figure 2) as an extension of Archi. The editor presents the classical divisions of an Eclipse-based editor. At the left, there is the model navigator and an outline of the graphical model. The central window presents views (defined as tabs) of the graphical model. At the right, the palette offers the telecom specific concepts and relations, from which the designer can select, drag and drop the desired ones.

Our investigations and results for telecommunications [29] are out of the scope of this paper. Here we focus on tool design and development concerns [30] and especially report on our experience regarding the benefits of using extension points through three approaches presented hereafter.

B. Hacks

In the first phase, while still getting familiar with Archi's inner structure, we extended the editor by adding code in several methods from different classes. Adding a new concept in Archi means adding one class for the concept logic and two other classes for graphical purposes. Five other classes need editing. Three of these hacks are of the same type: adding a *case* statement in a *switch* instruction. Knowing all these distributed editing places requires in depth knowledge, which takes a significant time to acquire.

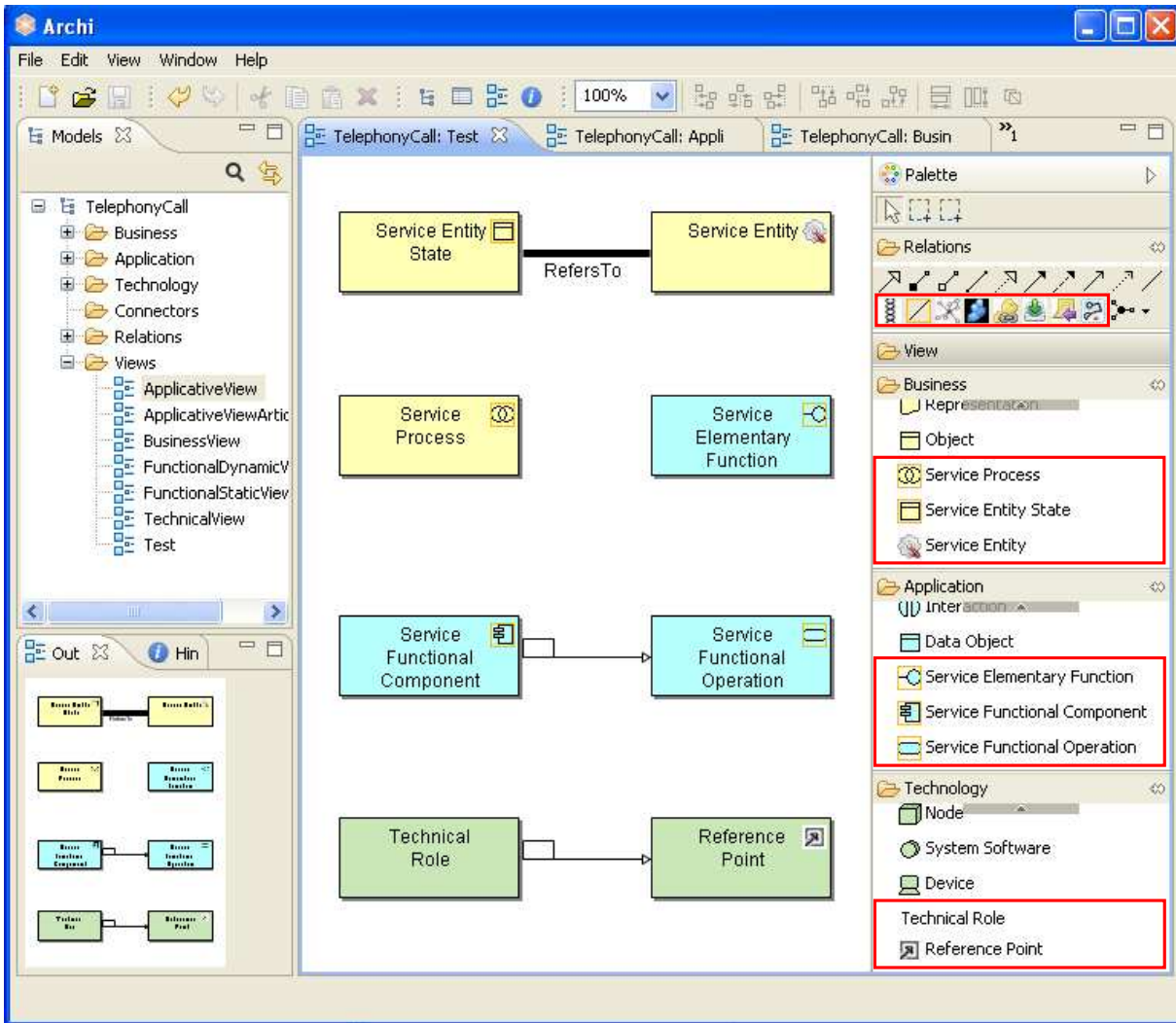


Figure 2. The Archi editor with the Telecom extension (showed in red boxes) in the palette.

After acquiring sufficient knowledge, we were able to propose a refactoring. The *switch* instruction may be replaced with an interface that is implemented by a class for each *case* statement (e.g., Strategy Design Pattern [18]). The advantage of this pattern is that code is added only in new classes, which impacts far less the existing code. However, refactoring takes a lot of time. An alternative solution would have been, for a good extensible plug-in, to have provided extension points from the original design.

C. Factory Design Pattern

Valid relations in Archi are listed in a hash-map. Adding a new relation implies updating this hash-map and the code that verifies if a particular relation has valid source and target entities. These verifications use conventions that are personal to the original developer (e.g., the pairs of valid source-target entities are coded as strings of letters). While

the resulting code is small, its readability is very low. However, an alternative way of adding new relations consists in adding a new class that verifies the type of a relation: *ArchimateModelTelecomExtensionUtils*.

This new class implements an interface, also added by us: *IArchimateModelExtensionUtils*. The advantage is that when a new extension is desired, the relations introduced by this new extension are grouped in a new class which implements the interface *IArchimateModelExtensionUtils*. The instantiation of the class that implements this interface is chosen in another class, implementing in this way the Factory Design Pattern [18]. The disadvantage is that at any given time, only one extension can be activated.

D. Eclipse Extension Points

We may want to separate the newly added classes into a new plug-in. Eclipse offers the possibility, through ex-

tensions points, to define an "extender" plug-in that adds functionality to a "host" plug-in. However, it still requires the "host" plug-in to call the extensions.

E. Lessons Learned and Insights

In all three methods presented for this case study, we had to provide expansion points in the "host" extended plug-in. This implies detailed knowledge of the extended plug-in code, which takes a long time to acquire. That is why we advocate that the original designer of the "host" plug-in should be the one that provides extension points. In this way, non-functional properties of the "host" plug-in (e.g., re-usability, extensibility) are improved and the development time of the extender plug-in is reduced.

We also emphasize the iterative manner in which the extensions were written. In the discovery phase of the "host" plug-in code, hacks were easier to use. When a more global understanding of the design was achieved, restructuring the design was envisaged and design patterns were employed. Finally, the added code could be separated in a new plug-in. We appreciate that a good extensible design for a "host" plug-in should enable the "client" plug-in developer to skip directly to what was, in our case, the third phase.

V. RELATED WORK

ObjectTeams/Java [31] is an Aspect-Oriented Programming language which introduces the concepts of roles and decapsulation. A role class can declare a base class by which this role is played (using the keyword "playedBy"). To visualize it, in Figure 1, consider "Plugin" as the base class, "ConcretePlugin" as the role class and replace the "realize" relation between them with the "playedBy" relation. A role class can adapt the behavior of its base class much like a sub-class, with the difference that roles are kept as separate entities at runtime. This results in base instances being kept intact, while roles can be added independently from each other. Of course, to have access to private data as sub-classes do, roles need decapsulation. Decapsulation means that a role class may access features of its base class even if the normal rules of encapsulation would prohibit it. Support for ObjectTeams has been added in the Eclipse OSGI framework. This generalizes the Eclipse plug-in extension mechanism with the introduction of joint points which can be seen as unanticipated extension points. This makes it possible not only to add new behavior, but also to replace functionality of a plug-in. However, the introduction of decapsulation breaks one of the most important principle and advantage of the Object Oriented Paradigm. Programmers of the extending plug-in become encumbered with the responsibility of correctly using the decapsulated data, which goes back to the need of knowing in great detail the implementation of the extended plug-in. To counter negative effects like this, we propose that the original architect of the plug-in provides the necessary extension points.

Other proposals that enable reuse of plug-ins, although not through extensibility, include, for example, [32]. The authors propose the concepts of Task Based plug-in and work-flow of Task Based plug-ins. A Task Based plug-in is a plug-in that declares the functionalities that can be executed as tasks. Using tasks, IDE users can create work-flows that execute multiple tools and integrate tool results. In this way, Task Based plug-ins can be integrated and composed through pre-defined and user-defined task flows. However, this reuse approach does not allow specialization of the behavior of a tool, which extensibility does.

Another framework for service development, developed as an Eclipse plug-in, is jABC [33]. It also contains an extensible set of plug-ins, so that the jABC models can be analyzed, simulated, verified, executed and compiled. However, it deals with services in general, while the plug-in we developed is focused on Telecommunications services.

VI. DISCUSSION AND CONCLUSIONS

The importance of plug-in extensibility is intrinsically part of the bigger discussion on software architecture (good properties). It may be argued that extensibility, and even the existence of a software architecture, plans too much in advance, pushes too much on the anticipation side. This may lead to BUFD [34] (Big Up-Front Design), massive documentation, smell of waterfall, implementing features YAGNI [35] (You Ain't Gonna Need It), huge future re-factorings because of architecture erosion. An alternative is that a metaphor should suffice, the architecture should emerge gradually sprint after sprint, as a result of a succession of small re-factorings, through an adaptive process. However, certain classes of systems, ignoring architectural issues too long, "hit a wall" and collapse by lack of an architectural focus [36]. So, there are categories of systems for which an adaptive approach may prove more appropriate (e.g., small web-based socio-technical systems) or, conversely, categories of systems for which an anticipative approach may prove more beneficial.

In this paper we addressed problems related to plug-in extension. To reduce the development time of the extender plug-in and increase quality properties (e.g., extensibility, re-usability, flexibility) of the "host" plug-in, we advocated including extension points from the start, in the original design of the "host" plug-in. We have illustrated issues and investigated solutions for the case of extending an Eclipse plug-in for a domain specific modeling language graphical editor. Through this, we hope to raise awareness among plug-in designers for domains which are highly probable to make use of plug-in extension (e.g., domain specific languages). In the future, a full comparative study of extension methods will be useful in pinpointing limitations from which current plug-in development systems may suffer and help correct them.

VII. ACKNOWLEDGMENTS

The authors would like to thank Sébastien Bigaret, Telecom Bretagne, for his helpful reviews and Phil Beauvoir, JISC CETIS, University of Bolton, for his helpful explanations, examples and suggestions.

REFERENCES

- [1] Wikipedia. (2011) Plug-in (computing). Accessed on 25.07.2011. [Online]. Available: http://en.wikipedia.org/wiki/Plug-in_%28computing%29
- [2] J. Mayer, I. Melzer, and F. Schweiggert, "Lightweight plug-in-based application development," in *Intl Conf. NetObject-Days on Objects, Components, Architectures, Services, and Applications for a Networked World*, London, UK, 2003, pp. 87–102.
- [3] Firefox. (2011) Add-ons. Accessed on 25.07.2011. [Online]. Available: <https://addons.mozilla.org/en-US/firefox/>
- [4] Photoshop. (2011) The plugin site. Accessed on 25.07.2011. [Online]. Available: <http://thepluginsite.com/knowhow/tutorials/introduction/introduction.htm>
- [5] Civfanatics. (2011) Customizing Civilization IV. Accessed on 25.07.2011. [Online]. Available: <http://www.civfanatics.com/civ4/downloads>
- [6] S. Wagner, S. Winkler, E. Pitzer, G. Kronberger, A. Beham, R. Braune, and M. Affenzeller, "Benefits of plugin-based heuristic optimization software systems," in *Proc. of the 11th intl conf. on Computer aided systems theory*, Las Palmas de Gran Canaria, Spain, 2007, pp. 747–754.
- [7] F. N. Paulisch, *The Design of an Extendible Graph Editor*. Secaucus, NJ, USA: Springer-Verlag, 1993.
- [8] JTC1/SC7/WG6, *ISO/IEC CD 25010.3: Systems and software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Quality models for software product quality and system quality in use. Version 1.46*, Std., 2009.
- [9] B. Aktemur and S. Kamin, "A comparative study of techniques to write customizable libraries," in *ACM Symposium on Applied Computing*, Hawaii, USA, 2009, pp. 522–529.
- [10] D. Rubel, "The Heart of Eclipse," *Queue*, vol. 4, pp. 36–44, 2006.
- [11] N. Sawadsky and G. C. Murphy, "Fishtail: From Task Context to Source Code Examples," in [37], 2011.
- [12] A. V. Deursen, P. Klint, and J. Visser, "Domain-specific languages: an annotated bibliography," *SIGPLAN Not.*, vol. 35, no. 6, pp. 26–36, 2000.
- [13] OMG, *Systems Modeling Language, Version 1.2*, Std., 2010.
- [14] M. F. Moreno, R. S. Marinho, and L. F. Gomes Soares, "Ginga-NCL Architecture for Plug-ins," in [37], 2011.
- [15] A. Seesing and A. Orso, "InSECTJ: a generic instrumentation framework for collecting dynamic information within Eclipse," in *Proc. of the eclipse Technology eXchange (eTX) Ws at OOPSLA*, San Diego, CA, USA, 2005, pp. 49–53.
- [16] Firefox. (2011) Firebug. Accessed on 25.07.2011. [Online]. Available: <https://addons.mozilla.org/en-US/firefox/addon/firebug/>
- [17] Getfirebug. (2011) Firebug/Extensions. Accessed on 25.07.2011. [Online]. Available: http://getfirebug.com/wiki/index.php/Firebug_Extensions
- [18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.
- [19] P. E. Salas, E. Marx, A. Mera, and J. Viterbo, "RDB2RDF Plugin: Relational Databases to RDF Plugin for Eclipse," in [37], 2011.
- [20] C. Prehofer, "Feature-oriented programming: A fresh look at objects," *ECOOP*, vol. 1241, pp. 419–443, 1997.
- [21] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin, "Aspect-oriented programming," *ECOOP*, vol. 1241, pp. 220–242, 1997.
- [22] E. Bertin, S. Bécot, and R. Nedelec, "Applying Enterprise Architecture Principles to Telco Services," in *14th Intl Conf. on Intelligence in Next Generation Networks*, 2010, pp. 1–6.
- [23] A. Ahuja, J. Simonin, and R. Nedelec, "MDA tool for telecom service functional design," in *Proc. of the 4th Euro conf. on Software architecture*, Copenhagen, Dk, 2010, pp. 519–522.
- [24] J. Simonin, E. Bertin, Y. L. Traon, J.-M. Jezequel, and N. Crespi, "Business and information system alignment: A formal solution for telecom services," *Intl Conf. on Software Engineering Advances (ICSEA)*, pp. 278–283, 2010.
- [25] J. Hällstrand and D. Martin, "Industrial requirements on a service creation environment," in *Proc. of the 2nd Intl Conf. on Intelligence in Broadband Services and Networks: Towards a Pan-European Telecommunication Service Infrastructure*, Aachen, Germany, 1994, pp. 17–25.
- [26] V. Chiprianov, I. Alloush, Y. Kermarrec, and S. Rouvrais, "Telecommunications Service Creation: Towards Extensions for Enterprise Architecture Modeling Languages," in *6th Intl Conf. on Software and Data Technologies (ICSOT)*, Seville, Spain, 2011, pp. 23–28.
- [27] Open Group, *Archimate 1.0 specification*, Std., 2009.
- [28] Cetis. (2011) Archi. Accessed on 25.07.2011. [Online]. Available: <http://archi.cetis.ac.uk/>
- [29] V. Chiprianov, Y. Kermarrec, and P. D. Alff, "A Model-Driven Approach for Telecommunications Network Services Definition," in *Eunice'09: Proc. of the 15th Open European Summer School and IFIP TC6. 6 Ws on The Internet of the Future*, vol. 5733, Barcelona, Spain, 2009, pp. 199–207.
- [30] V. Chiprianov, Y. Kermarrec, and S. Rouvrais, "Meta-tools for Software Language Engineering: A Flexible Collaborative Modeling Language for Efficient Telecommunications Service Design," in *FlexiTools Ws at the 32nd ACM/IEEE Intl Conf. on Software Engineering (ICSE)*, Cape Town, South Africa, 2010.
- [31] S. Herrmann, C. Hundt, and C. Pfeiffer, "Eclipse plugin adaptation with Equinox and ObjectTeams/Java," in *Eclipse Technology eXchange Ws at ECOOP*, Nantes, France, 2006.
- [32] L. Mariani and F. Pastore, "Supporting Plug-in Mashers to Ease Tool Integration," in [37], 2011.
- [33] B. Steffen, T. Margaria, R. Nagel, S. Jörges, and C. Kubczak, "Model-driven development with the jABC," in *Proc. of the 2nd intl Haifa verification conf. on hardware and software, verification and testing*, Haifa, Israel, 2007, pp. 92–108.
- [34] Wikipedia. (2011) Big Design Up Front. Accessed on 25.07.2011. [Online]. Available: http://en.wikipedia.org/wiki/Big_Design_Up_Front
- [35] R. E. Jeffries. (2011) Youre NOT gonna need it! Accessed on 25.07.2011. [Online]. Available: <http://www.xprogramming.com/Practices/PracNotNeed.html>
- [36] P. Abrahamsson, M. A. Babar, and P. Kruchten, "Agility and Architecture: Can They Coexist?" *IEEE Softw.*, vol. 27, pp. 16–22, 2010.
- [37] J. Bishop, K. Breitman, and D. Notkin, in *1st Ws on Developing Tools as Plug-ins (TOPI) at the 33rd Intl Conf. on Software Engineering (ICSE)*, vol. (in press), Hawaii, USA, 2011.

Effective Task Allocation in Distributed Environments: A Traceability Perspective

Salma Imtiaz, Naveed Ikram
 Department of Software Engineering
 International Islamic University
 Islamabad, Pakistan
 salma.imtiaz@iiu.edu.pk, naveed.ikram@bcs.org

Abstract— Task allocation or work assignment in Distributed Environments is a challenging task due to intricate dependencies between distributed sites and fundamental requirement of multifarious information. Conway’s law relates product architecture to communication and coordination needs of the people, whereas Parnas argues that communication and coordination needs give rise to technical dependencies. Product structure is depicted in its architecture, which in turn, consists of multiple views based on different perspectives. These views which are used to model different concerns of various stakeholders are inter-related. Task allocation depends on information about different architectural views and their interrelationship. Traceability links between various views can be used to model this interrelationship. There is a need to identify the traceability support between different architectural views to determine the extent of linkage between them. Task allocation is also dependent on factors not depicted in product architecture such as temporal and cultural dependencies between distributed sites. These dependencies highlight the need of an effective and sound task allocation strategy for distributed environment. A well conceived task allocation strategy will reduce various dependencies between sites resulting in effective task allocation and smooth distributed development. This paper analyses the dependencies/factors that should be considered for task allocation, the current task allocation strategies and their limitations and the traceability support between various views to identify gaps required to be filled.

Keywords-Task Allocation; Architectural View; Distributed Development.

I. INTRODUCTION

Allocation of task to distributed teams is a complicated and difficult affair as it involves enlarged time and space dimensions while adequate information of distributed sites is lacking [1][2][3]. Current literature adequately identifies the temporal, cultural, knowledge base, communication, coordination and other dependencies, which combined with various other factors, make the task allocation problematic [4]. Currently, task allocation is mostly done with focus on module or component dependencies overlooking most of the above mentioned important factors. This results in inadequate task allocation.

