



Ina Schaefer, Dimitris Karagiannis, Andreas Vogelsang,
Daniel Méndez, Christoph Seidl (Hrsg.)

Modellierung 2018

21.02.2018 – 23.02.2018
Braunschweig, Deutschland

Gesellschaft für Informatik e.V. (GI)

Lecture Notes in Informatics (LNI) - Proceedings

Series of the Gesellschaft für Informatik (GI)

Volume P-280

ISBN 978-3-88579-674-9

ISSN 1617-5468

Volume Editors

Prof. Dr.-Ing. Ina Schaefer

TU Braunschweig

Mühlenpfordtstr. 23, 38106 Braunschweig

i.schaefer@tu-braunschweig.de

o. Univ.-Prof. Dr. Dimitris Karagiannis

Universität Wien

Währinger Straße 29, 1090 Wien

dimitris.karagiannis@univie.ac.at

Series Editorial Board

Heinrich C. Mayr, Alpen-Adria-Universität Klagenfurt, Austria

(Chairman, mayr@ifit.uni-klu.ac.at)

Torsten Brinda, Universität Duisburg-Essen, Germany

Dieter Fellner, Technische Universität Darmstadt, Germany

Ulrich Flegel, Infineon, Germany

Ulrich Frank, Universität Duisburg-Essen, Germany

Michael Goedicke, Universität Duisburg-Essen, Germany

Ralf Hofestädt, Universität Bielefeld, Germany

Wolfgang Karl, KIT Karlsruhe, Germany

Michael Koch, Universität der Bundeswehr München, Germany

Thomas Roth-Berghofer, University of West London, Great Britain

Peter Sanders, Karlsruher Institut für Technologie (KIT), Germany

Andreas Thor, HFT Leipzig, Germany

Ingo Timm, Universität Trier, Germany

Karin Vosseberg, Hochschule Bremerhaven, Germany

Maria Wimmer, Universität Koblenz-Landau, Germany

Dissertations

Steffen Hölldobler, Technische Universität Dresden, Germany

Thematics

Andreas Oberweis, Karlsruher Institut für Technologie (KIT), Germany

© Gesellschaft für Informatik, Bonn 2018

printed by Köllen Druck+Verlag GmbH, Bonn



This book is licensed under a Creative Commons BY-SA 4.0 licence.

Vorwort

Die derzeit im zweijährigen Rhythmus stattfindende Fachtagung „Modellierung“ ist eine Plattform zur inhaltlichen Diskussion für eine große Anzahl von Fachgruppen in der Gesellschaft für Informatik (GI), die sich mit unterschiedlichsten Perspektiven des Themas Modellierung beschäftigen. Sie stellt somit ein zentrales Forum für den Erfahrungsaustausch zu akademischen wie auch praxisbezogenen Modellierungsansätzen dar.

Die Fachtagung „Modellierung“ umfasst traditionell ein wissenschaftliches Programm begleitet durch Workshops, Tutorien, Praxisforum, sowie ein Doktorandensymposium und die Präsentation von Tools/Demos. Dabei dienen die Workshops dazu, Spezialthemen der Modellierung im Detail zu beleuchten, während in den Tutorien praktische Anwendungen aktueller Modellierungsansätze vorgestellt werden. Den Tutoriums-Teilnehmerinnen und -Teilnehmern wird dadurch die Möglichkeit eröffnet, nicht nur einen theoretischen Einblick in die Modellierung zu bekommen, sondern den Einsatz der entsprechenden Werkzeuge und Methoden auch in Aktion zu erleben. Abgerundet wird das Programm durch ein Praxisforum zur Vorstellung der Anwendung und Umsetzung von Modellierungsmethoden, -techniken und -werkzeugen in der betrieblichen Praxis sowie ein Doktorandensymposium zur Vorstellung von aktuellen Dissertationsvorhaben. Eine Fokussierung auf die Präsentation von innovativen Werkzeugen wird seit 2016 in dem Tools/Demos Bereich gegeben.

Für das wissenschaftliche Programm der Modellierung 2018 wurden von insgesamt 24 Einreichungen die besten 13 Beiträge ausgewählt. Dies entspricht einer Annahmequote von 54%. Jede Einreichung wurde von drei GutachterInnen evaluiert. Die akzeptierten Beiträge behandeln aktuelle wissenschaftliche Erkenntnisse zu einer breiten Palette von Themen in den Bereichen Modellierungssprachen, -methoden und -ansätze, Prozessmanagement, sowie zur Modellierung im Software- und System-Engineering.

Im Rahmen der diesjährigen Tagung finden zwei eingeladene Vorträge statt: Prof. Dr. Andy Schürr (TU Darmstadt) spricht über die korrekte Entwicklung von Algorithmen für die Adaption von Netzwerktopologien und Nikolaus Regnat (Siemens AG) berichtet über die Verwendung von Modellierungssprachen, wie SysML in der industriellen Praxis.

Wir danken allen Autoren für ihre Beiträge und den Mitgliedern des Programmkomitees und den weiteren Gutachterinnen und Gutachtern für die Begutachtung. Weiterhin bedanken wir uns bei allen Beteiligten aus dem Organisationsteam.

Braunschweig, im Februar 2018

Ina Schaefer, TU Braunschweig

Dimitris Karagiannis, Universität Wien

Christoph Seidl, TU Braunschweig

Sponsoren

Wir danken den folgenden Unternehmen und Institutionen für die Unterstützung der Konferenz.

eck*cellent IT

automotive
engineering **iauv**

SIEMENS
Ingenuity for life

ckc 
group

VOLKSWAGEN

AKTIENGESELLSCHAFT













 **Gobas® Gruppe**
IT mit Struktur

 **BREDEX**

 **fme**

Querschnittsfachausschuss Modellierung

Die Plattform der GI zur Diskussion und zum Erfahrungsaustausch über aktuelle und zukünftige Themen der Modellierungsforschung . Beteiligte GI-Gliederungen:

-  EMISA, Entwicklungsmethoden für Informationssysteme und deren Anwendung
-  FoMSESS Formale Methoden und Modellierung für Sichere Systeme
-  ILLS Intelligente Lehr- und Lernsysteme
-  MMB Messung, Modellierung und Bewertung von Rechensystemen
-  OOSE, Objektorientierte Software-Entwicklung
-  PN Petrinetze
-  RE Requirements Engineering
-  ST Softwaretechnik
-  SWA Softwarearchitektur
-  WI-MobIS Informationssystem-Architektur: Modellierung betrieblicher Informationssysteme
-  WI-VM Vorgehensmodelle für die Betriebliche Anwendungsentwicklung
-  WM/KI Wissensmanagement

Tagungsleitung

Gesamtleitung:	Ina Schaefer, TU Braunschweig
Leitung des Programmkomitees:	Dimitris Karagiannis, Universität Wien
Leitung der Organisation:	Christoph Seidl, TU Braunschweig
Praxisforum:	Andreas Vogelsang, TU Berlin Daniel Méndez Fernández, TU München
Workshops:	Leok Cleophas, TU Eindhoven Michael Felderer, Universität Innsbruck
Tutorien:	Malte Lochau, TU Darmstadt Timo Kehrler, HU Berlin
Tools & Demos:	Hans-Georg Fill, Universität Wien Agnes Koschmider, Karlsruher Institut für Technologie
Lokale Organisation:	Michael Nieke, TU Braunschweig Sven Schuster, TU Braunschweig

Programmkomitee

Jörg Desel	Fernuniversität in Hagen
Jürgen Ebert	Universität Koblenz
Gregor Engels	Universität Paderborn
Ulrich Frank	Universität Duisburg-Essen
Holger Giese	Hasso-Plattner-Institut Potsdam
Martin Glinz	Universität Zürich
Jan Jürjens	TU Dortmund & Fraunhofer ISST
Gerti Kappel	Technische Universität Wien
Thomas Kuehne	Victoria University of Wellington
Florian Matthes	Technische Universität München
Heinrich C. Mayr	Alpen-Adria-Universität Klagenfurt
Jan Mendling	Wirtschaftsuniversität Wien
Günther Müller-Luschnat	iteratec GmbH
Friedericke Nickl	Swiss Life
Markus Nüttgens	Universität Hamburg
Andreas Oberweis	Karlsruher Institut für Technologie
Sven Overhage	Universität Bamberg
Barbara Paech	Universität Heidelberg
Henderik Proper	Luxembourg Institute of Science and Technology
Ulrich Reimer	Fachhochschule St. Gallen
Wolfgang Reisig	Humboldt-Universität zu Berlin
Anne Remke	Universität Münster
Matthias Riebisch	Universität Hamburg
Bernhard Rumpe	RWTH Aachen
Andy Schürr	TU Darmstadt

Elmar J. Sinz
Friedrich Steimann
Stefan Strecker
Bernhard Thalheim
Mathias Weske

Universität Bamberg
Fernuniversität in Hagen
Fernuniversität in Hagen
Christian-Albrechts-Universität Kiel
HPI, Universität Potsdam

Inhaltsverzeichnis

Eingeladene Vorträge

Andy Schürr

*Graph-Transformation-Driven Correct-by-Construction Development of
Communication System Topology Adaptation Algorithms* 15

Nikolaus Regnat

Why SysML does often fail – and possible solutions 17

Wissenschaftliche Beiträge

Julian Dörndorfer, Christian Seel

*A Framework to Model and Implement Mobile Context-Aware Business
Applications* 23

Stefan Tomaszek, Erhan Leblebici, Lin Wang, Andy Schürr

Model-driven Development of Virtual Network Embedding Algorithms 39

Benedikt Pittl, Hans-Georg Fill

Transforming Enterprise Models to Linked Data 55

Zoltán Ádám Mann, Andreas Metzger, Stefan Schoenen

Towards a run-time model for data protection in the cloud 71

Ralf Laue

Nutzung von Bilddatenbanken zur Erstellung von Symbolen 87

Andreas Grosche, Burkhard Igel, Olaf Spinczyk

Exploiting Modular Language Extensions in Legacy C Code 103

Tobias Wägemann, Ramin Tavakoli Kolagari, Klaus Schmid

Optimal Product Line Architectures for the Automotive Industry 119

Khanh-Hoang Doan, Marti Gogolla

Extending a UML and OCL Tool for Meta-Modeling 135

Alexander Rauh, Wolfgang Golubski, Stefan Queins <i>Measuring the Quality of System Specifications in Use Case Driven Approaches</i>	151
Daniel Gritzner, Joel Greenyer <i>Synthesis of Cost-optimized Controllers from Scenario-based GR(1) Specifications</i>	167
Dilshod Kuryazov, Andreas Winter, Ralf Reussner <i>Collaborative Modeling enabled by Versioning</i>	183
Peter de Lange, Petru Nicolaescu, Thomas Winkler, Ralf Klamma <i>Enhancing MDWE with Collaborative Live Coding</i>	199
Tilmann Stehle, Matthias Riebisch <i>Modellierung plattformübergreifender Quellcode-Entsprechungen für die koordinierte Co-Evolution portierter Software-Systeme</i>	215

Praxisforum – Eingeladene Industriebeiträge

Harald Störrle <i>Implementing Knowledge Management in Agile Projects by Pragmatic Modeling</i>	233
Markus Grabowski, Bernhard Kaiser, Yu Bai <i>Systematic Refinement of CPS Requirements</i>	245
Oscar Slotosch, Mohammad Abu-Alqumsan <i>Modeling and Safety-Certification of Model-based Development Processes</i>	261
Christian Reuter <i>Controlled Complexity for Future Mobility – Methodology, Guidelines and Tooling</i>	275
Daniel Ratiu, Holger Nehls, Robert Walter, Jochen Michel <i>Taming the Software Development Complexity with Domain Specific Languages</i>	281

Tutorials

Thomas Thüm, Sebastian Krieter, Thomas Leich <i>Feature Modeling and Development with FeatureIDE</i>	297
Daniel Strüber, Alexandru Burdusel, Stefan John, Steffen Zschaler <i>Henshin: A Model Transformation Language and its Use for Search-Based Model Optimisation in MDEOptimiser</i>	299
Lars Fritsche, Géza Kulcsár <i>eMoflon: A Tool for Tools and Transformations</i>	301

Werkzeugpräsentation

Prof. Dr. Gabriele Roth-Dietrich, Prof. Dr. Rainer Gerten, André Schäfer <i>Graphical App Designer</i>	307
Santiago Velasco, Jan Reich, Maxime Tchanguou <i>Interactive information zoom on Component Fault Trees</i>	311
Andreas Drescher <i>Eine musterbasierte Kontrollflusssemantik zur interaktiven Simulation</i> . .	315
Sven Jannaber, Benedikt Zobel, Lisa Berkemeier, Oliver Thomas <i>Development of a prototype for Smart Glasses-based process modelling</i> .	321
Benjamin Ternes, Stefan Strecker <i>A web-based modeling tool for studying the learning of conceptual modeling</i>	325

Autorenverzeichnis

Eingeladene Vorträge

Graph-Transformation-Driven Correct-by-Construction Development of Communication System Topology Adaptation Algorithms

Andy Schürr¹

Extended Abstract: How will the Internet of the future look like? Which communication mechanisms - as we know them today - will prevail, which novel forms of communication will emerge, and what are the challenges faced regarding the constantly increasing mobile use of communication networks? The Collaborative Research Center 1053 MAKI (Multi-Mechanism-Adaptation for the Future Internet) addresses these questions with a specific focus on the development of offline/online adaptation and optimization concepts for all kinds of communication mechanisms. Twelve subprojects, clustered in three project areas, study for this purpose communication system construction and adaptation mechanisms on different network layers for application domains such as Complex Event Processing, Wireless Sensor Networks, ...

In the Wireless Sensor Network domain (as well as in many other application domains), a large research area focuses on Topology Control (TC), which aims at optimizing the network topology (i.e., the graph-based structure of a communication network) to achieve certain optimization goals (e.g., reducing energy consumption and increasing lifetime of battery-powered nodes) while preserving crucial integrity constraints (e.g., connectivity of the topology or coverage of an observed area). A TC algorithm takes a "raw" topology as input and returns a topology as output that consists of the same nodes but only a subset of the edges (connections) of the input topology. The output topology should perform "better" w.r.t. the specified optimization goals while still fulfilling its integrity constraints.

A typical development workflow for TC algorithms (i) starts with a more or less formal specification (e.g., based on atomic graph operations or constraints that the output topology takes into account) that is then employed to prove its desired and required properties on a high level of abstraction. Afterwards, (ii) the TC algorithm is implemented (often in some general-purpose language such as C/C++ or Java) and tested inside a network simulator or - less often, unfortunately - in a hardware testbed. While numerous approaches exist that facilitate the way from simulation to testbed (e.g., using platform-independent APIs or UML-based code generation), only few attempts have been made to ensure that the implemented Topology Control algorithm is still correct w.r.t. the specified set of integrity constraints!

We propose a model-based approach to overcome this reliability problem of TC algorithms by (i) specifying their desired/required properties using graph constraints, (ii) turning these graph constraints into correct-by construction programmed graph transformations, (iii) adjusting the graph-transformation-based specification to the particularities of the target platform (e.g., limited local knowledge of nodes), and (iv) generating finally platform-specific Java code for simulation and C code for testbed evaluation purposes.

¹ Technische Universität Darmstadt, Fachbereich Elektrotechnik und Informationstechnik, Fachgebiet Echtzeitsysteme, Merckstr. 25, 64283 Darmstadt, andy.schuerr@es.tu-darmstadt.de

Why SysML does often fail – and possible solutions

Nikolaus Regnat¹

Extended Abstract: Did you try to bring model based development into your organization and had a hard time? Did you work hard preparing the process, methods and tools but despite all trainings and support, users did not adopt your model based development approach as expected? Did you experience low model quality and keep struggling to provide the needed model quality assurance? Well, you are probably not alone. Let us use the SysML as a prominent example to describe why modeling languages often do not live up to their expectations and why introducing them into an organization frequently fails.

It typically starts with a system architect who read or heard about model-based approaches and SysML. She/he may start talking with some of her/his colleagues and together they may convince their management that they should get rid of their hand-written, often huge and inconsistent, system architecture documents and replace them by a model-based approach. System architects are typically domain experts but very rarely modeling experts so they start to research the topic further. There are countless books regarding model-based development in general and the SysML in particular. There are numerous trainings available; big companies like Siemens may even offer in-house trainings. Let's assume that the system architects will get such trainings and have at least time to skim over one or two books. The best outcome they can get is that they now know a little bit more regarding the SysML language and realize that they would have to define how to apply the SysML in their organization. In other words: they know they need to develop a method that deals with their particular needs and describes how to apply the SysML in their specific use case. There are several possible scenarios that might follow; unfortunately most of them will often lead to failure.

The first scenario would be that the system architects try to introduce the SysML into their organization by themselves. This is a typical situation when the management is hesitant to invest larger amounts of money or the system architect is convinced (after reading those books and attending trainings) that he can do it by her/himself. Based on our experience this will not turn out right most of the time. While the definition of a method how to apply SysML is a hard task in itself one has also to consider countless other things: What is the purpose and goal of the modeling approach? Who are the stakeholders that directly or indirectly have impact on the approach? What are the skills of the potential users? How to train these users? How does the approach fit to the existing organizational structure and processes? How does the approach integrate into the existing tool landscape? Failure to consider these (and many other) things will typically lead to a modeling approach that does not fulfill all requirements of the

¹ Siemens AG, Corporate Technology, Research in Digitalization and Automation, Architecture Definition and Management, Otto-Hahn-Ring 6, 81739 München, nikolaus.regnat@siemens.com

organization and therefore is destined to fail.

The second scenario would be that the system architects realize that they need help to accomplish the task. If the management can be convinced to invest more money that typically means some consultants are hired. If these consultants are worth their money they will surely think about the above topics and define a modeling approach that fulfills all requirements of the organization. Unfortunately that does not mean that the introduction is successful in the long term. The worst situation is often when the consultants are also hired to do the creation of the models themselves. This is unfortunately still happening far too often, typically when the system architects are busy with their daily project work and the management thinks it is a good idea to “help” them this way. Not only will this successfully prevent any knowledge building within the organization itself, it will also often lead to a low acceptance of the approach: it often leads to the not-invented-here-syndrome of the organization. And when the contract with the consultants is not longer extended this usually means that there is no one taking care of the models any longer and the whole approach is soon to be ignored and forgotten.

The third scenario would be that either the system architects (first scenario) or the hired consultants (second scenario) did such an awesome job that all organizational requirements have been considered and a feasible modeling approach has been defined. Surely this will lead to a huge success and lead to another example of a successful introduction of a model based approach into an organization? Unfortunately, more often than not this is still not the case. Let’s go through the three most important reasons why the introduction of a model-based approach into an organization might still fail.

Lack of management support is one of the main reasons. Without the management actively contributing to the success of the approach failure is often inevitable. People need to be both motivated (in regards to their goals; e.g. “we want to replace our hand-written documents within the next fiscal year”) as well as given enough time to get accustomed to the new way of working. However, more often than not system architects do not get enough time to actively work with the modeling approach as their day-to-day business is taken priority. Moreover, the model-based approach has to be constantly tailored and improved to suit the changing needs of the organization and the management has to actively support this too.

Lack of proper model quality assurance is another main reason. It typically does not cause short-term issues but will often lead to failure in the long-term. Unfortunately many organizations realize this too late: models that are not constantly monitored using both a combination of automated checks and manual reviews will massively degrade over time. It often results in something we call “model graveyards”: an unstructured collection of data, often with unreadable and unexpressive diagrams, unused model elements (deleted only from diagrams but not from the model) that is inconvenient to extend and hard to maintain. Fixing these issues will often cost so much time that it is not feasible to do it.

The third most prominent reason for failure unfortunately is the SysML and

accompanying tools themselves. It is the prominent lack of focus on the end users. The SysML was built *by* modeling experts *for* modeling experts ignoring the fundamental fact that most end users are not and probably never will be modeling experts. To make things worse, the typical SysML tools were also built with a focus on modeling experts in mind. This combination of a general purpose language that lacks any method with a tool that is focused on expert users leads to a situation that is the downfall of many modeling approaches. Users will find themselves very often in a similar situation. They have gotten training on SysML and training on the method developed for their organization. However, most users will not use the modeling environment on a day-to-day basis and thus will only very rarely become modeling experts. They therefore struggle every single time they use the modeling language and accompanying tool. Users will typically have to go back and forth between guideline and tool to find out how to model, resulting in a massive productivity loss. Instead of being able to work on their original task (i.e. describing their system architecture) they struggle to handle the language and tool. This typically leads to frustrated users, lowering the acceptance of the modeling approach massively. Users will try to avoid using the modeling language and tool and find creative ways to circumvent the approach. Things that cannot be expressed properly in SysML are not improved in the modeling approach but instead other tools are used to draw “pretty” pictures. Existing models will get out of date and in the long run will not be used any more. This goes on until one got back to the point where all started: hand-written documents with some pictures or diagrams in it.

Knowing all that what can be done to improve the situation and successfully introduce a model-based development approach into an organization? Based on our experience regarding model based development in industry one has to focus on the three main aspects mentioned above. From day one ensure that the management supports the approach and make clear what is needed. Establish a model quality assurance process as soon as possible. And focus on the user experience. This cannot be stressed enough: even with proper management support and established quality assurance it is the acceptance of users that make or break the success of a model-based approach.

But what can be done immediately so that both modeling languages and tools improve on the situation? We think that it is the main responsibility when defining a modeling approach for an organization to improve the user experience as much as possible. The language has to focus on the user’s needs and do not provide unnecessary things. The tool has to support the defined method and guideline as much as possible. Every stumbling block for the users that can be removed has to be removed.

Some tool vendors have built their tools so that this can be already done in a sensible way. Unfortunately other tools only provide very limited capabilities to do so. As always: you have to choose the right tool for the right job. We are currently focusing on MagicDraw, as it supports a wide range of customizations that enhance the user experience. During the last years we have systematically worked on correcting both language and tool shortcomings. We’ve started with the easy things: we provided model templates for our users. These templates match both the structure defined in the

guideline as well as the generated documents as far as possible. We then used the tool customization possibilities of MagicDraw to guide the user wherever possible: instead of allowing to create every element and diagram everywhere the context menu only allow to do the things that are sensible in that context. We did not force users to remember (or read in the guideline) what additional stereotypes they have to add to various SysML elements. Instead we created new element as well as diagram types for them. We also worked on improving the visualization of model information. Elements may change their color, show icons etc. based on properties of the model. We even let users setup these things in an easy and consistent way: the user can define a connection type (e.g. CAN Bus) by himself and give that new type a distinct color. The tool will take care of this and ensure that both icons and diagram elements show this color whenever the type is used. Did you ever model a SysML Block with 10 ports an instantiate that block? Depending on the used tool the user might end up with a cluster of ports that one has to manually arrange again, every time an instance is created. This is just an awfully bad user experience, and SysML tools did it this way since they have been made available. Using the tool API we've implemented a very small extension that allows defining the port layout of elements once and whenever an instance of such an element is created the ports show up as defined. It is such a small thing but modeling experts and tool vendors alike ignored it and nobody told the tool vendors to change this (well, we did and MagicDraw now has such an option built in). When putting a strong focus on the user experience one can easily identify dozens of things that should have been improved a long time ago. And it is often a combination of small things that lead to a bad user experience.

The attentive reader might have already guessed it: ultimately we went the long road of implementing domain specific languages (DSLs) based on the UML/SysML using MagicDraw. We went even further and created a set of building blocks that allow us to efficiently create new DSLs with manageable effort. Our experiences in various projects over the last 5 years are more than promising: the user acceptance is very high and the overall model quality is far better at a reduced effort than anything we've done with SysML. All of this also helps to convince the management that the approach leads to an increased productivity and improved quality; in other words to a good return on invest.

Now, we don't propagate that everyone should go this route; instead we want to show that with a strong focus on the end users we could have been much further regarding the widespread acceptance of modeling languages and accompanying tools. Our approach shows what is already possible when customizing and adapting both languages and tools. What could we do if languages and tools would have been better at this? To re-enforce my former statement: both the modeling community and tool vendors need to step back and re-think what has been done in the past. It is not the goal to train domain experts to be modeling, language and tool experts just to be able to do something meaningful. The goal is to enable domain experts to focus on their challenges by providing modeling languages and tools that empower them to do so. Without us focusing on the user experience the introduction of modeling approaches into organization will continue to fail.

Wissenschaftliche Beiträge

A Framework to Model and Implement Mobile Context-Aware Business Applications

Julian, Dörndorfer¹ and Christian Seel¹

Abstract: The success and ubiquity of mobile devices like smartphones and tablets changed the daily work activities of many employees and employers. With the data provided by mobile devices in combination with external data from databases or web services, it is possible to recognize the context of a business process. Context recognition allows to adapt the business process by selecting the next process step or providing the necessary data, like the next customer near the current location. However, to use the advantages of context recognition, mobile business processes have to be designed, implemented and executed in a context-aware manner. Therefore, this paper presents a comprehensive framework for modeling context-aware business processes, which comprise the business process as well as the information collection to evaluate the needed context. Furthermore, it presents an architecture for the realization of context-aware applications.