While considering a mechanism to bridge the gap caused by geographic and cultural barriers in distributed development, we find that architecture plays an important role in this regard. It acts as a central knowledge and coordination mechanism [3][5]. However, the architecture of a system facilitates identification of some but not all the dependencies. For example, temporal, cultural and knowledge dependencies, which are also critical for an effective task allocation strategy, are not visible in architecture.

Task allocation is also important for co-located development, but it acquires a critical value in a distributed setting. Distributed teams need to intercommunicate and coordinate their activities to understand each other’s culture, norms, organizational structures and business process etc., while co-located teams share common social and cultural norms and have almost the same knowledge level. [6]. Lack of inter-team information, problem of mapping the system architecture to organizational structure, and time pressure are some of the important factors aggravating the complexities of a distributed environment [3][7].

The architecture view type literature highlight allocation view type as necessary to model ‘allocated to’ relationships [17]. This view type is necessary for task allocation as it presents the allocated to relationship between software elements and environmental elements [17]. The environmental elements in case of work assignment style are individuals, teams, and organizational units, etc. Thus it focuses on task allocation to teams. The software elements in work assignment view type are elements from the module and the component and connector view type, thus implicitly creating a linkage between various views. Because of this conceptual linkage, we can establish traceability relationships between views for supporting alignment between them. The architectural models surveyed and presented in Section II-A do not explicitly model this view resulting in lack of foundational information for task allocation.

II. MOTIVATION

According to Conway's law [8], the design of a system reflects the communication and coordination needs of the people. As opposed to this law, Parnas [9] argues that technical dependencies between modules give birth to communication and coordination needs. Both these statements have been validated through empirical evidence and this inter-relationship highlights the need for a clear, effective and sound strategy for task allocation in distributed environment. Considering that both these laws are true can we identify this information as early as required for effective task allocation?. Current literature points to a glaring misalignment between communication/coordination dependencies and technical dependencies; particularly, in distributed environment [10][11]. The current task allocation literature also does not encompass all the factors necessary for effective work assignment in a distributed setting.

An architecture is divided into multiple views for separating stakeholders' concerns. Modeling each concern in a separate view increases its comprehensibility, reuse and evolution [12]. Where these views are separated for understanding, proper linkage between them is also required for task allocation, evolution and view synchronization [13]. This is where traceability information comes in. This information is used to link the work assignment view with implementation view and execution view etc. to understand the effect of component and runtime dependencies on work assignment [6]. The traceability information between architectural views needs to be correct and current at all times to ensure architecture's inter-view alignment. We need traceability information within different views to ensure their synchronization in a manner that modification in one view automatically modifies similar information in other views as well. This synchronization is particularly important for re-allocation of work.

Different architectural views have been proposed by researchers and institutes for effective modeling of software architecture. Due to the unique nature of software, the scope of this work only includes architectural models specific to software systems. We have excluded architectural models such as Telemanagement Forum Views whose focus is telecommunication systems [34], Open Group Architecture Framework whose focus is enterprise architecture [35] and Zachmann's Framework, which again focuses on enterprise architecture [36]. Out of all the architectural models only five identify the need for separation of stakeholders concerns which is necessary for increased understanding, reuse and evolution of architecture. These are: SEI View Model [12], Siemens 4 View Model [12][14][15], 4+1 View Model [12][15][18], Rational ADS View Model [12] and RM-ODP [12] [14][15]. We are interested in architectural

viewpoint models which reflect different concerns separately, provide a linkage mechanism between them and focus on design of the system. SEI View Model and ISO-RM do not meet our requirement because of their independent views. Besides the focus of ISO-RM is 'development across variant domains'. The focus of Rational ADS View Model is 'requirement evolution', which is not relevant to task allocation. The only two viewpoint models which focus on architectural design are Siemens 4 View Model and Kruchten's 4+1 View Model. We have selected Kruchten's 4+1 View Model because it comprehensively describes the architecture of a system [12]. Different views of this model are designed using UML (Unified Modeling Language) which is an industry standard and a standard way to represent product architecture. It facilitates easy comprehension of different views [16]. Traceability support between these architectural views can be identified by studying the traceability support between UML models present in each view.

Current literature on architecture highlights the need for different views [18]. 4+1 Architectural View Model proposed by *Philippe Kruchten* is one such model which reflects concerns in different views [18]. It organizes the architecture using five concurrent views namely: logical view, process view, development view, use case view and physical view. All the surveyed architectural view models including 4+1 View Model lack work assignment view which is necessary for task allocation in distributed environment. The full support for this view needs to be incorporated for resolving task allocation problem in distributed development. Both, 4+1 View Model and Rational ADS View Model (extension of 4+1) consist of the deployment view which falls in the category of allocation view type. This view type is restricted to deployment on physical nodes only where deployment of work to different organizational units, teams or individuals is not modeled. Depicting work assignment view via module view is also considered a viable approach but it is also fraught with problems [19].

Task allocation is dependent upon communication and coordination needs of an organization (Conway's law) and various other factors discussed in literature survey. We present a resumé of the current literature in the following paragraphs. We have divided Section III (Literature Survey) into three subsections. Section *A* identifies different types of important dependencies existing between various distributed sites and their importance as related to task allocation. Subsection *B* highlights the current task allocation strategies used in distributed environment and their limitations. Subsection *C* identifies the traceability support present between architectural views of 4+1 Architectural View Model. Discussion is presented in Section IV whereas conclusion and future work are presented in Section V.

III. LITERATURE SURVEY

A. Dependencies between distributed sites

Important dependencies/factors for task allocation are: Knowledge base, Technical resources and Communication and Coordination [5][17]. The research work [1] also identifies other dependencies such as scheduling strategy, state synchronization and synchronizing release schedule which effect the task allocation in varying degrees. Most of the dependencies except cultural and temporal also affect task allocation in a co-located environment, but their impact is more pronounced in distributed development. Distributed teams are not only separated by geographic distances but they also differ in knowledge base, technological expertise, organizational structures, temporal, communication and coordination aspects, socio-cultural norms and business processes [4][22]. All these dependencies/factors exercise considerable influence on the task allocation and their deliberation is essential.

B. Current task allocation strategies

Currently _ different task allocation strategies are being used in distributed environment. These are Modular Structures (Functionality based and Product based), Phase based structures (Process based), Functional Expertise based Structures, Customization based Structures and Follow the Sun Configuration (Overnight gain effect) [20][22][23][24]. It is evident that these strategies focus on only one criterion and ignore other important factors while assigning tasks to remote teams. Even if these strategies are used in conjunction with each other, some important factors like communication/coordination and cultural dependencies get ignored. Some surveys [20] also recommend that culture, product architecture, willingness to work and mutual trust must be included in our deliberations for task allocation. It is, therefore, important that a comprehensive strategy be worked out by including all relevant factors.

C. Current traceability support between views

Architecture is affected by communication and coordination needs of the organization. If tasks are allocated according to Conway's law then the development view will change with change in communication and coordination needs. This change will also trigger change in other views such as deployment view, execution view and vice versa. There is a need to update linked views to incorporate the change effectively. This support can be given with help of traceability information. Availability of traceability links between various views will ensure the architecture's inter-view alignment.

Work in the field of traceability between architectural views is carried out for different purposes such as concern evolution, requirement evolution and impact analysis etc. Traceability information between architectural views ensures consistency [16][25][26][27]. This linkage information can be used for task allocation/re-allocation in distributed teams [6] in a manner that it supports timely communication and coordination where necessary.

We have divided the literature survey on the basis of its focus of traceability support between UML diagrams and architectural views.

The focus of research [28][29] is requirement traceability. Research work [28] proposes an approach to provide traceability between requirements and UML diagrams using the Z Notation and XML. The UML diagrams included are use cases, class and sequence diagram. Traceability rules are defined to specify the above mentioned diagrams in Z language. The formal specification of the diagrams is then converted to XML schemas and traceability information is generated along with identification of missing requirements, inconsistent implementation and incomplete coverage. Traceability between use case, sequence and class diagram is also supported [29][30] via explicit saving of traceability links and via guided software production process respectively. A framework for the purpose of requirement tracing is presented [29]. The explicit link saving is performed via stereotypes and can help in change tracking as well as influence analysis. A supporting tool "Tracer" for implementing the framework is also presented. The guided production process moves from a requirement model (made using TRADE) to a conceptual model (made using Object Oriented method). The three views of the Object Oriented method include object model, dynamic model and functional model. These views include class diagram, state diagram, collaboration diagram and sequence diagram. We identify the responsibility in each use case as client (which invokes the responsibility) and implement the responsibility as server (which carries out responsibility). Responsibility is given via sequence diagram as it shows the participating classes in realizing the corresponding use case through interaction. The work of Lee et al. [31] provides traceability between sequence diagram, class diagram activity and collaboration diagram. The traceability support for activity diagram was not present in any of the previously mentioned work. The focus of the research is to evaluate the architecture for logical, behavior and performance issues. The approach works by converting UML artifacts to colored petri nets. More detailed traceability links are provided between use case model (activity and use case diagram) and object model (class and sequence diagram) with help of explicit link saving [32].

Traceability is provided for evolution of lower level models (sequence and class diagram) with respect to changes in higher level models (use case and activity models). Later evolution is supported by transversal of these links.

Traceability between use case and sequence diagram is also supported [33] via trace model and process description. The model also supports traceability with state diagram for the purpose of impact analysis of functional system requirements for embedded systems. It provides semi automatic traceability with help of prototype tool.

Traceability between class, component and deployment diagram is only performed by Bedir et al. [26]. The purpose of the work is to support concern traceability. A Concern Traceability Meta Model (CTM) is proposed which is used to model concerns, architectural elements (entity or relationship), and trace links between architectural views. The work is validated via case study of climate control system with the help of different change scenarios. The CTM is implemented using XML document type definitions (DTD). The instantiation are provided to support traces using XQuery. The traces are automatically identified using generic and specific queries written in XQuery and the results are shown in XML.

Our literature survey has revealed that there is very little traceability support between views with respect to the levels of traceability highlighted in [16]. The traceability links which are maintained or are implicitly present are between use case, class and sequence diagrams which are part of

logical, process and use case view. Although the traceability support between above mentioned diagrams is present but the focus of research work is very narrow. Table 1 presents the results of the literature survey focusing on general and implementation information of each traceability technique/method.

IV. DISCUSSION

A survey of the current literature reveals that there is a linkage gap between different architectural views. Most of the work has been performed between use case, logical and process view. The work of Bedir et al. [26] whose focus is logical, implementation and physical view provides concern evolution but whether it is extendable to support full traceability across all views, for the purpose of our research is still to be seen. The physical view presented [26] concentrates on allocation of software units to hardware nodes disregarding work assignment style.

The literature survey also highlights the need for validation of traceability work to identify its usefulness for task allocation in distributed environment. Even if the traceability links within these views are identified it is not possible to relate them to work assignment view as this view is not designed in any of the viewpoint models. Moreover, the focus of research work surveyed ranges from concern evolution to impact analysis and architectural evaluation etc. thus providing traceability support only to solve the specific issues.

TABLE 1: EVALUATION OF LITERATURE ON GENERAL AND IMPLEMENTATION DETAILS

		General Information			Implementation Details				
Focus of Paper		Paper ID	Author(s)	Year	Architectural View	UML Models Covered	Implemented via	validation	Tool Support
Requirement Traceability		[28]	<i>S. Sengupta et al.</i>	2008	use case , process and logical view	Use case, Sequence and Class diagram	(Framework) XML, Z notation,	Limited Case Study	Apache Xerces DOM Parser
		[29]	<i>T. Tsumaki, Y. Morisawa</i>	2000	use case, process and logical view	Use case, Sequence and Class diagram	(Framework) Business Object Modeling and Design Methodology	Case Study	Proposed Tool (Tracer)
Concern Traceability		[26]	<i>B. Tekinerdogan et al.</i>	2007	logical view, implementation and physical view	Class, Component and Deployment diagram	(Meta Model) X-Query	Case Study	Research Tool M-Trace
Other	Evaluating Architecture	[31]	<i>L. W. Wangenhals et al.</i>	2002	Logical view, process view	Collaboration, Sequence, Activity and Class diagram	(A Process) Colored Petri Nets, Algorithm for Conversion to Executable Models	Example	No
	Supporting Evolution in OO development	[32]	<i>H. Omote et al.</i>	2004	Use case, logical and process view.	Use case, Activity, Class and Sequence diagram.	Via Stereotypes of UML	Example	No
	Impact analysis	[33]	<i>A. Von</i>	2002	Use case, logical and process view.	Use case, Class, Sequence and State diagram	(Trace Change Approach, Trace Model)	Experiment	Case Tool, St P/UML and Prototype Tool
	Moving from Requirements to a conceptual schema in a traceable way	[30]	<i>E. Insfran et al.</i>	2002	Use case, logical and process view.	Use case, Class and State diagram	(Conceptual Modeling Approach) TRADE and OO Method	Used in two Medium Sized Projects	Case tool

V. CONCLUSION AND FUTURE WORK

Our literature survey reveals different task allocation strategies being used for assigning work to distributed sites. It also highlights various dependencies between distributed sites and traceability support between different views of viewpoint models. We find that distributed sites depend on each other for various things such as knowledge, process,

module etc. but the current task allocation strategies do not take into account all these factors. Task allocation also depends on the communication and coordination needs of the teams which is depicted in architecture of the product. Synchronizing different architectural views will result in informed and effective task allocation in a distributed environment. This synchronization is proposed by studying traceability linkage between view. The initial survey highlights inadequacy of linkage between architectural views. There is a need to identify/model the traceability links which would be specifically required for task allocation. We also need to model the work assignment view along with other views and provide synchronization between all of them. How this can be accomplished for an effective task allocation strategy will be seen in future.

REFERENCES

- [1] M. Bass, V. Mikulovic, L. Bass, J. Herbsleb, and C. Marcelo, "Architectural misalignment: An experience report," *Proceedings of Working IEEE/IFIP Conference on Software Architecture*, pp. 17-17, 2007
- [2] P. J. Compton and J. Byrd, "Utilizing cluster analysis to structure concurrent engineering teams," *IEEE Transactions on Engineering Management*, vol. 47, no. 12, pp. 269-280, 2000.
- [3] F. Salger, "Software Architecture Evaluation in Global Software Development Projects", *OTM Workshops, LNCS 5872*, pp. 391-400, 2009
- [4] J. Ralyte, X. Lamielle, N. A. Bloch, and M. Leonard, "A framework for supporting management in distributed information system development," *Proceedings of the IEEE Conference on Research Challenges in Information Science*, pp. 381-392, Marakech, 2008
- [5] J. Herbsleb, "Global software engineering: Future of socio technical coordination," *Proceedings of International Conference on Software Engineering*, pp. 188-198, 2007.
- [6] R. Sagwan, M. Bass, N. Mullick, and D. J. Paulish, "Global software development handbook," Chapter 5, Auerbach Publications, Taylor and Francis Group, pp. 37-65, 2007.
- [7] M. T. Lane and P.J. Agerfalk, "On the suitability of particular software development roles to global software development," *IEEE International Conference on Global Software Engineering*, pp. 3-12, 2008.
- [8] M. Conway, "How do committees invent?," *Thompson Publications*, Reprinted by permission of *Datamation Magazine*, 1968.
- [9] J. Herbsleb and R. Grinter, "Architectures, coordination and distance: Conway's law and beyond," *IEEE Software*, vol. 16, no. 5, pp. 63-70, Sep./Oct 1999.
- [10] M. E. Sosa, S. D. Eppinger, and C. M. Rowles, "The misalignment of product architecture and organizational structure in complex product development," *Journal of Management Sciences*, vol. 50, no. 12, pp. 1674-1689, 2004.
- [11] C. Amrit and J. V. Hillegersberg, "Mapping social network to software architecture to detect structure clashes in agile software development," *Proceedings of 15th European conference on information technology*, Switzerland, 2007.
- [12] N. May, "A survey of software architecture," In *Proceedings of the Sixth Australasian Workshop on Software and System Architectures*, pp. 13-24. Melbourne, Australia, 2005
- [13] N. Medvidovic and D. S. Rosenblum, "Domains of concern in software architectures," Published in the *Proceedings of the Conference on Domain Specific Languages*, Santa Barbara, pp. 119-218, October 1997.
- [14] D. Soni, R. L. Nord, and C. Hofmeister, "Software architecture in industrial applications," *Proceedings of International Conference on Software Engineering*, pp. 196-207, 1995.
- [15] P. Clements et al., "A practical method for documenting software architectures," Retrieved from "<http://www-2.cs.cmu.edu/afs/cs/project/able/ftp/icse03-dsa/submitted.pdf>" Accessed on 20th September, 2004, Sept. 2002 Draft.
- [16] R. Hilliard, "Using the UML for architectural description," *Proceedings of <<UML>>, Lecture notes in Computer Science*, Springerlink, vol. 1723, pp. 32-48, 1999.
- [17] V. Clerc, P. Lago, and H. V. Vliet, "Global software development: Are architectural rules the answer?," *International Conference on Global Software Engineering*, pp. 225-234, 2007.
- [18] P. Kruchten, "The 4+1 view model of architecture," *IEEE Software*, vol. 12, no. 6, pp. 42-50, Nov. 1995.
- [19] P. Clements et al., "Documenting software architectures," Published by Addison Wesley, Second Review Edition, 2002.
- [20] A. Lamersdorf, "A survey on the state of the practice in distributed software development: criteria for task allocation," *Fourth IEEE International Conference on Global Software Engineering*, pp. 41-50, 2009
- [21] H. Sertit and R. F. Pogaj, "Efficient software development organization based on unified process," *Electronics in Marine 46th International Symposium*, pp. 390-395, 2004.
- [22] B. Lings, B. Lundell, P. J. Agerfalk, and B. Fitzgerald, "A reference model for successful distributed development of software systems," *Proceedings of the International Conference on Global Software Engineering*, pp. 130-139, 2007.
- [23] I. Gortona and S. Motwanib, "Issues in co-operative software engineering using globally distributed teams," *Journal of Information and Software Technology*, vol. 38, issue 10, pp. 647-655, 1996.
- [24] R. E. Grinter, J. D. Herbsleb, and D. E. Perry, "The geography of coordination: Dealing with distance in R & D work," *Proceedings of the International ACM SIGGROUP Conference on Supporting group Work*, pp. 306-315, 1999.
- [25] E. V. Hippel, "Task Partitioning: An innovation process variable," *Journal of Research Policy*, vol. 19, issue 5, pp. 407-418.
- [26] B. Tekinerdogan, C. Hofmann, and M. Aksit, "Modeling traceability of concerns for synchronizing architectural views," *Journal of object technology*, vol. 6 no. 7, pp. 7-25, August 2007.
- [27] N. Boucke, D. Weyns, R. Hilliard, T. Holvoet, and A. Helleboogh, "Categorizing relations between architectural views," *Springer-Verlag*, pp. 66-81, 2008.
- [28] S. Sengupta, A. Kanjilala, and S. Bhattacharya, "Requirement traceability in software development process: An empirical

- approach,” The 19th IEEE/IFIP International Symposium on Rapid System Prototyping, pp. 105-111, 2008
- [29] T. Tsumaki and Y. Morisawa, “A framework of requirement tracing using UML,” Proceedings of APSEC, pp. 206-213, 2000.
- [30] E. Insfran, O. Pastor, and R. Wieringa, “Requirement engineering- based conceptual modeling,” Requirement Engineering, Springerlink Verlag, pp. 61-72, 2002.
- [31] L. W. Wagenhal, S. Haider, and A. H. Levis, “Synthesizing executable models of object oriented architecture,” Workshop on Formal Methods, Applied to Defense Systems, Australia, vol. 12, pp. 266-300, June 2002.
- [32] H. Omote, K.Sasaki, H. Kaiya, and K. Kaijiri, “Software evolution support using traceability link between UML diagrams,” Proceedings of the 6th JCKBSE, pp. 15-23, 2004.
- [33] A. Von, “Change-oriented requirements traceability support for evolution of embedded systems,” Proceedings of the International Conference on Software Maintenance, pp. 482-585, 2002.
- [34] “New generation operational support system,” Architecture Overview, Telemangement Forum, GB920, Public Version 1.5, November 2000.
- [35] “Open group architecture framework,” Retrieved from “<http://pubs.opengroup.org/architecture/togaf8-doc/arch>”, Accessed on 28th August 2011.
- [36] “Zachman framework,” Wikipedia, Retrieved from “http://en.wikipedia.org/wiki/Zachman_Framework”, Accessed on 28th August 2011.

An Agile Method for Model-Driven Requirements Engineering

Grzegorz Loniewski, Ausias Armesto, Emilio Insfran

ISSI Research Group, Department of Computer Science and Computation

Universidad Politecnica de Valencia

Camino de Vera, s/n, 46022, Valencia, Spain

{gloniewski, einsfran}@dsic.upv.es, aarmesto@novasoft.es

Abstract - The complexity and pervasiveness of software applications has increased over the last few years. In this context, software development processes have also become complex and difficult to use. It is widely recognized that requirements engineering has become a critical activity within this process. In this paper, we aim to provide a methodological approach which focuses on requirements engineering within the Model-Driven Development (MDD) context. Our approach is an OpenUP extension in which the requirements discipline is placed in the model-driven context. We believe that the integration of requirements engineering and MDD into one consistent process will provide practitioners with the benefits of both. This paper presents the definition of the proposed process, OpenUP/MDRE, including its activities, roles, and work products. We also provide an example of its use in a SOA-based software development project. The use of our approximation guides the activities of requirements engineering and promotes automation by means of model transformations.

Keywords - *Model-Driven Development, Requirements Engineering, agile methodology, OpenUP.*

I. INTRODUCTION

Software systems are becoming more and more complex, and the success of their development should not depend on individual efforts and heroics. Successful software development can only be accomplished by using a well-defined software development process. Requirements Engineering (RE) is recognized as being one of the most critical aspects of this process. Errors made at this stage may have negative effects on subsequent development steps, and on the quality of the resulting software.

Several software development approaches with which to support the development of complex systems have been proposed, of which Model-Driven Development (MDD) is one of the most promising. MDD promotes the separation of concerns between the business specifications and the implementation of these specifications on specific platforms [4]. This separation is obtained by using models that allow the level of abstraction to be elevated [9]. In this context, Model-Driven Architecture (MDA) [13] is the best known realization of the MDD. It encourages the use of models and model transformations, among several models: the Computation Independent Model (CIM), the Platform Independent Model (PIM), the Platform Specific Model (PSM) and code.

However, most MDA-based approaches focus on the transformation strategies from PIM to PSM and from PSM to

code. Unfortunately, less attention is paid to the CIM to PIM transformations upon which requirements engineering places emphasis. Loniewski *et al.* [7] have shown that there is no systematized development process that applies RE techniques in the MDD context. Although various techniques for CIM to PIM model transformations exist, those software development projects which attempt to use them often fail owing to the lack of well-defined methods and processes describing the entire development life cycle.

Another problematic issue as regards existing MDD supporting approaches is that the clear assignation of the methodology's artifacts to the MDA abstraction levels is frequently impossible. This situation arises as a result of the unclear definition of CIM and PIM, which confuses developers. It is consequently very difficult to apply the MDA life cycle by starting from the CIM level, and thus obtain most of the benefits that the MDD process should provide, i.e., automation in model transformations and traceability management.

In this paper, we introduce a methodology that provides MDD processes with an agile method which incorporates the RE activities. This method has been developed as an extension of OpenUP [2], an agile methodology, which is a minimally sufficient software development process that provides only fundamental content for small or medium size projects that deliver software as a main product. It mainly focuses on the collaborative nature of software development. The main extension is the replacement of the Requirements discipline with Model-Driven Requirements with which to elicit, model and manage requirements in the MDD context. Although OpenUP was not initially created to support MDD processes, it offers a flexible, agile and extensible means to introduce a model-driven process integrated with RE activities. This work may be an interesting contribution for those software process engineers who are faced with the challenge of guiding software development projects that follow an MDD approach from the requirements elicitation. Moreover, in projects already using the OpenUP method, the agility feature of our method makes the incorporation of the improved MDD-complaint OpenUP extension quite quick and smooth.

The remainder of this work is as follows. Section 2 provides an overview of the software process as an engineering process and also introduces some related approaches. Section 3 introduces the improved OpenUP-based methodological approach, presenting details of the content and process elements of the new Model-Driven Requirements discipline. Section 4 puts this approach into practice, discussing an example of its application to a SOA-

based middleware platform development. Finally, Section 5 contains some conclusions and future work.

II. RESEARCH BACKGROUND

This section describes the background of software process engineering along with other important approaches related to OpenUP/MDRE.

A. Software Process Engineering

When methodologies first emerged, each software development process used its own concepts and notations to define the contents of the methodology. The need to unify all these concepts and notations therefore emerged. The OMG thus introduced the Software Process Engineering Metamodel (SPEM) [12] standard. SPEM provides a complete metamodel based on the Meta Object Facility (MOF) [11] to formally express and maintain development method content and processes.

Various tools supporting this standard currently exist, one of which is the Eclipse Process Framework (EPF) [1]. EPF is a comprehensive process authoring tool which provides extensive method authoring and publishing capabilities. EPF uses the concept of a plug-in library to allow process engineers to define and extend methodologies. The fact that OpenUP is itself a plug-in library permits it to define new processes or extend existing ones. In this paper, this tool is used to extend OpenUP by incorporating a model-driven requirements engineering approach.

B. Related Work

Various existing approaches provide model-based requirements specifications incorporated into an agile methodology. The Agile Unified Process (AUP) is a simplified version of the RUP which applies agile techniques to Agile Model-Driven Development (AMDD). This approach considers the model as the principal artifact of the requirements specifications. However, its use in the model-driven context is not clear. There has been another attempt to create a lightweight methodology upon the MDA principles: OpenUP/MDD. However, its stable version has not been released. This proposal was focused solely on the transformations from the PIM to PSM level of the MDA framework. In this respect, our proposal and the OpenUP/MDD approach are complementary. The methodological approach presented in this work focuses on the CIM level transformations and generates the desired model at the PIM level.

Several attempts to establish a methodology with model-driven principles can be found in literature, but none of them focuses on the CIM-level requirements and a complete process to derive PIM-level specifications. Methods and techniques that describe particular transformations of requirements also exist (e.g., [5] or [15]), but they hardly ever possess a well-defined description of their use in a development process.

III. OPENUP EXTENSION FOR THE MODEL-DRIVEN RE

This section introduces an extension of OpenUP for model-driven requirements engineering - OpenUP/MDRE,

signifying that it focuses on a discipline for requirements engineering based on models in the model-driven context. We believe that this new discipline will improve the effectiveness of requirements engineering and will also make a significant contribution towards supporting analysts and developers by providing a well-defined process.

The first proposal of the methodology presented in [8] was defined on a base method adapted from the Rational Unified Process (RUP) [6]. This proposal was validated in a case study in an academic context and some of its weaknesses were identified. The RUP-based model-driven requirements engineering proposal was too strict and complex. This limitation has been solved by changing the base process of our approach to the agile OpenUP. Both RUP and OpenUP provide an iterative and incremental life cycle. However, OpenUP decreases the ceremony of the process, incorporating one of the strong points of agile methodologies such as Extreme Programming and Scrum. Our new method restricts neither the requirements elicitation methods, nor their specification form. The tasks provided in our approach allow an easy adaptation of any type of requirements specifications when using them in a model-driven process leading to PIM definition.

OpenUP is also available under the Eclipse Public License and is developed as a plug-in library of the Eclipse Process Framework (EPF) [2] tool, giving process engineers a powerful mechanism with which to provide content variability of its process elements by means of contribution, extension and replacement.

Figure 1 illustrates the OpenUP hump chart in which the Requirements discipline is redefined by the Model-Driven Requirements discipline.

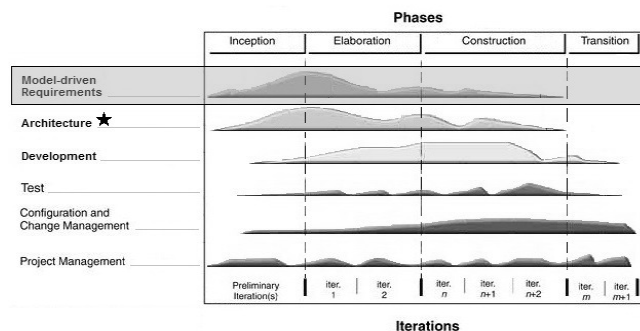


Figure 1. OpenUP Extension for Model-Driven Requirements

As is depicted in Figure 1, the redefined Model-Driven Requirements discipline is a concern from the inception phase to the construction. Since the hump chart emphasizes the workload within disciplines, the diagram shows that the new discipline is particularly important during the inception and elaboration phase, in which the product vision is created and the architecture is established.

In this new discipline, a CIM requirements model is first created on the basis of the stakeholders' needs, and this is then transformed into an analysis model at the PIM level. Specifying the CIM to PIM transformations reduces the system analysts' workload and responsibilities by including

domain experts and stakeholders in the system modeling. The analyst’s workload therefore decreases, particularly in the elaboration phase. Since we are concentrating on model use in the MDD context, the workload in the Development discipline in the elaboration phase also decreases, depending on the degree of automation of the specification generation tasks in the Model-Driven Requirements discipline.

The Architecture discipline (marked with a star) is only performed if the architecture (adequate architectural elements, models, or patterns) has not been defined. However, once this architecture has been defined, the Architecture discipline is optional and may be narrowed to refine the reference architecture provided.

Owing to space constraints, we shall comment only briefly on each activity of the main extensions of the OpenUP methodology, which is the Model-Driven Requirements discipline, pointing out the roles responsible for each task, along with input and output artifacts.

Our approach maintains the roles originally defined by OpenUP, but also introduces two roles related to the model-driven context activities: Model Analyst and Transformations Specifier.

A set of new activities has been provided and the workflow has been replaced. Figure 2 shows the Model-Driven Requirements workflow represented through a tailored version of a UML activity diagram. It is based on the OpenUP’s Requirements workflow tasks, but also introduces new activities and tasks, which are crucial to the MDD process.

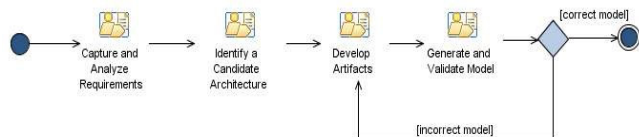


Figure 2. Model-Driven Requirements workflow

The following subsections discuss each of the activities of the proposed workflow, including a description of the new activities, accompanied by a detailed diagram of tasks, their input and output artifacts, and responsible roles. These diagrams show not only the roles which are responsible for a particular task, but also the roles who participate in its realization.

A. Capture and Analyze Requirements

This activity, which is mainly composed of tasks taken from the original OpenUP requirements discipline, involves reaching an agreement on a statement of the problem to be solved, identifying the stakeholders and clearly defining the system’s boundaries and constraints.

Stakeholders’ needs and potential features, which represent the high-level user or customer view of the system, are captured and documented in the Vision document. The potential for possible misunderstandings between the Analyst and the other different domain background stakeholders is minimized by establishing and maintaining a common vocabulary in the Glossary.

The purpose of this activity is, amongst others, to identify and capture functional and non-functional requirements for the system. The idea is to initially understand and determine the requirements at a high-level, and then describe these requirements with enough detail to validate understanding of the requirements, to ensure concurrence with stakeholder expectations, and to permit software development to begin.

The artifacts that result from the tasks performed constitute the principal input for further modeling tasks. The tasks defined in this activity are shown in Figure 3.

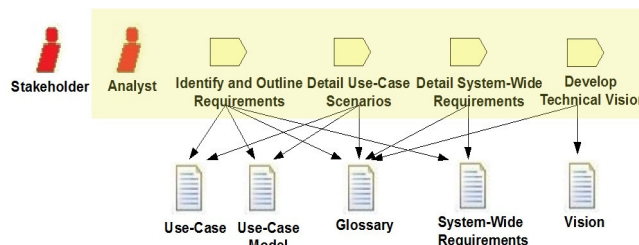


Figure 3. Roles, tasks and work products of the Capture and Analyze Requirements activity

B. Identify a Candidate Architecture

This activity is essential for the software development process. It determines the content of artifacts in the RE phase, which also conditions the MDD process to be followed. In this activity, the main architectural elements are identified and the metamodels for models at the CIM-level and PIM-level of the MDD process are established. This approach is architecture-centric, signifying that it is the architecture that demands a certain type of model to be created. The architectural pattern identified for the system becomes a basis from which to derive further analysis artifacts. For example, if the architecture chosen is Service Oriented Architecture (SOA), the model that describes requirements at the CIM-level may be given as Business Process Modeling Notation (BPMN) and it is supposed that the model at the PIM-level may be a service model. A detailed diagram of particular tasks, roles and artifacts for this activity is shown in Figure 4.

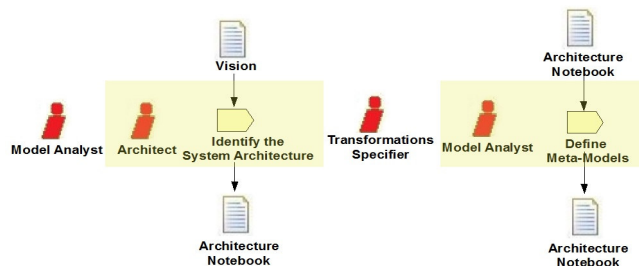


Figure 4. Roles, tasks and work products of the Identify a Candidate Architecture activity

C. Develop Artifacts

This activity, like the Identify a Candidate Architecture, is essential in our approach. The CIM-level requirements model (RM) is created in this activity, and this model

conforms to the metamodel selected for this purpose in the previous activity. Model-driven transformations are also specified and planned. In particular, the transformation language is chosen, along with the transformation automation level and tool support that are specified. Transformation rules are described in a specially prepared Transformation Rules Catalog (TRC). This document is the principal artifact supporting transformation execution, but it is also essential for the requirements traceability, which is the means used to control changes in requirements, maintain agreements with the customer and set realistic expectations as to what will be delivered. This requirements traceability is performed in a new task, Manage Dependencies, and the Model Dependencies Specification document is produced as a result of this.

A Transformation Iteration Plan (TIP) is created in this activity to describe not only the elements of a source model to which the transformation applies, but also the order of the transformation rule application. For example, if the architecture chosen is SOA, the CIM model contains BPMNs and the PIM model contains service models. An example of the transformation iteration plan could specify that the transformation rules between a BPMN of a higher level and a BPMN of a lower level should be executed before the transformations rules from a BPMN of a lower level to a service model. The tasks defined within this activity are shown in Figure 5.

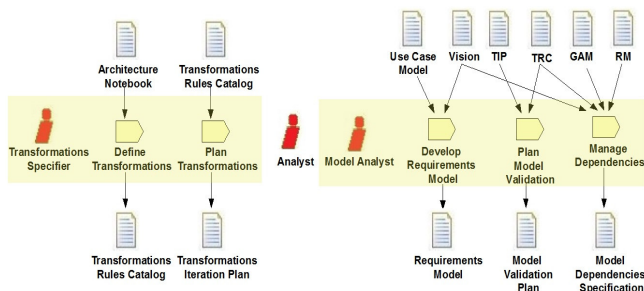


Figure 5. Roles, tasks and work products of the Develop Artifacts activity

D. Generate and Validate Model

This activity concludes the entire requirements modeling process by generating the principal artifact of the requirements engineering process, which is the Generated Analysis Model (GAM). GAM represents requirements specification at the PIM-level of the MDA lifecycle. For example, GAM can be specified by a UML sequence diagram in the case of a client-server software project, or a service model in the case of SOA platform development.

Artifacts, such as a requirements model (RM) or transformation rules catalog (TRC), developed as a result of the previously performed tasks, are the input artifacts for performing model transformations in order to create the GAM artifact. These transformations can be manual or automated, depending on their level of complexity. Their execution may be supported by appropriate tools. Although the GAM is systematically obtained by the transformation rules, we believe that it is necessary to validate it with regard

to its consistency and correctness. This type of validation should be previously described while defining the transformation rules in a separate Model Validation Plan (MVP) document. The RM can also be validated against the specific conceptual standards of the domain in which it is applied. The validation result is stored in the Model Validation Record (MVR) document. The tasks defined in this activity are shown in Figure 6.

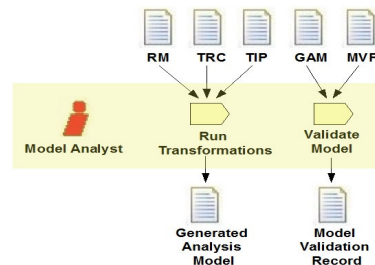


Figure 6. Roles, tasks and work products of the Generate and Validate Model activity

IV. APPLYING OPENUP/MDRE

The main objective of this section is to show the applicability and feasibility of the OpenUP/MDRE approach in the development process of a SOA-based system. In this example of methodology usage, the system specification is developed on the basis of user requirements, which were captured as user scenarios. We assume that the requirements scenarios and use cases defined in the Capture and Analyze Requirements activity have been correctly captured and documented in the initial stages of the project. These artifacts constitute the input for the subsequent model-driven process.

Each of the main tasks of the OpenUP/MDRE application is commented on in the following subsections.

A. Identify the System Architecture

Our proposal for the OpenUP extension was applied to a domain in which current systems have to deal with many complex processes, multiple stakeholder views and users, a distributed environment, changing requirements and many other factors. These factors and the domain's complexity lead to system specifications of the same complexity, and the principal issues to be considered are interoperability and distributed use of the software functionalities. These characteristics indicate that the Service-Oriented Architecture (SOA) is a primary candidate architecture. SOA strengthens such factors as reusability, scalability or interoperability. In this case, the Architecture Notebook (artifact from the architecture OpenUP's discipline) contains the SOA reference architecture description adapted to this particular project.

B. Define Meta-Models

Since SOA is the selected architecture, it demands a certain type of functionality specification. It also implies the type of models that should be used in the RE process. The Architecture Notebook therefore includes the metamodels identified for the CIM-level requirements model and PIM-

level analysis model. In this example, requirements specified as scenarios and use case models provided by stakeholders and captured by the Analyst serve to create the requirements model that conforms to the features metamodel (Figure 7.A). The most important concept of this metamodel is the Service Feature, with one of three refinement types (Decomposition, Specialization, and Implemented-by).

Since SOA was chosen as the reference architecture for the project, the PIM-level analysis models will cover the business process and service layers of the architecture. In this case, models that will be generated in the MDD process are established to conform to the BPMN process metamodel (Figure 7.B) and service metamodel (Figure 7.C).