Keywords: Context-aware Business Processes, BPMN, Domain Specific Modeling Language (DSML), Mobile Architecture

1 Introduction

With the introduction of the iPhone ten years ago a boost of mobile devices like smartphones or tablets took place, which increasingly leads to a disruption of “traditional” work conditions and executions [Pr16], [In13b], [Mo14], [KK14], [Rh13], such as waiters using smartphones to accept orders from the customer and billing the meal with it. The emerging generation of digital natives which are entering more and more the job market will further drive this change from stationary to mobile working conditions [Pr16]. The widespread use of mobile devices is leading to business processes being executed independent of the location.

The design, implementation, execution and controlling of business processes is a standard approach in theory and practice [HC93], [BKR11], [VR10], [Sc00], [Bi16]. The nearly ubiquitous presence of mobile devices can be used to support the execution of business processes. The use of mobile devices improves the quality and flexibility of business processes, and also saves time and costs during the execution [FL14], [HL15]. Another aspect is that mobile devices are providing many sensors and can be additionally equipped via *Bluetooth* or other proprietary protocols with more sensors. Moreover, mobile devices are capable to request additional data from other sources, like databases or web services, via their internet connection.

¹University of Applied Sciences Landshut, Computer Sciences, Am Lurzenhof 1, 84036 Landshut,
julian.doerndorfer|christian.seel@haw-landshut.de

The aggregation and interpretation of sensor data enable detecting the context of the users and support them during the execution of a business process. DEY [De01] describes context as “any information that can be used to characterize the situation of an entity.” Therefore, context recognition can be used to automatically present useful information for the user or adapt the application behavior and moreover pre-select the next possible process steps. To support the design and implementation of supportive mobile context-aware applications, the business process languages also have to consider contextual influences [DS16]. In addition, the aggregation of context through sensor data can be complex. For instance, the context parameter *weather* basically consists of the sensor data *humidity*, *wind speed* and *temperature*. Moreover, *weather* can also be a sensor for another context parameter, like the *street conditions* in a navigation system. To use the opportunities of context recognition via mobile devices and ease the design of mobile context-aware business processes, this paper answers the following research questions.

RQ.1: How can a modeling framework for mobile context-aware applications be designed?

RQ.2: How can the architecture of a mobile context-aware application be built?

The remainder of this paper is structured as follows: In section 2 a brief overview of the existing literature will be given. Afterwards, the modeling approach will be presented, which comprises a Business Process Model and Notation (BPMN) extension and a sensor modeling language (**RQ.1**). Section 3.3 discusses an approach for an architecture for mobile context-aware applications (**RQ.2**). Hereafter, section 4 shows an example application which supports the business process from the previous section. The paper ends with a conclusion and outlook to further research in mobile context-aware applications.

2 Related Work

Besides DEY context was also defined by others [SAW94], [SBG99], [We91]. Not only is DEY’s definition well known and accepted in the scientific community, but he also declares when an application is context-aware. It is context-aware “if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task” [De00]. Some approaches to make business processes more flexible have been made. ROSEMANN et al. [RRF08] claim that modeling languages have to be more flexible to model context. Further, they state that an increased attention on flexibility took place in the research area, which leads to a decreasing time-to-market for products [RRF11]. Therefore, the result is a demand for higher process flexibility [So05]. In particular, ROSEMANN et al. show the limitation of the actual event-driven process chain (EPC) language and the lack of supporting context modeling. However, they do not present an appropriate way to integrate the identified context in business processes. The extension C-EPC is an approach to make EPC more configurable for decisions at runtime, but it doesn’t address the context in particular [Rv07]. The approach from LA VARA et al. aims to reveal all possibilities of a business process and integrate them into one process model. However, this leads to large and complex models. In [SN07] an approach to identify and apply

context in a business process is introduced, but it is more of a theoretical framework to identify context and does not show how a context-aware business process could be designed. In [HS15] HEINRICH and SCHÖN mention that business processes have to consider “not-static” context events that change the process execution. Moreover, they present an algorithm which supports automated process planning for context-aware processes, but no modeling representation in BPMN. CONFORTI et al. present an approach to manage process risks by sensor evaluation [Co13]. Furthermore, they show a way to model these risks and when they occur. They only consider sensor data and context evaluation for process risks, but context and its evaluation can be used for more than risk analysis for business processes. DÖRNDORFER and SEEL present in their paper a BPMN extension for business processes, which is able to model context in business processes via a complete modeling technique [DS17]. They also developed a context-free grammar to state brief context expression for decisions depending on context. Furthermore, two extensions for UML are published by AL-ALSHUHAI and SIEWE. In the first paper, they are extending the class diagram with additional annotations [AS15b]. The second paper [AS15a] expands the activity diagram to mark context-aware areas or sequences. All the presented papers do not consider how complex context evaluation is and how context-aware business processes can be designed. Furthermore, the implementation of supportive mobile applications is not supported by the modeling languages. The area of domain specific (modeling) languages (DS(M)L) is partly covered by the following articles. A DSL for multiple mobile platforms [KCO10] and for context modeling in the context-aware system [HGB13] were found. Both articles introduce a DSL which enables a description of context. However, both approaches do not present a possibility to model the context aggregation graphically. SHENG and BENATALLAH [SB05] are introducing an adapted UML to enable context modeling. In addition, BERARDINELLI et al. [BCD10] also present an extension for the UML. However, both approaches are not integrated into BPMN, which seems to be difficult. Therefore, they cannot be used for further development.

For the architecture of mobile context-aware applications two main paradigms exist. Firstly, the client approach, which is to recognize context information on the mobile device. The architecture reminds of a traditional *model view controller* (MVC) approach [BA11], [Ch08], [KKC11], [Sh12]. Recognizing and preparing context information on a server is the second approach. All devices which can collect data via sensors send their information to a backend application that evaluates the context information [VL12], [JKR01], [DAS01], [He05]. Both approaches have specific disadvantages. With the server approach all collected data have to be sent to the server, the data have to be evaluated and at last the result (context) has to be sent back to the mobile device. This requires an internet connection, which is not always available for a mobile device. In addition, the described data transmission sometimes leads to unnecessary delays, when the context evaluation can also be conducted on the mobile device. The client approach, on the other hand, is limited to the resources of these devices. If big data packages have to be evaluated, the processor or the memory can be pushed to their limits, thus delaying the process. In addition, some information relevant to evaluate the correct context, might not be accessed from a mobile device.

Therefore, this paper presents a context-aware framework which firstly enables to design mobile context-aware business processes. Secondly, it shows a modeling language for evaluation of context. And thirdly it shows how context-aware application can be designed to support the conduction of context-aware business processes. Thus, this paper presents a new artifact in the design science paradigm by HEVNER et al. [He04].

3 Modeling a Mobile Context-aware Application

To increase the efficiency of business processes and ease the planning and implementation of mobile context-aware applications, this paper introduces a framework (**RQ.1**). On the left side of **Fig. 1** are the context parameter which influences the process. The first step is to create a context-aware business process (first layer). The modeling language of choice is Context4BPMN [DS17] because it extends the standard BPMN to enable the creation of context-aware business processes. It uses so-called *Context Expressions* as conditions for paths or elements depending on context. To evaluate the *Context Expression* the domain specific modeling language *SenSoMod* can be used (second layer). It enables to model how information from sensors can be aggregated to context information (cf. section 3.2). The model hereafter can be used to create a mobile context-aware application which supports the execution of the context-aware business process (third layer).

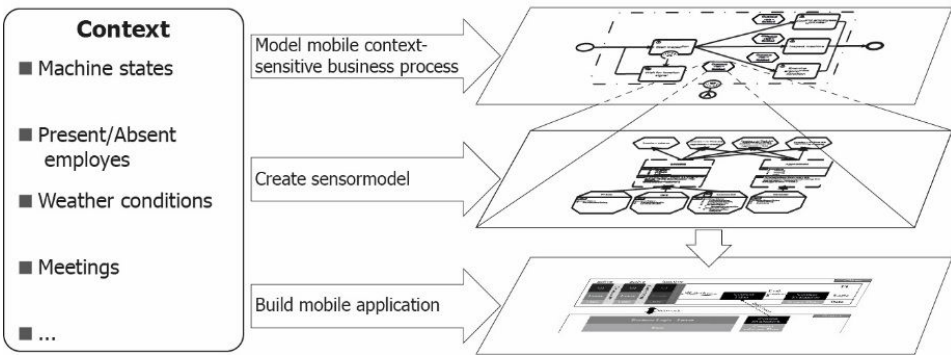


Fig. 1. The framework to model and create a mobile context-aware application

An example of a mobile context-aware business process are traveling salespersons in a sales department of a company. They partly work at the customer's to sell the products from the company as well as in the company to participate in meetings or further training. Moreover, some paperwork can also be done at home. Therefore, in this example the first context parameter is the *location* with three states *at the customer*, *at the office* and *at home*. Other parameters are the *season* and the *weather* because some goods can be better sold in specific seasons or in certain weather condition [Mu10], [Bu12]. In addition, with the context parameter *customer history*, the mobile application is capable to adapt the list of goods to the preferences of the customer. For instance, the non-food products can be

deactivated if the customer only bought food-related products so far. Therefore, we use this example to demonstrate the framework.

3.1 Context-Aware Business Processes

To reach higher flexibility in business processes, context has to be considered in modeling languages. BPMN is a standard modeling language for business processes [In13a] and has a built-in extension mechanism [Ob11]. The mechanism was used to create the Context4BPMN extension [DS17]. **Fig. 2** depicts the business process of the salesperson mentioned in section 3 at design time.

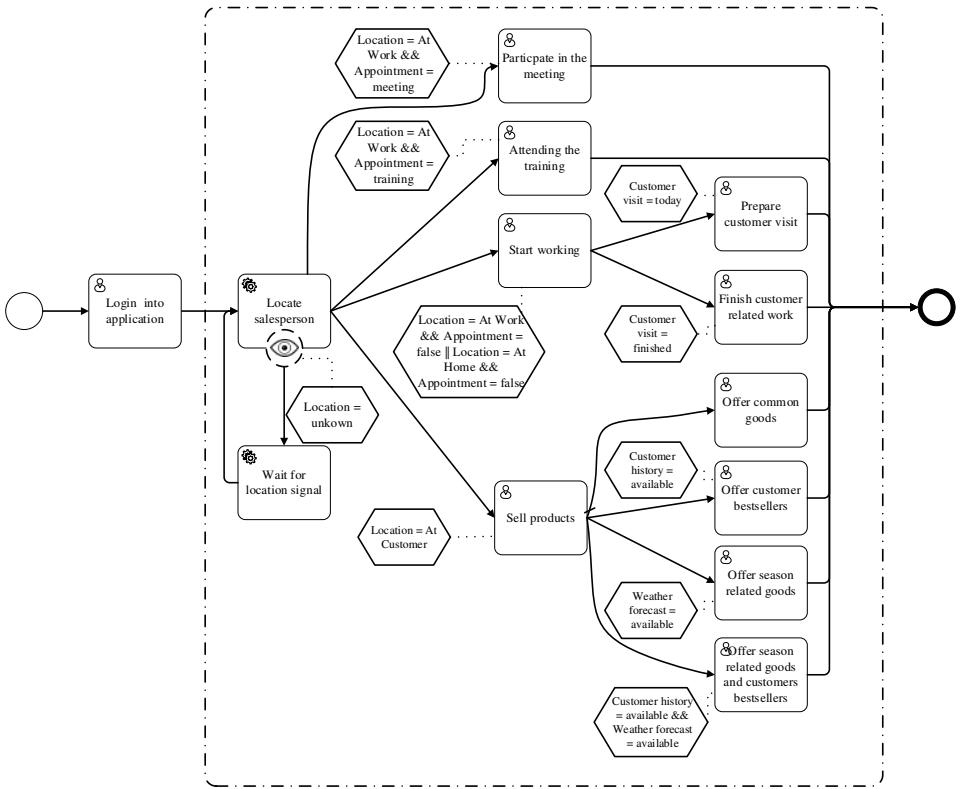
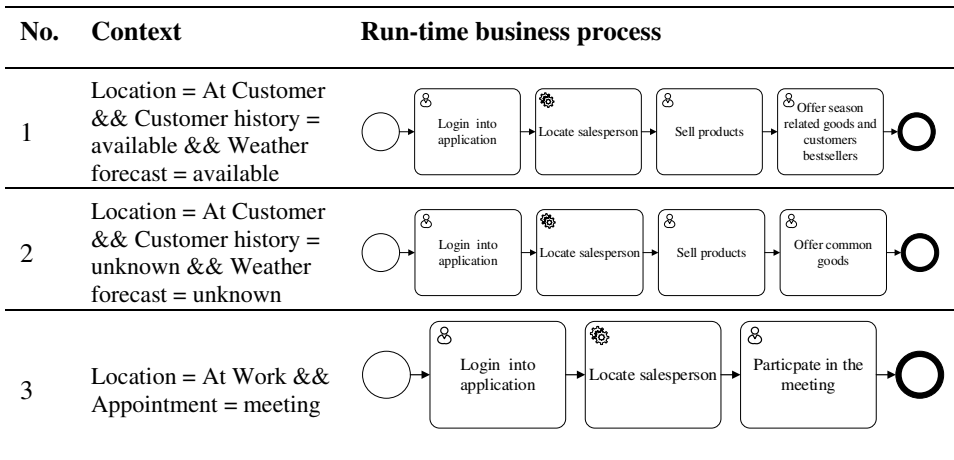


Fig. 2. The salesperson business process designed with the BPMN extension *Context4BPMN*

The first action after starting the application is log in. Hereafter, a dotted line marks the ongoing business process sequence as context depending. After the login, the application locates the salesperson. Attached to the task *locate salesperson* is the *context event* symbolized through an eye. It is a *non-interrupting context event* which is triggered when the location is undefined. The activation condition is stated in the hexagon with a context-free grammar to formalize the conditions. A parallel activity starts in which the application

tries to determine the location of the salesperson. If a location is identified, four different tasks are possible. They all mainly depend on location. For example, the salesperson takes part in the sales training if s/he is at work and a training is noted on the calendar. A different path is when the salesperson is at the customer. Obviously, s/he wants to sell the products of the company to the customer. Depending on the accessible data, the mobile application can support this process by showing the customer’s bestsellers and/or seasonal depending goods. The first action needs the customer history which can be queried via a connection to the customer’s relationship management system (CRM). The seasonal goods can be shown when the weather forecast for the local area is available.



Tab. 1: Variations of the salesperson business process at run-time

The business process will be transformed at run-time into a standard BPMN workflow. **Tab. 1** depicts three variations of the business process for three different context situations. The first row depicts the process when the salesperson is at the customer, the customer history is available and also the weather forecast is available. Therefore, the salesperson offers the customer his bestsellers and seasonal related goods. In the second row, the only determined context is that the salesperson is at the customer. Hence, s/he sells the common goods of the company. In the third variation, the customer is at work and has an appointment with colleagues on the calendar. So s/he participates in the meeting.

3.2 Sensor Modeling

The extension of the BPMN enables to consider static and non-static context events. It is therefore possible to design context-aware business processes. However, the extension of the BPMN does not specify how context can be evaluated by sensors. For instance, how can the context *location* be measured? This sounds like an easy task because the obvious

answer would be via GPS, but where is the location *at home*? When can it switch to *at work*? To address these questions the sensor modeling language has been developed. To further ease the design and implementation of mobile context-aware applications which support the execution of context-aware business processes, an additional language is needed.

Notation of SenSoMod

Context is measured through sensors. We understand a sensor as any source of information for context. This can be a “usual” physical sensor, – like a hygrometer or a temperature sensor – a database, or an application from which information could be requested. Even a machine in an assembly hall that is accessible through a network connection can be a source of information. Therefore, different types of sensors have to be distinguished. There are atomic sensors which cannot be aggregated from other sensors. Two kinds of atomic sensors exist: *Physical sensors*, which measure physical quantities like temperature or humidity, and *virtual sensors* which are dedicated to non-physical quantities like databases, machines or stock states. Besides the atomic sensor, there is the computed sensor. It is a sensor that relies on other sensors, which could be atomic sensors or other computed sensors. For example, the sensor *weather* could be the combination of the atomic sensors *humidity* and *temperature*. To address the different kind of sensor types we gave each of them a graphical representation (cf. **Tab. 2**).

Element	Notation	Element	Notation
Physical atomic sensor		Context	
Virtual atomic sensor		Context description	

Element	Notation	Element	Notation
Computed sensor	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <div style="text-align: center; border-bottom: 1px solid black; padding: 2px 5px;">Sensorname</div> <div style="border-bottom: 1px solid black; padding: 2px 5px;"> Out Variablename:Type <ul style="list-style-type: none"> • Element </div> <div style="padding: 2px 5px;"> DL If(Expression) then(Assignment) Else(Assignment) </div> </div>	Flow	→
Multiple Instances	III		

Tab. 2: The notation of SenSoMod

The *physical sensor* is designated to depict physical quantities, whereas the *virtual sensors* are dedicated to non-physical quantities. To differentiate these two atomic sensors, the virtual sensors are marked by a database symbol to the left of the sensor name field. Beneath the name is the output area marked with *Out*. The area is intended to describe the structure of the outgoing objects of the sensor like the type and elements of it. First the name and – separated by a colon – the type of the return object have to be stated. If the object type has specific values – like enums, arrays or lists – they can be stated in a bullet list. Alternatively, the return values can be specified in the JavaScript Object Notation (JSON) RFC-7159 [In14] notation. Next to the two atomic sensors is the *computed sensor*. It can be used to combine different atomic sensors and/or *computed sensors*. Weather, for example, can be aggregated from different types of sensors. *Computed sensors* also have an area for describing outgoing elements. Furthermore, they have a *DL* area dedicated to describing the decision logic for their outgoing objects. This is necessary to express when a certain state from the *Out* area of a sensor will be returned. For example, the location *at work* will be returned when the logged-in network name is *companyNetwork*. The accurate description of the decision logic language is presented in **Fig. 3**. The *context* notation is the next element in the table. The name of the element has to match with the name of the involved context in the *context description*. Only *context* elements can be connected with a *context description*. Like the sensors, the *context* has the *DL* and *Out* areas. A *context* is based on at least one sensor, of some type. The *context description* is the graphical representation of an expression to describe a contextual influence in a business process. The context expression language is part of the Context4BPMN extension [DS17]. To connect the different elements and show the sequence stream, the *flow* element has to be used. *Multiple instances* is the last element in the table and is an attribute for any type of sensors. It indicates that a sensor occurs in more than one instance, and is placed to the right of a sensor name. An example, using the introduced elements, is given in **Fig. 4**. **Fig. 3** depicts the logical decision language which has been developed. The language is a context-free grammar in the Extended Backus Naur Form (EBNF) [Ba60] and serves to briefly and precisely express the decision in the *DL* area of the *sensor* and *context* elements. A decision logic consists of a *LogicTerm* and can have a default assignment. The *LogicTerm* itself is built out of an *Expression* and an *Assignment*. If the expression is

evaluated as *true*, the variable will be assigned to the designated value. The assigned variable has to exist in the output area of the computed sensor or context. To shorten some basic definitions, like integer numbers or date, we link with "-->" to standards, like strings. In order to express the structure of the object which a sensor returns to a request, the "Out" area is provided. First the name and – separated by a colon – the type of the return object have to be stated. Optionally, the values of the type (if existing) could be stated in a bullet list. Alternatively, the return values could be specified in the JSON notation.

```

<Decisionlogic> ::= <LogicTerm> ["else(" <Assignment> ")"] |
    <LogicTerm> ", "<LogicTerm> ["else(" <Assignment> ")"]
<LogicTerm> ::= "If(" <Expression> ")then(" <Assignment> ")"
<Assignment> ::= <Variable> "=" <Value>
<Expression> ::= <Variable> [<MathematicalOperator> <Constant>]
    <Comparison> <Value> | <Expression> <LogicOperator> <Expression>
<Comparison> ::= "=" | "!=" | "<=" | ">=" | "<" | ">"
<MathematicalOperator> ::= "*" | "/" | "+" | "-"
<LogicOperator> ::= "&&" | "||"
<Constant> ::= --> "DoubleNumber"
<Interval> ::= -->"IntegerNumber" "msec" | "sec" | "min" | "hours" | "days"
<Variable> ::= -->"StringIdentifier in UTF-8"
<Value> ::= -->"StringIdentifier in UTF-8"
    
```

Fig. 3. EBNF Grammar for the Decision Logic (DL)

Fig. 4 shows the sensor model for the example from section 3. At the bottom line, the sensors are represented. For the context *location*, which is represented in the dotted rectangle above, the physical sensors *WiFi*, *GPS* and the virtual sensor *Location-DB* are needed. The GPS sensor returns an object consisting of a long- and latitude double, to compute the position of the salesperson. In the center of the figure, the context elements are represented. The *DL* areas show when an *appointment* is a *meeting* or *training*, respectively when the *location* is *at work*, *at home*, *at the customer* or *unknown*. For example if the calendar entry is today and the entry description contains the name *training*, the salesperson should be part of the training. At the top of the figure are the context expressions related to the context elements. They are also the connection to the related context-aware business process. This example shows that it is possible to model the context evaluation for mobile context-aware business processes with SenSoMod. It enables to model the aggregation of information from the basic sensor information to the context expressions.

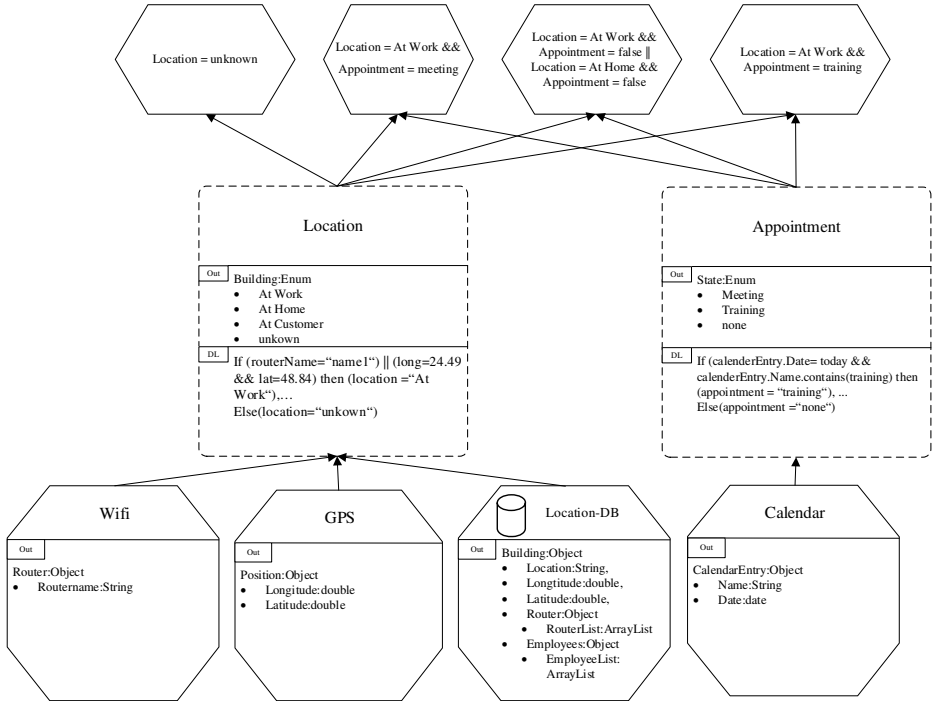


Fig. 4. Sensor model for the contexts *location* and *appointment*

3.3 Architecture of the Mobile Context-Aware Application

To implement a mobile application an architecture for context-aware applications has to be designed (RQ.2). Unfortunately, the standard architectures of the operating systems for mobile applications do not support context-awareness [Go17], [Ap17]. Therefore, to implement the application, an architecture considering context has to be created. The context recognition allows adapting the application to the context, which means that certain parts of the application have to be activated, whereas other parts can be deactivated to reduce the information overload for the user. The result is that logical components have to be encapsulated. Fig. 5 shows the architecture of the supportive mobile context-aware application. The application is divided into a server and a client part. The server contains the normal logic and data layer to exchange data with the client applications and handle the data storage. In some cases, the server also has a graphical user interface (GUI) layer, but this does not matter for the architecture of mobile context-aware devices. In a separated part the evaluation of the context and the storage of the context relevant data is encapsulated. This is necessary because not every context can be evaluated on the mobile device. For instance, the context *weather* will be evaluated by sensors in a weather station. The evaluated context on the server side will be sent to the mobile context-aware application. The client is usually separated into a 3-layer architecture consisting of a user-

interface (UI), logic, and data-layer. To realize the adaptive components, the layers are divided vertically. The vertical parts are *context components* which can be activated or deactivated depending on the evaluated context. The *context components* also have a UI-, logic- and data-layer dedicated to their specific task, but they have to be independent of the other context components to achieve activation or deactivation.