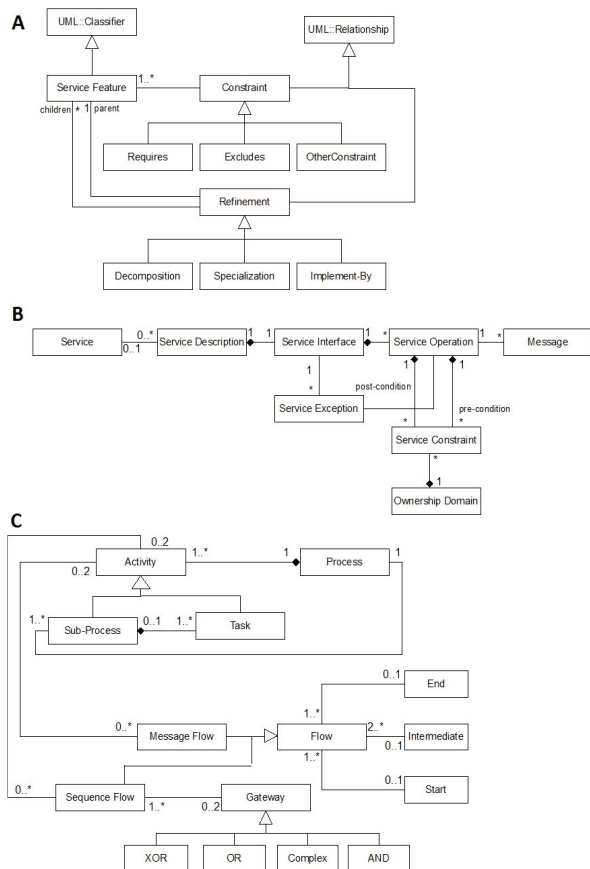


Figure 7. Simplified metamodels used for requirements specification

The most important concepts of these models are the service with its description containing operations, messages and exceptions, and also the process with the flow of activities.

C. Define and Plan Transformations

At this stage, the Transformations Specifier prepares the TRC, which documents transformations from the CIM model to the PIM-level model. In this case, these rules describe a transformation from the features model to reach target models such as BPMN and service specification. The transformation described in this example is not straight forward, but consists of two steps: one from the features

model to BPMN, and one BPMN to the service specification. In this case, the TIP document describes the order of use of specified transformations.

The transformations in this particular example are performed manually by the Model Analyst. However, in other cases they may be executed automatically through the use of tools that support a particular transformation.

An example of this feature to BPMN transformation, which is described in detail by Montero *et al.* [10], is shown in Figure 8, in which the left-hand side of the transformation presents an element of the source model and the right-hand side presents a corresponding element of the target model.

The service specification is then obtained from the BPMN model by applying one of existing techniques. In this case, the services and their operations were identified using the method described by Azevedo *et al.* [3].

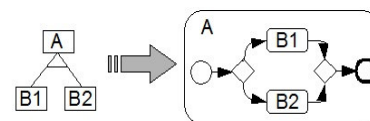


Figure 8. Example of transformation rules

D. Develop Requirements Model

Once the architecture and the metamodels have been identified, the Requirements Model is created. This is done manually. The features model, which constitutes the input for the CIM to PIM transformation, is created on the basis of scenarios and use-cases previously described by the stakeholders. Figure 9 shows an example of a features model for the system's Actor Management functionality. This functionality contains three independent and optional functionalities with which to manage actors.

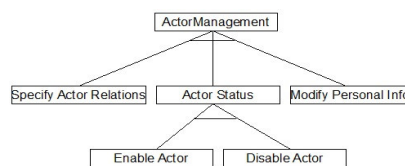


Figure 9. Example of the requirements model

E. Run Transformations

Once the transformations have been defined and planned, the Model Analyst generates the PIM-level models in order to produce the Generated Analysis Model artifact. The specification produced is the input for further design and implementation in the Development discipline. Figure 10 shows a simplified specification of the Actor Management business process represented as BPMN (Figure 10.A), along with the Actor Management service specification (Figure 10.B) that conforms to the aforementioned metamodels.

The Model Dependencies Specification document, which is prepared during the aforementioned process, includes the traceability links between the requirements and all subsequent models that are created as intermediate or final products of the model-driven process.

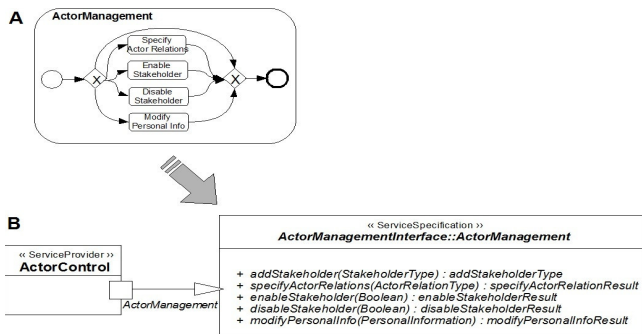


Figure 10. Example of Generated Analysis Model for Actor Management

The example presented here shows how the requirements specification process may be conducted for SOA-based systems development. Taking advantage of the model-driven requirements process signifies that it is systematized but also agile when preparing different kinds of specifications. The process is accompanied by a set of artifacts that provide well-documented guidelines for all interested project development members.

V. CONCLUSIONS AND FURTHER WORK

This paper presents an extension of OpenUP, emphasizing the use of models as requirements specifications in the context of MDD. This extension redefines the original Requirements discipline in the OpenUP and proposes a new discipline called Model-Driven Requirements. Our methodological approach is an agile RE method for project managers who would prefer to adapt a MDD RE process to particular software architecture rather than using another general approach. We believe that this flexible MDD approach is a solution to the common "one method fits all" problem of generic methodologies.

In our approach we apply model-driven techniques to extend OpenUP to support different architectures and project needs. It improves the development process defined by OpenUP in that it is not only model-based, but also model-driven. This makes OpenUP/MDRE more compliant to maturity model approaches (such as that of the MDD Maturity Model [14]) needed in industry for the incremental adoption of MDD processes. The extension was developed as a plug-in library for EPF. It includes new content elements, such as: artifacts, roles, tasks, and process elements, i.e., activities and capability patterns, to guide software engineers who attempt to follow an MDD approach in their software projects.

The application of this approach to an MDD project has been described, and shows that the use of a model-driven RE has an important influence on the entire development process.

As further work, we plan to provide a tool support with which to easily create the artifacts defined for this model-driven development process (transformation rules, transformations iteration plan, model validation plan, etc.). This will be addressed by providing document templates and creating artifacts with wizards.

Finally, we are involved in the redefinition of the OpenUP/MDRE based on the artifact-driven approach which, in our opinion, better covers the different aspects of an MDD process definition.

ACKNOWLEDGEMENT

This research work is supported by the MULTIPLE project (with ref. TIN2009-13838) funded by the "Ministerio de Ciencia e Innovación (Spain)".

REFERENCES

- [1] The Eclipse Process Framework (EPF) project, www.eclipse.org/proposals/beacon/ (last accessed 8/8/2011)
- [2] The OpenUP methodology web site (November 2010), <http://epf.eclipse.org/wikis/openup/> (last accessed 8/8/2011)
- [3] L.G. Azevedo, F. Santoro, F. Baio, J. Souza, K. Revoredo, V. Pereira, and I. Herlain, A Method for Service Identification from Business Process Models in a SOA Approach. In: Enterprise Business-Process and Information Systems Modeling, Lecture Notes in Business Information Processing, vol. 29, pp. 99-112. Springer Berlin Heidelberg (2009)
- [4] M. Azoff, The Benefits of Model Driven Development. White paper, Butler Group (March 2008), <http://www.ca.com/us/products/detail/CA-Gen.aspx> (last accessed 8/5/2011)
- [5] P. Jamshidi, S. Khoshnevis, R. Teimourzadegan, A. Nikravesh, and F. Shams, Toward Automatic Transformation of Enterprise Business Model to Service Model. In: PESOS '09: Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems, pp. 70-74, IEEE CS, Washington, DC, USA (2009)
- [6] P. Kruchten, The Rational Unified Process: an Introduction. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)
- [7] G. Loniewski, A. Armesto, and E. Insfran, Incorporating Model-Driven Techniques into Requirements Engineering for the Service-Oriented Development Process. In: ME'11: Proceedings of the 2011 Conference on Method Engineering, vol. 351, pp. 102-107, Springer Boston (2011)
- [8] G. Loniewski, E. Insfran, and S. Abrahao, A Systematic Review of the Use of Requirements Engineering Techniques in Model-Driven Development. In: Petriu, D.C., Rouquette, N., Haugen, (eds.) MoDELS. Lecture Notes in Computer Science, vol. 6395, pp. 213-227, Springer (2010)
- [9] T. Menzies, Editorial: Model-Based Requirements Engineering. Requirements Eng. 8(4), 193--194 (2003)
- [10] I. Montero, J. Pena, and A. Ruiz-Cortes, From Feature Models to Business Processes. In: Proceedings of the 2008 IEEE Int. Conf. on Services Computing, vol. 2, pp. 605-608, IEEE Computer Society, Washington, DC, USA (2008)
- [11] OMG (Object Management Group): Meta Object Facility (MOF) Core Specification Version 2.0 (2006), <http://www.omg.org/cgi-bin/doc?formal/2006-01-01>
- [12] OMG (Object Management Group): Software Process Engineering Metamodel (SPEM) (January)
- [13] OMG (Object Management Group): Model Driven Architecture: The Architecture of Choice for a Changing World (2010), <http://www.omg.org/mda>
- [14] E. Rios, T. Bozheva, A. Bediaga, and N. Guilloreau, MDD Maturity Model: A Roadmap for Introducing Model-Driven Development. In: ECMDA-FA. pp. 78-89 (2006)
- [15] L. Zhang and W. Jiang, Transforming Business Requirements into BPEL: A MDA-Based Approach to Web Application Development. In: WSCS'08: IEEE Int. Workshop on Semantic Computing and Systems, 2008, pp. 61-66 (2008)

Evidence in Requirements Engineering: A Systematic Literature Review Protocol

Talat Ambreen, Muhammad Usman, Naveed Ikram, Muneera Bano

International Islamic University Islamabad, Pakistan
 {talat.ambreen, m.usman, naveed.ikram,muneera}@iiu.edu.pk

Abstract— Requirements Engineering (RE) is recognized as one of the critical phases in software development. RE has its own journals and conferences where lots of work has been published. As the area is maturing, increasingly large numbers of empirically supported studies have been reported in RE. There is a need to synthesize evidence based RE literature. We plan to systematically investigate evidence based RE studies to see and report state of the art in evidence based RE reported research. This paper aims at providing a systematic literature review (SLR) protocol to describe a process for synthesizing the empirically supported work in the area of RE that will eventually present a state of the art of the field. This SLR intends to not only summarize the empirical data regarding RE but will also be helpful for various practitioners in this field to find out areas of RE rich in terms of tools, techniques, frameworks, models and guidelines to aid in their work. It will also facilitate RE researchers to identify knowledge gaps to recognize needs and chances for future research directions in this field.

Keywords-systematic literature review; requirements engineering; evidence-based software engineering.

I. INTRODUCTION

Software Requirements Knowledge Area (KA) is concerned with the elicitation, analysis, specification, and validation of software requirements [1]. Requirements engineering is known as the key to success to software and systems development [2].

Requirements are very fundamental aspect of a software system as Fredrick Brooks illustrates “The hardest part of building a software system is deciding precisely what to build. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later” [3].

RE is recognized as one of the important activities in software development that deals with the requirements, from their elicitation until the system is validated for completion of requirements. Software requirement elicitation, analysis of the requirements, and writing good requirements are the most difficult parts of software engineering. As Karl Wiegers [4] describes, “If you don’t get the requirements right, it doesn’t matter how well you do anything else”, because if requirements are wrongly captured or developed it results in a flawed product.

Software requirements have been considered a problem repeatedly during the past 36 years [17]. Ross and Schoman [10] broadly described the scope of requirement engineering. Since then the work progressed in this field in terms of research and development.

A lot of research has been done in all the areas of requirement engineering. Some of the famous international Journals and Conferences [5,6] of requirement engineering have published significant research in this field.

But, after all the research in this field, the question arises; how much of RE literature is supported by evidence? What is the nature of evidence in RE studies? What is the strength of evidence in RE studies? These are important questions. Answers to these questions will help RE researchers and practitioners to work with clear focus and see where more attention needs to be given. For finding solution to these questions, one needs to look for ways to gather this important information. One of the way of gathering information related to some specific topic of interest is through a systematic way of discovering, synthesizing and then reporting that information, i.e., through the methodology of SLR. Despite of significant research in the field of RE, there has not been done any effort of carrying out an SLR for the whole field of RE. The SLR described in this paper, intends to capture all the useful information related to software requirement engineering in a systematic way. This process of capturing information based on evidence is rigorous and repeatable. To the best of our knowledge, no such systematic review has been reported at this level of RE yet, so this SLR is first of its kind that will review all evidence based RE studies covering all the subareas of RE.

This paper aims at providing a systematic literature review protocol to describe a process for capturing the current empirically evaluated knowledge in the area of software requirement engineering and to recognize needs and chances for future research in this field. The rest of the paper consists of following sections; Section II presents background, Section III describes the systematic literature review protocol, while Section IV concludes the paper along with future work.

II. BACKGROUND

Requirement Engineering is an important phase in software development because a software success is strongly tied to the fulfillment of requirements as put by various stakeholders. Having recognized this fact, there is a vast amount of research available in all the areas of this field, aiming at providing specific tools, techniques, and guidelines for improving sub areas of RE. Despite of all this research, RE lacks a study describing the state of the art of this field.

To carry out such a study which is based on evidence, there are specific methodologies like Systematic Literature Review [12] and Systematic Mapping [18], which have been frequently used by many research fields like evidence-based

medicine. But they have not been much employed in software engineering for validation of empirical studies.

In the field of software engineering, there has been a new drift towards evidence-based software engineering (EBSE) [7] with an emphasis on new empirical and systematic research methods.

EBSE is concerned with capturing current best evidence from research and then integrating it with practical experience and human values in the software development decision making process [8,9]. The main tool of EBSE is Systematic Literature Review [12] which has been employed in this review.

The main motive to undertake this SLR is to discover gaps and commonalities in software requirement engineering empirical research and providing a summary of the existing empirical evidence in this field to form a stepping-stone for future research in this field and for practitioners for their practical use

III. SYSTEMATIC LITERATURE REVIEW PROTOCOL

There are three main phases of a systematic literature review process; planning, conducting and reporting of review as described by Kitchenham [12].

This paper aims at describing the first phase of the review i.e., planning the review, in the specified field of requirement engineering. The output achieved after completion of this phase is a systematic review protocol which has significance as it tells about rationale and process of carrying out the whole systematic review step by step. A systematic review protocol developed in such a way in the start of a systematic review lessens the risk of bias on the part of a researcher [12].

The steps followed in this review protocol have been developed according to guidelines provided by Kitchenham [12]. This protocol describes various steps for carrying out a systematic review that aims at reviewing primary studies related to software requirement engineering to present an outline of existing information related to this field that will eventually help in drawing a broad category of conclusions from this information.

The review intends to find a state of the art in the field of software requirement engineering and the research questions have been formulated according to the motive of the review. The research questions are:

RQ₁: What is state of the art in empirical studies of RE?

The purpose of this question is to empirically evaluate the status of the software requirement engineering and finding out future research directions in this field

For this purpose specific information will be collected through existing research papers in the software requirement engineering field by carrying out a systematic review of research papers in this field.

RQ₂: What is the strength of empirical evidence reflected in empirical requirement engineering literature?

The aim of this question is to find out the strength of empirical evidences, i.e., what we actually know regarding the evidences we collected. This question will

provide information about the strength of the studies in terms of sources of the evidence and research approach used.

A. Search Strategy

The search strategy contains different decisions like search string, resources to be searched and selection items to be searched.

This has been done in steps as:

- Identifying Major Terms from Research Question

Major terms from research questions are:

- RQ1: Software, Requirements Engineering, Empirical studies
- RQ2: Software, Requirements Engineering, Empirical studies

- Identifying Alternate Spellings and Acronym for Major Terms

Alternate Spellings and Acronym for Major Terms are:

- A: (All synonyms of requirement engineering) requirements process, requirements development, requirements elicitation, requirements gathering, requirements identification, requirements discovery, requirements analysis, requirements specification, requirements validation, requirements verification, requirements testing, requirements checking, requirements negotiation, requirements documentation, requirements management, requirements change management.
- B: (All synonyms of empirical studies) empirical, case study, experiment, industrial report, experience report, observational study, survey.

- Formulating Search String

The final search string will be like:

(All synonyms of requirement engineering ORed)
AND
(All synonyms of empirical studies ORed)

The search string will be modified according to the search criteria provided by different sources like Compendex, ACM digital Library, etc. during the conduction of review.

- Deciding Resources to be Searched

Resources to be searched include:

IEEE Explore, ACM digital library, ScienceDirect, SpringerLink and Compendex.

- Selecting Items to be Searched

The items to be searched include: Journal articles, Workshop papers and Conference papers.

- Deciding Language of Review Studies

The research papers in English language will be selected for review.

- Deciding Publication Period

Publication period included in the review will be from the start period as specified in the resource to be searched up to year 2011.

B. Publication Selection

• Inclusion Criteria

The study will be included that fits the criteria as:

- The study is about RE
- OR the study is about any of sub-areas of RE
- AND the study has empirical evidence.

• Exclusion Criteria

The study will be excluded that:

- Is in the form of books or thesis or unpublished articles
- OR the study that does not directly address RE or any of its sub areas
- OR the study that lacks empirical evidence

• Selecting Primary Studies

The primary studies included in the review will be selected in two iterations:

- Level 1 screening

Initially the papers will be selected by reviewing the title, keywords and abstract. By doing this, the studies which are relevant to the research questions will be selected and those lacking this relevance will be excluded. If there is any uncertainty about any paper for inclusion/exclusion in level 1 screening then the paper will not be excluded at this level rather it will be iterated through level 2 screening.

- Level 2 screening

The candidate primary studies selected initially in level 1 will then be checked against the aforementioned inclusion/exclusion criteria by reviewing the studies thoroughly by going through their full text. A secondary reviewer will then review the studies against the Inclusion/Exclusion criteria to cross-check the results of inclusion and exclusion. Studies that lack empirical evidence or that are not about RE will be excluded in this process. And those matching with the inclusion criteria will be selected as primary studies for the review.

C. Publication Quality Assessment

Quality Instrument will be used to evaluate quality of empirical evidences as described in the primary studies. The Quality instrument used in this review consists of 5 sections; a section having general checklist items which are applicable to all the studies included while other 4 sections are specifically for various research method used in the study i.e., experiment, survey, case study and experience report. These criteria have been adopted from SLR guidelines [12] [13] [14] [15] [16]. The formulation of this checklist is a joint group effort of various researchers. Table 1 shows the detailed checklist.

Generic	
Are the aims clearly stated?	YES/NO
Are the study participants or observational units adequately described?	YES/NO/PARTIAL
Was the study design appropriate with respect to research aim?	YES/NO/PARTIAL

Are the data collection methods adequately described?	YES/NO/PARTIAL
Are the statistical methods justified by the author?	YES/NO
Is the statistical methods used to analyze the data properly described and referenced?	YES/NO
Are negative findings presented?	YES/NO/PARTIAL
Are all the study questions answered?	YES/NO
Do the researchers explain future implications?	YES/NO
Survey	
Was the denominator (i.e., the population size) reported?	YES/NO
Did the author justified sample size?	YES/NO
Is the sample representative of the population to which the results will generalize?	YES/NO
Have “drop outs” introduced biasness on result limitation?	YES/NO/NOT APPLICABLE
Experiment	
Were treatments randomly allocated?	YES/NO
If there is a control group, are participants similar to the treatment group participants in terms of variables that may affect study outcomes?	YES/NO
Could lack of blinding introduce bias?	YES/NO
Are the variables used in the study adequately measured (i.e., are the variables likely to be valid and reliable)?	YES/NO
Case Study	
Is case study context defined?	YES/NO
Are sufficient raw data presented to provide understanding of the case?	YES/NO
Is the case study based on theory and linked to existing literature?	YES/NO
Are ethical issues addressed properly (personal intentions, integrity issues, consent, review board approval)?	YES/NO
Is a clear Chain of evidence established from observations to conclusions?	YES/NO/PARTIAL
Experience Report	
Is the focus of study reported?	YES/NO
Does the author report personal observation?	YES/NO
Is there a link between data, interpretation and conclusion?	YES/NO/PARTIAL
Does the study report multiple experiences?	YES/NO

TABLE 1: QUALITY CHECKLIST ADOPTED FROM [12] [13] [14] [15] [16].

The questions included in the checklist will be answered either *Yes*, *No* or *Partial* and will be rated as 2, 1 or 0 respectively. The sum of the scores from all these questions will be used for assessing the quality of the studies.

D. Data Extraction Strategy

Two reviewers will extract the data randomly and then will compare the results. In case of any disagreement, reviewer will arbitrate to reach on some agreement. Each study will be uniquely numbered and studies reported in more than one papers will be counted once. For each research question, relevant data will be extracted from all accepted papers and will be recorded in data synthesis forms. Data will be extracted contributing to each research question.

For RQ1 following information will be extracted:

- RE area (Elicitation, Analysis, Specification, etc.)
- Research output (New Tool, New Technique, New Process, Modification of Tool/Technique/Process, Usage Experience Tool, Usage Experience Technique, Usage Experience Process, RE issues and Challenges).
- Participant Type (Academia, Industry, Mixed)
- Country (involved in research)
- Conference/ Journal
- Year of Publication

For RQ2 following information will be extracted:

- Type of evidence (case study, experiment, experience report, etc.)
- Data collection method (interview, questionnaire, etc)
- Type of research: For extracting information about type of research we have consulted an already developed classification of research approaches by Wieringa [11] who has categorized research types as validation research, evaluation research, solution proposal, philosophical papers, opinion papers and experience papers.

E. Data Synthesis Strategy

Data from all the included papers will be extracted and recorded. Different kind of data will be extracted for each research question as has been described in section D of this paper. The data will be extracted by using well defined data extraction forms, where data will be fed to the best in a quantitative way so that data can finally be analyzed for various patterns.

Data related to RQ1 will help out in finding information like:

- Which area of requirements engineering is empirically evaluated more/less frequently and in what context these empirical studies have been carried out?
- What type of tools, techniques, frameworks and models, etc. are being used in the field of RE?
- Where these tools, techniques and models, etc. are adopted mostly?
- What are the areas of RE that are rich in terms of tools and techniques available and what are the areas where more attention needs to be given?
- What areas of RE are under more consideration and where more work is required?

- Which era of RE can be said as having maximum progress in terms of new advances?
- Which research participants are more involved in RE progress?
- What is the knowledge gap pointed by the evidence in RE?

Data related to RQ2 will help out in finding:

- Which research methods have been employed more frequently in RE?
- Which research methods have been employed for investigating which sub areas of RE?
- Which data collection method has been used for investigating which sub area of RE and by using which research method?
- What type of research has been presented more frequently by the empirical studies of RE?

The extracted data will then be analyzed using various quantitative and qualitative methods for synthesizing the data. Based on this data, different statistics and reports will be generated like:

- Percentage of studies containing case studies, experiments and experiences
- Percentage of studies for each RE area.
- Percentage of studies for different research types.
- Percentage of studies for different participant types.
- Analysis of evidence type versus participant type.
- Analysis of evidence type versus RE area type.
- Analysis of evidence type versus type of research.
- And more complex analysis comprising more than two parameters.

All of this information will be finally presented in the form of systematic maps like *bar graphs*, *bubble plots*, etc.

IV. CONCLUSION AND FUTURE WORK

Requirements engineering being a mature field of software engineering, presents a vast history of research and developments in all of its sub areas. But there lacks a study summarizing the whole field of RE with an emphasis on empirical evidences presented in this field to date. This paper in the form of a protocol provides a plan for carrying out a systematic literature review for the field of requirements engineering, describing state of the art of this field. The future work includes successful execution of the research plan presented in this paper to present a state of the art of software requirements engineering. It will help out various practitioners and researchers in the field to find out which areas of RE are rich in research and also to identify gaps and thus future research directions in this field.

REFERENCES

- [1] A. Abran, J. W. Moore, R. Dupuis, R. L. Dupuis, and L. L. Tripp, "Guide to the software engineering body of knowledge (swebok)," 2004 ed P Bourque R Dupuis A Abran and JW Moore Eds IEEE Press, 2001. Last access: March, 2011.
- [2] H. F. Hofmann and F. Lehner, "Requirements engineering as a success factor in software projects," *Software, IEEE*, vol. 18, no. 4, pp. 58-66, 2001. Last access: March, 2011.
- [3] F.P. Brooks, "Mythical Man-Month: Essays on Software Engineering]", 20th anniversary edition. Addison-Wesley Professional, 1995.

- [4] K. E. Wiegers, "In search of excellent requirements," *The Journal of the Quality Assurance Institute*, vol. 1, 1995. Last access: April, 2011.
- [5] Springer, *International Journal of Requirement Engineering*. Website Url: <http://www.springer.com/computer/swe/journal/766>
- [6] IEEE, *International Conference on Requirement Engineering*. Website Url: <http://www.requirements-engineering.org/>
- [7] B. A. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-based software engineering," in *Proceedings of the 26th international conference on software engineering*, 2004, pp. 273-281. Last access: January, 2011.
- [8] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering-A systematic literature review," *Information and Software Technology*, vol. 51, no. 1, pp. 7-15, 2009. Last access: March, 2011.
- [9] T. Dybå and T. Dingsøy, "Strength of evidence in systematic reviews in software engineering," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, 2008, pp. 178-187. Last access: March, 2011.
- [10] D. T. Ross and K. E. Schoman Jr, "Structured analysis for requirements definition," *Software Engineering, IEEE Transactions on*, no. 1, pp. 6-15, 1977, doi: 10.1109/TSE.1977.229899. Last access: May, 2011.
- [11] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion," *Requirements Engineering*, vol. 11, no. 1, pp. 102-107, 2006.
- [12] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," *Engineering*, vol. 2, no. EBSE 2007-001, 2007. Last access: January, 2011.
- [13] B.A. Kitchenham, O.P. Brereton, D. Budgen, and Z. Li, "An Evaluation of Quality Checklist Proposals-A participant-observer case study," *13th International Conference on Evaluation and Assessment in Software Engineering*, Durham University, UK, 20 - 21, April 2009. Last access: March, 2011.
- [14] B. Kitchenham, D.I.K. Sjoberg, O.P. Brereton, D. Budgen, T. Dyba, M. Host, D. Pfahl, and P. Runeson, "Can we evaluate the quality of software engineering experiments?," *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ACM, 2010, pp. 1-8, doi:10.1145/1852786.1852789. Last access: March, 2011.
- [15] M. Host and P. Runeson, "Checklists for software engineering case study research," *Empirical Software Engineering and Measurement*, ESEM 2007, First International Symposium on, IEEE, 2007, pp. 479-481, doi:10.1109/ESEM.2007.29. Last access: January, 2011.
- [16] D. Budgen and C. Zhang, "Preliminary reporting guidelines for experience papers," *Proceedings of EASE*, 2009, pp. 1-10. Last access: March, 2011.
- [17] T.E. Bell and T.A. Thayer, "Software Requirements: Are They Really a Problem?," *Proc. ICSE-2: 2nd International Conference on Software Engineering*, San Francisco, 1976. Last access: January, 2011.
- [18] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering*, 2008, pp. 71-80.

Success Factors Leading to the Sustainability of Software Process Improvement Efforts

Natalja Nikitina, Mira Kajko-Mattsson
 School of Information and Communication Technology
 KTH Royal Institute of Technology
 Stockholm, Sweden
 nikitina@kth.se, mekm2@kth.se

Abstract— Despite the fact that many software organizations put a lot of effort into software process improvement, they still do not always succeed in sustaining the improvement results. We believe that this is due to the fact that current software process improvement frameworks and/or models do not provide any aid in form of a list of success factors that primarily contribute to the sustainability of SPI efforts. In this paper, we compile thirty two SPI success factors as elicited in theoretical and empirical studies. Our primary goal is to aid software companies in defining, planning, monitoring and improving their SPI efforts. Our secondary goal is to create a basis for identifying SPI health attributes which, in turn, would allow software companies to determine the health of their SPI projects. (Abstract)

Keywords-SPI project health, lessons (key words)

I. INTRODUCTION

Current Software Process Improvement (SPI) process frameworks and/or models do not always provide clear evidence about the long-term health of the SPI projects and the sustainability of their results. With this we mean that successful SPI implementations do not always guarantee long lasting results [1]. Although the immediate SPI positive results may be clearly tangible, this does not imply that they will sustain in the long-run [2].

The problem of sustaining the SPI results has been widely recognized and suggestions for solving it have been made by some maturity models and development methods [3], [4]. The suggestions usually incorporate process improvement activities into software development processes. Still, however, they do not cover exhaustive list of attributes aiding software organizations in evaluating the success of their SPI projects, and thereby, aiding them in sustaining the SPI results in the long-run.

Sustainability of the SPI efforts is very important. Lack of it means that the organizations quickly go back to the old pre-SPI process state and its problems, and thereby make the SPI efforts a waste of time and resources. For this not to happen, organizations should not only implement SPI activities and check their immediate results. They should also plan, monitor and control the long-term progress of their SPI efforts and the processes undergoing SPI. For that, they need to identify the success factors aiding them in sustaining their SPI results.

Today, there are many SPI models and frameworks, development methods and experience reports dealing with SPI. They either suggest different ways of improving processes or they report on the SPI results. Many of them delineate SPI from their own perspectives, in which, they may suggest or report on the SPI success factors that are limited to and/or relevant in their own contexts. This, in

turn, implies that overall knowledge of the SPI sustainability factors is provided by different authors and, thereby, scattered in many different sources. To the knowledge of the authors of this paper, however, no one has tried to gather them and put them into an overall list of SPI sustainability success factors.

In this paper, we elicit thirty two SPI success factors that primarily contribute to the sustainability of SPI efforts. We do it in two ways, via a literature study and via interviews. Our primary goal is to aid software companies in defining, planning, monitoring and improving their SPI efforts, and sustain its results. Our secondary goal is to create a basis for identifying SPI health attributes which, in turn, would allow software companies to determine the health of their SPI projects.

The remainder of this paper is organized as follows. Section II presents background of the field. Section III describes the method used during this study. Section IV lists and provides descriptions of thirty two SPI sustainability success factors. Finally, Section V presents final remarks and suggests future work.

II. BACKGROUND

There is a large amount of software maturity models that have been designed to help software organizations implement SPI activities. The best known ones are CMMI and SPICE [3], [4].

CMM Integration (CMMI) framework provides guidance for improving software organization's processes in a structured and well planned manner. It helps assess organizational maturity or process area capability, establish priorities for improvement, and implement these improvements [3]. CMMI best practices are organized into 25 process areas, which have two different representations: continuous, labeled by standardized "capability levels", and staged, labeled by tailored "maturity levels".

SPICE maturity model, also known as ISO 15504, has similar structure to CMMI. It consists of capability levels, which, in turn, consist of process attributes, and further, of generic practices. SPICE model provides tools for standardized process assessment and suggestions for defining process maturity.

Even though many software organizations are using maturity models for process improvements, there are still many organizations that are not willing to follow formal maturity models [5]. The reasons are many. Some of them are: (1) the organizations are too small, (2) process certification is too costly, (3) the organizations do not have time for it, or (4) the organizations use other SPI methods [5]. Smaller organizations are often using ad-hoc SPI methods, or, they transition from one development method to another without proper planning or preparation [6].

III. METHOD

In this section, we present our research method. We first present research steps in Section III.A. We then describe the questionnaire used in one of the steps of this study in Section III.B. Finally, we describe the validity of our results in Section III.C.

A. Research Steps

Our research consisted of the three following steps: (1) *Literature Study*, (2) *Empirical Study*, and (3) *Data Analysis*. During the first two steps, we elicited sustainability success factors, first by reviewing literature and then by interviewing software practitioners. These two steps were conducted independently. This implies that the results of the first step did not constitute input to the second step, and vice versa. In the third step, we combined and analyzed the results as achieved in the first two independently done steps. Below, we briefly describe the three steps.

1) Literature Study

During the literature study, we reviewed more than 45 publications dealing with SPI projects. These were mainly experience reports and case studies that had been retrieved from IEEE, ACM, Springer, John Wiley and Sons, and other publishers. Out of them, we chose 25 empirical reports describing conditions contributing to or subtracting from the success of SPI projects [1], [2], [7-29]. Our goal was to elicit factors that contributed to the sustainability of SPI efforts, as defined by literature.

The majority of the publications studied mainly reported on the empirical process improvement projects. They did not focus on outlining the conditions contributing to the success of SPI efforts. However, some of the conditions could be indirectly recognized out of their contexts and results. Only three publications provided direct and explicit feedback on critical SPI success factors. These were [7], [8], [9].

During the literature study, we elicited critical factors influencing SPI for both successful SPI initiation and implementation, and successful preservation of its results. This step resulted in a preliminary list of SPI sustainability success factors. Having this list as a basis, we reviewed the publications anew, now with the purpose of studying their direct and indirect descriptions, their contexts, and identifying their impact on the sustainability of the SPI efforts. This step resulted in 28 SPI success factors.

2) Empirical study

During the empirical study, we interviewed 40 software engineers who had been involved in or who had been affected by SPI projects. Among the interviewees, there were twenty two software developers, ten testers, seven development managers and one SPI manager. They came from eight different middle size software organizations, located in Vietnam (18 participants), Sweden (18 participants), Bangladesh (2 participants), China (1 participant) and Island (1 participant).

Each interviewee was interviewed only once, in a tête à tête manner. Some of the interviews were recorded, while others were not. The ones that had not been recorded were the interviewees from Vietnam. On purpose, we chose not to record them because we believed that due to cultural reasons, the interviewees might feel hampered in providing honest answers. However, the interview results were

TABLE I. INTERVIEW QUESTIONNAIRE

- | |
|--|
| <ol style="list-style-type: none"> 1. Are you aware that the information you provide will be kept confidential? 2. Have you been involved in process improvement or process transition before? To what extent? <ol style="list-style-type: none"> a. If yes, have the results of the process improvements been lasting? <ol style="list-style-type: none"> i. If yes, why do you think the results have been lasting? ii. If no, why do you think the results have not been lasting? 3. What factors contribute to the process improvement sustainability? Please list them and motivate your answers. 4. What factors prevent the process improvement sustainability? Please list them and motivate your answers. 5. What are your suggestions for keeping the process improvement results lasting/sustainable? Please list them and motivate your answers. |
|--|

documented directly after each interview. The interviews lasted for forty minutes per interview in average. They had resulted in 24 SPI sustainability factors in total, out of which 20 overlapped with the factors as identified in the literature studied, and four constituted new SPI success factors that had not been identified in the literature.

3) Data analysis

During the *Data Analysis* step, we analyzed the results of the literature study, transcribed the interviews that had been recorded, and analyzed the empirical data using the hermeneutics approach. Here, we identified and analyzed the sustainability factors as elicited in both studies. Finally, we identified common and overlapping sustainability factors, combined them and created a list of SPI sustainability success factors. It is this list that constitutes the body of this paper and a basis for the future creation of the SPI health attributes.

B. Questionnaire

For this study, we used semi-structured interviews, based on a questionnaire presented in Table 1. The semi-structure implies that the interview structure was flexible, allowing new questions to be asked depending on the answers of the interviewee.

As shown in Table 1, the interviews were aimed at identifying both the success and failure factors. They consisted of the following groups of questions: (1) the reasons for why SPI efforts have been lasting, (2) the reasons for why SPI efforts have not been lasting, (3) factors contributing to the SPI sustainability, (4) factors preventing the SPI sustainability, and finally, (5) suggestions for how to keep the SPI efforts sustainable.

C. Validity

All the qualitative research methods, encounter validity threats [30]. Those threats concern construct validity, internal validity, external validity, and conclusion validity.

Construct validity refers to the degree to which inference can be made from the operational definition of a variable to the theoretical constructs [31]. The main threat to construct validity is to guarantee that the right measures have been chosen for the study. Here, the risk was that we might use wrong measures, and as a result, that we might misinterpret the SPI sustainability success factors. To minimize this threat, we conducted both theoretical and

empirical studies. Moreover, we employed the multiple sources of data during the empirical study by interviewing different roles in eight different organizations.

Internal validity refers to the degree of inferences of the cause-effect or causal relationships in the study [31]. The main thread to internal validity for the literature study was the fact that we might misinterpret the conclusions presented in the literature or use too few literature sources. Therefore, in this study, we first made a comprehensive search in various scientific sources out of which we extracted 25 experience reports. The main threat to internal validity for the empirical study was that the interviewees might have misunderstood the impacts on the SPI sustainability. To minimize this threat, we used various roles involved in SPI in different software organizations.

External validity refers to the degree of whenever the sample findings can be generalized [31]. The main external validity threat to our empirical study was the fact that the SPI sustainability factors that had been identified during the interviews were based on the experiences of only 40 individuals and eight software organizations. However, we believe that the findings and conclusions of this study can still be found useful for many other software companies planning to conduct SPI and wishing to sustain its results.

Conclusion validity refers to the degree to which the conclusions are based on the correct interpretation of the

relationships of the data [31]. The conclusion validity threat to our study was that the conclusions would not be related to the data. To minimize the threat, we based our conclusions on the multiple data sources such as literature and interviews.

IV. SPI SUSTAINABILITY FACTORS

In this section, we present the SPI success factors that have been elicited both during the literature and empirical studies. All the SPI success factors identified in the literature study have direct or indirect impact on the sustainability of the SPI efforts. Therefore, when describing them, we state their relationship with the SPI sustainability wherever relevant.

During the literature study, we have identified 28 SPI success factors, out of which 20 factors overlapped with the factors that have been elicited during the empirical study. The interviews have additionally resulted in four new SPI sustainability factors.

Just because these two studies were done independently, they had led to two groups of SPI success factors: (1) the ones that are common to the two studies, and, (2) the ones that have been elicited within one type of a study but not within the other. When describing them in this section, we clearly identify their sources. Additionally, we list them and their sources in Table 2.

TABLE II. COMPILED LIST OF SPI SUSTAINABILITY SUCCESS FACTORS

	Cluster	SPI sustainability factor	Source		Cluster	SPI sustainability factor	Source
Human factors	Education	Stakeholders are trained and mentored	Lit. & Emp.	Organizational factors (2)	Alignment	SPI is aligned with business goals	Lit.
		Stakeholders have a common understanding of the process undergoing change	Lit. & Emp.			SPI is aligned with organizational policies and strategies	Lit.
	SPI campaign	Stakeholders are encouraged to support SPI	Emp.			SPI methods are tailored to specific organizational contexts and needs	Lit. & Emp.
		SPI activities are accepted	Lit. & Emp.	Knowledge	Technical staff participates in SPI	Lit. & Emp.	
	Commitment and support	Management is committed to and continuously supports the SPI process	Lit. & Emp.		Technical staff and SPI leaders possess experience and expertise in SPI	Lit. & Emp.	
		Technical staff is committed to the SPI process	Lit. & Emp.		SPI method is well defined	Lit. & Emp.	
	Communication	Stakeholders are aware of the complexity, challenges and benefits of SPI	Lit.	Implementation factors	Preparation and planning	Mechanisms for stabilizing the process are planned and prepared	Lit.
		Stakeholders have realistic expectations	Emp.			SPI goals and objectives are clear and realistic	Lit. & Emp.
		Information about SPI activities and its results is disseminated	Lit. & Emp.		Management	SPI leaders do not blame staff for their mistakes	Lit.
	SPI drivers	Technical staff owns the process	Lit. & Emp.			Process standards are defined and enforced	Lit. & Emp.
		External SPI leaders are trusted and respected	Lit. & Emp.			SPI projects are effectively managed	Lit.
		Internal SPI leaders are designated	Lit. & Emp.			SPI activities are prioritized	Lit.
	Rewards	Newly introduced processes give positive results	Emp.	Process review and measurement	Software process is monitored and measured	Lit. & Emp.	
		Stakeholders involved are rewarded for successful SPI activities	Lit. & Emp.		Software process and its efficiency is continuously reviewed	Lit. & Emp.	
Organizational factors (1)	Resources	Time and resources are dedicated to SPI	Lit. & Emp.		Continuous SPI	Mechanisms for continuous process tuning are in place	Lit. & Emp.
		SPI responsibilities are clearly specified and compensated	Lit.				
		People turnover is low	Emp.				