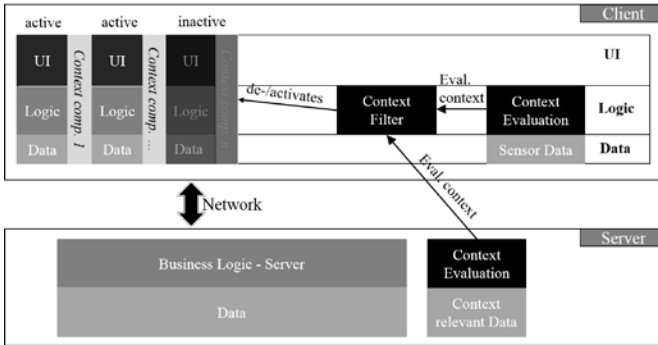


Fig. 5. Schematics of the architecture of the mobile context-aware application

Context, which can be evaluated on the mobile devices, is encapsulated in the *context evaluation* component which also has its own sensor data. A typical example for this is *location*. It can be evaluated via the GPS or WiFi sensor. The decision taken when a certain context is recognized can be seen in the sensor model in the *DL* areas. Furthermore, the model supports the programmer by showing how many different context objects exist and what the return values of the context objects are. The evaluated context from the mobile device and the server will be sent to the *context filter* component. It decides to activate or deactivate a certain *context component* depending on the evaluated context.

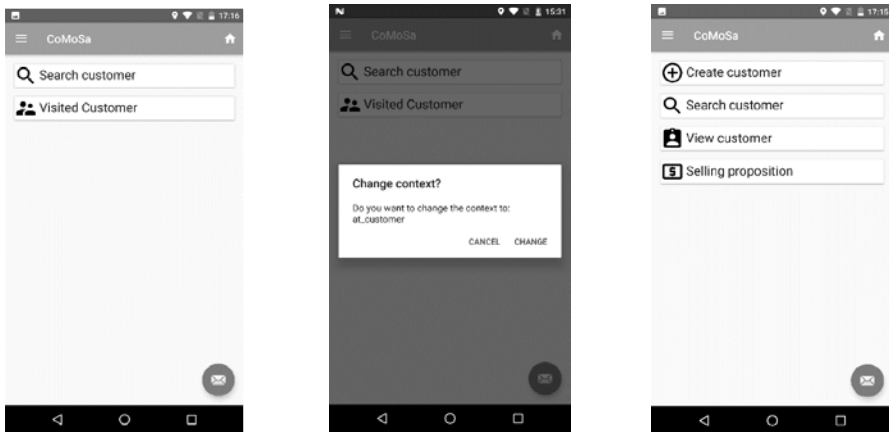
4 Implementation of a Prototype

The designed mobile context-aware business processes and its sensor model eases the implementation of a supportive mobile application. The sensor model can be used as a blueprint for the sensor and context classes which will be needed to evaluate the context of the user. Furthermore, the return types of the sensor and context objects, as well as the decision logic, can be used in the source code of the context evaluation. The context-aware business process can be used to derive the scope of the application and which parts of the application are needed under which condition.

Proof of Concept

Based on the modeled mobile context-aware business process, the sensor model and the architecture of the application were developed. In Fig. 6 three screenshots are depicted demonstrating how the mobile application reacts to the context recognition. Screenshot a) shows the home screen when the context *at work* was recognized. It supports the user with

his tasks to *prepare a customer visit* or *finish customer related work*. The first task will be supported by researching the customer to get relevant data, like address, contact or sell history. By showing the visited customers the user can finish related work like editing customer details, confirm discounts or set shipping dates. The next screenshot shows a pop-up when a new context was recognized. The mobile application does not force the adaption of the UI, instead the pop-up lets the user decide to adapt the UI. If the user declines, the mobile application remains unchanged, otherwise the UI will be changed. The home screen when the context *at customer* was recognized is depicted in screenshot c). In contrast to the screenshot a) the component *visited customer* is deactivated and the components *create customer*, *view customer* and *selling proposition* are activated. The user will be supported to sell the company's products by the component *selling proposition*. By clicking on it, a submenu will be shown and depending on the availability of the *weather data* and *customer history* it then recommends products.



The mobile context-aware application shows how the user can be supported in executing their tasks. Furthermore, the modeling languages help to implement these kinds of applications by clarifying how the context will be evaluated (SenSoMod) and what the user needs during the execution of the business process (Context4BPMN).

5 Outlook and Further Research

The main contribution of this article is to show how the framework helps to model mobile context-aware application to support the execution of business processes. This comprises an extension of the BPMN, a domain specific modeling language and an architecture for mobile context-aware applications. It provides the possibility for model engineers to plan such processes in a precise, detailed and comprehensive way. It also enables programmers to reuse the decision logic in the sensor model in the source code of the application.

Therefore, it can improve the interaction between modelers and programmers and accelerate the adaption of business processes. There are some tasks for further research in this area. The introduced framework need an evaluation with practitioners to get feedback from the target group and improve its usefulness. Since mobile context-aware business processes obviously need to measure context and are supported by an application on smart devices, an automated or semi-automated way to generate code from the business process would be helpful to increase the flow between modeling and implementation phase. The logical context expressions can be used to generate decisions in the application program. Furthermore, the sensor model can be utilized to pre-generate classes and interfaces. In an additional step, the use of the gathered context data from the execution of a business process will be investigated. The data could be used to identify problems in the execution and therefore be interesting for the controlling phase.

Literaturverzeichnis

- [Ap17] Apple: About the iOS Technologies.
<https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html>, 03.03.2017.
- [AS15a] Al-alshuhai, A.; Siewe, F.: An Extension of UML Activity Diagram to Model the Behaviour of Context-Aware Systems: Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015; S. 431–437.
- [AS15b] Al-alshuhai, A.; Siewe, F.: An Extension of Class Diagram to Model the Structure of Context-Aware Systems: The Sixth International Joint Conference on Advances in Engineering and Technology (AET), 2015.
- [BA11] Barrenechea, E. S.; Alencar, P. S.C.: An Adaptive Context-Aware and Event-Based Framework Design Model. In *Procedia Computer Science*, 2011, 5; S. 593–600.
- [Ba60] Backus, J. W. et al.: Report on the algorithmic language ALGOL 60. In *Communications of the ACM*, 1960, 3; S. 299–314.
- [BCD10] Berardinelli, L.; Cortellessa, V.; Di Marco, A.: Performance Modeling and Analysis of Context-Aware Mobile Software Systems. In (Rosenblum, D. S.; Taentzer, G. Hrsg.): *Fundamental approaches to software engineering*. 13th international conference, FASE 2010, held as part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20 - 28, 2010 ; proceedings. Springer, Berlin, 2010; S. 353–367.
- [Bi16] Bichler, M. et al.: Erratum to. *Theories in Business and Information Systems Engineering*. In *Business & Information Systems Engineering*, 2016.
- [BKR11] Becker, J.; Kugeler, M.; Rosemann, M. Hrsg.: *Process management. A guide for the design of business processes*. Springer, Berlin, 2011.

- [Bu12] Busse, M. et al.: Projection Bias in the Car and Housing Markets. National Bureau of Economic Research, Cambridge, MA, 2012.
- [Ch08] Choi, J.: Context. From Birth to Design.: International Conference on Advanced Language Processing and Web Information Technology, 2008; S. 347–352.
- [Co13] Conforti, R. et al.: Real-time risk monitoring in business processes. A sensor-based approach. In *Journal of Systems and Software*, 2013, 86; S. 2939–2965.
- [DAS01] Dey, A.; Abowd, G.; Salber, D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. In *Human-Computer Interaction*, 2001, 16; S. 97–166.
- [De00] Dey, A. K.: Providing Architectural Support for Building Context-aware Applications. Georgia Institute of Technology, Atlanta, GA, USA, 2000.
- [De01] Dey, A. K.: Understanding and Using Context. In *Personal and Ubiquitous Computing*, 2001, 5; S. 4–7.
- [DS16] Dörndorfer, J.; Seel, C.: The impact of mobile devices and applications on business process management. In (Barton, T. et al. Hrsg.): *Prozesse, Technologie, Anwendungen, Systeme und Management 2016. Angewandte Forschung in der Wirtschaftsinformatik*, 2016; S. 10–19.
- [DS17] Dörndorfer, J.; Seel, C.: A Meta Model Based Extension of BPMN 2.0 for Mobile Context Sensitive Business Processes and Applications. In (Leimeister, J. M.; Brenner, W. Hrsg.): *Proceedings der 13. Internationalen Tagung Wirtschaftsinformatik (WI)*, St. Gallen, 2017; S. 301–315.
- [FL14] Falk, T.; Leist, S.: Effects of mobile solutions for improving business processes. In (Avital, M.; Leimeister, J. M.; Schultze, U. Hrsg.): *ECIS 2014 proceedings. 22th European Conference on Information Systems ; Tel Aviv, Israel, June 9-11, 2014*, AIS Electronic Library, 2014.
- [Go17] Google: Android Platform Architecture. <https://developer.android.com/guide/platform/index.html>, 03.03.2017.
- [HC93] Hammer, M.; Champy, J.: Reengineering the corporation. A manifesto for business revolution. Harper Business, New York, NY, 1993.
- [He04] Hevner, A. R. et al.: Design Science in Information Systems Research. In *MIS Q*, 2004, 28; S. 75–105.
- [He05] Henriksen, K. et al.: Middleware for Distributed Context-Aware Systems. In (Hutchison, D. et al. Hrsg.): *On the Move to Meaningful Internet Systems 2005. CoopIS, DOA, and ODBASE*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005; S. 846–863.
- [HGB13] Hoyos, J. R.; García-Molina, J.; Botía, J. A.: A domain-specific language for context modeling in context-aware systems. In *Journal of Systems and Software*, 2013, 86; S. 2890–2905.
- [HL15] Heinrich, B.; Lewerenz, L.: A Novel Concept for the Usage of Mobile Information Services. In (Linnhoff-Popien, C.; Zaddach, M.; Grahl, A. Hrsg.): *Marktplätze im Umbruch. Digitale Strategien für Services im mobilen Internet*. Springer Vieweg, Berlin, 2015; S. 319–329.

- [HS15] Heinrich, B.; Schön, D.: Automated Planning of Context-aware Process Models. University of Münster, Münster, Germany, 2015.
- [In13a] International organization for standardization (iso): Information technology. Object Management Group Business Process Model and Notation, 2013.
- [In13b] Intel IT Center: Mobile Computing Trends: Insight into Today's Workforce, 2013.
- [In14] Internet Engineering Task Force: The JavaScript Object Notation (JSON) Data Interchange Format, 2014.
- [JKR01] Jang, S.-I.; Kim, J.-H.; Ramakrishna, R. S.: Framework for Building Mobile Context-Aware Applications. In (Kim, W. et al. Hrsg.): The Human Society and the Internet Internet-Related Socio-Economic Issues. First International Conference, Human.SocietyInternet 2001 Seoul, Korea, July 4-6, 2001 Proceedings. Springer, Berlin, Heidelberg, 2001; S. 139–150.
- [KCO10] Kramer, D.; Clark, T.; Oussena, S.: MobDSL. A Domain Specific Language for multiple mobile platform deployment: 2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications. IEEE, 2010; S. 1–7.
- [KK14] Kerr, D.; Koch, C.: A Creative and Useful Tension? Large Companies Using "Bring Your Own Device". In (Bergvall-Kåreborn, B.; Nielsen, P. A. Hrsg.): Creating Value for All Through IT. IFIP WG 8.6 International Conference on Transfer and Diffusion of IT, TDIT 2014, Aalborg, Denmark, June 2-4, 2014. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, s.l., 2014; S. 166–178.
- [KKC11] Kim, S.; Kim, E.; Choi, Y.: Composite context information design and model approach for adaptive service decision: 13th International Conference on Advanced Communication Technology (ICACT), 2011. 13 - 16 Feb. 2011, Phoenix Park, Gangwon-Do, Republic of Korea ; proceeding. IEEE, Piscataway, NJ, 2011; S. 1593–1598.
- [Mo14] Morabito, V. Hrsg.: Trends and challenges in digital business innovation. Springer, Cham, 2014.
- [Mu10] Murray, K. B. et al.: The effect of weather on consumer spending. In Journal of Retailing and Consumer Services, 2010, 17; S. 512–520.
- [Ob11] Object Management Group (OMG): Business Process Model and Notation (BPMN), Version 2.0, 2011.
- [Pr16] Prümper, J. et al.: Abschlussbericht der Studie: „Mobiles Arbeiten“. spring Messe Management GmbH, Frankfurt am Main, 2016.
- [Rh13] Rhee, K. et al.: High-Level Design for a Secure Mobile Device Management System. In (Marinos, L.; Askoxylakis, I. Hrsg.): Human aspects of information security, privacy, and trust. First international conference, HAS 2013, held as part of HCI International 2013, Las Vegas, NV, USA, July 21 - 26, 2013 ; proceedings. Springer, Berlin, 2013; S. 348–356.
- [RRF08] Rosemann, M.; Recker, J. C.; Flender, C.: Contextualisation of business processes. In International Journal of Business Process Integration and Management, 2008, 3 (1); S. 47–60.

- [RRF11] Rosemann, M.; Recker, J.; Flender, C.: Designing context-aware Business Processes. In (Siau, K.; Chiang, R.; Hardgrave, B. C. Hrsg.): Systems analysis and design. People, processes and projects. M.E. Sharpe, Armonk, NY u.a, 2011; S. 51–73.
- [Rv07] Rosemann, M.; van der Aalst, W.M.P.: A configurable reference modelling language. In Information Systems, 2007, 32; S. 1–23.
- [SAW94] Schilit, B.; Adams, N.; Want, R.: Context-Aware Computing Applications: First Workshop on Mobile Computing Systems and Applications (WMCSA), 1994; S. 85–90.
- [SB05] Sheng, Q. Z.; Benatallah, B.: ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services Development: International Conference on Mobile Business (ICMB), 2005; S. 206–212.
- [SBG99] Schmidt, A.; Beigl, M.; Gellersen, H.-W.: There is more to context than location. In Computers & Graphics, 1999, 23; S. 893–901.
- [Sc00] Scheer, A.-W.: ARIS - Business Process Modeling. Springer, Berlin u.a., 2000.
- [Sh12] Shrestha, A. et al.: A Framework for Building and Operating Context-Aware Mobile Applications. In (Venkatasubramanian, N. Hrsg.): Mobile wireless middleware, operating systems, and applications. 4th international ICST conference, Mobilware 2011, London, UK, June 22 - 24, 2011 ; revised selected papers. Springer, Heidelberg, 2012; S. 135–142.
- [SN07] Saidani, O.; Nurcan, S.: Towards Context Aware Business Process Modelling: Workshop on Business Process Modelling, Development, and Support, Norway, 2007; S. 1.
- [So05] Soffer, P.: On the Notion of Flexibility in Business Processes: Proceedings of the CAiSE'05 Workshops, 2005; S. 35–42.
- [VL12] Verclas, S.; Linnhoff-Popien, C. Hrsg.: Smart Mobile Apps. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [VR10] Vom Brocke, J.; Rosemann, M. Hrsg.: Handbook on Business Process Management 2. Strategic Alignment, Governance, People and Culture. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [We91] Weiser, M.: The Computer for the 21st Century. In Scientific American, 1991, 265; S. 94–104.

Model-driven Development of Virtual Network Embedding Algorithms with Model Transformation and Linear Optimization Techniques

Stefan Tomaszek¹ Erhan Leblebici¹ Lin Wang² Andy Schürr¹

Abstract: Enhancing the scalability and utilization of data centers, virtualization is a promising technology to manage, develop and operate network functions in a flexible way. For the placement of virtual networks in the data center, many approaches and algorithms are discussed in the literature to optimize solving the so-called virtual network embedding problem with respect to various optimization goals. This paper presents a new approach for the model-driven specification, simulation-based evaluation, and implementation of possible mapping algorithms that respect a set of given constraints and using linear optimization solving techniques to select one almost optimal mapping. Rule-based model transformation techniques are used to translate a given mapping problem into a linear optimization problem by taking domain specific knowledge into account. The resulting framework thus supports the design and evaluation of (correct-by-construction) virtual network embedding algorithms on a high level of abstraction. Well-defined model transformation rule refinement strategies can be used to reduce the search space for the employed linear optimization techniques.

Keywords: Model-driven development; virtual network embedding; triple graph grammar; integer linear programming; data center

1 Introduction and Motivation

With the rapid evolvement of the Internet, online services such as social networking, e-commerce, and online gaming have become ubiquitous. These online services are constantly generating a huge amount of data that is managed and analyzed by service providers like Google, Facebook or Amazon. To this end, cloud computing has become the norm as it can provide the required availability, scalability, and cost-effectiveness and can support rapid development and operation cycles. Data centers (DCs) are major facilities for cloud computing and usually host a large number of computing or storage servers interconnected by a dedicated communication network. To operate these very large and complex environments,

¹ Technische Universität Darmstadt, Real-Time Systems Lab, Merckstr. 25, 64283 Darmstadt, Germany, {stefan.tomaszek,erhan.leblebici,andy.schuerr}@es.tu-darmstadt.de

² Technische Universität Darmstadt, Telecooperation Lab, Hochschulstraße 10, 64289 Darmstadt, Germany, wang@tk.tu-darmstadt.de

Acknowledgment

This work has been funded by the German Research Foundation (DFG) as part of project A1 within the Collaborative Research Center (CRC) 1053 – MAKI.

virtualization has become a key technology, decoupling the underlying infrastructure and the upper-layer application and increasing the management flexibility so that economy-of-scale can be easily achieved. As services are encapsulated in virtual machines (VMs) and are interconnected with virtual networks (VNs), cloud operators can consolidate multiple VMs on the same physical machine in the substrate network (SN), migrate the VM at runtime, and span VNs regardless of the underlying network cabling details. This flexibility makes it possible to enact a fast development process, to unify the configuration, and to reduce the energy consumption of the DC, which is a significant cost factor for the cloud operators.

However, the virtualization and thus the unification of the configuration is accompanied by a high complexity, which manifests itself especially in the virtual network embedding (VNE) problem. The VNE problem is defined as the embedding of VNs in the SN with various constraints respected and with multiple metrics optimized on both the computing nodes and the network. When considering modern frameworks like OpenStack [SAE12] for the VNE problem, administrators often perform these embeddings manually.

In recent years, research into automating VNE has been greatly intensified. A variety of algorithms and methodologies has been developed to improve the distribution of virtual servers and networks within DCs. These algorithms and methodologies depend on specific optimization objectives such as higher resource utilization or lower energy consumption and specific structures of the underlying infrastructure [Gu10]. Performing a customized embedding algorithm is generally an NP-hard optimization problem with a substantial search space [Fi13]. Therefore, many different approaches and methods have been proposed to reduce the search space with customized embedding algorithms and optimization heuristics. Unfortunately, most of these algorithms are difficult to expand and adapt to different environments and constraints because they are highly tailored for specific infrastructures, frameworks, or application scenarios.

A typical development cycle for VNE is shown in Figure 1. Taken as a rule, the development of a new dedicated VNE algorithm starts with the informal documentation or formal specification of a set of requirements and actions. According to this specification, a prototype is implemented and evaluated in a simulation framework. Only when the simulation has been successful, the algorithm is integrated and tested in a realistic testbed before it goes into production. In the classical way of developing new embedding algorithms, the specification is often manually encoded and often manually integrated into a simulation environment, which brings time-consuming, error-prone tasks.

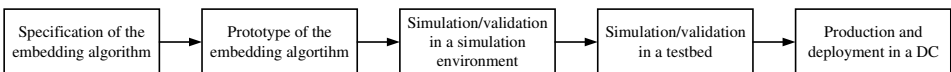


Fig. 1: Typical development cycle for VNE.

In this paper, we target this issue and propose a new methodology for model-driven virtual network embedding (MdVNE). In the model-driven development, executable code for different platforms can be automatically generated, an integrated simulation framework can fasten the prototyping of new algorithms by using the generated code and the correct implementation of specifications can be ensured. The MdVNE combines model transformation in the form of so-called triple graph grammars (TGGs) and integer linear programming (ILP) techniques with an optimization goal with linear equalities and inequalities which represent constraints in this paper. In the first step model transformation techniques are used to reduce the search space by pattern matching methods that generate families of possible mappings of VNs to SN elements. These mappings respect a set of given constraints handling rather structural or attribute conditions described via single graph patterns. In the second step, the ILP solving techniques take further decisions among the matched mapping candidates by selecting an optimal mapping with respect to a given set of constraints and optimization goals. On the contrary to the TGG constraints, the ILP constraints with a global scope go beyond single graph patterns and describe rather mathematical constraints over available resources.

Compared to existing solutions, the benefits of the proposed MdVNE approach include

- Developers specify embedding algorithms on a rather abstract level using a combination of first order logic constraints, inequalities, and model transformation rules.
- Prototypes of implementations are then generated from the high-level specification, leveraging state-of-the-art incremental pattern matching, model transformation, and ILP solving technologies.
- The generated low-level implementation respects the high-level specifications of embedding constraints and optimization goals by construction.
- The selected implementation techniques simplify the development of incremental reconfiguration of embeddings even including scenarios where embedding constraints and optimization goals are modified at runtime.
- The offered algorithm development and simulation framework supports the design of rather different categories of embedding and optimizing algorithms by combination and weighting of purely ILP-based and model transformation based approaches.

The remainder of this paper is organized as follows. After introducing the related work in Section 2 a running example is introduced in Section 3. The new mapping approach MdVNE is presented in detail in Section 4, followed by the evaluation in Section 5. Finally, the paper is concluded in Section 6.

2 Related Work

Virtualization of DC networks has been widely explored and a survey can be found in [Ba13]. As a result, many different algorithms [Gu10], [Ze15], [Xi12], [Zh13] for VNE have been proposed to maximize the resource utilization or minimize the cost for DCs. As the embedding problem is actually a case of the multi-way separator problem, it is NP-hard

and, therefore, not scalable without reducing the space size by heuristics or meta-heuristics [Fi13]. Guo et al. propose SecondNet [Gu10], which introduces a heuristic approach to map a subset of virtualized data centers (VDCs) to a tree-based DC. However, the authors only consider constraints on bandwidth and the number of virtual machines per physical server for reduced complexity. Zeng et al. [Ze15] consider the DC architecture and the traffic between virtual machines to minimize the overall communication costs between virtual servers and employs the commercial ILP solver Gurobi [Gu16] to solve the optimization problem. In addition, Xie et al. [Xi12] incorporate the time dimension into the VNE process and Zhani et al. [Zh13] include dynamic migration to adapt embedding decisions over time. The major advantage of MdVNE over the above-mentioned algorithms is that different architectures, constraints for resources, demands, and various optimization goals can be integrated and embedding decisions can be smoothly adapted in accordance to constraint changes on the fly.

In other network areas such as software-defined networks (SDNs) or wireless networks, the model-driven development is already used with promising results in order to increase the abstraction level, to create applications and algorithms independently of existing technologies and to verify them during development. In the SDN area, which makes the control and forwarding level independent of the physical network and can be a part of virtualization in DCs, Lopes et al. [Lo16] describe a model-driven approach to develop, verify and generate application-, controller- and network-independent code for SDN applications. In the area of wireless networks, Kluge et al. [Kl17] describe a model-driven approach to develop topology control algorithms with graph transformations while ensuring compliance with user-defined constraints and consistencies. However, none of the proposals can support both server- and network-end constraints simultaneously and thus are not directly suitable for VNE in complex DC environments.

3 Running Example

A typical scenario for DC operators consists of requests from customers for a customized VN infrastructure with switches, servers or network functions like firewalls. One typical VN request is a virtual cluster in which servers are connected to one central switch for creating a network environment [Ba11]. Such a topology is common in many enterprise scenarios e.g. to build a web application with clustered web servers. These network topologies may have different properties, resources, and constraints to be integrated and mapped by the DC that entails a high degree of configuration diversity.