Many different roles are involved in process improvement. Their naming and responsibilities vary in different literature and industrial contexts. For this reason, we identify and define the following roles involved in SPI:

- *Stakeholder*: a person or a group that is involved in or affected by SPI.
- *Development team*: a group of developers and/or testers that work together on development of the software product.
- *Technical staff*: a group consisting of developers, testers and other roles involved in executing the process undergoing the improvement. They are the “doers”, and therefore, they get affected by the process change the most.
- *External SPI leader*: a person or a group that is in charge of the overall SPI process. He/she initiates the improvement projects, requests resources, encourages local improvement efforts and establishes communication channels between different groups. External SPI leader is not the doer in the process to be improved. For this reason, he/she is seen as an external and independent role.
- *Internal SPI leader*: a person or a group within the development team who is responsible for supporting and following the SPI strategy on a local level.

To facilitate our presentation, we group the elicited SPI sustainability success factors into three categories as defined in [10]. These are *human factors*, *organizational factors* and *implementation factors*.

A. *Human factors*

We have identified fourteen different human SPI sustainability success factors. Human factors deal with human behavior and reactions in the SPI context.

1) *Stakeholders are trained and mentored*

Process improvement often implies changes to the process or introduction of new techniques and practices. Hence, as pointed out in the literature studied, the development team needs to be trained in them in order to fully understand their role in the process change. They need to be prepared for the process improvement and understand the reasons behind each suggested change. Otherwise, they would less likely follow the new process [11]. For this reason, staff training and mentoring in the new process, new techniques and practices are needed not only for implementing process changes but also for sustaining their results. In organizations or cultures where knowledge of the process is low, the training in the process is even more important [32].

The need for training and mentoring of the SPI stakeholders was also raised during the interviews. According to our interviewees, all the company employees need to have necessary training in the new method in order to be able to follow it properly and dedicatedly. Moreover, the internal SPI leaders and team members responsible for improvement activities have to be coached on how to implement improvements and how to follow the new process. According to our interviewees continuous mentoring and training increases the credibility of strategic

SPI decisions and contributes to building trust in those decisions and in the new process.

2) *Stakeholders have a common understanding of the process undergoing change*

The process cannot be efficiently improved unless it is properly understood. According to the literature studied, the technical staff and management have to reach consensus on the status of the current process, its problems and possible solutions, as well as the organization’s vision and the improvement goals [9]. Common understanding of the current and new process, suggested changes and its potential benefits are important to increase support for process improvement among all the stakeholders involved.

Our empirical study has led to the same conclusion. According to our interviewees, technical staff should understand the reasons behind process changes, since it is mainly the technical staff, who have to change the previous habits and adapt to a new way of working. Our interviewees have also pointed out that common understanding of the new process, SPI activities and its potential benefits strongly contribute to the increase of commitment and motivation towards SPI.

3) *Stakeholders are encouraged to support SPI*

Commitment to and support of SPI by all the stakeholders is a great asset to help successful SPI implementation and to decrease inertia to change. Therefore, according to the interview results, all the stakeholders need to be continuously encouraged to support SPI, and to show interest in the process improvement activities.

4) *SPI activities are accepted*

Changes to the process may affect daily work of many employees. Therefore, according to the literature studied, it is important that all the members of the technical staff agree and accept future changes to the process [10], [23]. This can decrease inertia to change. Acceptance of process changes can be encouraged by high involvement of the technical staff in the SPI activities.

Our interviews have also led to the same success factor. According to our interviewees, if all the personnel accept the newly changed process, then there is a greater opportunity that the changed process will be sustained. Mutual acceptance of the changed process and process improvement activities is a key to sustain the results achieved by the SPI.

5) *Management is committed to and continuously supports the SPI process*

To provide long-term sustainable results, software improvement requires continuous investment in time, resources and effort. This, in turn, requires that management is strongly committed to and continuously supports the SPI efforts [9], [10], [12], [13], [33]. According to the literature studied, the strong management commitment helps retain high priority of the SPI projects and the continuous management support helps assure continuous supply of the required resources [9], [10], [12], [13]. It is especially important in the initial SPI phases during which the cost of SPI activities is higher than initially expected and planned [7].

Even our interviewees have stated that SPI projects need commitment and support of top management for investment in time and resources in order to achieve sustainable results.

6) *Technical staff is committed to the SPI process*

Acceptance of SPI activities is a critical success factor when starting SPI projects. However, according to the literature studied, it needs to be complemented with the commitment of the technical staff. Management commitment to SPI projects has already been listed as a significant success factor to SPI projects. However, commitment of technical staff is just as important [14-17], [33]. Together with the increased motivation and engagement, the commitment of the technical staff can become a driving wheel of process improvement [2]. Committed staff takes responsibility and ownership of the process and keeps process in a healthy state [2].

Commitment of the technical staff has also been elicited during our interviews. Our interviewees have stated that, if the company personnel does not commit to the process changes, it will most likely go back to the pre-SPI process state.

7) *Stakeholders are aware of complexity, challenges and benefits of SPI*

Since SPI requires continuous effort and often brings mainly long term results, it is important that everybody involved in it is aware of its complexity, challenges and future benefits. Hence, according to the literature studied, organizations must make sure that all the stakeholders involved are aware of them. This can be realized via education and training. Raising awareness of SPI and effective communication of its complexity, challenges and benefits strongly affects the success of the SPI projects [16-22].

8) *Stakeholders have realistic expectations*

Our interviews have indicated that in order to be satisfied with SPI and its results, the employees affected by SPI should have realistic expectations. Otherwise, the stakeholders would get disappointed with SPI and would not continue with it, even though SPI brought positive results.

9) *Information about SPI activities and its results is disseminated*

SPI projects bring many changes to the process and daily routines. These changes have to be communicated to all the stakeholders that can be directly or indirectly impacted by the changes.

According to the literature studied, insufficient communication of the SPI changes may lead to lack of transparency of the SPI projects, confused personnel and poor quality process. Team collaboration and communication, on the other hand, may help the staff members to exchange knowledge and experience during the improvement project and contribute to coherent organizational culture [9].

The need for communicating on the SPI activities and their results has also been raised by our interviews. According to them, sufficient communication positively impacts motivation in SPI and acceptance of the new process changes.

10) *Technical staff owns the SPI process*

Disregarding the reasons behind the SPI projects, the new process has to be accepted and followed by the team. According to the literature studied, it is important that not only external and internal SPI leaders but also all the technical staff members take on the ownership of the process to be improved. The members should take the

responsibility for tailoring the process and for continuously improving it. It is only in this way they will feel more affiliated with the process and more responsible for future process improvements. This, in turn, will lead to a built-in, self-driven continuous process improvement process, which, in turn, will strongly contribute to the sustainability of the SPI results [12].

Our interviewees have also stated that the success of SPI projects is strongly related to process ownership. According to them, not only management and SPI leaders should own the process, but also all the technical staff members. They should not only be responsible for the SPI process but also for its progress.

11) *External SPI leaders are trusted and respected*

According to the literature studied, the level of experience, commitment and engagement of the external SPI leaders can determine the success of the SPI projects [7-10]. However, as [7], [8] claim, the authority and respect paid to the external SPI leaders is as important. Even if the SPI leaders are in a privileged position, it still does not imply that they have high enough authority, trust and respect among the technical staff members. If so, then their ideas may not be supported and successfully transmitted to the process change [7-10]. Trust and respect may only be gained via personal qualities such as honesty, credibility, reliability, experience and reputation.

Trust and respect of the external SPI leaders was also raised during the interviews. According to our interviewees, to make the SPI results last, there should be an external SPI leader, a person or a group of people who have knowledge of SPI and who take on the responsibility of driving it.

12) *Internal SPI leaders are designated*

According to the literature studied, the internal SPI leaders are recognized as important SPI actors since they take on immediate responsibility for leading and supporting continuous process improvement [9], [12-14]. By possessing knowledge of the process, they are able to adapt the improvement suggestions to the different needs of development teams, projects and cultures. They help SPI activities get started and their engagement aids in winning support of their team members [13].

The importance of designating internal SPI leaders was also recognized during the interviews. According to our interviewees, the involvement of the internal SPI leaders helps spread commitment to the process and create strong process ownership. Internal leadership creates continuous control that the process is followed in a correct way and that its stakeholders are engaged in SPI.

13) *Newly introduced processes give positive results*

As mentioned before, the results of the SPI activities should be disseminated to all the stakeholders. However, as discovered during the interviews, just the dissemination of the results of SPI is not enough. The results achieved by the early SPI effort should be positive and should speak for themselves. Positive results of the newly introduced process encourages and motivates stakeholders to continue with the SPI activities and changes the opinions of those who did not support it from the very beginning.

14) *Stakeholders involved are rewarded for successful SPI activities*

The importance of rewards for SPI success has been mentioned in some of the studied literature sources [20].

Our interviews have also shown that in order to keep constant stakeholder commitment to and engagement in SPI, the stakeholders should celebrate the SPI success. Rewards for the improved process contribute to the increase of motivation and engagement in future SPI activities.

B. Organizational factors

We have elicited six organizational factors. Organizational factors are critical success aspects that are outside the scope of SPI. Those are related to the organizational structures and politics as well as cultural issues [10]. Still, however, they have a substantial impact on SPI sustainability.

1) Time and resources are dedicated to SPI

According to the literature studied, SPI projects need to have dedicated time and resources. As many as 72% of SPI improvement projects have suffered from lack of resources and constant time pressure [7], [11], [18], [21]. SPI projects cannot run on their own. Investment in time and people has been recognized not only for starting and implementing the SPI projects but also for sustaining the achieved results [7], [8], [10], [14], [18-20], [24].

Our interviewees were of the same opinion. According to them, process related problems often start when no one is responsible for the process.

2) SPI responsibilities are clearly specified and compensated

According to the literature studied, people involved in SPI should have clear responsibilities and compensation for their effort [7], [8]. If they are assigned to SPI related tasks, they should be relieved from other tasks. Time dedicated to the SPI activities should be compensated in the same manner as other work. Otherwise, the SPI activities will be done in a rush, they may be neglected, they may be delayed or they may even be forgotten.

3) People turnover is low

According to our interviewees, high people turnover can become a significant barrier to the sustainability of the SPI efforts. When the key employees leave the company, so does the knowledge of the process and SPI. With high people turnover, more effort needs to be spent on the education and training of the new hires.

4) SPI is aligned with business goals

The goals of SPI projects should not only go in line with the standardization of process and quality standards, but also with business goals. According to the literature studied, alignment of SPI goals with the organizational business goals contributes to the better management of, commitment to and support of the SPI projects [10], [14], [15], [25], [26].

5) SPI is aligned with organizational policies and strategies

Improvement projects often conflict with the existing organizational policies by requiring changes to routines and processes that are common to the whole organization. Therefore, as stated in the literature studied, organizational policies have to be aligned with the SPI goals and vice versa. In cases when organizations do not have any policies, they have to establish ones and make the process standardization and improvement coherent with them. Lack of organizational policies to support process changes

can potentially become a big barrier for a successful process improvement [7], [11], [14], [21].

6) SPI methods are tailored to specific organizational contexts and needs

Each organization is different with respect to its structure, culture and policies. For this reason, as stated in the literature studied, SPI initiatives should consider the contextual specifics of the organizational culture, product characteristics, customer availability and people influenced by the process. The adaptation of process improvement methods to specific organizational contexts and needs helps address individual problems and contributes to sustainable SPI efforts [9-10].

The interviews have led to the similar conclusion. According to our interviewees, if the SPI is not aligned with the organizational needs, or if it does not fit the established organizational and national culture, then it is more difficult to win people's support and commitment. Moreover, the people would resist the process changes and the results achieved by SPI would be easily lost.

C. Implementation factors

We have elicited twelve implementation factors. The implementation factors are related to the execution of the SPI projects.

1) Technical staff participates in SPI

Technical staff constitutes an important process knowledge and experience asset [9]. By knowing all the nooks and crannies of the process, they may provide useful feedback on the suggested SPI changes [23]. For this reason, it is important that they are involved in identifying process pains and in suggesting solutions for them [9], [10], [18-20], [25], [26].

The literature findings show that the involvement and participation of the technical staff reduce resistance to change, and thereby, strongly impacts the SPI success [9], [10], [14]. By being involved in the SPI activities, the technical staff members feel more motivated to adhere to the process changes, and therefore, they are more likely to accept them [9], [10]. Technical staff involvement was found especially important in immature organizations [19].

Our interviews have also led to the same conclusion. According to our interviewees, the involvement of the technical staff contributes to the alignment of SPI methods to the organizational needs. It also decreases inertia to change and increases motivation, and thereby, significantly affects the sustainability of the SPI efforts.

2) Technical staff and SPI leaders possess experience and expertise in SPI

Process improvement implies changes to the deeply ingrained organizational culture, habits, working patterns and manners that have been developed throughout a long time. To change them is very difficult. However, according to the literature studied, it is easier to change them if the stakeholders involved possess enough knowledge and experience in implementing software process improvement changes. If there is lack of such knowledge and experience, then there is a risk of using unsuitable SPI strategy and of having poor SPI execution, which could potentially fail the SPI projects [11], [12], [18-20], [34].

Few of our interviews have also mentioned the importance of knowledge in SPI by all the stakeholders involved.

3) *SPI method is well defined*

Software process improvement is a complex and time consuming process. Following a well defined and structured SPI implementation method strongly contributes to its success [12], [19], [20]. According to the literature studied, the SPI method should be suitable to the organization, its size and goals.

Our interviews have also led to the same success factor, highlighting the importance of accessible and updated process documentation.

4) *Mechanisms for stabilizing the process are planned and prepared*

To prevent losing the immediate advantage of process improvement efforts, it is important to stabilize the changed process. According to the literature studied, this can be done by providing a comprehensive support to those responsible for the process and by encouraging staff to practice new procedures [9]. All the roles responsible for the SPI projects should influence the process stabilization by continuously reaffirming commitment to change, communicating progress of improvements, and by providing continuous feedback and motivation [9].

5) *SPI goals and objectives are clear and realistic*

SPI projects should have clearly specified goals and objectives. Our literature study shows that clear, realistic and well communicated SPI goals contribute to the good understanding of the SPI process and assurance that they are well understood across all the organizational levels [8].

Realistic SPI goals lead to realistic expectations and aid in maintaining high motivation for and support of the SPI activities. Unrealistic, too ambitious or unreachable objectives, on the other hand, may jeopardize the SPI projects, by decreasing employees' engagement and motivation even in projects with positive results [7], [14]. Our interviews have led to the similar conclusion.

6) *SPI leaders do not blame staff for their mistakes*

During the SPI projects, the weaknesses and problems of the current process are continuously identified and improvements are suggested. Since the problems and negative issues of the process are continuously discussed, it is important not to start blame games [11].

According to the literature studied, blaming people for mistakes can only lead to frustration and inertia to process change [11], [24]. One should focus on process's weaknesses rather than on people's mistakes [24]. One should also encourage initiative, innovation, creativity and openness. Without it, employees cannot share valuable ideas, and thereby, contribute to process improvement [24].

7) *Process standards are defined and enforced*

When the stakeholders lack dedication and commitment to the new process, people are tempted not to follow the process standards, unless there is a strong control mechanism in place [6]. Even when properly trained, the staff may not follow the newly introduced process. Therefore, as stated in the literature studied, in order to guarantee that the process is dedicatedly followed by all the stakeholders, it should be enforced and controlled by the SPI managers [6].

Our interviews have led to the similar conclusion. The interviewees have also suggested that the employees that are not following the new process procedures correctly should be informed and consequently corrected.

8) *SPI projects are effectively managed*

Management of the SPI projects involves a wide range of activities such as planning for change, identifying actors involved, ensuring the level of understanding of process changes, monitoring the status of SPI, evaluating the progress, and the like. It needs to be performed in an effective and professional manner [21]. According to the literature studied, without project management, SPI projects are doomed to fail and may lead to chaos [9].

9) *SPI activities are prioritized*

At the beginning of the SPI projects, companies can be overwhelmed with the amount of suggestions for the improvements. Such being a case, as stated in the literature studied, it is important not to do too many changes at ones. Instead, companies should prioritize the SPI suggestions and, focus on one or few improvements at a time [13], [14]. This leads to easier and more efficient implementation, control, measurement, and thereby, to more sustainable results.

10) *Software process is monitored and measured*

Continuous process monitoring and measurement indicates whether the SPI activities are effective or not, and allows to provide early feedback on the sustainability of the SPI efforts. Hence, as stated in the literature studied, it is important to evaluate and measure the process on a continuous basis to reinsure its purpose and to increase the engagement of the SPI supporters. Measured and acknowledged process improvements will positively affect team morale and motivation [12], [13], [25], [26]. Our interviews have also stated that measurement and evaluation of the SPI results can positively impact the engagement in and motivation for future SPI.

11) *Software process and its efficiency is continuously reviewed*

To achieve continuous process improvements, the SPI process and its efficiency should be reflected on and evaluated on a continuous basis. As stated in the literature studied, process reviews, such as retrospectives, allow learning from previous experience and from experimenting with the process, which, in turn, contributes to a self-driven continuous process improvement, and thereby, to the long lasting SPI results [25], [26].

Our interviews have led to the similar conclusion. According to our interviewees, process reviews help to identify problems in the current process and to acknowledge benefits achieved by SPI. This, in turn, significantly contributes to the sustainability of the achieved results.

12) *Mechanisms for continuous process tuning are in place*

Software organizations have dynamic and continuously changing structures. Organizational culture, availability of the customer and background of the employees are always changing. Hence, a static process that is not improving or adapting to the changing organizational needs is failed to decay [6]. Without frequent reviews and changes to the process, it will soon outdate. Therefore, it is important to have mechanisms for continuous process tuning and improvements in place [6]. This was also concluded during our interviews.

V. FINAL REMARKS

In this paper, we have presented thirty two success factors influencing the sustainability of SPI efforts. We have elicited them in two independently conducted studies, the literature and empirical studies.

Initially, we grouped our sustainability factors into three clusters: human, organizational and implementation. When analyzing them, however, we could further group them into thirteen additional sub-clusters. As shown in Table 2, those are: (1) *Education*, (2) *SPI campaign*, (3) *Commitment and support*, (4) *Communication*, (5) *SPI drivers*, (6) *Rewards*, (7) *Resources*, (8) *Alignment*, (9) *Knowledge*, (10) *Preparation and planning*, (11) *Management*, (12) *Process review and measurement*, and (13) *Continuous process improvement*.

The SPI sustainability success factors presented in this paper constitute the body of the knowledge of the software engineering community as elicited in the current software engineering literature and in the industry. They may already be used by software development organizations when implementing and managing their SPI projects.

We strongly believe that it is not enough to just define SPI process frameworks and/or models. Process frameworks/models do not always provide clear evidence about the health of the SPI projects. For this reason, we plan to continue studying and analyzing the SPI sustainability success factors presented in this paper. Our future goal is to create a basis for supplementing currently defined SPI frameworks and/or models with a checklist of health attributes allowing software companies to define, monitor, control and improve their SPI processes, and thereby, allowing them to achieve long-term sustainable results.

REFERENCES

- [1] S. S. Chakravorty, "Where Process Improvement Projects Go wrong," Wall Street J., 2010.
- [2] N. Nikitina and M. Kajko-Mattsson, "Developer-Driven Big Bang Process Transition from Scrum to Kanban," Proc. 2011 International Conference on Software and Systems Process (ICSSP 2011), ACM, 2011.
- [3] CMMI Product Team, Capability Maturity Model Integration (CMMI), Version 1.1. Pittsburgh, USA: Software Engineering Institute, Carnegie Mellon University, 2002.
- [4] K. Schwaber and M. Beedle, Agile Software Development with SCRUM, Prentice Hall, 2001.
- [5] M. Staples, M. Niazi, R. Jeffery, A. Abrahams, P. Byatt, et. al., "Why organization do not want to adopt CMMI," J. of Syst. and Softw., vol. 80, 2007, pp.883-895.
- [6] S. Zahran, Software Process Improvement: Practical Guidelines for business success, Addison Wesley, 1998.
- [7] D. Goldenson and D. Herbsleb, After the Appraisal: A systematic Survey of Process Improvements, its Benefits, and Success Factors that influence Success. Technical report, SEI, 1995.
- [8] K. El Emam and P. Fusaro, B. Smith, "Success factors and barriers for software process improvement," 1999. In: C. Tully, R. Messnarz, (Eds), Better Software Practice For Business Benefit: Principles and Experience. In: IEEE Computer Society Press, Silver Spring, MD.
- [9] D. Stelzer and W. Mellis, "Success Factors of Organizational Change in Software Process Improvements," J. Softw. Process Improve. Pract., 1998, pp. 227-250.
- [10] T. Hall, A. Rainer and N. Badoo, "Implementing Software Process Improvement: An Empirical Study," J. Softw. Process Improve. Pract., vol. 7, 2002, pp. 3-15.
- [11] S. Beecham, T. Hall and A. Rainer, "Software Process Improvement Problems in Twelve Software Companies: An Empirical Analysis," Emp. Softw. Eng., vol. 8, 2003, pp.7-42.
- [12] A. Rainer and T. Hall, "Key success factors for implementing software process improvement: a maturity-based analysis," J. Syst. Softw., vol. 62, 2002, pp. 71-84.
- [13] D. Paulish and A. D. Carleton, "Case Studies of Software Process Improvement Measurement," IEEE Computer, vol. 27: 9, 1994, pp. 50 - 57.
- [14] M. Nazir, R.. Ahmad and N. H. Hassan, "Resistance factors in the Implementation of Software Process Improvement Project in Malaysia," J. of Comp. Science, vol. 4: 3, 2008, pp. 211-219.
- [15] K. C. Dangle, P. Larsen, M. Shaw and M. V. Zelcovitz, "Software Process Improvesmnt in Small ogranizations: A case study," IEEE Software, vol.22:6, 2005, pp. 68-75.
- [16] G. Santos, M. Montoni, J. Vasconcellos, S. Figuerido, R. Cabral, et. al., "Implementing Software Process Improvements Initiatives in Small and Medium-Size Enterprises in Brazil," Proc. QUATIC, 2007, pp.187-196.
- [17] S. B. Basri and R. V. O'Connor, "Organizational Commitment Towards Software Process Improvement: An Irish Software VSEs Case Study," Proc. ITSIM'10, IEEE, 2010, pp.1456-1461.
- [18] M. Niazi, D. Wilson and D. Zowdhi, "A framework for assisting the design of effective software process improvement implementation strategies,"J. of Syst. and Softw., vol. 78, 2005, pp. 204-222.
- [19] M. Niazi, D. Wilson and D. Zowdhi, "Implementing software process improvement initiatives: An Empirical study," Proc. PROFES 2006, 2006 , pp. 222 - 233.
- [20] M. Niazi, D. Wilson and D. Zowghi, "Critical Success Factors for Software Process Improvement Implementation: An Empirical Study," J. Softw. Process Improve. Pract., vol. 11, 2006, pp. 193-211.
- [21] M. Niazi, M. Ali Babar and J. M. Verner, "Software Process Improvement Barriers: A cross-cultural comparison," J. Infor. and Softw. Techn., vol. 52: 11, 2010.
- [22] T. Varkoi, "Management of Continuous Software Process Improvement," Proc. 2002 IEMC, 2002, pp. 334-337.
- [23] A. Sweeney and D. W. Bustard, "Software Process Improvement: making it happen in practice," Soft. Qual. J., vol. 6, 1997.
- [24] B. Curtis and M. Paulk, "Createing a software process improvement program," Butterworth-Heinemann Ltd, vol. 35: 6/7, 1993.
- [25] T. Dyba, "An instrument for measuring the key factors of success in Software Process Improvement," J. Emp. Softw. Eng., vol. 5, 2000, pp. 357-390.
- [26] T. Dyba, "An empirical investigation of the key factor for success in software process improvements," J. Trans. on Soft. Eng.,vol 31: 5, 2005.
- [27] I. Aaen, J. Arent, L. Mathissen and O. Ngwenyama, "A conceptual MAP of Software Process Improvement," Scandinavian J. of Infor. Syst., vol. 13, 2001.
- [28] T. Galinac, "Empirical Evaluation of selected best practices in implementation of software process improvements,"J. Infor. Soft. Techn., vol. 51, 2009, pp. 1351-1364.
- [29] F. Ekdahl and S. Larsson, "Experience Report: Using Internal CMMI Appraisals to Institutionalize Software Development Performance Improvement", Proc. EuroMicro 2006, 2006.
- [30] M. D. Myers, Qualitative Research in Business & Management, Sage publications, London, 2009.
- [31] T. William, Research Methods: The Concise Knowledge Base, Cornell University, 2005.
- [32] M. Niazi, M. A. Babar and J. V. Verner, "Software process improvement barriers: cultural comparism," J. Infor. Soft. Techn., vol. 52, 2010, pp. 1204-1216.
- [33] P. Abrahamsson and N. Iivari, "Commitment in Software Process Improvement--In Search of the Process," Proc. HICSS 2002, 2002.
- [34] N. Nikitina and M. Kajko-Mattsson, "Historical Perspective of Two Process Transitions," Proc. 2009 International Conference on Software Engineering Advances (ICSEA 2009), IEEE, 2009.

Software Quality Assessment and Error/Defect Identification in the Italian Industry Preliminary Results from a State of the Practice Survey

Giuseppe Scanniello
University of Basilicata
Dipartimento di Matematica e Informatica
85100, Potenza, ITALY
giuseppe.scanniello@unibas.it

Fausto Fasano
University of Molise
STAT Dept.
86090, Pesche (IS), Italy
fausto.fasano@unimol.it

Andrea De Lucia, and Genoveffa Tortora
University of Salerno
School of Science
84084, Fisciano (SA), ITALY
adelucia@unisa.it, tortora@unisa.it

Abstract—In this paper, we present the results of a survey aimed at comprehending the relevance and the typology of the software quality assessment approaches and software error/defect identification methods/approaches used in the industrial practice. The context of this study was the IT industry. In particular, we involved industries/organizations that develop and sell software as a main part of their business or develop software as an integral part of their products or services. The results of a preliminary analysis indicated that software quality assessment and software error/defect identification are very relevant and regard almost the totality of the interviewed companies. Furthermore, the widely used and most popular practice is testing, while an increasing interest has been manifested in distributed inspection methods.

Keywords—empirical investigation; quality assessment; error/defect identification; inspection; state of the practice survey; testing.

I. INTRODUCTION

To construct high quality products engineering disciplines check intermediate and final artifacts so that defects can be identified and then removed. Similarly, software development needs complementary combination of design and verification/validation activities to produce and deliver high quality software products [24]. In fact, today it is widely recognized that verification and validation activities are needed to assess and maintain the quality of a software product.

In the software engineering community, there is a growing interest towards surveys investigating the state of the art and practice about the use of processes, methods, and tools within software products development and maintenance [12], [17], [19], [22] as well as for software verification, validation, and review [5], [14].

Surveys are investigations to gather data from respondents, using a questionnaire composed of closed or open questions [23]. Depending on the survey purpose, it may focus on opinions or factual information [16]. Data can be collected by: face-to-face and phone interviews, mail, e-mails, and web pages. E-mail surveys are both very economical and very fast. They are often best for sensitive items, and there is no interviewer bias. On the other hand,

email surveys are limited to simple questionnaires. The data can be analyzed to derive descriptive and explanatory conclusions [2] that are applicable only to the selected population.

In this paper, we present the preliminary results of a survey organized by three Italian Universities – University of Basilicata, University of Molise, and University of Salerno – to understand the state of the practice of software quality assessment and software error/defect identification in the IT Italian industry.

The survey was conducted from the spring 2008 to the winter 2009. We invited to participate 70 companies/organizations that develop and sell software as a main part of their business or develop software as an integral part of their products. We received by e-mail 30 fully completed questionnaires from key people of the invited companies/organizations.

The main findings of the study can be summarized as follows:

Software quality assessment and software error/defect identification are very relevant and regard roughly almost the totality of the interviewed companies. The widely used and popular practice is testing. An increasing interest has been however manifested in distributed inspection methods.

The remainder of the paper is organized as follows: Related work is presented in Section II, while Section III presents the design of the study. The preliminary analysis of the data and the threats that may affect the validity of the results are discussed in Section IV. Final remarks and future work conclude the paper.

II. RELATED WORK

Methods and techniques for software quality assessment and software error/defect identification have been largely experimented in case studies and controlled experiments [8], [18]. A number of systematic reviews and state of the art surveys have been proposed in the literature on these topics [4], [7], [15], [17]. On the other hand, only a few numbers of state of the practice survey have been conducted in the

past [5], [14]. Accordingly, in the following subsections we describe state of practice surveys related to software review and to process and methods for software development and maintenance.

A. Software Reviews

The focus of the survey presented in [5] is the analysis of the current state of the practice in industry regarding the application of reviews and inspections. The major focus is on the concrete application of walkthroughs, peer reviews, and formal inspections. Similarly to our study, the results indicate that there are still many objections against the usage of the techniques considered in the study. The main highlighted concern is that these techniques are perceived as too time consuming and thus not applicable in practice. Differently from us, this state of the practice surveys is only focused on walkthrough and software inspections.

Based on the results discussed in [5] and [21], Jedlitschka et al. [14] conduct a survey to investigate the state of the practice of inspection technology in German software industry decision makers. They involved 92 companies and observed that information regarding the impact of technologies on product quality, cost, and development time, as well as on technology cost-benefit ratio is considered highly relevant for the interviewed decision makers.

B. Software Development and Maintenance

Hauge [12] explores and investigates the open source phenomena in the IT industry. He adopts both a literature study and a web-based survey. The sample is composed of companies from the Norwegian software industry. The results of this study show that the open source is widely used. In particular, he observes that about 50% of the Norwegian IT companies adopt open source code in the marketed software products.

Conradi et al. [19] presents a state of the practice survey on risk management in software development with off-the-shelf software components. The authors interviewed software companies from Norway, Italy, and Germany. The results show that off-the-shelf components normally do not contribute negatively to the quality of the software system. Furthermore, the study also reveals that issues such as the underestimation of integration effort and inefficient debugging remain problematic.

Torchiano et al. [22] reports on a state of the practice survey conducted among 59 Italian software companies. This survey is conducted within a research project [9] and aims at analyzing the state of the practice in software migration. The results of the survey indicate that about 66% of the interviewed companies have some experiences in migration tasks. The study also highlights the lacking of tools for the execution of migration tasks. This however does not seem to constitute a problem for the interviewed companies.

III. DEFINITION AND DESIGN

The goals of the survey we have conducted in the IT Italian industry can be summarized as follows:

Primary goal: comprehending the relevance and the typology of the software quality assessment approaches and software error/defect identification methods/approaches used in the practice.

Secondary goal: identifying the main problems and the actual needs (methods, techniques, and tools).

With respect to the goals, the following research questions have been defined and investigated:

RQ1 What is the relevance of quality assessment and error/defect identification in IT Italian industry?

RQ2 What are the most popular and widely used practices?

RQ3 What are the main problems encountered to employ approaches/methods for quality assessment and software error/defect identification?

RQ4 Is there an interest in never used approaches/methods for quality assessment and software error/defect identification?

The survey has been conducted through the following three steps:

- (i) Designing a common questionnaire that includes the main questions and perspectives;
- (ii) Conducting the survey leveraging the industrial contact networks of the Universities involved in the study;
- (ii) Analyzing the data and packaging the results.

A. Conceptual Model

The conceptual model clarifies the meaning of some terms (e.g., project and inspection) and describes all the entities of interest for the survey.

Project. It represent a completed software project.

Software artifact. It is a tangible product created during software development.

Testing. “The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software items” [1].

Inspection. “A static analysis technique that relies on visual examination of development products to detect errors, violations of development standards, and other problems” [13].

Distributed Inspection. It is a method to support geographically distributed teams in the inspection of software artifacts.

Pair Inspection. It is an informal method for inspecting software artifact. The author’s artifact and an inspector are require to accomplish the inspection.

Walkthrough. “A static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a segment of documentation or code, and the participants ask questions

and make comments about possible errors, violation of development standards, and other problems” [13]

We identified three areas of interest for collecting the data:

Demographic information concerns the interviewed company/organization (company, the hereafter) and the respondents.

Relevance and typology regards information on projects on which the considered quality management methods and approaches have been used.

Main problems and needs is about the issues to adopt the software quality assessment approaches and software error/defect identification methods.

B. Identification of the Target Population

The target population consisted of IT companies that develop and sell software as a main part of their business (e.g., software house) or develop software as an integral part of their products or services (e.g., healthcare domain).

The selection of the companies (*sampling*) has been conducted using the network contacts (for convenience and opportunity) of the research groups of the authors. The contacts network included companies that participated to our research projects [9] and/or employed or hosted (for external stages) students with a Master or a Bachelor degree from the following Universities: University of Basilicata, University of Molise, and University of Salerno.

C. Questionnaire Design and Data Collection

We have developed the questionnaire following the standard schema proposed in [5]. Figure 1 shows the designed questionnaire. The questionnaire contains both open (some required just filling in a comment or text) and closed questions. According to the conceptual model, the questionnaire consists of different questions that depend on the usage or not of software quality assessment approaches and software error/defect identification methods and on the will of employing them in the future.

The questionnaire was introduced with a brief motivation sketching the general problem to be investigated.

The importance of this study and our objectives were inserted in an accompanying letter attached to the questionnaire. Great care was taken to ensure ethical requirements and privacy rules imposed by the Italian regulations. Furthermore, we also clarified that all the information was considered confidential and that the data were used only for research purposes and revealed only in aggregated form.

The respondents sent the answered questionnaires by e-mail. The rationale for using this communication medium was that the companies may consider sensitive the information treated in the survey.

IV. RESULTS

Among the 70 invited companies, 48 gave their availability to participate to the survey, while 32 correctly filled in the questionnaire.

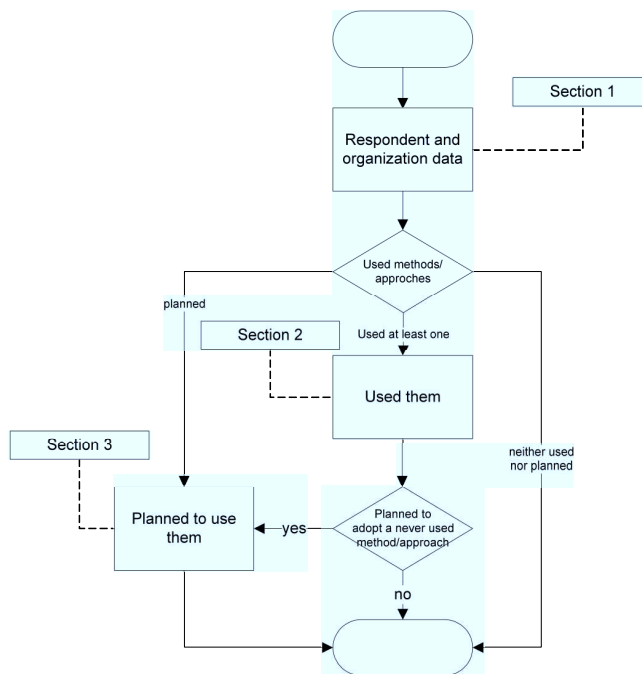


Figure 1. Designed questionnaire

A. Respondents’ Background and Companies Characteristics

The age of the respondents ranged from 24 to 50 years old with an average of 35 years (only 2 were female). Regarding the role of the respondents, 73% of them stated that they had management roles (i.e., projects manager, IT manager, quality manager, or production manager) while 27% were developers or software architects. Among the respondents, 93% had a master degree, 7% had a bachelor degree. None of the interviewed had a PhD. 80% had a specific IT degree.

The interviewed companies were 60% independent and 40% subsidiaries (i.e., controlled by a larger and more powerful company). Among these companies, 67% were private companies and 33% were quoted on the Italian stock exchange. None was a government organization. 53% of the companies were either small or medium-sized enterprises (i.e., < 250 employees), while 47% were larger ones.

Note that larger companies were composed of businesses units. For smaller companies, the number of employees of the business unit coincided with the total number of the company employees. For the larger companies, the size of the business units of the respondents were distributed as follows: 13% were micro (< 10), 67% were small (between 10 and 50), and 20% were medium (between 50 and 250).

The companies come from different industrial domains. In particular, most of them worked in the area of software consultancy (40%). On the other hand, 33% of the companies worked on software development and 27% provided IT services to others.

The typical size of the software systems handled/developed was: from 10 to 100KLOCs (7%), from 100 to 500 KLOCs (80%); more than 500 KLOCs (13%).

B. Relevance of the used practices and influential factors

All the respondents of the companies, that never used the methods/approaches considered in the survey, have planned to adopt at least one among: testing, inspection, distributed inspection, and walkthrough. These were 20% of the companies.

On the other hand, 40% of the companies regularly employed at least one of the considered methods/approaches, while 13% of the respondents declared that their companies often used testing, inspection, distributed inspection, and walkthrough. Finally, 27% of the companies occasionally used them.

Among the companies that have used at least one of the methods/approaches (i.e., 24 out of 30), most of them (i.e., 14 out of 24) regularly used testing techniques, while 4 companies stated that testing has been often used in the past. Only 2 respondents indicated that their company occasionally used testing.

Inspection methods (i.e., variations of the Fagan’s process) to identify defects in software artifacts were already used in only 2 companies, while 10 out of 24 companies often used them. Inspection methods were occasionally employed within 6 companies, while 4 stated that these approaches have never been employed, but will be used in the future. Only 2 companies were not interested in using inspection methods in the future.

Distributed inspection methods have been rarely used within 6 companies. Moreover, 6 respondents stated that his/her company has never used distributed inspection, but this technique will be used in the future to identify defects in software artifacts. Finally, 12 out of 24 companies have never used a method for distributed inspection and did not plan to employ it in the future.

The pair inspection was regularly used in 6 out of 24 companies, while only 2 companies often used this technique. Pair inspection was occasionally used within 8 companies. The respondents of 8 companies stated that this technique was never used. Among these companies, 6 stated that were not interested in using pair inspection in the future.

Walkthrough was regularly used in 4 companies, while 6 companies often employed this practice within their projects. Walkthrough was occasionally used within 10 companies (i.e., 42% of the cases). Finally, 4 companies never used this practice and were not interested in using it.

We also asked to indicate the approach/method the respondents considered simpler, more effective, less expensive, and with a best cost benefit ratio. They identified the pair inspection as the simplest method to apply (i.e., 12 out of 24), while testing was considered the more effective

and with the best cost benefit ratio. The less expensive approach/method was considered the pair inspection. Further details can be found in Figure 2.

The greater part of the respondents (16 out of 24) stated that the methodological aspect is the predominant factor to effectively identify defects and improve the quality of software artifacts. The human factor was indicated as the secondary concern (8 out of 24).