In the following sections of the paper, a simplified example for a virtual cluster and DC is used to introduce the new mapping approach called MdVNE. The used scenario (Figure 2) and the metamodel for modeling the environment and generating the network instances (Figure 3) are now described in detail. Starting with Figure 2 (a), a snapshot of a DC with a queue of VNs, which should be mapped to the DC, is shown. Let us assume that some VNs are already mapped and all available positions (*slots*) to embed are occupied except

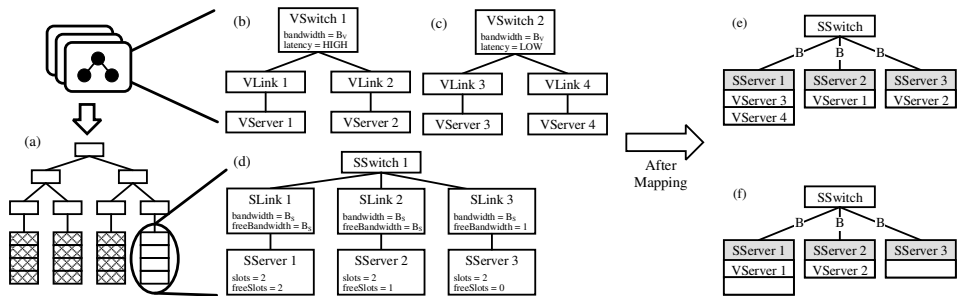


Fig. 2: Running example: (a) DC with multiple VNs in the mapping queue; (b) and (c) are examples for a virtual cluster; (d) is a subtree of the DC; (e) and (f) are two exemplary mapping solutions.

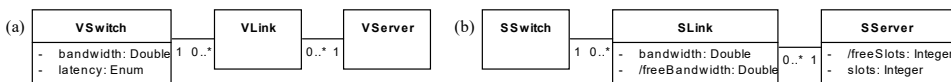


Fig. 3: Metamodels for (a) the VN and (b) the DC/SN.

the marked subtree. In the following, we only focus on the marked subtree from Figure 2 (d), which is called SN in the following. The mapping queue contains the VNs as shown in Figure 2 (b) and (c). Every VN has a central switch (*VSwitch*) with two links (*VLink*) each connected to a server (*VServer*). The bandwidth for these links is denoted as B_V (*VSwitch.bandwidth*) and the global VN has the service level agreement (SLA) that the latency must be *HIGH* or *LOW* (*VSwitch.latency*). A *LOW* latency means that the whole VN must be mapped to one substrate server in order to minimize the traffic delay whereas a *HIGH* latency has no restrictions. The SN is similar except that the bandwidth is defined for every *SLink* (*SLink.bandwidth*), every *SServer* owns a number of *slots*, each slot being able to host a virtual server and the bandwidth for the server internal traffic is assumed as unlimited. We further assume that every link in the SN has the same bandwidth B_S .

After defining the networks, the mappings of the VN to the SN are specified. These mapping constraints must be strictly adhered to at all times because they represent e.g. technical conditions or SLAs with the customers. In this paper, the following constraints are defined:

- (1) Every virtual switch must be mapped to one substrate switch or server.
- (2) Every virtual server must be mapped to one substrate server.
- (3) Every virtual link must be mapped to a substrate server or to one substrate link.
- (4) Virtual networks with latency *LOW* must completely be mapped to one substrate server.
- (5) The sum of all bandwidths of virtual links mapped to a substrate link must not exceed the available bandwidth of the substrate link.
- (6) The sum of all virtual servers mapped to a substrate server must not exceed the available number of slots of the substrate server.

In Figure 2, two exemplary results (e) and (f) after the mapping process of both VNs to the SN are shown. In (e), the VN (b) with *HIGH latency* is mapped to *SServer 2* and *3*, and the VN (c) to *SServer 1*. Result (f) presents that VN (c) with *latency LOW* must be rejected because it cannot be mapped to one server after VN (b) is mapped to *SServer 1* and *2*.

While our set of constraints reflects a subset of requirements from real-world scenarios, further types of resources include CPU or storage capacity. Further constraint types include quality of service regarding response times and security levels reducing allowed mappings to certain subtrees in DC.

4 Mapping Approach

A typical workflow for a mapping process is presented in Figure 4, which consists of three phases: a preparation, mapping and deployment phase. In the preparation phase customers define the VN requests with their network functions, demands, SLAs, or change already existing virtual networks e.g. bandwidth. Furthermore, changes of the DC can be executed (add, remove, change server, switches,...) or the migration and shutting down of virtual networks. After that, the mapping phase is started in which the new mappings for the VN request are planned and activated/deployed in the deployment phase. This paper only focuses on the mapping phase while other phases rather concern technical details of communication networks.

Having defined and exemplified VNs and SN separately, our next goal is an explicit modeling of their mapping relationships. TGGs [Sc95] meet this requirement to specify the mappings between two graph-like structures via graph transformation rules. The combination of TGGs and ILP [LAS17] can be used to generate families of possible mapping candidates between two graphs that respect a set of given structural constraints and transfer this search space to an ILP solver for solving the optimization problem. The ILP solving techniques have been used to solve the resulting optimization problem (e.g., minimizing energy consumption) expressed as linear inequalities. The advantages over a classical VNE algorithm like [Ba11] or [Ze15] is that different constraints for resources, demands or optimization goals can be combined and easily added or extended so that a very wide range of applications are supported. Because TGG offer support for incremental updates of models, it is possible to adapt the incremental methods for this embedding approach in order to be able to efficiently deal with the highly dynamic system.

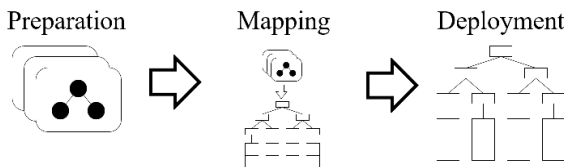


Fig. 4: Workflow for embedding of VN request.

4.1 Triple Graph Grammar

TGG is an approach to specify graph grammars, whose languages are sets of graph triples. Each graph triple consists of a so-called source and target graph plus a correspondence graph with traceability information between source and target. Given such a specification, the source graph contains the VNs, the target graph the SN and the correspondence graph the mapping relationships. For generating the mapping candidates between the VNs and the SN two steps are needed: (i) Creation of the SN and (ii) creation of the VNs with their (mapping) correspondences between the VNs and the SN. In the following, we assume that the SN is already created by the TGGs so we describe part (ii) now in detail.

In Figure 5, the TGG rules for generating the virtual networks and the mapping candidates are shown. Black elements are required context elements for executing the rule, whereas green elements marked by ++ are created by the rule. The naming convention for the elements and the correspondences are as follows: The name indicates the type e.g. Sw is an element of type *Switch* and $SwSw$ is the correspondence between a virtual and a substrate *Switch*. The subscript letter V or S indicates if the element is part of the virtual or the substrate network, the letters e.g. a represent an index of the specific type. Two types of tuples are defined, one for the links and one for the correspondences. The first type of the tuples are for links and the letters a and b represent the source and target node e.g. for $L_{V(a,b)}$ Sw_{Va} is the source and Sr_{Vb} is the target element (Figure 5 d). The second type is for the correspondences and the first part represents virtual and the second the substrate element e.g. for $LSr_{(a,b),c}$ $L_{(a,b)}$ is in the VN and Sr_c in the SN.

Rule (a) creates a new virtual switch (Sw_{Va}) and mapping candidate ($SwSw_{(a,b)}$) to an existing substrate switch (Sw_{Sb}). Rule (b) and (c) are similar to rule (a) except that in rule (c) an attribute constraint $Sr_{Sb}.freeSlots \geq 1$ is added which means that this rule only matches if the attribute constraint is fulfilled. *FreeSlots* represent, similar to *freeBandwidth*, the number of slots that are not occupied by an active mapping before the mapping phase has

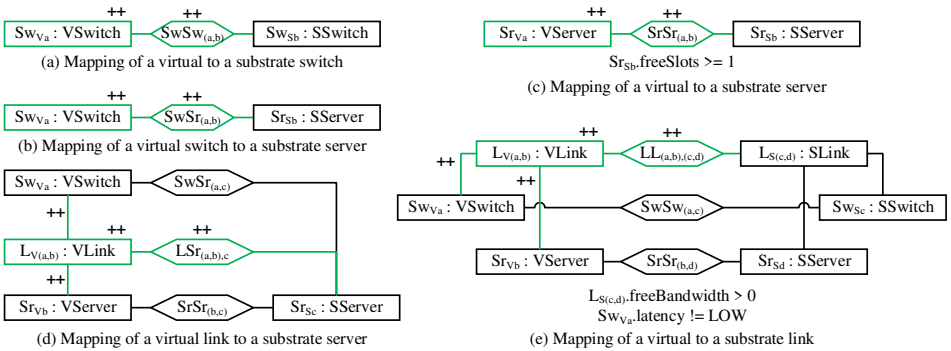


Fig. 5: TGG rules for creating the mapping of the VN to the SN.

started. Rule (d) creates a virtual link ($L_{V(a,b)}$) with a mapping candidate ($LSr_{(a,b,c)}$) to an existing substrate server (Sr_{Sc}), if the mappings $SwSr_{(a,c)}$ and $SrSr_{(b,c)}$ already exist. The last rule (e) creates a mapping of a virtual link ($L_{V(a,b)}$) to a substrate link ($LS_{(c,d)}$) if the mappings $SwSw_{(a,c)}$ and $SrSr_{(b,d)}$ exist, $freeBandwidth$ of $LS_{(c,d)}$ is greater than 0 and the latency of the switch Sw_{Va} is not LOW . This implicates that if Sw_{Va} has *latency LOW* only rule (d) can produce a link mapping. After the TGG rules are explained in detail, the connection of the constraints from section 3 to the TGG rules are summarized in Table 1.

Constraint	TGG rule	Annotation
(1)	(a) and (b)	A virtual switch can be mapped to a substrate server or switch.
(2)	(c)	A substrate server must have a free slot to map a virtual server.
(3)	(d) and (e)	A virtual link can be mapped to a substrate server or link.
(4)	(d) and (e)	Rule (d) must be executed to map a virtual link because rule (e) cannot be executed if <i>latency = LOW</i> .
(5), (6)	-	Is not represented by TGG rules

Tab. 1: Representation of the constraints from section 3 by the TGG rules from Figure 5.

To generate all mapping candidates, the TGG rules are executed on the example instance from Figure 2. An example of these mapping candidates of VN (c) from Figure 2 is shown in Figure 6, which shows a subset of all created elements and neglects $VLink\ 2$ and $VServer\ 2$ for brevity. During the creation of all correspondences e.g. $SwSr_{(1,1)}$ further constraints are internally created as integer (in-)equalities. These constraints are described and listed in the next subsection.

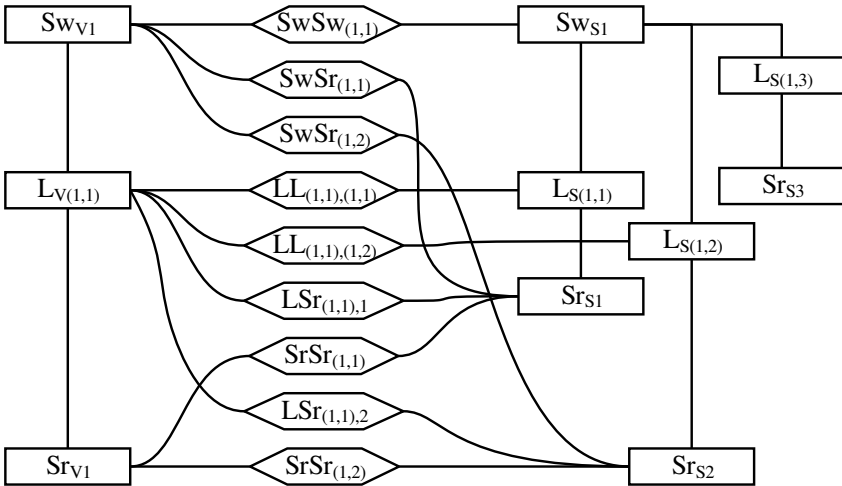


Fig. 6: All possible mappings after executing the TGG rules for the VN (c) from Figure 2. The $VLink\ 2$ and $VServer\ 2$ are neglected to avoid diagram clutter.

4.2 Linear inequalities

The execution of the TGG rules generates all potential candidates of mappings and integer variables that match the required graph structures and attribute constraints as specified in the rules. Additionally, ILP constraints are generated to ensure that each virtual network element is mapped to one and only one substrate element. In the next step, the linear optimization problem is solved by establishing linear inequalities for all constraints and passing them to the ILP solver. These linear inequalities specify which mappings will be activated and which are discarded e.g. if $SwSw_{(1,1)}$ is activated, $SwSr_{(1,1)}$ is discarded as they map the same virtual element and are thus mutually exclusive. To show the relationship between the mapping candidates and their integer variables in the inequalities, the name of the mapping candidate variables are retained with lowercase letters e.g. $swsw_{(1,1)}$ for their integer variables with the value 0 or 1. Other necessary parameters for establishing the linear inequalities are given in Table 2.

Variables for virtual network		Variables for substrate network	
M_V	Number of all virtual switches	M_S	Number of all substrate switches
N_V	Number of all virtual servers	N_S	Number of all substrate servers
K_V	Number of all virtual links	K_S	Number of all substrate links

Tab. 2: Different parameters for the linear inequalities.

Some of the advantages of this approach are the reduction of the search space by using graph grammars, and the inequalities automatically derived from the TGG rules if executed, which prevent a virtual element from being mapped several times, or that the dependencies between different mappings, e.g. $lsr_{(1,1),2}$ and $swsr_{(1,2)}$ are taken into account.

In the following, the constraints and their linear inequalities are described in detail and shown in a compact form in Table 3.

Constraint (1) is represented by TGG rule (a) and (b) (Table 1) and after their execution, the generated inequalities are presented in Table 3. The first line is generated by rule (a) considering that after the generation of all possible mappings between one virtual switch and all substrate switches, a maximum of one mapping can be selected. Therefore, the sum of all integer variables $swsw_{(i,j)}$ must be smaller or equal to 1 e.g. $swsw_{(1,1)} \leq 1$. The inequalities generated by rule (b) are very similar to rule (a) with the only difference that instead of a substrate switch a substrate server is used which leads to the following exemplary inequality e.g. $swsr_{(1,1)} + swsr_{(1,2)} \leq 1$.

Constraint (2) is represented by TGG rule (c) which leads to similar inequalities like rule (a) and (b) e.g. $srsr_{(1,1)} + srsr_{(1,2)} \leq 1$. The additional attribute condition $SrSb.freeSlots \geq 1$ is not included in the linear inequalities because the pattern matching checks this condition before executing the rule and creating a potential mapping. Assuming that this constraint is not encoded in the TGG rules, this attribute condition would be manually encoded and added to the inequalities.

Constraint	Inequalities	TGG rules
(1)	$\forall i, 1 \leq i \leq M_V \sum_{j=1}^{M_S} swsw(i,j) \leq 1$	(a)
	$\forall i, 1 \leq i \leq M_V \sum_{j=1}^{N_S} swsr(i,j) \leq 1$	(b)
(2)	$\forall i, 1 \leq i \leq N_V \sum_{j=1}^{N_S} sr sr(i,j) \leq 1$	
(3)	$\forall i, j, 1 \leq i \leq M_V, 1 \leq j \leq N_V \sum_{p=1}^{N_S} lsr(i,j),p \leq 1;$ $\forall i, j, p, 1 \leq i \leq M_V, 1 \leq j \leq N_V, 1 \leq p \leq N_S $ $lsr(i,j),p \leq swsr(i,p), lsr(i,j),p \leq sr sr(j,p)$	(d)
	$\forall i, j, 1 \leq i \leq M_V, 1 \leq j \leq N_V \sum_{p=1}^{M_S} \sum_{q=1}^{N_S} ll(i,j),(p,q) \leq 1,$ $ll(i,j),(p,q) \leq swsw(i,p), ll(i,j),(p,q) \leq sr sr(j,q)$	(e)
(4)	No additional inequalities are needed.	-
(5)	$\forall i, j, 1 \leq i \leq M_S, 1 \leq j \leq N_S L_{S(i,j)}.bandwidth -$ $\sum_{p=1}^{M_V} \sum_{q=1}^{N_V} ll(i,j),(p,q) * Sw_{Vp}.bandwidth \geq 0$	-
(6)	$\forall i, 1 \leq i \leq K_S Sr_{S_i}.slots - \sum_{N_V}^{j=1} sr sr(i,j) \geq 0$	-

Tab. 3: Representing linear inequalities for the constraints (section 3) and the TGG rules (Figure 5).

Constraint (3) is represented by TGG rule (d) to map a virtual link to a substrate server and rule (e) to map it to a substrate link. Compared to the previous constraints these rules have implications to express that the context elements and mappings are already selected. In rule (d) the implication is that a virtual switch and virtual server are already mapped to the same substrate server. The result are two additional inequalities e.g. $lsr_{(1,1),2} \leq swsr_{(1,2)}$, $lsr_{(1,1),2} \leq sr sr_{(1,2)}$ meaning that if $lsr_{(1,1),2}$ is selected then $swsr_{(1,2)}$ must also be chosen because the link $L_{V(1,1)}$ can only be mapped to server Sr_{S2} if the switch Sw_{V1} is already mapped to Sr_{S2} . The inequalities for rule (e) are similar except that a virtual link is mapped to a substrate link and a virtual switch to a substrate switch e.g. $ll_{(1,1),(1,1)} + ll_{(1,1),(1,2)} \leq 1$, $ll_{(1,1),(1,1)} \leq swsw_{(1,1)}$, $ll_{(1,1),(1,1)} \leq sr sr_{(1,1)}$, and $ll_{(1,1),(1,2)} \leq sr sr_{(1,2)}$.

Constraint (4) is represented by the combination of TGG rule (d) and (e) because if *latency* = *LOW* then rule (e) is not executed ($Sw_{V_a}.latency \neq LOW$) and, therefore, the virtual network must be mapped to one server (rule (d)), if possible. Consequently, no additional inequalities are needed to realize this constraint.

Constraint (5) cannot not be expressed by TGG rules because adding all mapping candidates is (actually) not expressible by eMoflon, the used to tool to specify TGGs and generate executable code. Therefore, these must be manually added to the generated inequalities. The constraint requires that the bandwidth of all mapped virtual links to a substrate link must no exceed the available bandwidth of this substrate link e.g. $L_{S(1,1)}.bandwidth - ll_{(1,1),(1,1)} * Sw_{V1}.bandwidth \geq 0$.

Constraint (6) is realized in a similar way as constraint (5) except that the sum of all virtual servers mapped to a substrate server must not exceed the available slots of the substrate server e.g. $Sr_{S1}.slots - sr sr_{(1,1)} \geq 0$.

After the generation of all linear inequalities, these inequalities are handed over to Gurobi, the selected ILP solver, with the optimization goal to maximize the number of mapped virtual elements. Depending on the application or scenario, the optimization goal can also be modified and, e.g. be designed to minimize energy consumption.

Comparing the search space for this running example for a brute-force method and the TGG approach to generate all inequalities shows that the number of integer variables and inequalities could be reduced significantly. Generating all mapping candidates without checking additional attribute constraints result in more integer variables, e.g. $SrSr_{(1,3)}$ cannot exist because Sr_{S3} has no free slots, which implicates that the link mapping candidate $LSr_{(1,1),3}$ and $LL_{(1,1),(1,3)}$ can also not exist. The same holds for $SrSr_{(2,3)}$, $LSr_{(1,2),3}$ and $LL_{(1,2),(1,3)}$. In a brute-force approach, all attribute constraints *freeSlots* and *freeBandwidth* must additionally be encoded as inequalities while the pattern matcher did already check these constraints during the executing of the TGG rules. At the end, checking the *latency* during the generation of the mapping candidates reduces the number of integer variables and inequalities significantly. Looking at VN (c) from Figure 2 the TGG rule (e) is never executed and, therefore, no mapping candidates for $LL_{(a,b),(c,d)}$, no inequalities to express the implications for the source switch and target server and no attribute constraints have to be encoded as inequalities.

5 Evaluation

In this section, the presented MdVNE approach is evaluated and compared with a brute force and the Oktopus algorithm [Ba11], an established VNE algorithm for DCs, in relation to the runtime and the number of ILP variables and constraints. After introducing the simulation setup, the following three research questions are discussed:

- RQ (1):** How does the runtime, ILP constraints and variables change if the number of servers in the SN increases?
- RQ (2):** How does the runtime behave in comparison to a brute force mapping approach and the Oktopus algorithm in a specific scenario?
- RQ (3):** Does the reduction of the search space for the mapping candidates by MdVNE offer advantages with respect to the total runtime compared to a brute force mapping approach?

5.1 Simulation setup

The structure and underlying scenario for the evaluation is based on the presented running example. As shown in Figure 2, a DC with a two-tier network infrastructure is used in which virtual networks are stored in a queue and mapped one after the other to the DC. The two-tier DC infrastructure consists of two aggregation switches each connected to a varying

number of racks (marked area in Figure 2 (a)), each with a top of the rack switch (ToR switch) and 10 servers with four slots. Each server is connected to the ToR switch with a bandwidth of 10, and the ToR switches with a bandwidth of 100 to each aggregation switch. The virtual networks are implemented as virtual clusters [Ba11], which have a central switch connected to all virtual servers with a bandwidth of two. The following evaluation will vary the number of racks in the DC and the number of virtual servers. All other parameters remain constant. In order to obtain a wide range of virtual server distribution configurations, a random sequence of virtual networks with virtual servers between 2 and 10 is generated and used for all further evaluations to map one virtual network after the other to the DC. The evaluation is done on an Intel Core i7-7700HQ with 2.80 GHz with Windows 10 (version 1703) and the Java SE Development Kit 8u141.

In the following, three approaches are presented and compared with each other. The first and second approach are the presented MdVNE and a brute force ILP mapping approach. They are quite similar to each other except that in the brute force approach no attribute constraints are used further restricting the execution of the rules, e.g. $Sr_{sb} \geq freeSlots$ (Figure 5 c). These attribute constraints are encoded into the ILP problem by additional inequalities after variables and formulas have been generated for all possible mappings of virtual to substrate elements. As a last comparison, the established Oktopus algorithm [Ba11] is executed and evaluated. Because this algorithm is not based on graph transformations or the Eclipse Modeling Framework (EMF), other data structures in the background are used which makes the comparison more difficult. In addition, Oktopus uses heuristics to map the virtual networks in contrast to the MdVNE approach. A qualitative comparison of the three algorithms is nevertheless out of scope of this paper.

5.2 Results

In the following, the results in combination with the research questions are presented and discussed.

RQ (1): To answer the first research question, the MdVNE approach is used to map 40 virtual networks and increases the number of racks from 2 to 50. This corresponds to a total server count of 20 to 500. After all 40 virtual networks are mapped, an average value for the total runtime of the mapping process, the runtime of Gurobi, the ILP solver, the number of ILP variables and constraints are being calculated. The mapping process of the first network is ignored in this calculation, as many Java and EMF initializations take place and the system is not in a steady state. In order to obtain reliable results all simulations were performed three times with a maximum percentage deviation from the average value of 9%. The evaluation of the runtime for the MdVNE approach over the number of racks can be found in Figure 7. The complete mapping process seems to have a polynomial growth that can be explained by the fact that the distribution of the virtual servers to different substrate servers generates mapping candidates in a combinatorial manner. The growth of the ILP solver runtime values depends on the internal heuristics of the solver but it seems to be

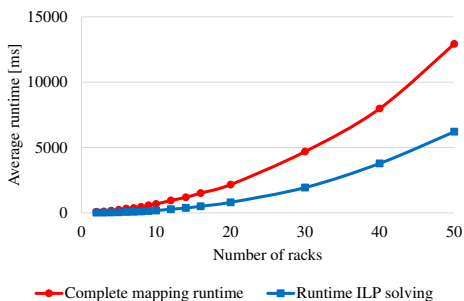


Fig. 7: MdVNE: Runtime of the complete mapping process and the ILP solver in ms over the number of racks in the SN.

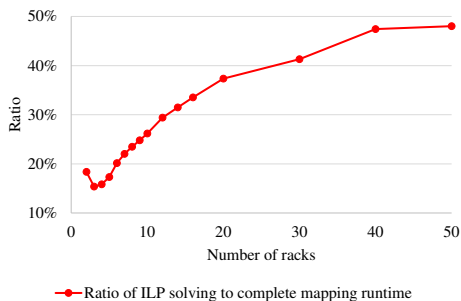


Fig. 8: MdVNE: Percentage of the ILP solver runtime of the complete mapping runtime over the number of racks in the SN.

a polynomial growth in this scenario. The influence of the ILP solver for the complete runtime of the MdVNE approach grows continuously from 15 % for three racks to 48 % at 50 racks (Figure 8) because the number of the generated ILP constraints and variables is also growing proportionally to the number of racks (Figure 9). The linear growth of both parameters can be explained by the steady increase of the elements in the model that are proportional to the number of combinatorial pairs of virtual servers as mapping candidates.

RQ (2): To answer the next two research questions the evaluation was modified to a scenario of 6 racks, which makes it possible to map the first 40 virtual networks from the queue into the DC. The result of the average runtime measurement is shown in Figure 10 with a logarithmic scaled runtime in ms over the number of mapped virtual networks e.g. 30 mapped virtual networks mean that 29 networks are already mapped.

As expected, the optimized and for this application scenario tailored Oktopus algorithm has the lowest constant runtime between 2 ms to 5 ms in this comparison. The efficient hand-tailored background data structure (in contrast to the usage of EMF models as data structures for the MdVNE and brute force algorithm) minimizes the internal Java overhead

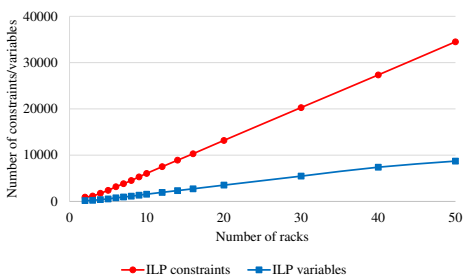


Fig. 9: MdVNE: Number of ILP integer variables and constraints over the number of racks in the SN.

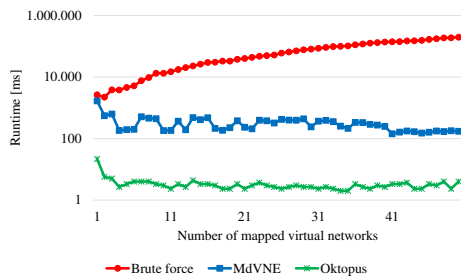


Fig. 10: Runtime of the MdVNE, brute force and Oktopus approach over the number of mapped virtual networks.

and, therefore, reduces the runtime, too. The MdVNE approach has also an almost constant runtime in a range of 140 ms to 500 ms, which is two magnitudes higher than the Oktopus algorithm e.g. 184 ms for MdVNE and 2 ms for Oktopus. This overhead is mainly caused by the usage of EMF as model framework. In both approaches, the higher values indicate that a virtual network had to be distributed on several substrate servers because all positions inside the tree have to be taken into account in a combinatorial manner. For the brute force mapping approach, a polynomial growth can be approximately assumed because for every combinatorial pair of elements in the increasing model an ILP variable is generated. This can be seen in more detail for the MdVNE and the brute force approach in Figure 11.

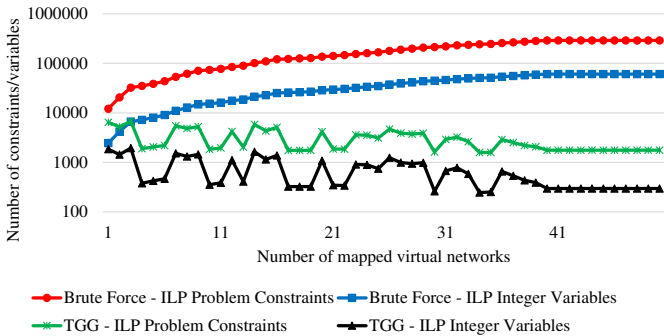


Fig. 11: ILP constraints and variables for the MdVNE and brute force approach over the number of mapped virtual networks.

RQ(3): We can see in Figure 11 that the ILP variables and constraints can be significantly reduced by using a more sophisticated ILP variable and constraint generation approach, which uses attributed graph transformations (TGGs) to filter/eliminate unfeasible mapping configurations early on in the MdVNE optimization algorithm. The result is a reduction in the runtime of the mapping process by two orders of magnitude (Figure 10). Generally, as many constraints as possible should be integrated into the TGG rules to reduce the complete mapping process.

5.3 Conclusion

In this section, we see that using the MdVNE approach it is possible to specify algorithms for the generation of ILP formulas to solve an optimization problem on a high level of abstraction (TGGs). The VNE problem to map a varying number of virtual clusters into a two-tier DC network by respecting constraints, attribute conditions and structural patterns could be realized and evaluated. The algorithm scales because of the linear runtime complexity and is thus in the same complexity class as the optimal tailored Oktopus algorithm. The overhead of the TGG execution phase is acceptable against the ILP solver phase especially when the number of servers increases in the DC. Furthermore, we see that with very little implementation effort (adapting the TGG rules) the search space of the ILP problem and,

therefore, the complete runtime of the mapping process can be significantly reduced. In the evaluation and running example only conservative structural and attribute constraints were specified in the TGG rules, but it is also possible to define more domain specific complex constraints to define sort of heuristic algorithm to reduce the search space even more and to improve the scalability.

6 Conclusion and Future Work

This paper presents a new methodology called model-driven virtual network embedding (MdVNE) combining model transformation and integer linear programming techniques to solve virtual network embedding problems. The model transformation and pattern matching techniques are used to generate families of possible mappings and reduce the search space by respecting a set of given constraints. Afterwards ILP solving techniques are used to select optimal mapping candidates. The advantage of this methodology is that the embedding algorithm can be specified on a rather abstract level and a prototypical implementation is automatically generated from this high-level specification. The development of new algorithms with this method can be fastened and, therefore, easily adjusted to other environments, applications and scenarios.

The evaluation has shown that it is possible to specify an algorithm for the VNE problem by using TGGs and an ILP solver. This algorithm scales in the range of 20 to 500 servers in a two-tier data center network with a linear runtime complexity. The reduction of the search space by the usage of pattern matching techniques reduces the runtime significantly.

To develop and evaluate different algorithms, the simulation framework and the metamodel of the DC and the VNs will be extended to support the definition of new types of constraints that take resources like e.g. CPU or demands e.g. latency into account. In addition, metrics to measure the qualitative properties of different algorithms will be added. Because of the high dynamics in this system, which requires a re-embedding or migration of existing mappings, e.g. changes in the DC or the virtual networks, incremental mapping scenarios are studied right now and will be supported in future versions of MdVNE.

References

- [Ba11] Ballani, H.; Costa, P.; Karagiannis, T.; Rowstron, A.: Towards Predictable Datacenter Networks. *ACM SIGCOMM Computer Communication Review* 41/4, pp. 242–253, 2011.
- [Ba13] Bari, M. F.; Boutaba, R.; Esteves, R.; Granville, L. Z.; Podlesny, M.; Rabhani, M. G.; Zhang, Q.; Zhani, M. F.: Data Center Network Virtualization: A Survey. *IEEE Communications Surveys & Tutorials* 15/2, pp. 909–928, 2013.

- [Fi13] Fischer, A.; Botero, J. F.; Beck, M. T.; de Meer, H.; Hesselbach, X.: Virtual Network Embedding: A Survey. *IEEE Communications Surveys & Tutorials* 15/4, pp. 1888–1906, 2013.
- [Gu10] Guo, C.; Lu, G.; Wang, H. J.; Yang, S.; Kong, C.; Sun, P.; Wu, W.; Zhang, Y.: SecondNet: A Data Center Network Virtualization Architecture with Bandwidth. In: *Proceedings of the 6th International Conference. Co-NEXT*, pp. 1–15, 2010.
- [Gu16] Gurobi Optimization, I.: *Gurobi Optimizer Reference Manual*; 2015. URL <http://www.gurobi.com/>, 2016.
- [KI17] Kluge, R.; Stein, M.; Varró, G.; Schürr, A.; Hollick, M.; Mühlhäuser, M.: A Systematic Approach to Constructing Families of Incremental Topology Control Algorithms using Graph Transformation. *Software & Systems Modeling*, pp. 1–41, 2017.
- [LAS17] Leblebici, E.; Anjorin, A.; Schürr, A.: Inter-model Consistency Checking Using Triple Graph Grammars and Linear Optimization Techniques. In: *Fundamental Approaches to Software Engineering. FASE*, pp. 191–207, 2017.
- [Lo16] Lopes, F. A.; Lima, L.; Santos, M.; Fidalgo, R.; Fernandes, S.: High-Level Modeling and Application Validation for SDN. In: *Network Operations and Management Symposium. NOMS*, pp. 197–205, 2016.
- [SAE12] Sefraoui, O.; Aissaoui, M.; Eleuldj, M.: OpenStack: Toward an Open-Source Solution for Cloud Computing. *International Journal of Computer Applications* 55/3, pp. 38–42, 2012.
- [Sc95] Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: *Graph-Theoretic Concepts in Computer Science. Vol. 903. Lecture Notes in Computer Science*, pp. 151–163, 1995.
- [Xi12] Xie, D.; Ding, N.; Hu, Y. C.; Kompella, R.: The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers. In: *Applications, Technologies, Architectures, and Protocols for Computer Communication. SIGCOMM*, pp. 199–210, 2012.
- [Ze15] Zeng, D.; Guo, S.; Huang, H.; Yu, S.; Leung, V. C.: Optimal VM Placement in Data Centers with Architectural and Resource Constraints. *International Journal of Autonomous and Adaptive Communications Systems* 8/4, pp. 392–406, 2015.
- [Zh13] Zhani, M. F.; Zhang, Q.; Simona, G.; Boutaba, R.: VDC Planner: Dynamic Migration-Aware Virtual Data Center Embedding for Clouds. In: *Integrated Network Management. IM*, pp. 18–25, 2013.

Transforming Enterprise Models to Linked Data via Semantic Annotations

Benedikt Pittl¹, Hans-Georg Fill²

Abstract: The use of conceptual models in enterprises is today a well-known fact. This includes many different types of models ranging from process models, organizational models, and infrastructure models to various types used in software engineering and technical systems development. Although these models are largely specified in a formal or at least semi-formal way, the knowledge contained in them is often only accessible via manual inspection. The primary reason for this shortcoming is the use of different formats for expressing models and the lack of machine-processable semantic specifications of the model content. In this paper we present a flexible approach for transforming information from such enterprise models to RDF. Thereby, we use a model weaving technique to annotate conceptual models with concepts from ontologies. For assessing its technical feasibility, the approach has been prototypically implemented on the SeMFIS platform and applied to a use case in the area of business process management.

Keywords: Conceptual Model; RDF; Ontology; Semantic Annotation

1 Introduction

Today, enterprises heavily rely on conceptual models such as business process models, organization models or infrastructure models, thus potentially leading to hundreds if not thousands of models just within one organization [Ro06, WH01]. Such models are often created with the aim of fostering communication and understanding [My92] and are an important source of knowledge. Usually, these models are stored in the databases of the used modeling tools [vDDM13]. For analyzing these models using query techniques and benchmarks [APW08, EKO07, vDDM13], or for executing them [Fi12], models need to be available in a machine-processable format that is ideally based on a standard representation. In the context of the Web of Data, conceptual models were recently identified as a valuable source for data repositories [BK16]. Thereby, the model content is transformed to ontologies - usually in RDF format. The transformation to ontologies has two main benefits: (i) *Exchange of Model Information*. Standardized formats such as RDF foster the exchange of models across different tools and platforms. (ii) *Semantic Processing*. The usage of ontology formats

¹ University of Vienna, Faculty of Computer Science, Waehringerstrasse 29, 1090 Vienna, Austria; benedikt.pittl@univie.ac.at

² University of Bamberg, Department for Information Systems - System Development and Database Application Group, An der Weberei 5, 96047 Bamberg, Germany; hans-georg.fill@uni-bamberg.de

enables semantic processing based on reasoning and query techniques, especially through linking the model information to other ontologies and linked data repositories. This enables the linkage, querying and merging of different data sources with information contained in the models [BHB09].

The scientific community proposed two main paradigms for the transformation from conceptual models to ontologies used in the Web of Data: (i) Hinkelmann [Hi15] presented seven approaches which describe how to establish a linkage between conceptual models and ontologies. These approaches range from *simple links* – through adding textual attributes in model elements which contain a URI to an ontology element – over *semantic tunnels* where semantic information for model elements is retrieved via webservices – to *semantic transit models* where models contain references to an ontology. However, a concrete approach for transforming models to RDF was not described. Additionally, all seven approaches have the drawback that a new modeling language is required that permits to link elements to ontology concepts. Based on the assumption that enterprises already have large model repositories [WH01], the requirement of such a new modeling language would lead to a considerable effort for remodeling or at least migrating the existing models. (ii) For overcoming these drawbacks, an *RDFizer* has been presented for transforming conceptual models to RDF [BK16]. Thereby, static transformation patterns were defined that are applicable to arbitrary models. While this approach is comprehensive in the sense that each model element and attribute is serialized according to the pattern, it is not easily adaptable to specific needs e.g. to define how model concepts are serialized to RDF. Furthermore, the modeling languages of the models to be transformed has to be altered if additional RDF information is to be represented.

The research question which we want to answer in this paper is "*How to semantically enrich and process existing visual models in standardized semantic formats?*". Thereby, we pursue the following three goals: (i) The approach has to be generic so that it is applicable for models created with different modeling languages. (ii) The approach has to be simple so that business users are able to do the enrichment (iii) The approach has to be adaptive so that modifications of the enrichment are possible. Hence, in this paper we present a *customized model weaving approach* for transforming the content of conceptual models to RDF. Our approach does not require an adaptation of existing modeling languages but allows referring to existing ontology concepts. Thereby, our weaving approach is based on semantic annotations for linking model elements and their attributes with ontology schema concepts. The annotations can be created, removed or modified without affecting the conceptual models nor the ontology, which makes our approach useful for semantically enriching and processing *already existing models*. For easing the specification of the semantic annotations, we provide a domain-specific visual language. We conceptualized and evaluated the approach following three steps: (i) Specification of a visual language for configuring flexible transformations via annotations (ii) Specification of generic rules for conducting the transformation to RDF (iii) Implementation of the approach using the SeMFIS platform [Fi17].

The remainder of the paper is structured as follows. An overview of existing paradigms for semantically enriching models is presented in section 2. Our transformation approach based on visual annotations is explained in section 3. Section 4 describes the technical implementation followed by a use case based evaluation in section 5. A discussion is described in section 6. The paper closes with a conclusion in section 7.

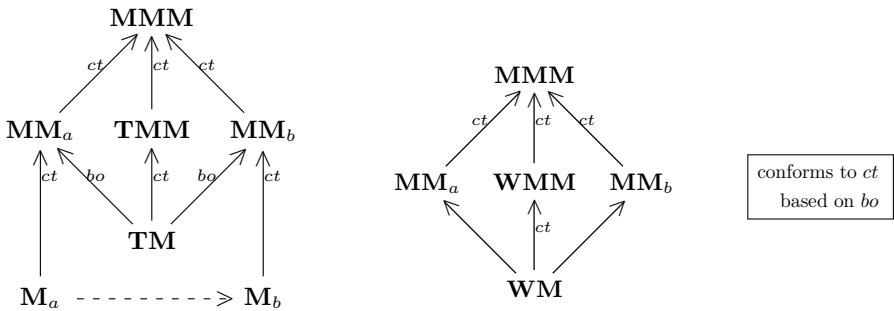
2 Background and Related Work

In the literature there are currently two research directions being investigated for linking ontologies with conceptual models: *model transformation* and *model weaving* [Fi11]. We will thus first describe the differences between model transformation and model weaving. Second, we will review existing weaving approaches which define how models can be semantically enriched. In the third part we will investigate existing approaches for the model-to-ontology transformation with a special focus on RDF ontologies.

The transformation between different types of models has been discussed to a large extent in the context of Model Driven Engineering [DFBV06]. Thereby, transformation models are models which describe operations for transferring source models to target models. These operations are executed by a transformation engine. Prominent examples of this approach are transformations via Query View Transformation (QVT) or the ATL Transformation Language. An overview of the model transformation approach from a generic perspective is shown in Figure 1a. The source model (\mathbf{M}_a) and the target model (\mathbf{M}_b) conform to the metamodels \mathbf{MM}_a and \mathbf{MM}_b which is illustrated with the *ct* connectors. Similarly, the transformation models (\mathbf{TM}) conform to a metamodel (\mathbf{TMM}). All three metamodels conform to a meta-metamodel. The transformation operations are part of the transformation model which references elements of the metamodels \mathbf{MM}_a and \mathbf{MM}_b . This is illustrated with the *based-on* (*bo*) connector. For example, the ATL rules used for a model transformation are grouped to a transformation model \mathbf{TM} which is executed by an ATL transformation engine.

The second research direction for linking models and ontologies is to use weaving models. The main difference between model weaving and model transformation is that transformation metamodels have *fixed semantics* that can be implemented by transformation engines, whereas weaving models have *user-defined semantics* [DFBV06]. The model weaving approach is illustrated in Figure 1b. For model weaving, three metamodels are used whereby \mathbf{WMM} is the weaving metamodel. Weaving models (\mathbf{WM}) are models which use domain specific link types for establishing references between two metamodels (\mathbf{MM}_a and \mathbf{MM}_b). The overall goal of model weaving is just to establish links between elements of two models. The weaving model can be used for model transformations but it is not limited to it. Hence, the illustration in Figure 1b does not show a transformation example as Figure 1a. Weaving approaches are e.g. used for model traceability as well as for model alignment. According to [DFBV06] model weaving fulfills the following requirements: (i) the weaving model supports the expression of links between two model elements, (ii) different types of links

have to be supported whereby the link type provides the semantics, (iii) the links support different arities, (iv) and the links have references to the model elements.



(a) Model transformation based on [Jo06] (b) Model weaving from [DFJ05]

Fig. 1: Model transformation and model weaving

Based on these foundations we can now investigate approaches that make use of these concepts for linking conceptual models and ontologies. Two main reasons can be stated why such connections are beneficial: First, the use of standardized exchange formats such as RDF and OWL permits the easy transfer of model information across different tools and platforms. Second, ontology formats permit semantic processing based on reasoning and query techniques, especially through linking the model information to other ontologies and linked data repositories.

Following the direction of model transformations, it is often being referred to the XML Metadata Interchange (XMI) format as a starting point for transforming models to ontologies. XMI is a standardized format maintained by the OMG³ which fosters the exchange of models between different modeling tools. For example, [Ga04] describe a transformation approach for models represented in XMI to OWL via XSLT. Similar approaches are described in [BB12] and [Cr01]. In [TF07] Event driven Process Chains (EPC) models are transformed to an RDF ontology. Thereby, the authors assume that the EPC model is stored in the XML-based format EPML so that the transformation to RDF-XML can also revert to XSLT. However, all these XSLT transformations are static. They are predefined and cannot be adapted by end users. Furthermore, such direct XSLT transformations consider metamodels only implicitly. Linkages to other ontologies or linked data repositories are not foreseen.

In [BK16] a transformation approach from models to RDF is introduced, denoted as *RDFizer*. It supports three different ways for first linking existing URIs to model elements: (i) *Linking by Equivalence*: This way is very similar to the direct linkage approach presented in [Hi15]. It expects a string attribute in each model element which contains a URI of an equivalent ontology concept. (ii) *Linking by Modeling Properties*: Similar to the previous approach,

³ <http://www.omg.org/spec/XMI/>

URIs are entered into string attributes of model elements. However, the interpretation is different. The entered URI does not represent the model element which contains the attribute. Instead, it refers to related concepts. (iii) *Linking by Arbitrary Properties or Types*: For adding additional information to model elements, an attribute Table can be added. This Table is used for generating customized RDF triples. Thereby, the model element which contains the Table is either the subject or the object of the triple. In a second step, [BK16] then apply static patterns for the transformation of the conceptual models to RDF ontologies. These patterns are pre-defined generic rules which determine how model information is transformed to RDF triples. Using this approach, customization is possible ex-post, e.g. using the external Java component "RDF export customizer" as shown in [KB16] and [BK15]. It allows adding, removing or modifying RDF triples created with the patterns. In addition to the RDFizer several related approaches exist. For example, in [Ka06] an approach for semantic lifting of metamodels was introduced. Thereby, the authors transfer metamodels to ontologies using mapping patterns. Based on this mapping metamodels are transferred to ontologies.

In the following we investigate approaches which make use of the model weaving paradigm. The previously mentioned seven different approaches by Hinkelmann [Hi15] do not fulfill all described requirements for model weaving as stated above - a detailed discussion of them is out of the scope of this paper. However, they are closely related to this direction. In the following, we focus on the three most relevant types. In all of them it is implied that model elements contain references to ontology concepts: (i) *Indirect Linkage*: Following this approach, the whole ontology is represented as a visual model called *semantic transit model*. The connections between the model and the ontology are established using additional hyperlink attributes in the model. (ii) *Direct and Indirect Linkage*: This is a combination of the indirect linkage approach and the so-called *semantic tunnel approach*. It retrieves ontology concepts via a webservice (semantic tunnel). Thereby, only selected concepts are represented in the semantic transit model. The model elements have again hyperlink attributes for referencing ontology concepts. Based on these references additional ontology concepts can be retrieved and offered to the user. (iii) *Loose Coupling*: This approach introduces an intermediate ontology that acts as a reference for connecting it to model elements.

In summary it can be stated that existing approaches have achieved various ways for transforming conceptual models to ontology formats. However, an approach for transforming conceptual models to RDF with a special focus on (i) adaptability (ii) semantic enrichment for linking them to existing ontology and data repositories and (iii) adequacy for non-technical users is missing so far.

3 Transformation via Visual Annotations

The main motivation for our work is based on the assumption that potentially large repositories of conceptual models already exist in an organization which are used by

human actors and technical systems alike, cf. [WH01]. These models are usually stored in vendor-specific formats and there is a lack of machine-processable semantic specifications of the model content. Rather than remodelling all existing models with an adequate modeling language - which would be costly - we could semantically enrich the existing models. Therefore, we first need to provide means for a semantic enrichment of these models to align them with existing data and ontology schemas. Semantic annotations have been shown as a solution that does not require changes in the models nor the underlying modeling language [Fi11]. As depicted in Figure 2, we use annotations that are stored in visual Annotation Models on the *Configuration Layer* of our approach. Thereby, they configure the RDF serialization of conceptual models. The annotations have references to both, model concepts and ontology schema concepts. The RDF serialization itself belongs to the second layer called *Standardized Semantic Representation Layer*. The resulting RDF can subsequently be merged with other ontologies or queried, which is foreseen in the *Analysis Layer*.

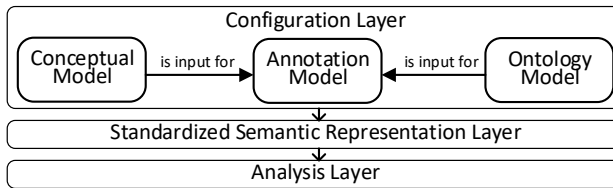


Fig. 2: Overview of the Three-layered Approach for Transforming Conceptual Models to RDF

As the annotations are created as visual models, the provision of a corresponding visual modeling language is required. For describing the visual modeling language as well as the transformations to RDF in a non-ambiguous way we will revert to the FDMM formalism [FRK12] in the following. A summary of the notation and the elements of the visual language - which are described using FDMM - is given in Table 1.

3.1 FDMM Core Concepts

For formally describing modeling languages, different formalisms have been introduced – e.g. for EMF [Sc08] or in the OCL specification⁴. We decided to use the FDMM formalism as it aligns well with the concepts used in the ADOxx metamodeling platform, which we used later for the implementation and evaluation of our approach [FRK12].

FDMM describes metamodels \mathbf{MM} using four components $\mathbf{MM} = \langle \mathbf{MT}, \leq, \text{domain}, \text{range}, \text{card} \rangle$. Each metamodel consists of a set model types \mathbf{MT} which are used to create a set of model instances \mathbf{mt} . Each model type \mathbf{MT}_i consists of a set of object types \mathbf{O}_i^T , which have in turn a set of attributes \mathbf{A}_i . Each attribute is assigned a datatype

⁴ <http://www.omg.org/spec/OCL/2.0/>

from the set \mathbf{D}_i^T . So, in FDMM a model type is described as follows: $\mathbf{MT}_i = \langle \mathbf{O}_i^T, \mathbf{D}_i^T \mathbf{A}_i \rangle$. \leq is an ordering on the set of object types \mathbf{O}_i^T for defining an object type hierarchy similar to inheritance hierarchies in object-orientation. *domain* is a function which assigns attributes to object types. The *range* function assigns datatypes to attributes while the *card* function defines the cardinality of attribute values. Models \mathbf{mt}_i consist of triples τ representing the model content. The first element of a triple $t \in \tau$ represents an instance of an object type, the second component represents an attribute, and the last component represents the attribute value. Due to limited space we do not describe FDMM in more detail here - for more information we refer to [FRK12].