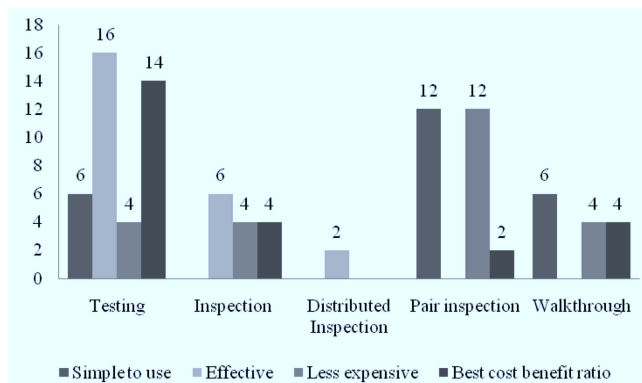


Figure 2. Results of the used practices with respondents considerations

C. Main Problems and Needs

We asked the respondents to indicate the methods they were interested in using among the never used ones. Distributed inspection was the method on which the respondents manifested greatest interest.

The main problems in the industry to adopt variations of the inspection process proposed by Fagan were (in increasing priority order): lack of specialized employees, technique not properly known, technique too much expensive, and lack of time. The companies that never used this method were 10 out of 22.

Similarly, the main problems to employ a distributed inspection process were (in increasing priority order): technique not properly known, short time to market, technique too much expensive, and trust in the technique. Anyway, 16 out of 22 companies were interested in this method.

Regarding, pair inspection 14 companies never used it. Most of them never used this practice since it was not properly known. The factors identified as less influential to use this practice were: technique too much expensive, short time to market, and lack of specialized employees.

The respondents were also asked to describe other techniques used in the company to identify defects within software artifacts. Nobody answered to this question, thus enforcing the assumption that the methods/approaches investigated in our survey are the only employed within the involved companies.

Finally, we asked the respondents whether their company were interested in experimenting inspection, distributed in-

spection, and pair inspection. All of them generally manifested the same level of interest on these methods.

D. Findings

We summarize the main findings emerged from the conducted state of the practice survey according to the defined research questions.

RQ1. The data analysis showed that software quality assessment and software error/defect identification are very relevant in for the IT companies involved in the survey.

RQ2. Testing is the most employed practice. The second larger employed practice was formal inspection based on the original process proposed by Fagan.

RQ3. The main problem to use pair inspection and distributed inspection is related to their scant popularity. On the other hand, the main problem to introduce the other approaches/methods is the short time to market and the lack of properly skilled employees in the company.

Regarding distributed inspection, most of the companies never used this technique for the following three reasons: (i) team members are not geographically distributed; (ii) there is lack of tools for supporting distributed inspection processes; (iii) inspection tools (if available) are not integrated with Software Configuration Management [11] (SCM) systems.

Despite the scant usage of distributed inspection, it aroused great curiosity and interest. This is probably due to the fact that respondents perceived distributed inspection less expensive than traditional inspection. Further, distributed inspection avoids problems related to the different time zones, when synchronous discussions are not accomplished [8].

RQ4. Most of the companies were interested in using inspection, distributed inspection, and pair inspection within pilot projects.

E. Threats to Validity

Internal validity threats regard external factors that may affect the results. In industrial surveys, it is usually impossible to know whether the respondents truthfully answered the questionnaire. Scarce motivation to answer the questionnaire could also affect the results. To mitigate this threat we properly designed the survey. Another factor that may have influenced the internal validity is the number of invited companies that did not answer the questionnaire. Even, the interviewed within our industrial contact network may influence the internal validity. Another threat could be related to the difficulty of comprehending the questions (e.g., ambiguous, not clear, not well formulated). To mitigate this threat, the questionnaire was designed to (i) minimize comprehension problems; (ii) reduce complexity and memory overload; (iii) increase respondent's attention.

External validity concerns the generalization of the results. This threat is present in case of industrial surveys. In fact, we cannot be sure that our sample is representative of the Italian IT industry in general, and we are aware that

Southern Italy is over-represented compared to Northern Italy. Accordingly, replications are needed to increase our confidence in the achieved results.

Construct validity threats concerns the metrics used in the study. In our case, the questionnaire was designed using standard ways and scales [20]. The questions were formulated to minimize possible ambiguities.

V. CONCLUSION AND FUTURE WORK

The survey presented in this study aims at studying and understanding the state of the practice of software quality assessment and software error/defect identification in the Italian industry. Accordingly, we invited 70 companies to participate and received 30 fully and correctly completed questionnaires.

The target population consisted of decision makers in software development. Indeed, we considered IT Italian companies that develop and sell software as a main part of their business (e.g., software house) or develop software as an integral part of their products or services (e.g., commerce in the healthcare domain).

The main results of the presented study show that software quality assessment and software error/defect identification are relevant and regard roughly almost the totality of the interviewed companies. Among the practices considered in the study, software testing is the widely used and popular one. The greater part of the companies that regularly uses software testing is not interested in the approaches/techniques we have investigated in the survey presented here. Future work will aim at investigating this point.

Furthermore, the state of the practice survey and the subsequent interview, in particular, highlighted some further discussion points with respect to the global software development and the quality of software artifacts produced by geographically distributed software engineers:

- (i) First of all, the business units of the respondent are often geographically co-located. This indicates that global software development is only marginally applied in the interviewed software companies. However, in case a company has more distributed business units, they communicate using standard synchronous (e.g., instant messaging) and asynchronous communication media (e.g., email and/or forum).
- (ii) Secondly, there is lack of tools that effectively support distributed teams during software inspections. Despite a number of distributed inspection processes and tools have been proposed [9], [16], the industrial practice is still far to adopt them. This indicates a gap between research laboratories and industrial reality that deserves a concrete cooperation between academy and industry based on technology transfer projects.
- (iii) Finally, the proposed distributed inspection tools are not integrated with the SCM system used in the company. This point is the most critical. Indeed, most of

the software companies use SCM systems to access the level two of CMM and to get ISO 9000 certification, while distributed inspection tools are not widely employed in the industrial practice despite they are recognized useful to improve software quality. We think that the integration of these tools within widely known systems for the management and version control would significantly increase their diffusion, thus improving the quality assurance of software systems developed in distributed contexts. This conjecture needs to be further investigated conducting industrial user studies within industrial software projects.

REFERENCES

- [1] ANSI/IEEE Std. 829-1983: "Software Test Documentation", IEEE Press. Institute of Electrical and Electronics Engineers. Inc.. New York, 1987.
- [2] V.R. Basili, "The Role of Experimentation in Software Engineering: Past, Current, and Future," Proc. 18th International Conference on Software Engineering, 1996, pp. 442–449.
- [3] B. Berliner, "CVS II: Parallelizing software development". In Proceedings of the USENIX Winter 1990 Technical Conference, Berkeley, CA, 1990. USENIX Association, pp.341–352.
- [4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey". In ACM Computing Survey. vol. 41, no. 3, 2009, pp. 1–58.
- [5] M. Ciolkowski, O. Laitenberger, and S. Biffi, "Software reviews: The state of the practice". In IEEE Software, vol. 20, no. 6, 2003, 2003, pp. 46–51
- [6] B. Collins-Sussman, B. Fitzpatrick, and C. Pilato. Version Control with Subversion. O'Reilly, 2004. draft from <http://svnbook.red-bean.com/>.
- [7] R. Czaja and J. Blair, "Designing surveys: a guide to decisions and procedures". 2nd ed. London: Sage, 2005.
- [8] A. De Lucia, F. Fasano, G. Scanniello, and G. Tortora, "Evaluating distributed inspection through controlled experiments", In IET Software, vol. 3, no. 5, 2009, pp. 381–394.
- [9] A. De Lucia, M. Di Penta, F. Lanubile, and M. Torchiano, "METAMORPHOS: MEthods and Tools for migrAting software systems towards web and service Oriented aRchitectures: exPerimental evaluation, usability, and techNology transfer", In Proceedings of 13th European Conference on Software Maintenance and Reengineering, 2009, pp. 301–304.
- [10] A. De Lucia, F. Fasano, G. Scanniello, and G. Tortora, "Integrating a distributed inspection tool within an artefact management system". In Proceedings of 2nd International Conference and Data Technologies, 2007, pp. 184–189.
- [11] J. Estublier, D. Leblang, A. Van Der Hoek, R. Conradi, G. Clemm, W. Tichy, and D. Wiborg-Weber, "Impact of Software Engineering Research on the Practice of Software Configuration Management", In Transactions on Software Engineering and Methodology, vol. 14, no. 4, 2005, pp. 383–430.
- [12] O. Hauge. Open source software in software intensive industry - a survey. Technical report, Norwegian University of Science and Technology Department of Computer and Information Science, 2007. <http://daim.idi.ntnu.no/masteroppgaver/IME/IDI/2007/3290/masteroppgave.pdf>.
- [13] IEEE Std 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology", 1990
- [14] A. Jelitshka, M. Ciolkowski, C. Denger, B. Freimut, and A. Schlichting, "Relevant information sources for successful technology transfer: a survey using inspections as an example", In 1st International Symposium on Empirical Software Engineering and Measurement, 2007, pp. 31–40.
- [15] L. P. W. Kim, C. Sauer, R. and Jeffery "A framework for software development technical reviews". In Software Quality and Productivity: Theory, Practice, Education and Training, 1995, pp. 294–299
- [16] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering", IEEE Transactions on Software Engineering, vol. 28, no. 8, pp. 721–734, August 2002. DOI: 10.1109/TSE.2002.1027796.
- [17] O. Laitenberger. A Survey of Software Inspection Technologies. In Handbook on Software Engineering and Knowledge Engineering, vol. 2, World Scientific Publishing, 2002, pp. 517–555.
- [18] F. Lanubile, T. Mallardo, and F. Calefato, "Tool Support for Geographically Dispersed Inspection Teams", In Software Process: Improvement and Practice, vol.8, no.4, Wiley Inter-Science, 2003, pp.217–231.
- [19] J. Li, R. Conradi, O. Petter, N. Slyngstad, M. Torchiano, M. Morisio, and C. Bunse, "A State-of-the-Practice Survey of Risk Management in Development with Off-the-Shelf Software Components". In IEEE Transactions on Software Engineering, vol. 34 no. 2, 2008, pp. 271–286.
- [20] N. Oppenheim, "Questionnaire Design, Interviewing and Attitude Measurement", Pinter Publishers, 1992.
- [21] T. Punter, "What information do Software Engineering practitioners need?", In Proceedings of the 2nd International Workshop on Empirical Software Engineering, 2003, pp. 85–95
- [22] M. Torchiano, M. Di Penta, F. Ricca, A. De Lucia, and F. Lanubile, "Software migration projects in Italian industry: Preliminary results from a state of the practice survey", In Proceeding of 23rd IEEE/ACM International Conference on Automated Software - Workshop Proceedings (ASE Workshops), L'Aquila, Italy, 2008, pp. 35–42.
- [23] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen, "Experimentation in Software Engineering - An Introduction", Kluwer, 2000.
- [24] M. Young and M. Pezze, "Software Testing and Analysis: Process, Principles and Techniques". John Wiley & Sons, 2008.

Revisiting the Requirements Communication Problem from a Knowledge Management Perspective

Hermann Kaindl and Lukas Pilat

Institute of Computer Technology
Vienna University of Technology
Vienna, Austria

kaindl@ict.tuwien.ac.at, lukas.pilat@student.tuwien.ac.at

Abstract—The communication problem between stakeholders in requirements engineering is well known. It is typically attributed to stakeholders having different background and domain knowledge and, therefore, using different “languages”. However, even when this is not the case, there is an inherent problem in such communication. In order to explain it, we revisit the requirements communication problem from a *knowledge management* perspective.

Keywords—requirements engineering; communication problem; knowledge management.

I. INTRODUCTION AND BACKGROUND

Gathering requirements from stakeholders involves communication, which may be modeled via *communicative acts* as in [3]. However, what is communicated cannot be the requirements per se, but some *representations* of them [4]. So, it is important to understand the consequences of the *transfer* and *transformations* involved.

Knowledge transfer and transformation are central concepts of *knowledge management* (KM) in the context of knowledge sharing in organizations [1]. They are based on the distinction between *tacit* knowledge and *explicit* knowledge, which leads to the spiral of knowledge [5]. In our own previous work [6], we have adopted this concept in form of a spiral of *requirements* knowledge. We have not yet elaborated there, however, on the specific facet of the communication problem related to knowledge transformation.

Since the stakeholders (or their representatives) as well as the requirements engineers are humans, the *transfer* of knowledge can take place either directly through face-to-face exchange using human communication, or through the intermediary of an artifact representing the knowledge to be transferred (e.g., a document). Neither the human communication nor the representation in form of an artifact can transmit the knowledge without communication error and unambiguously.

This results from the fact that every transfer of knowledge is inherently bound to a knowledge *transformation*. The knowledge that a stakeholder has in his mind is typically transformed into an explicit representation in natural language and non-verbal channels of human communication during direct interaction. The knowledge of a knowledge holder is also transformed when he codifies it into an artifact, possibly using a formal

language with a certain expressiveness. Additionally, the receiver of the transferred knowledge transforms codified knowledge through his interpretation of the given representation, which is again error-prone when using natural language due to its inherent ambiguity.

So, in the course of requirements knowledge transfer between stakeholders and requirements engineers, transformation of knowledge occurs and thus errors are creeping in. When the stakeholders speak a different “language,” then the communication problem is certainly reinforced. Our point is, however, that the problem is inherent and exists also if this was not the case.

II. THE REQUIREMENTS COMMUNICATION PROBLEM FROM A KNOWLEDGE MANAGEMENT PERSPECTIVE

In order to explain this inherent requirements communication problem from a KM perspective, let us assume that the stakeholders share the essential domain knowledge and “language”. Still, they have to use some language to express and represent requirements in the course of the corresponding knowledge transfer. We make the following strict distinction here, being aware that various combinations in semi-formal languages exist as well:

- *Natural language*: This is the most widespread kind of language in practical use for communicating about requirements. It is well known that any natural language is inherently *ambiguous*. It is equally important to know that also the *expressiveness* of any natural language is inherently limited. Wittgenstein coined an excellent example: the sound of a clarinet can be easily recognized from having heard it before, but a description of that very sound in natural language is very hard to give.
- *Formal language*: A truly formal language based on mathematical axioms is *not ambiguous*, but it may still be wrongly interpreted by humans for various reasons. Such a language is much more restricted in terms of *expressiveness* than a natural language. So, even more can be “lost” when a requirement is represented in a formal language.

Based on that, let us explain this inherent requirements communication problem from a KM perspective in more detail

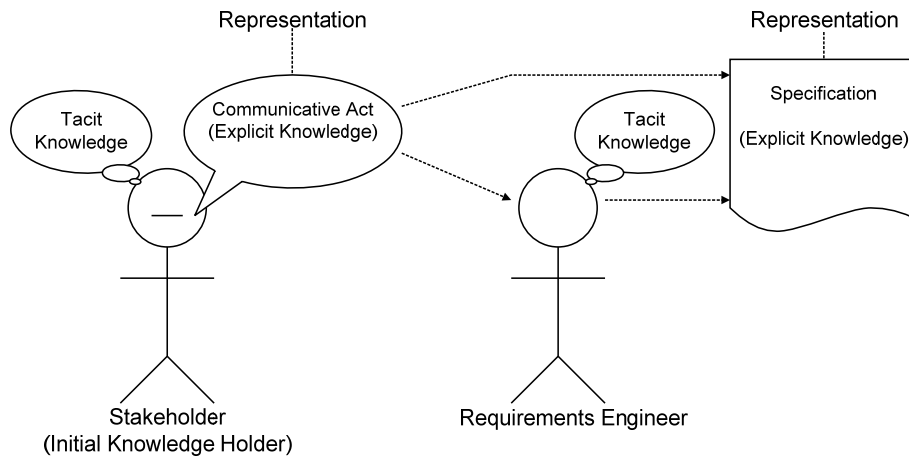


Figure 1. Transfer, Transformation and Representation of Requirements Knowledge

using Figure 1. A stakeholder is the initial holder of some requirements knowledge. It has been recognized that the knowledge of stakeholders is mostly tacit [2] (for the notion of *tacit knowledge* in KM see Nonaka [5], e.g.: “Tacit knowledge is highly personal. It is hard to formalize and, therefore, difficult to communicate to others.”). So, there is tacit knowledge about requirements in the stakeholder’s mind. In the course of requirements elicitation, he tries to communicate his requirements knowledge to the requirements engineer. As indicated above, this can be viewed as a knowledge *transfer* via *communicative acts* as in [3].

In this course, tacit requirements knowledge needs to be made explicit, i.e., a knowledge *transformation* occurs. Usually, the stakeholder will use some natural language. So, what is actually communicated is both restricted through the expressiveness of the particular natural language used and inherently ambiguous. When the requirements engineer attempts to understand the requirement, he has to internalize what is communicated to him explicitly and combine it with his own tacit knowledge. Since the representation in natural language is inherently ambiguous, an error may be induced. If, instead, a formal language is used for this communication, its expressiveness may cause a major difference between the requirements knowledge held by the stakeholder and the one internalized by the requirements engineer.

Based on already changed requirements knowledge as received and internalized from several stakeholders, the requirements engineer will have to prepare a requirements specification, i.e., an explicit representation of the knowledge that he acquired about the requirements. (Of course, he may intentionally change something, in order to figure out the *needs* from mere wishes of the stakeholders, but this is yet another issue beyond the scope of this paper.) Again, depending on the (“mix” of) languages used, the same problems arise as explained above, so that even more errors creep in.

This is an inherent difficulty when doing requirements, addressed by proceeding in an iterative manner. The initial knowledge holder can check his knowledge against the knowledge resulting in the specification. This necessitates that he internalizes the knowledge codified within the specification, and this is, again, error-prone.

Alternatively, a stakeholder may directly represent his requirements knowledge in the specification. This approach avoids the problems through the internalization by the requirements engineer, but it still involves the problems of making tacit requirements knowledge explicit. Of course, this approach entails other problems as well, especially when several stakeholders would simply put a requirements specification together, leading to inconsistencies and even conflicts in the specification.

Finally, it should be noted that tacit knowledge about requirements can also be transferred between the stakeholder and the requirements engineer through *socialization* [6]. As this does not explicitly involve communicative acts, it is beyond the scope of this paper.

III. CONCLUSION

In this paper, we revisit the requirements communication problem from a KM perspective. In this way, we explain a facet of this problem that appears to have attracted less attention. Still, it poses an inherent issue when doing requirements. With an improved understanding of this issue, it may be possible to reduce the resulting errors in requirements specifications.

REFERENCES

- [1] T. H. Davenport, and L. Prusak, *Working knowledge: how organizations manage what they know*. Harvard Business School Press, 2000.
- [2] R. Gacitua, L. Ma, B. Nuseibeh, P. Piwek, A. N. de Roeck, M. Rouncefield, P. Sawyer, A. Willis, and H. Yang, “Making Tacit Requirements Explicit”, in *Proceedings of the Second Int Managing Requirements Knowledge (MARK) Workshop*, 40–44, 2009.
- [3] I. J. Jureta, J. Mylopoulos, and S. Faulkner, “Revisiting the Core Ontology and Problem in Requirements Engineering”, in *Proceedings of the 16th IEEE International Requirements Engineering Conference (RE’08)*, 71–80, Sept. 2008.
- [4] H. Kaindl, and D. Svetinovic, “On confusion between requirements and their representations”, *Requirements Engineering*, Springer London, vol. 15, 307–311, 2010.
- [5] I. Nonaka, *The knowledge-creating company*. Harvard Business Review, 1991.
- [6] L. Pilat, and H. Kaindl, “A Knowledge Management Perspective of Requirements Engineering”, in *Proceedings of the Fifth IEEE International Conference on Research Challenges in Information Science (RCIS’11)*, 48–59, May 2011.