3.2 Formalizing Visual Annotations in FDMM

The visual modeling language for annotations consists of a single model type \mathbf{MT}_{Annot} . This model type has a set of object types \mathbf{O}_{Annot}^T which have a set of attributes \mathbf{A}_{Annot} which have in turn data types \mathbf{D}_{Annot}^T . The object types used in the model type are described in the following equation. In FDMM, model connectors such as *isInputFor* and *refersTo* are also considered as object types.

$$\mathbf{O}_{Annot}^T = \{ModelReference, ConnectorReference, AttributeReference, \\ OntologyReference, Annotator, AnnotationElement, \\ ModelReferences, isInputFor, refersTo\} \quad (1)$$

All the attributes used in the object types are part of the set \mathbf{A}_{Annot} .

$$\mathbf{A}_{Annot} = \{Name, AllClassInstances, InstanceReference, AttributeName, \\ ConnectorName, AnnotationType, isInputFor-from, \\ isInputFor-to, OntologySchemaConceptReference, \\ refersTo-from, refersTo-to\} \quad (2)$$

The datatypes of the attributes are summarized in the set \mathbf{D}_{Annot}^T . $Enum_{anntype}$ represents an enumeration list.

$$\mathbf{D}_{Annot}^T = \{String, Enum_{anntype} = \{instanceOf, isEqualTo, isBroaderThan, \\ isNarrowerThan, isSubclassOf, isSuperclassOf, \\ isInstanceUsingFromClass, isInstanceUsingToClass\}\} \quad (3)$$

We have defined an ordering of the object types similar to an inheritance hierarchy to avoid the duplicate specification of attributes:






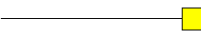

$$\begin{aligned} AnnotationElement &\leq InstanceType \\ OntologyReference &\leq AnnotationElement \\ Annotator &\leq AnnotationElement \\ ModelReferences &\leq AnnotationElement \\ ConnectorReference &\leq ModelReferences \\ ModelReference &\leq ModelReferences \\ AttributeReference &\leq ModelReference \end{aligned} \quad (4)$$

The object type *InstanceType* can be considered as a super-object type similar to the class *Object* in Java - so we did not list it explicitly in \mathbf{O}_{Annot}^T . Attributes and their value ranges were specified using the FDMM domain and range functions [FRK12]: $domain(Name) = \{AnnotationElement\}$, $domain(AllClassInstances) = \{ModelReferences\}$, $domain(InstanceReference) = \{ModelReferences\}$, $domain(AttributeName) = \{AttributeReference\}$, $domain(AnnotationType) = \{Annotation\}$, $domain(OntologySchemaConceptReference) = \{OntologyReference\}$, $domain(isInputFor-from) = \{isInputFor\}$, $domain(isInputFor-to) = \{isInputFor\}$, $domain(refersTo-from) = \{refersTo\}$, $domain(refersTo-to) = \{refersTo\}$. $range(Name) = \{String\}$, $range(AllClassInstances) = \{true, false\}$, $range(InstanceReference) = \{InstanceType\}$, $range(AttributeName) = \{Enum_{attribute\ names}\}$, $range(AnnotationType) = \{Enum_{annotation\ type}\}$, $range(OntologySchemaConceptReference) = \{InstanceType\}$, $range(isInputFor-from) = \{ModelReferences\}$, $range(isInputFor-to) = \{Annotator\}$, $range(refersTo-from) = \{Annotator\}$, $range(refersTo-to) = \{OntologyReference\}$. In FDMM the cardinality function - abbreviated with *card* - defines how many attribute values an object type can have. In our modeling type all attributes have at most one value. Therefore, we do not list the cardinality functions explicitly here.

3.3 Graphical Notation

The graphical notation of the described object types is depicted in Table 1. There are three different object types which have hyperlinks to classes, connectors and attributes of conceptual models: The Model Reference (MREF) object type has a hyperlink to model elements (instances of object types except connectors). The Attribute Reference (AREF) object type has a hyperlink to attributes of model elements and the Connector Reference (CREF) object type has a hyperlink to instances of connectors, which are object types in FDMM. All three object types have further attributes for a more precise description of the type of linkage which should be established. For example, the MREF, AREF as well as the CREF object type have an attribute *applies to all instances*. This attribute indicates if the reference is only representative for the model element to which the MREF, AREF or CREF instances points to, or if it is representative for all instances of the same object type in the conceptual model. Instances of the object type *Annotator* are the connecting link between CREF, AREF and MREF elements which reference to contents of conceptual models and elements which reference to ontology schema concepts. The latter are instances of the OREF object type in our visual language. Annotator elements contain additional information regarding the type of linkage which is established. The connectors shown on the lower right corner of Table 1 are the connectors for constructing annotations. An example of an annotation created with our visual language is shown in the use case (section 5).

Similar to the transit model approach described in [Hi15] we make use of visual ontology models. This means that we represent ontologies such as OWL ontologies or frames ontologies as visual models. To ensure interoperability with applications such as Stanford

Graphical Notation	Description	Graphical Notation	Description
 <p>Modeltype: Business process model Model: Business process model -... Class: Activity Reference applies to all instances.</p>	<p>MREF- references to instances of model object types (non-connectors) FDMM: $ModelReference \in O^T_{Annot}$</p>	 <p>Modeltype: Business process model Model: Business process model -... Class: Activity Reference applies to all instances. Attribute: Description</p>	<p>AREF- references to attributes of instances of object types FDMM: $AttributeReference \in O^T_{Annot}$</p>
 <p>Modeltype: Business process model Model: Business process model -... Relation Class: Subsequent Reference applies to all instances.</p>	<p>CREF- references to instances of model object types (connectors) FDMM: $ConnectorReference \in O^T_{Annot}$</p>	 <p>Modeltype: Ontology Model Model: Ontology Model - new (2) Class: Class Instance: BPMNActivity</p>	<p>OREF- references to ontology schema concepts FDMM: $OntologyReference \in O^T_{Annot}$</p>
	<p>Annnotator- connecting MREF, AREF, CREF elements with OREF elements FDMM: $Annot \in O^T_{Annot}$</p>		<p>Connector between MREF, CREF or AREF and Annotator FDMM: $isInputFor \in O^T_{Annot}$</p>
			<p>Connector between Annotator and OREF FDMM: $refersTo \in O^T_{Annot}$</p>

Tab. 1: Overview of the Object Types Used in the Visual Language for Creating Annotations

Protégé we developed an import as well as an export function to standardized ontology serialization syntaxes (e.g. OWL-XML).

3.4 Transformation Rule in FDMM

After the annotations are created they are used in the standardized semantic representation layer to transform the annotated conceptual models to an RDF ontology. For a better understanding of how the transformation works we present in the following an exemplary transformation rule for annotations. It is assumed that an annotation model is present in which an MREF element is connected with an OREF element via an annotator element of type *instanceOf*. The MREF elements reference elements in the conceptual model and the OREF element corresponding ontology concepts.

The function *getAttributeValue* returns the attribute value t_3 (third component) of a triple $t \in \tau$ whereby $\mathbf{mt}_i \in \mu_{MT}(MT_{Annotation})$. We used FDMM also for describing OWL ontologies ($\mathbf{mt}_{OWL} \in \mu_{MT}(MT_{OWL})$) as well as the resulting RDF ontologies ($\mathbf{mt}_{RDF} \in \mu_{MT}(MT_{RDF})$). Similarly, we described the conceptual model $\mathbf{mt}_{REF} \in \mu_{MT}(MT_{REF})$ to which the MREF element refers to in FDMM. The Id attribute of the referenced elements

represents a unique identifier. A sample transformation rule is then specified as follows (the structure of the other rules is similar):

Transformation Rule to RDF described in FDMM: Annotation which connects MREF elements with OREF elements via an annotator of type instanceOf

$$\begin{aligned}
 & \forall \mathbf{mt} \in \mu_{\mathbf{MT}}(MT_{\text{Annotation}}) \\
 & \forall \mathbf{mref} \in \mu_{\mathbf{O}}(MREF, MT_{\text{Annotation}}) | (\mathbf{mref}, AllClassInstances, false) \in \beta(\mathbf{mt}) \\
 & \forall \mathbf{schemaConcept} \in \mathbf{O} | (\\
 & \quad \exists \mathit{inputConnector} | (\mathit{inputConnector}, \mathit{isInputFor-from}, \mathbf{mref}) \in \beta(\mathbf{mt}) \wedge \\
 & \quad \exists \mathit{annotation} | (\mathit{inputConnector}, \mathit{isInputFor-to}, \mathit{annotation}) \in \beta(\mathbf{mt}) \wedge \\
 & \quad \exists \mathit{refersToConnector} | (\mathit{refersToConnector}, \mathit{refersTo-from}, \mathit{annotation}) \in \beta(\mathbf{mt}) \wedge \\
 & \quad \exists \mathit{oref} | (\mathit{refersToConnector}, \mathit{refersTo-to}, \mathit{oref}) \in \beta(\mathbf{mt}) \wedge \\
 & \quad (\mathit{oref}, \mathit{OntologySchemaConceptReference}, \mathbf{schemaConcept}) \in \beta(\mathbf{mt}) \\
 & \quad) \\
 & \forall \mathbf{modelElement} \in \{ \mathit{getAttributeValue}(i) | i \in \{\beta(\mathbf{mt})\}_{t_1 = \mathbf{mref}} \\
 & \quad \wedge t_2 = \mathit{InstanceReference} \} \\
 & \quad \implies \\
 & \quad \exists \mathbf{mt}_{\text{RDF}} \in \mu_{\mathbf{MT}}(MT_{\text{RDF}}) \wedge \\
 & \quad \exists t \in \beta(\mathbf{mt}_{\text{RDF}}) | (\\
 & \quad \quad t_1 \in \mu_{\mathbf{O}}(\mathit{Description}, MT_{\text{RDF}}) \wedge \\
 & \quad \quad (t_1, \mathit{rdf} : \mathit{about}, y) \in \beta(\mathbf{mt}_{\text{RDF}}) | (\mathbf{modelElement}, \mathit{Id}, y) \in \beta(\mathbf{mt}_{\text{REF}}) \wedge \\
 & \quad \quad (t_1, \mathit{rdf} : \mathit{type}, x) \in \beta(\mathbf{mt}_{\text{RDF}}) | (\mathbf{schemaConcept}, \mathit{Id}, x) \in \beta(\mathbf{mt}_{\text{OWL}}) \\
 & \quad \quad) \\
 & \quad)
 \end{aligned}$$

In the same way we created FDMM-based rules for all kinds of annotations, i.e. with CREF elements for referencing connectors and AREF elements for referencing attributes. Due to the space limit these are omitted here.

4 Technical Implementation

Based on the FDMM specifications we prototypically implemented the approach using the SeMFIS platform [Fi17] to evaluate its technical feasibility. The reasons for using the ADOxx based SeMFIS platform are besides familiarity due to previous projects with this platform twofold: (i) ADOxx is open and (ii) ADOxx is widely used in the modeling community e.g. for the process modeling toolkit ADONIS. We extended SeMFIS with our RDF transformation approach. Additionally, we implemented an OWL/XML import/export function. The implementation follows a three-layer approach as shown in Figure 3. The model types were implemented using the ADOxx development toolkit underlying SeMFIS. The transformation rules were encoded using XSLT. In addition, a Java component was created for merging the OWL and RDF ontologies.

The following numbers correspond to the numbers used in Figure 3. (1,2) First, the conceptual models are created or loaded. Additionally, the visual ontology model is created from scratch or imported from an existing ontology file. (3) After the conceptual model as well as the ontology model are in place, annotations are created using the visual language

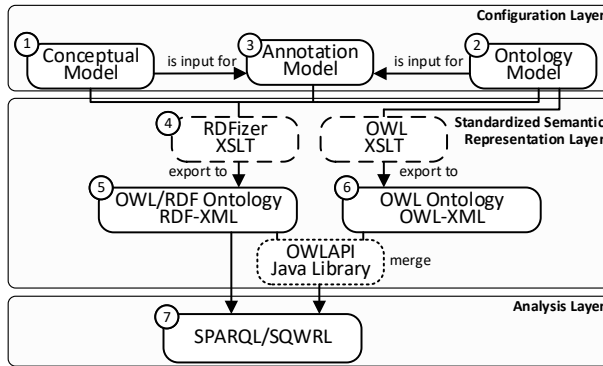


Fig. 3: Overview of the Technical Implementation

introduced in the previous section. Thereby, the content of conceptual models is linked with ontology schema constructs using annotations. (4) The annotations created with the visual language are used to configure the RDF serialization. Therefore transformation rules as exemplary introduced in the previous section are applied. (5) The transformation results in an RDF ontology containing the model elements for which we created annotations. The RDF ontology does not contain schema constructs but only instances. (6) It is thus possible to export the visual ontology model to which the annotations reference as an OWL ontology. (7) Then we get two ontologies: an RDF ontology (see (5)) and an OWL ontology (see (6)). Both ontologies can be merged. This is accomplished using the Java library OWLAPI. The resulting ontology then contains both, the OWL ontology including the schema constructs, as well as the instances stored as RDF triples. This ontology can then be processed using further semantic tools and techniques such as reasoners, query or rule engines.

5 Evaluation through a Use Case

In addition to the evaluation of the technical feasibility, we applied the approach to a use case to assess whether it can be used in a practical scenario.

For this purpose we reverted to an account opening business process that has been previously used within the Open Models Initiative⁵ - see [KMM16] for more information. Figure 4 shows an excerpt of the process model which was created with the domain-specific modeling language BPMS [KJS96]. Typically, models in such domain-specific languages are stored in an internal, platform-dependent serialization format, which makes machine-processing difficult. Therefore, the use of a standardized format is beneficial. Hence, we annotated the process model with OWL ontologies as shown in the right part in Figure 4. The depicted *OntoRule Ontology* ontology is an excerpt of the process ontology developed

⁵ <http://www.semfis-platform.org/>

within the OntoRules project⁶. We further created an ontology called *user ontology* with the data property *hasExecutionTime*. The lines shown in Figure 4 depict references as used in the MREF, CREF and OREF elements. To keep the Figure simple we have not shown the references of the AREF elements. As the annotation model shows, the process activity of the process model is annotated with the OWL class *Task*. Further, we used an AREF element for the annotation of the execution time attribute. It is annotated with the OWL dataproperty *hasExecutionTime*. The connector of the type *Subsequent*, which connects activities in a business process model, is annotated with the *follows* OWL property.

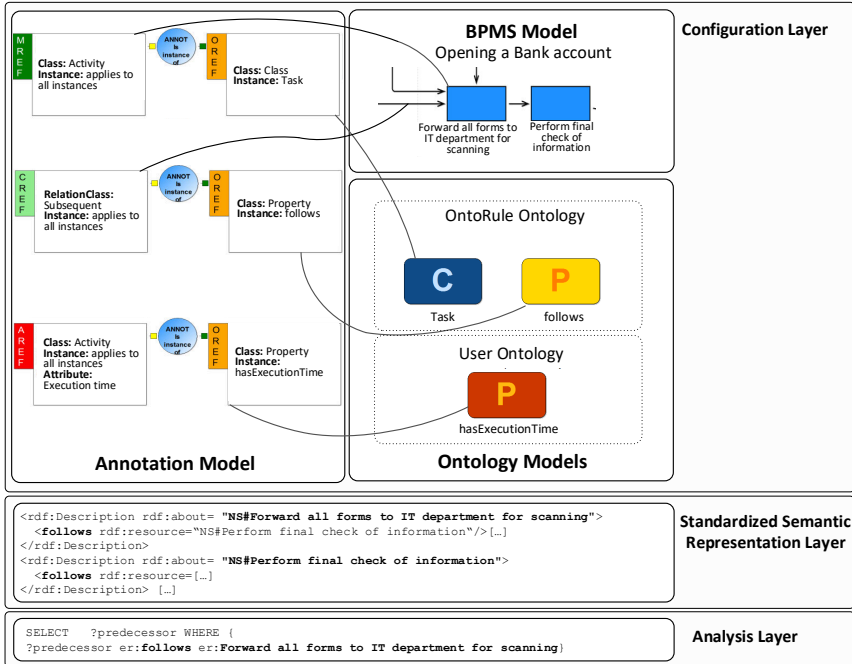


Fig. 4: Overview of Use Case Showing Excerpts of an Annotation Model Instance (left), of a Process Model in BPMS Notation (top right), Two Ontology Models (mid right), the Resulting RDF Representation, and a Sample Query in SPARQL

Based on these visual annotations, the XSLT stylesheets together with the Java component as described in the previous section transform the process model into an RDF serialization. This is depicted in the standardized semantic representing layer in Figure 4. Listing 1 shows a more detailed excerpt of the resulting RDF. For all annotated model elements, RDF resources are created including the corresponding types. The connector reference leads to the creation of the *follows* property. Similarly, the annotation of the attribute leads to the creation of the *hasExecutionTime* property. The serialized RDF can be analyzed as shown in the last layer of Figure 4. An example query using the SPARQL query language is

⁶ <http://ontorule-project.eu/resources/assembler/process-ontology-and-facts.owl>

LISTING 1: Excerpt of the RDF Serialization of the Use Case Example - namespaces were neglected

```

<rdf:Description rdf:about= "NS#Forward all forms to IT department for scanning" >
  <rdf:type rdf:resource= "NS#Task" />
  <follows rdf:resource= "NS#Perform final check of information" />
  <hasExecutionTime rdf:datatype="http://www.w3.org/2001/XMLSchema#string" >
    | 00:000:00:03:00
  </hasExecutionTime>
</rdf:Description>
<rdf:Description rdf:about= "NS#Perform final check of information" >
  <rdf:type rdf:resource= "NS#Task" />
  <follows rdf:resource= "NS#Forward remaining forms to inspection department" />
  <hasExecutionTime rdf:datatype="http://www.w3.org/2001/XMLSchema#string" >
    | 00:000:00:02:00
  </hasExecutionTime>
</rdf:Description>

```

shown in Listing 2. This query selects all predecessor activities of the *Perform final check of information* activity.

LISTING 2: Example of SPARQL Query on an RDF Representation of a Model from the Use Case

```

SELECT ?predecessor WHERE {
  ?predecessor ns:follows ns:Perform final check of information
}

```

Result: ns:Forward all forms to IT department for scanning

6 Discussion

With the technical realization of the approach and its application to a use case we can conclude that the presented approach is useful for semantically enriching existing visual models ex-post. Thereby, neither the models nor the used ontologies have to be modified with the creation of annotations. Thus, the annotations are not limited by the type of model or ontology. However, in the described implementation, the ontologies have to be imported as visual models in order to use our annotation approach. The described weaving approach requires however that the modeling tool used for it supports model references. Hence, tools which do not support model references have to be adapted to realize the loosely coupled semantic annotation approach. A performance analysis is part of our further research.

In summary we can derive a number of benefits as well as also some drawbacks of the approach in its current version. These are listed in Table 2.

Benefits	Drawbacks
⊕ Customization of RDF generation with visual annotations	⊖ Visual annotations may become complex to handle
⊕ Independent of the used modeling language for conceptual models	⊖ Direct annotation references to ontology concepts not implemented yet
⊕ Types of annotations are extendable	⊖ Semantics of annotation types needs to be provided separately via rules
⊕ Annotations are re-usable	⊖ Ontology schema concepts are required
⊕ OWL import/export options exist	⊖ RDF-serialization for non-OWL ontologies not implemented yet

Tab. 2: Benefits and drawbacks of the introduced approach

Usability test and economical analysis are two aspects which are out of the scope of this paper but which have to be done before implementing the approach in industry modeling tools. The linking mechanism - from the visual annotation to model elements - is probably the most challenging feature. This is because the linking mechanism has to be generic so that model elements created with different modeling languages and tools can be referenced. Further, we see the support of different ontologies - as described in Table 2 - as an important feature to make the approach feasible.

The introduced approach enables institutions to semantically enrich their existing models created with different modeling tools. Hence, they save costs for remodelling and standardizing the existing models. However, the introduced approach requires human-created annotations so that institutions face a trade-off between costs for remodelling and costs for creating semantic annotations.

7 Conclusion and Further Research

In this paper we introduced a model weaving approach for transforming conceptual models to RDF. For this purpose we introduced a visual modeling language for creating model annotations. The annotations are neither limited to a specific kind of conceptual model nor to a specific kind of ontology. The technical feasibility of the approach has been shown by implementing it on the SeMFIS platform and applying it to a use case.

In our future research we want to develop further types of annotations and introduce ontology references which point directly to ontology schema constructs, e.g. as contained in an ontology repository. In this way the transformation of ontologies to visual models could be omitted. Another aspect that will be investigated will be the usability of the approach. For this purpose especially the procedures of annotating existing models will have to be tested with users to judge whether the used modeling language is adequate in a practical setting. Economical as well as usability analysis are part of our further research.

References

- [APW08] Awad, Ahmed; Polyvyanyy, Artem; Weske, Mathias: Semantic querying of business process models. In: Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE. IEEE, pp. 85–94, 2008.
- [BB12] Belghiat, Aissam; Bourahla, Mustapha: Transformation of UML models towards OWL ontologies. In: Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), 2012 6th International Conference on. IEEE, pp. 840–846, 2012.
- [BHB09] Bizer, Christian; Heath, Tom; Berners-Lee, Tim: Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.
- [BK15] Buchmann, Robert Andrei; Karagiannis, Dimitris: Pattern-based Transformation of Diagrammatic Conceptual Models for Semantic Enrichment in the Web of Data. In: 19th International Conference in Knowledge Based and Intelligent Information and Engineering Systems, KES 2015, Singapore, 7-9 September 2015. pp. 150–159, 2015.
- [BK16] Buchmann, Robert A.; Karagiannis, Dimitris: Enriching Linked Data with Semantics from Domain-Specific Diagrammatic Models. *Business & Information Systems Engineering*, 58(5):341–353, 2016.
- [Cr01] Cranefield, Stephen: Networked Knowledge Representation and Exchange using UML and RDF. *J. Digit. Inf.*, 1(8), 2001.
- [DFBV06] Del Fabro, M. Didonet; Bézivin, Jean; Valduriez, Patrick: Weaving Models with the Eclipse AMW plugin. In: Eclipse Modeling Symposium, Eclipse Summit Europe. volume 2006, 2006.
- [DFJ05] Del Fabro, Marcos Didonet; Jouault, Frédéric: Model transformation and weaving in the AMMA platform. *Proceedings of GTTSE*, 2006, 2005.
- [EKO07] Ehrig, Marc; Koschmider, Agnes; Oberweis, Andreas: Measuring similarity between semantic business process models. In: Proceedings of the fourth Asia-Pacific conference on Conceptual modelling-Volume 67. Australian Computer Society, Inc., pp. 71–80, 2007.
- [Fi11] Fill, Hans-Georg: On the Conceptualization of a Modeling Language for Semantic Model Annotations. In: Advanced Information Systems Engineering Workshops - CAiSE 2011 International Workshops, London, UK, June 20-24, 2011. *Proceedings*. pp. 134–148, 2011.
- [Fi12] Fill, Hans-Georg: An Approach for Analyzing the Effects of Risks on Business Processes using Semantic Annotations. In: 20th European Conference on Information Systems, ECIS 2012, Barcelona, Spain, June 10-13, 2012. p. 111, 2012.
- [Fi17] Fill, Hans-Georg: SeMFIS: A flexible engineering platform for semantic annotations of conceptual models. *Semantic Web*, 8(5):747–763, 2017.
- [FRK12] Fill, Hans-Georg; Redmond, Timothy; Karagiannis, Dimitris: FDMM: A Formalism for Describing ADOxx Meta Models and Models. In: ICEIS 2012 - Proceedings of the 14th International Conference on Enterprise Information Systems, Volume 3, Wroclaw, Poland, 28 June - 1 July, 2012. pp. 133–144, 2012.

- [Ga04] Gasevic, Dragan; Djuric, Dragan; Devedzic, Vladan; Damjanovic, Violeta: Converting UML to OWL ontologies. In: Proceedings of the 13th international conference on World Wide Web - Alternate Track Papers & Posters, WWW 2004, New York, NY, USA, May 17-20, 2004. pp. 488–489, 2004.
- [Hi15] Hinkelmann, Knut: Modeling Framework for BPaaS. CloudSocket, December 2015. <https://www.cloudsocket.eu/documents/10182/20690/CloudSocket-D3.1-BPaaS+Design+Environment+Research/91a3c2ae-6394-482a-940e-d0186e82f7f6>, Accessed on 13-04-2017.
- [Jo06] Jouault, Frédéric; Allilaire, Freddy; Bézivin, Jean; Kurtev, Ivan; Valduriez, Patrick: ATL: a QVT-like transformation language. In: ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. ACM, pp. 719–720, 2006.
- [Ka06] Kappel, Gerti; Kapsammer, Elisabeth; Kargl, Horst; Kramler, Gerhard; Reiter, Thomas; Retschitzegger, Werner; Schwinger, Wieland; Wimmer, Manuel: Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. In: International Conference on Model Driven Engineering Languages and Systems. Springer, pp. 528–542, 2006.
- [KB16] Karagiannis, Dimitris; Buchmann, Robert Andrei: Linked Open Models: Extending Linked Open Data with conceptual model information. *Inf. Syst.*, 56:174–197, 2016.
- [KJS96] Karagiannis, Dimitris; Junginger, Stefan; Strobl, Robert: Introduction to Business Process Management Systems Concepts. In: Business process modelling, pp. 81–106. Springer, 1996.
- [KMM16] Karagiannis, Dimitris; Mayr, Heinrich C.; Mylopoulos, John, eds. Domain-Specific Conceptual Modeling, Concepts, Methods and Tools. Springer, 2016.
- [My92] Mylopoulos, John: Conceptual modelling and Telos. *Conceptual Modelling, Databases, and CASE: an Integrated View of Information System Development*, New York: John Wiley & Sons, pp. 49–68, 1992.
- [Ro06] Rosemann, Michael: Potential pitfalls of process modeling: part A. *Business Process Management Journal*, 12(2):249–254, 2006.
- [Sc08] Schätz, Bernhard: Formalization and rule-based transformation of EMF Ecore-based models. In: International Conference on Software Language Engineering. Springer, pp. 227–244, 2008.
- [TF07] Thomas, Oliver; Fellmann, Michael: Semantic EPC: Enhancing Process Modeling Using Ontology Languages. In: Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management SBPM 2007, held in conjunction with the 3rd European Semantic Web Conference (ESWC 2007), Innsbruck, Austria, June 7, 2007. 2007.
- [vDDM13] van Dongen, Boudewijn F.; Dijkman, Remco M.; Mendling, Jan: Measuring Similarity between Business Process Models. In: *Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE*, pp. 405–419. 2013.
- [WH01] Whitman, Larry; Huff, Brian: On the Use of Enterprise Models. *International Journal of Flexible Manufacturing Systems*, 13(2), 2001.

Towards a run-time model for data protection in the cloud

Zoltán Ádám Mann, Andreas Metzger, and Stefan Schoenen¹

Abstract: The protection of sensitive data in the cloud is a challenge of increasing importance. It is made particularly difficult by the complex and dynamic interactions of many entities (hardware and software, as well as organizations and individuals). A model-based approach can be used to reason about these interactions and their impact on data protection during deployment and at run time. The basis for such an approach is a model of all relevant socio-technical cloud entities, which is created during deployment and kept alive at run-time to support adaptations. In this paper, we focus on the meta-model of this model. The meta-model is created during design and instantiated during deployment. We discuss what entities must be present in the meta-model to allow reasoning about data protection. In particular, we discuss to what extent the results of previous cloud modeling efforts can be reused and what extensions are necessary because of the particular requirements of data protection.

Keywords: Models@runtime; cloud computing; data protection; privacy

1 Introduction

The compelling advantages of cloud computing, such as the instantaneous access to services and seemingly infinite compute power without the need for costly IT equipment, have made the cloud the platform of choice in many domains. The cloud is a complex and highly dynamic environment. For example, the active user base of cloud services is continuously changing, and so is the intensity with which cloud users use the services. Also the specific requirements of the users keep changing. New services are added, existing services upgraded, old services removed, and so on. To adapt to changes in the workload, software components are scaled in or out or migrated between servers.

The flexibility and dynamism offered by cloud computing is an advantage for cloud users, as they can access and pay for compute and software resources on demand [Ma15b]. Yet, at the same time, this flexibility and dynamism implies data protection risks. Protecting sensitive data in the cloud is becoming an important limiting factor of cloud adoption [Mo13]. Also requirements and constraints regarding data protection² can change continuously. As an example, users may change their preferences regarding how their personal data is handled by cloud services. The General Data Protection Regulation (GDPR) of the European Union

¹ The authors are with paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen

² In this paper, we use the term *data protection* to refer to the protection of sensitive data. This includes privacy (protection of personal data). Security concerns are included, as long as they relate to the protection of sensitive data (but security issues that do not impact sensitive data are not relevant here).

[Co16] increases the breadth and depth of control that users (in this context called data subjects) have about their data. This makes it easier for users to withdraw their consent to processing and storage of their data, but in turn exhibits new challenges for cloud service providers. In the flexible and dynamic setting of cloud systems, the applicability of traditional security mechanisms that were designed to keep the system in a stable secure state is limited [BKW14]. In particular, security-by-design methodologies are not sufficient anymore, due to uncertainty at design time of how the cloud and privacy requirements may dynamically evolve and change at run time.

A possible approach to cope with continuously changing data protection requirements in a dynamic cloud environment is to apply run-time monitoring and adaptation. This way the system can adapt at run time to changes in both the cloud and the data protection requirements, ensuring that requirements are met in the presence of changes, with minimal impact on other quality metrics like performance and costs [Ma15a]. To enable effective adaptation at run time, a run-time model, i.e., a model of the system and its environment available for reasoning at run time, is of central importance [Am12].

Therefore, our aim is to devise a run-time model of the cloud, which is useful for detecting and mitigating data protection violations. More specifically, we focus on the key modeling concepts required to cover the main elements of a cloud system in a run-time model of the cloud, leading to a *cloud meta-model for data protection*. The challenge in devising the meta-model of the cloud is to determine a sufficient level of detail as well as the necessary scope of modeled entities. Data protection concerns relate to all layers of the cloud stack, including, for example, secure hardware capabilities, co-location of virtual machines of different tenants on the same server, encryption of the communication between application components, and data anonymization. Moreover, the actors (organizations and individuals) as well as their goals and relations may play an important role. The challenge, therefore, is to devise a holistic model encompassing all relevant entities.

The approach followed in this paper can be summarized as follows. (i) We identify the types of information that the meta-model must contain in order to serve as a basis for assessing data protection issues. (ii) We specify the entities and their most important relations in a possible cloud meta-model. (iii) A scenario from an industrial context is used to validate the applicability of the suggested meta-model.

2 Industrial cloud scenario

To devise an appropriate model, it is important to first understand the purpose of modeling. In our case, this means that we need to understand the types of data protection violations that we want to be able to detect. To this end, we look at an industrial cloud setup and its implications on data protection. This scenario has been defined in the context of the project “RestAssured – Secure Data Processing in the Cloud”³ jointly with several industry partners.

³ <https://restassuredh2020.eu/>

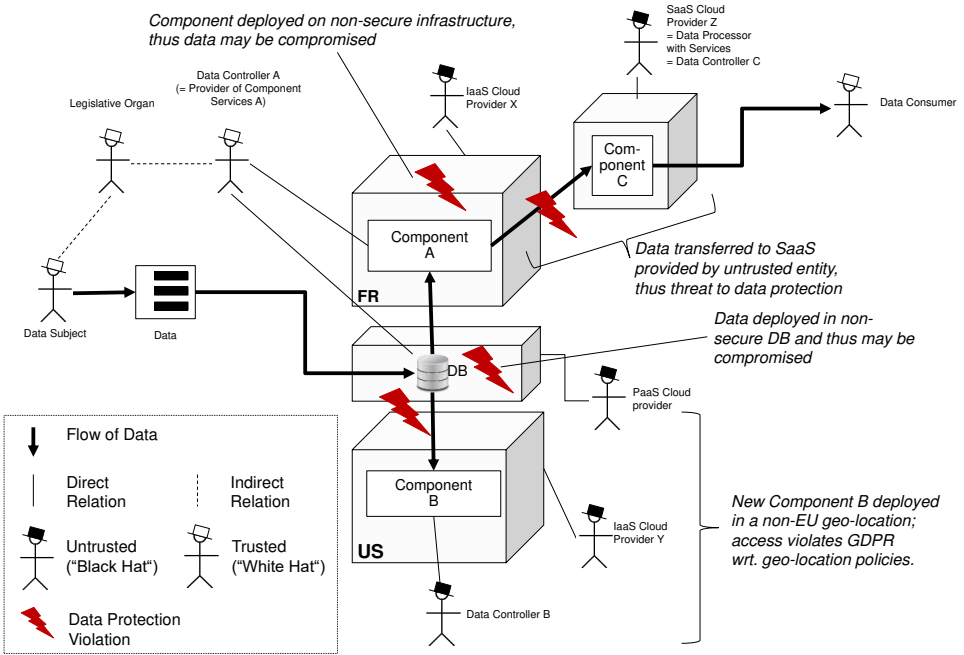


Fig. 1: An industrial cloud scenario

While the scenario abstracts from specific applications, it has been validated to reflect the typical data protection concerns of practical cloud systems.

In Fig. 1 we see a typical cloud scenario where multiple parties are involved. In this scenario, personal data about individual users (Data Subject⁴) are captured by a company (Data Controller A), with explicit consent of the users and under legislative control. The company stores the data in an unencrypted database (DB) operated by a PaaS (Platform as a Service) provider and deploys its application (Component A) using the infrastructure (including a virtual machine in which Component A runs and a physical machine in which this virtual machine runs) provided by IaaS (Infrastructure as a Service) provider X. Another actor (Data Consumer) uses an application (Component C) to communicate with Component A and get access to the data. Component C is run by SaaS (Software as a Service) provider Z. Another company (Data Controller B) uses an application (Component B) run on the infrastructure of IaaS provider Y that accesses the same database DB.

It is important to note that Data Controller A and Data Consumer are trusted by Data Subject, as shown by their white hats⁵ in the figure. Nevertheless, as the data traverse between the trusted parties, several untrusted actors (PaaS provider, IaaS provider X, SaaS provider

⁴ The names of the roles are based on the terminology of the GDPR

⁵ Note that here white and black hats encode trustworthiness and do not refer to white-hat / black-hat hackers.

Z) may get unauthorized access to the data along the way. Moreover, other cloud tenants using the same public database offering (DB) can also get access to the data. The access by Component B also poses a further problem because Component B is hosted in the US, but European regulations prohibit processing personal data of EU citizens outside the EU.

As we can see, a cloud setup can be complex and dynamic, with many different socio-technical interactions, posing a wide-ranging set of threats to data protection.

3 Related work

Multi-layer cloud meta-models Shao et al. proposed a model-based approach for cloud monitoring [Sh10]. Their approach, called RMCM (Runtime Model for Cloud Monitoring), differentiates between three roles: cloud operator, service developer, and end user. Monitoring is used to adjust a model of the cloud consisting of four layers: (i) infrastructure, (ii) middleware, (iii) application, and (iv) interactions between the roles and the cloud. The models inside the layers are not specified; the focus of that work was rather on how the different layers can be monitored.

The NIST Cloud Computing Reference Architecture [Na11] also defines a kind of model of cloud computing. It mainly focuses on the definition of roles (cloud service provider, cloud service consumer, cloud broker, cloud auditor, cloud carrier) and the associated activities. In connection with the service provisioning and orchestration activities, the model foresees three layers: (i) physical resource layer, (ii) resource abstraction and control layer, (iii) service layer. However, the model does not include the actual components that make up a cloud system. Privacy is mentioned as an important requirement, but no details are provided.

Marquezan et al. [Ma14] developed a conceptual model of all the key entities relevant for adaptations in the cloud, based on a survey of the literature, discussions with industrial partners, and the analysis of commercial solutions. The resulting model consists of four layers: (i) The *physical layer* contains the physical equipment, like servers. (ii) The *virtualization layer* consists of virtual resources, like virtual machines. (iii) The *logical application architecture layer* is composed of the software components needed to support the logical architecture of the application, like application servers. (iv) The *application business logic layer* contains the components actually implementing the business logic of the application. Since the model of [Ma14] focuses on the adaptation possibilities of the cloud, it contains beside the actual cloud entities also the adaptation techniques (e.g., load balancing) in the same model. On the other hand, data protection was not in focus, so that the model does not support reasoning about pieces of data, actors, and relations among them.

Meta-models for cloud applications Several works focused on the highest cloud layer, i.e., the application layer. Chapman et al. addressed the problem of defining the architecture of multi-cloud software systems and proposed a model-based approach [Ch12]. The resulting

models target the application layer and only include some references to lower layers. The language defined in that paper relies on the Open Virtualization Format (OVF), but extends it with several new concepts, e.g., for elasticity rules. Nagel et al. presented a meta-model supporting the adaptation of business processes realized as cloud services [Na12]. The meta-model focused on the business process models, their adaptation and their mapping on cloud services, and abstracted from the technical infrastructure underlying the cloud services. However, some important threats to data protection relate to the underlying technology stack.

Heinrich proposed an architectural run-time model-based approach for analyzing cloud applications [He16]. That work uses a megamodel to connect a design-time architecture model, an architectural run-time model, and the actual implementation to enable correct interpretation of monitoring events and keeping the run-time model in sync with the state of the program. That method could be combined with our approach to provide effective monitoring for analyzing data protection issues. Bergmayr et al. proposed the Cloud Application Modeling Language (CAML) to facilitate expressing cloud-based deployments directly in UML [Be14]. Later on, that approach has also been extended to enable application provisioning by means of TOSCA [Be16]. That approach underlines the applicability of UML for modeling not only applications but also cloud environments. However, the approach lacks support for several concepts related to data protection, like the trust among stakeholders.

Model-based approaches for cloud security and privacy Similarly to our approach, the work of Schmieders et al. also applies model-based adaptive methods to data protection in the cloud [SMP15a, SMP15b]. That work, however, is limited to one specific type of privacy goals: geo-location constraints. Our work, in contrast, addresses data protection goals in a much broader sense.

Kritikos and Massonet proposed a domain-specific modeling language for modeling security aspects in cloud computing [KM16]. This includes security controls, security properties, security metrics, and security capabilities. In contrast, our work focuses on modeling the cloud – in particular, the possible attack surfaces and the configurations that may lead to data protection violations.

Other cloud models The MODACLOUDS project proposed a model-based approach to design, deploy, and maintain cloud applications [Ar12]. To enable multi-cloud deployments and migrations between clouds, MODACLOUDS adopted a model-driven approach: a Cloud-enabled Computation Independent Model (CIM) is transformed semi-automatically via a Cloud-Provider Independent Model (CPIM) to a Cloud-Provider Specific Model (CPSM). Unfortunately, the approach gives little support on what exactly needs to be in the models.

Industry cooperation resulted in TOSCA, a standard for cloud deployment topology and orchestration specification [OA13]. TOSCA advocates a generic template-based specification approach, in which services are specified in terms of node templates and relationship templates, and deployed by applying a deployment plan. While the generality of TOSCA allows the specification of any service, it lacks support for explicitly reasoning about data-protection-relevant aspects, like the location of data. Lejeune et al. took an even more generic approach and developed an abstract service model [LAL17]. This allows to describe any cloud service in terms of used services and components and offered SLAs. However, the abstract model hides the underlying technical components and their interrelations, which can be important for assessing the fulfillment of data protection policies.

Model-based approaches have also been proposed for evaluating the design of cloud systems. Relating to privacy, Ahmadian et al. devised a methodology based on the concept of Privacy Level Agreements [Ah17]. Such approaches are orthogonal to ours and an interesting path for future research is to investigate the integration of design models and run-time models.

Recently, we have introduced a method for detecting violations of data protection policies in cloud systems at run time [SMM17]. Our method is based on identifying “risk patterns” – configurations that would lead to unacceptably high risks of data protection violations – in a model of the cloud. Thus, having an appropriate run-time model of the cloud is a prerequisite for the method to work.

4 Design considerations for the meta-model

Based on the requirements from the example of Sec. 2 and the analysis of related work in Sec. 3, the cornerstones for an appropriate meta-model can be established as follows.

- Often, data protection violation is not confined to a specific cloud layer, but arises from the *interplay of entities belonging to different cloud layers*. For instance, a data record accessed by an application hosted by a virtual machine on a physical machine might constitute a data protection violation because it potentially allows the administrator of the physical machine to access the data. However, if the data are encrypted, or the application is protected by appropriate access control mechanisms, or the physical machine supports secure hardware enclaves, data protection can still be ensured [MM17]. Therefore, it is important to model the attributes of as well as interactions among entities on different cloud layers.
- Beyond technical entities like physical and virtual machines, also *actors* and their attributes and relationships need to be modeled. For example, if the data belonging to actor A can get accessed by actor B, this may or may not constitute a data protection violation depending on whether A *trusts* B or not.
- In existing cloud models, *data* was also missing. For reasoning about data protection, we need to add support for modeling data. This is not to be confused with “data

modeling,” which is about modeling the logical concepts captured by the data. For our purposes, other attributes of data objects matter, like their sensitivity (e.g., personal data) and where they are stored and processed.

- *UML* provides sufficient expressiveness to model cloud systems, as shown by [Be14]. Although there are also alternatives, we stick to UML because of its wide-spread adoption and the available tool support. More specifically, we will use UML class diagrams for the meta-model and object diagrams for the model of the cloud.
- Beyond the mere detection of data protection violations, the model should also support finding the right mitigation action. For this purpose, it is vital to also model the possible *data protection mechanisms* and their impact on security attributes. Moreover; there may be multiple mechanisms that can be used to achieve the same security goal; in this case it is useful to select the one that has the smallest impact on other goals like performance or costs. For supporting such decisions, it is important to also model the *impact* of the available security mechanisms on those other *goals*.
- To be useful, the cloud meta-model needs to mirror the used technologies. However, there are several different technologies used in cloud computing and the technologies are also subject to change. For example, some cloud systems use virtual machines, others use containers, and a combination of the two is also possible. Therefore it is not feasible to strive for a cloud meta-model that is generic enough to capture all possible technical cloud realizations and at the same time also detailed and specific enough to allow reasoning about data protection impacts of a given cloud configuration. Rather, we argue that the exact cloud meta-model has to be created during system design, taking into account the specific technologies that are foreseen for the given system. We support that process by identifying the sorts of entities that need to be modeled, resulting in a framework for the meta-model, and giving examples of the modeling of entities that play an important role in most cloud systems.

5 Proposed meta-model

Based on the considerations of Sec. 4, we now propose a cloud meta-model for data protection. This meta-model can be seen as an extension of the existing multi-layer models of cloud systems discussed in Sec. 3. We extend those models with several concepts that are vital for data protection, like explicitly modeling data and actors.

5.1 Structure of the meta-model

Independently from the used technologies, our meta-model framework is structured into four high-level packages as shown in Fig. 2 and explained below:

Assets: configuration of the cloud, including all the physical and virtual entities that are

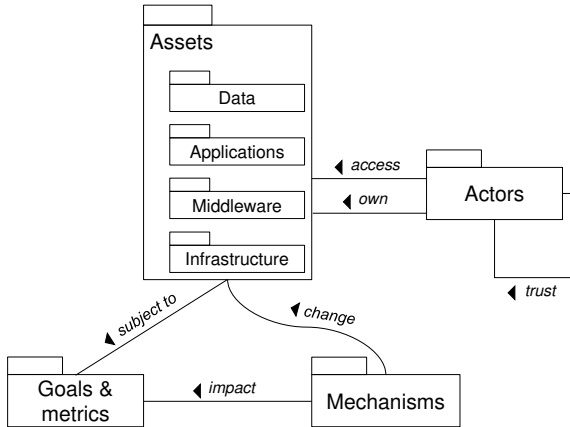


Fig. 2: The structure of the proposed meta-model

important for data protection

Actors: stakeholders and their roles relevant for data protection in the cloud at run time

Goals & metrics: non-functional properties that the system should fulfill

Mechanisms: possibilities for adaptation, including structural changes and changes to specific attributes

Fig. 2 also highlights the most important relations between the packages. In particular, assets may be owned and/or – independently from that – accessed by actors. In terms of relations among actors, *trust* is of special importance. We use a white-list approach to trust, i.e., every trust relation must be explicitly established (e.g., by means of a contract). Also, trust relations can be limited to specific types of actions on specific data. Goals & metrics relate to some assets, e.g., by specifying a response time constraint on an application. The mechanisms change some assets and impact the goals & metrics. For example, encrypting a piece of data changes an attribute of that data object and it has some given positive impact on confidentiality but negative impact on performance.

The Assets package is further subdivided into Infrastructure, Middleware, Applications, and Data. Traditionally, cloud models include only the first three of those, but we also included data because of its obvious importance to data protection. Data is the primary asset at risk, but the other layers are also important because they act as additional attack surface.

Altogether, the proposed meta-model consists of seven *sub-models*: the four packages within Assets, plus Actors, Goals & metrics, and Mechanisms. These sub-models are detailed next.

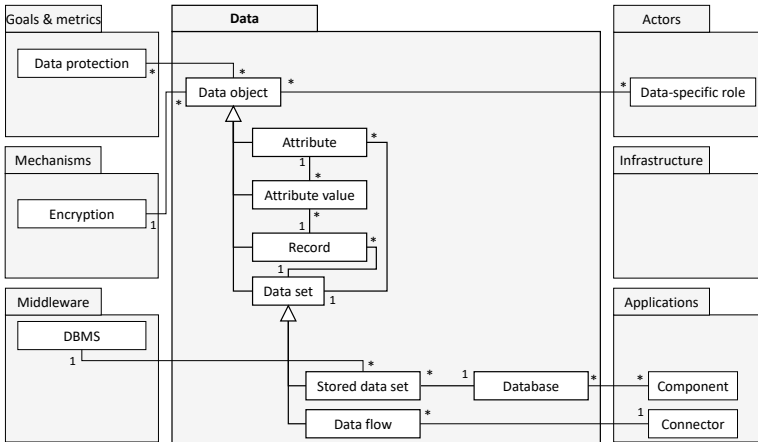


Fig. 3: The sub-model Data and its relations to the other sub-models

5.2 Contents of the sub-models

The structure of the meta-model presented above is technology-independent and hence it can be expected to remain stable for a wide range of cloud systems. In contrast, the contents of the sub-models may depend on specific technologies and hence may differ from system to system. For example, the contents of the Data sub-model may depend on whether structured, semi-structured, or non-structured data are used; the contents of the Infrastructure sub-model will be different depending on whether virtual machines or containers are used etc. Therefore in the following we show examples of what the contents of the sub-models may be. Still, we try to be generic enough so that these models likely apply to many different cloud systems with no or little modification and can be used as starting point for modeling other cloud systems as well.

The Data sub-model shown in Fig. 3 is based on a relational model (like in [Ri13]) but abstracts from details that are not important for us (e.g., domains of attributes) and adds others that are important (e.g., relating to the storage of data). The smallest unit of data is the “Attribute value,” corresponding to a cell in a relational table. Attributes (columns) and Records (rows) contain multiple Attribute values; a Data set (table) contains multiple Attributes and multiple Records. A Data set can either be stored or transferred, represented by the entities Stored data set and Data flow inheriting from Data set. A database consists of multiple stored data sets. A Stored data set can be stored either in a local Database associated with a specific application Component or using a database management system (DBMS) provided by the Middleware. Attribute Values, Attributes, Records and Data Sets are all considered “Data objects.” For each Data object there can be multiple Data protection requirements. Data objects may be accessed by different Actors in Data-specific roles. Encryption can be used to alter the security attributes of a Data object.

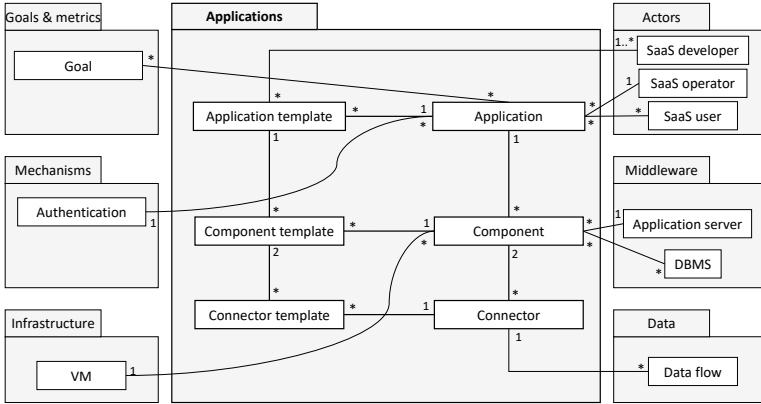


Fig. 4: The sub-model Applications and its relations to the other sub-models

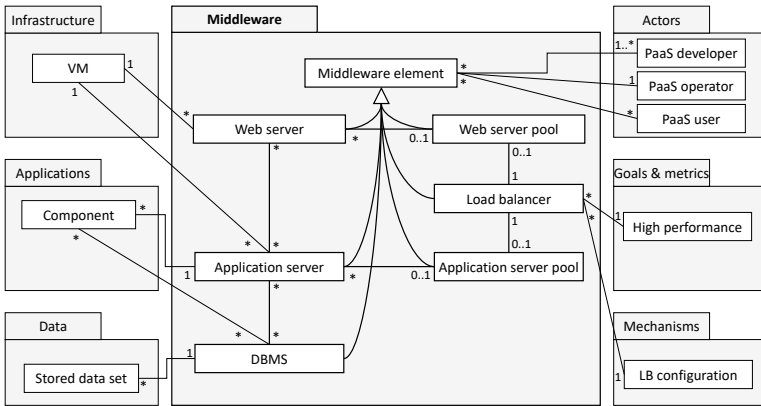


Fig. 5: The sub-model Middleware and its relations to the other sub-models

Applications (Fig. 4) consist of multiple Components that are linked by Connectors. The logical structure of an Application is defined by an Application template, from which the specific Application, Component, and Connector instances are derived and scaled as necessary. SaaS (Software as a Service) developers work with Application templates. Applications are managed by SaaS operators and used by SaaS users. Certain Goals may apply to an Application and Authentication mechanisms can be turned on or off for an Application. Each Component is deployed in a VM (Virtual Machine) and controlled by an Application server. A Connector may accommodate Data flows.

The Middleware sub-model (Fig. 5) supports multi-tier web applications with Web server, Application server, and DBMS entities. Web servers and Application servers can be clustered into Web server pools and Application server pools, respectively, which can be associated to a Load balancer. All these are Middleware elements, which are created, operated, and used

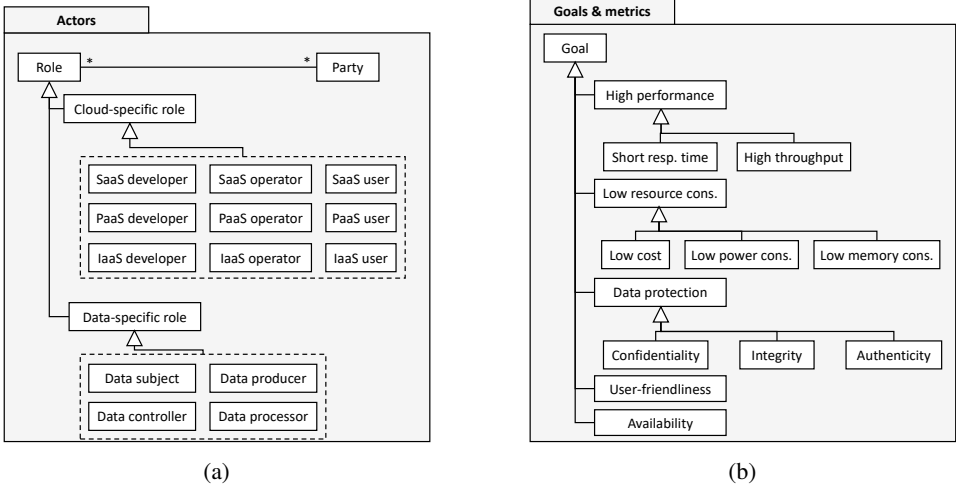


Fig. 7: The sub-models (a) Actors and (b) Goals & metrics. The dashed boxes indicate that all contained classes inherit from the same superclass

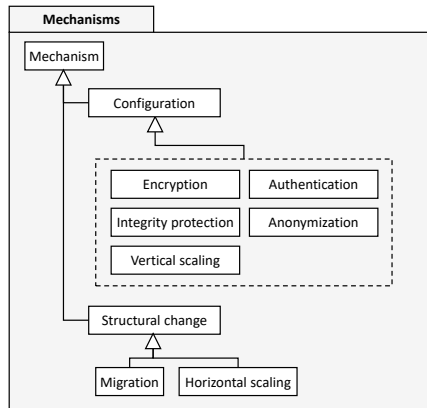


Fig. 8: The sub-model Mechanisms

the system must fulfill. This includes High performance, Low resource consumption, Data protection, User-friendliness, and Availability. These high-level goals can be decomposed into more specific goals; in particular, Data protection is decomposed into Confidentiality, Integrity, and Authenticity. Goals may relate to different Assets. In particular, Data protection goals relate to Data objects. A Goal is posed by a Party in a specific Role and may be impacted by different Mechanisms.

As shown in Fig. 8, Mechanisms can be of two kinds: Configuration or Structural change. Configuration may mean that a feature such as Encryption within an application Component

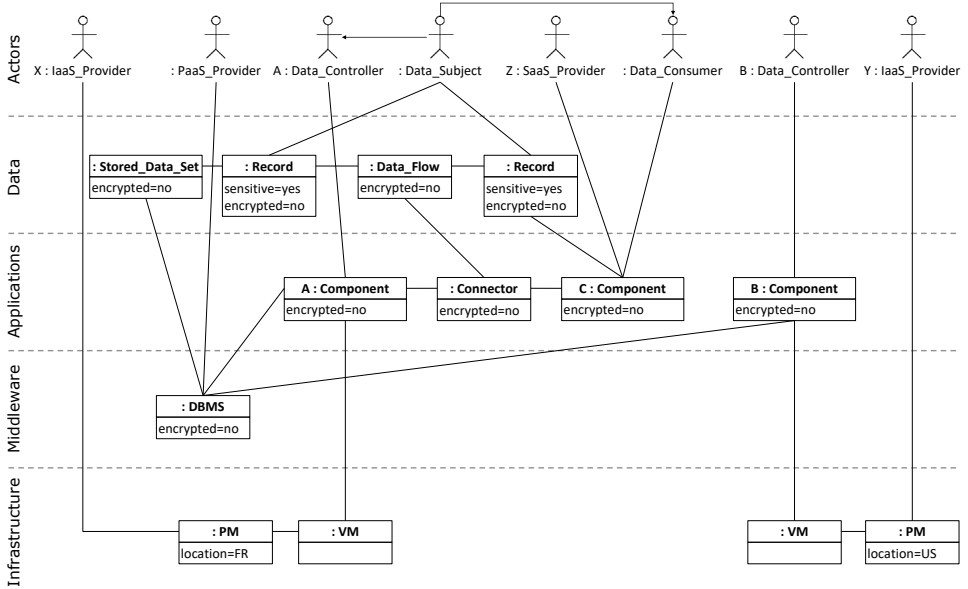


Fig. 9: Object model resulting from applying the meta-model of Sec. 5 to the example of Sec. 2

is switched on or off or some more subtle parameter change, e.g., the key length of the encryption algorithm is changed. Similar mechanisms are Authentication, Integrity protection (e.g., using cryptographic hashes), Anonymization of data, or Vertical scaling of VMs. Structural changes, on the other hand, create or remove entities or change interconnections among entities. In particular, Horizontal scaling of VMs creates or removes VM instances, whereas Migration of VMs between PMs changes the interconnection between the affected PMs and VM. These are powerful mechanisms to achieve several different Goals, including Data protection (e.g., by means of co-locating Components so that the Data flow between them does not have to traverse the network).

6 Application to the industrial cloud scenario

To validate the applicability of the proposed meta-model, we revisit the industrial cloud scenario from Sec. 2. We modeled that example using the proposed meta-model. An excerpt of the resulting model is shown in Fig. 9, consisting of the four Assets and the Actors sub-models and their interconnections.

The resulting model captures all details that are necessary to automatically identify all data protection issues that were previously identified by experts in Fig. 1. For example, Fig. 9 shows that the data subject trusts data controller A and the data consumer, but all other

parties are not trusted. We can see the different actors that can access components A, B, and C, the data record containing sensitive data of the data subject, the joint use of the DBMS, as well as the used infrastructure elements and the associated providers. On the basis of this model, the problems marked in Fig. 1 can be identified using the run-time model (instance of the meta-model). The run-time model is shown as an object model in Fig. 9. The model shows that the PaaS provider has access to the DBMS, which contains a sensitive data record of a data subject in an unencrypted form, and the PaaS provider is not trusted by the data subject. This clearly constitutes a high risk of data protection violation. Such violations can be found automatically based on a pre-defined set of forbidden subgraphs using graph pattern matching algorithms [Ma11], as discussed in [SMM17].

Once a data protection violation has been detected, the model may also be used to investigate the available mechanisms. For each of them, it can be checked whether (i) it is applicable in the given situation, (ii) it would break the forbidden subgraph found in the run-time model, and (iii) it would not introduce another forbidden subgraph. In our example, turning on encryption would be such a mechanism. If multiple appropriate mechanisms are available, the one with the best (i.e., least disadvantageous) impact on the other goals is selected and applied, thus automatically solving the identified problem. If no appropriate adaptation action can be found, a human operator needs to be alerted. A more detailed discussion on the process of applying the run-time model for identifying data protection violations can be found in the companion paper [SMM17].

7 Conclusions and future work

In this paper, we argued that data protection in a dynamic cloud setting should be addressed by automatically configuring the system at deployment time and dynamically re-configuring it at run time. A central element of such an approach is a run-time model of the relevant entities. We have presented a way of determining and structuring the corresponding meta-model. Similarly to previous approaches to cloud modeling, cloud assets like infrastructure, middleware, and applications need to be modeled. However, for data protection purposes, more is needed, and hence we introduced further sub-models for data, actors, goals, and mechanisms. We have presented an initial validation of our meta-model by applying it to an industrial case study and found that the resulting model contains all necessary information for detecting and automatically mitigating data protection violations.

The next step will be to implement the proposed model together with the reasoning technique for finding risk patterns described in [SMM17]. This can then be used to carry out a more realistic evaluation of the proposed approach, also showing how much the meta-models of the run-time model of different cloud systems differ from each other. Moreover, it should be investigated how other, more sophisticated mechanisms to protect privacy, such as controlled interaction [BMZ16] can be incorporated.

Acknowledgments. This work received funding from the European Union’s Horizon 2020 research and innovation programme under grant 731678 (RestAssured). Useful discussions with project partners are gratefully acknowledged.

References

- [Ah17] Ahmadian, Amir Shayan; Strüber, Daniel; Riediger, Volker; Jürjens, Jan: Model-Based Privacy Analysis in Industrial Ecosystems. In: Proceedings of the 13th European Conference on Modelling Foundations and Applications. pp. 215–231, 2017.
- [Am12] Amoui, Mehdi; Derakhshanmanesh, Mahdi; Ebert, Jürgen; Tahvildari, Ladan: Achieving dynamic adaptation via management and interpretation of runtime models. *Journal of Systems and Software*, 85(12):2720–2737, 2012.
- [Ar12] Ardagna, Danilo; Di Nitto, Elisabetta; Casale, Giuliano; Petcu, Dana; Mohagheghi, Parastoo; Mosser, Sébastien; Matthews, Peter; Gericke, Anke; Ballagny, Cyril; D’Andria, Francesco et al.: MODAClouds: A model-driven approach for the design and execution of applications on multiple clouds. In: Proceedings of the 4th International Workshop on Modeling in Software Engineering. IEEE Press, pp. 50–56, 2012.
- [Be14] Bergmayr, Alexander; Troya, Javier; Neubauer, Patrick; Wimmer, Manuel; Kappel, Gerti: UML-based Cloud Application Modeling with Libraries, Profiles, and Templates. In: *CloudMDE@MoDELS*. pp. 56–65, 2014.
- [Be16] Bergmayr, Alexander; Breitenbücher, Uwe; Kopp, Oliver; Wimmer, Manuel; Kappel, Gerti; Leymann, Frank: From Architecture Modeling to Application Provisioning for the Cloud by Combining UML and TOSCA. In: Proceedings of the 6th International Conference on Cloud Computing and Services Science. pp. 97–108, 2016.
- [BKW14] Busch, Marianne; Koch, Nora; Wirsing, Martin: SecEval: An Evaluation Framework for Engineering Secure Systems. In: *Modellierung*. pp. 337–352, 2014.
- [BMZ16] Biskup, Joachim; Menzel, Ralf; Zarouali, Jaouad: Controlled Management of Confidentiality-Preserving Relational Interactions. In: *International Workshop on Data Privacy Management*. pp. 61–77, 2016.
- [Ch12] Chapman, Clovis; Emmerich, Wolfgang; Márquez, Fermin Galán; Clayman, Stuart; Galis, Alex: Software Architecture Definition for On-demand Cloud Provisioning. *Cluster Computing*, 15(2):79–100, 2012.
- [Co16] Council of the European Union: , General Data Protection Regulation. <http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf>, 2016.
- [He16] Heinrich, Robert: Architectural run-time models for performance and privacy analysis in dynamic cloud applications. *ACM SIGMETRICS Performance Evaluation Review*, 43(4):13–22, 2016.
- [KM16] Kritikos, Kyriakos; Massonet, Philippe: An integrated meta-model for cloud application security modelling. *Procedia Computer Science*, 97:84–93, 2016.
- [LAL17] Lejeune, Jonathan; Alvares, Frederico; Ledoux, Thomas: Towards a generic autonomic model to manage Cloud Services. In: *7th International Conference on Cloud Computing and Services Science*. 2017.

- [Ma11] Mann, Zoltán Ádám: Optimization in computer engineering – Theory and applications. Scientific Research Publishing, 2011.
- [Ma14] Marquezan, Clarissa Cassales; Wessling, Florian; Metzger, Andreas; Pohl, Klaus; Woods, Chris; Wallbom, Karl: Towards exploiting the full adaptation potential of cloud applications. In: Proceedings of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems. pp. 48–57, 2014.
- [Ma15a] Mann, Zoltán Ádám: Approximability of virtual machine allocation: much harder than bin packing. In: Proceedings of the 9th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications. pp. 21–30, 2015.
- [Ma15b] Mann, Zoltán Ádám: Modeling the virtual machine allocation problem. In: Proceedings of the International Conference on Mathematical Methods, Mathematical Models and Simulation in Science and Engineering. pp. 102–106, 2015.
- [MM17] Mann, Zoltán Ádám; Metzger, Andreas: Optimized Cloud Deployment of Multi-tenant Software Considering Data Protection Concerns. In: 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. pp. 609–618, 2017.
- [Mo13] Modi, Chirag; Patel, Dhiren; Borisaniya, Bhavesh; Patel, Avi; Rajarajan, Muttukrishnan: A survey on security issues and solutions at different layers of Cloud computing. The Journal of Supercomputing, 63(2):561–592, 2013.
- [Na11] National Institute of Standards and Technology: , NIST Cloud Computing Reference Architecture. NIST Special Publication 500-292, <https://www.nist.gov/publications/nist-cloud-computing-reference-architecture>, 2011.
- [Na12] Nagel, Benjamin; Gerth, Christian; Yigitbas, Enes; Christ, Fabian; Engels, Gregor: Model-driven specification of adaptive cloud-based systems. In: Proceedings of the 1st International Workshop on Model-Driven Engineering for High Performance and Cloud computing. 2012. article 4.
- [OA13] OASIS: , Topology and Orchestration Specification for Cloud Applications (TOSCA) v1.0. <http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>, 2013.
- [Ri13] Ristić, Sonja; Aleksić, Slavica; Čeliković, Milan; Luković, Ivan: An EMF Ecore based relational DB schema meta-model. In: Proceedings of the 6th International Conference on Information Technology ICIT. 2013.
- [Sh10] Shao, Jin; Wei, Hao; Wang, Qianxiang; Mei, Hong: A runtime model based monitoring approach for cloud. In: IEEE 3rd International Conference on Cloud Computing. pp. 313–320, 2010.
- [SMM17] Schoenen, Stefan; Mann, Zoltán Ádám; Metzger, Andreas: Using Risk Patterns to Identify Violations of Data Protection Policies in Cloud Services. In: 13th International Workshop on Engineering Service-Oriented Applications and Cloud Services. 2017.
- [SMP15a] Schmieders, Eric; Metzger, Andreas; Pohl, Klaus: Architectural runtime models for privacy checks of cloud applications. In: Proceedings of the Seventh International Workshop on Principles of Engineering Service-Oriented and Cloud Systems. pp. 17–23, 2015.
- [SMP15b] Schmieders, Eric; Metzger, Andreas; Pohl, Klaus: Runtime model-based privacy checks of big data cloud services. In: International Conference on Service-Oriented Computing. pp. 71–86, 2015.

Nutzung von Bilddatenbanken zur Erstellung von Symbolen für graphische Modellierungssprachen

Ralf Laue¹

Abstract: Um graphische Modellierungssprachen effektiv für die Kommunikation zu nutzen, müssen die verwendeten Symbole verständlich und gut erlernbar sein. Nachdem festgestellt wurde, dass dies bei bisher entwickelten Sprachen häufig nicht der Fall war, wurde von mehreren Autoren als Alternative die Methode der kollektiven Symbolerstellung untersucht. Bei dieser Methode werden Symbolvorschläge von potentiellen Nutzern der Modellierungssprache gesammelt. Dieser Beitrag untersucht, ob dieses aufwendige Verfahren durch die Suche nach Symbolen in Bilddatenbanken ersetzt werden kann. Nachdem diese Frage bejaht werden kann, wird eine Methode zur Erstellung von Notationen unter Einbeziehung einer Suche in Bilddatenbanken vorgeschlagen.

Keywords: kollektive Symbolerstellung; konkrete Syntax; Notation; graphische Modellierungssprache; domänenspezifische Modellierungssprache

1 Einführung

Graphische Modelle werden häufig für die Kommunikation zwischen Systementwicklern und Kundenvertretern genutzt. Diese Modelle können ihren Zweck, die Kommunikation zu unterstützen, nur erfüllen, wenn sie von allen Beteiligten gleichermaßen verstanden und interpretiert werden können. Während bei den Systementwicklern solide Kenntnisse der Modellierungssprache vorausgesetzt werden können, ist dies bei den Kunden oft nicht der Fall. Folglich können vor allem solche Notationen gewinnbringend in Diskussionen mit Kunden eingesetzt werden, die ohne größere Anwenderschulung verständlich sind.

Wichtig ist daher die Frage nach Verständlichkeit und Erlernbarkeit der Symbole in graphischen Modellierungssprachen. Ein Blick auf die Symbole der im Requirements Engineering verbreiteten Modellierungssprache ι^* [Yu11] und die Notation von DMN-Entscheidungsdiagrammen [Ob16] in Abb. 1 und 2 legt jedoch nahe, dass diese Kriterien beim Notationsentwurf derzeit noch keine vorrangige Rolle spielen. Die Folge ist, dass die Arbeit mit Modellierungsneulingen weniger effizient verläuft, als dies bei sorgfältigem Notationsdesign der Fall sein könnte.

Mit der zunehmenden Verbreitung domänenspezifischer graphischer Modellierungssprachen in den letzten Jahren liegt der Entwurf der Sprache und ihrer Notation häufig nicht mehr bei

¹ Westsächsische Hochschule Zwickau, Fachgruppe Informatik, Dr.-Friedrichs-Ring 2a, 08056 Zwickau ralf.laue@fh-zwickau.de

Akteur	Agent	Rolle	Position	Ziel	unscharfes Ziel	Aufgabe	Ressource	Annahme	Annahme (alternative Darstellung)

Abb. 1: Symbole der i^* -Notation

Entscheidung	Geschäftswissen	Wissensquelle	Eingangsdaten

Abb. 2: Symbole in DMN-Entscheidungsdiagrammen

Akteur	Agent	Rolle	Position	Ziel	unscharfes Ziel	Aufgabe	Ressource	Annahme

Abb. 3: i^* -Symbole unter Berücksichtigung der „Physics of Notations“

Standardisierungskomitees, sondern wird im Entwicklungsprojekt durchgeführt. Dadurch erhöht sich noch einmal der Bedarf an einer geeigneten Methode zur Erstellung der Notation.

Dass ikonische Symbole besser verständlich sind als arbiträre Formen wie in Abb. 1 und 2 zeigten Siau und Tian [ST09] anhand der Sprache UML. In ähnliche Richtung weisen die Ergebnisse von Weitlaner et al. Sie analysierten in [WGK13] die Verständlichkeit von Geschäftsprozessmodellen in verschiedenen Modellierungssprachen für eine Zielgruppe mit überwiegend geringen Modellierungskennntnissen. Das Ergebnis war, dass nichtstandardisierte comicartige Darstellungen besser verständlich sind als die üblicherweise verwendeten Modellierungssprachen BPMN, EPK und UML-Aktivitätsdiagramme.

Ein Rahmenwerk zur Bewertung der Eignung graphischer Modellierungssprachen liefert Moodys “Physics of Notations” [Mo09]. Nach diesem Rahmenwerk wurde in [MHM10, MHM09] die von Yu [Yu11] vorgeschlagene i^* -Notation bewertet und die in Abb. 3 gezeigten Symbolalternativen vorgeschlagen. Vorteile gegenüber der Original-Notation von Yu sind sofort erkennbar: Erstens sind die Symbole in Abb. 3 klarer voneinander unterscheidbar. Und zweitens sind die Symbole (zumindest die meisten) durch die bildliche Darstellung leichter zu interpretieren. So weckt das „Aufgabe“-Symbol die Assoziation mit einem Klebezettel, auf dem zu erledigende Aufgaben notiert sind.

2 Kollektive Symbolerstellung

Die in Abb. 3 dargestellten Symbole wurden von Moody als Illustration der Anwendung seiner Prinzipien aus der „Physics of Notations“ entworfen. Moody erhob nicht den Anspruch, dass es sich um die optimale Wahl der Symbole handelt. Um mögliche Verbesserungen zu



Abb. 4: Prozess der kollektiven Symbolerstellung nach [Ge12, Ca13]

finden, müssten die potentiellen Nutzer, ihre Gedankenwelt und ihre Assoziationen zu den Konzepten der Modellierungssprache untersucht werden. Forschungen in dieser Richtung unternahm Genon et al. [Ge12, Ca13]. Sie baten Modellierungs-Neulinge, Symbole für die Darstellung der in der Sprache i^* vorhandenen Konzepte zu entwerfen.

Ein solcher Ansatz hat sich auf anderen Gebieten bewährt: Howell und Fuchs [HF68] untersuchten am Beispiel von militärischen Piktogrammen die Entwicklung von effizienten Symbolen für die visuelle Kommunikation. Ihre Methode der kollektiven Symbolerstellung (engl. *sign production method* oder *stereotype production method*) sieht vor, dass Personen aus dem Kreise der potentiellen Nutzer einer symbolischen Notation entsprechend ihrer Intuition Vorschläge für die Gestaltung der Symbole machen. Diejenigen Symbole, die von einem hohen Prozentsatz der befragten Personen vorgeschlagen werden, werden als *Stereotyp* der Intuition der Benutzergruppe angenommen.

In der Folge wurde die Methode in den verschiedensten Bereichen genutzt, etwa für die Erstellung von Piktogrammen auf Mobiltelefonen [SZ08], für die Information von Bahnreisenden [ZB83] oder auf dem Bedienfeld von Fotokopierern [Ho91]. In verschiedenen Bereichen konnte gezeigt werden, dass aus dem Kreis der potentiellen Nutzer vorgeschlagene Symbole besser verstanden werden als Symbole, die von Einzelpersonen, Firmen oder Normierungsgremien erstellt werden.

Genon et al. [Ge12, Ca13] nutzten die Methode der kollektiven Symbolerstellung für die Symbole der Modellierungssprache i^* . Ihr Vorgehen ist in Abb. 4 skizziert. Befragt wurden 104 Studenten der Wirtschaftswissenschaften an der Universität Namur, Belgien. In einem ersten Experiment wurden mittels kollektiver Symbolerstellung Stereotypen für die Darstellung der einzelnen i^* -Konzepte gewonnen. Abb. 5 zeigt diese Stereotypen², also diejenigen Symbole, die von den Probanden am häufigsten vorgeschlagen wurden.

Akteur	Agent	Rolle	Position	Ziel	unscharfes Ziel	Aufgabe	Ressource	Annahme

Abb. 5: Symbole der Stereotypen (aus [Ge12, Ca13])

Es schloss sich ein zweites Experiment an, in dem 30 weiteren Studenten des selben

² In den Abbildungen sind nur die Symbole für Knoten im Modellgraph gezeigt. Darüber hinaus wurden im Experiment auch Symbole für drei Konzepte erfragt, die im i^* -Modell Kanten zwischen diesen Knoten sind.

								
Akteur	Agent	Rolle	Position	Ziel	unschar- fes Ziel	Aufgabe	Ressource	Annahme

Abb. 6: Symbole der Prototypen (aus [Ge12, Ca13])

Studienganges die im ersten Experiment erstellten Symbole vorgelegt wurden. Die Probanden sollten daraus die geeignetsten Symbole benennen. Im Ergebnis dieser Abstimmung wurde ein weiterer Symbolsatz (in [Ge12, Ca13] als „Symbol-Prototypen“ bezeichnet) gewonnen. Dieser ist in Abb. 6 dargestellt. Während also die Stereotypen in Abb. 5 die am häufigsten gezeichneten Symbole zeigen, sind die Prototypen in Abb. 6 diejenigen Symbole, die am besten bewertet wurden. Dieses Vorgehen ist sinnvoll, weil in früheren Experimenten zur kollektiven Symbolerstellung nicht selten auch solche Symbole die größte Zustimmung erhalten haben, die von nur einem Probanden vorgeschlagen wurden [Jo83].

Schließlich wurden in einem dritten Experiment mit 65 Studenten der Haute Ecole Robert Schuman-Libramont sowie der Haute Ecole Marie HAPS-Bruxelles den Probanden sämtliche bisher genannten Symbolzusammenstellungen vorgelegt. Zu jedem Symbol erhielten die Probanden eine Liste mit den Namen und Erklärungen der τ^* -Konzepten. Die Aufgabe bestand darin, zu jedem Symbol herauszufinden, welches Konzept es darstellen soll. Während für die Standard-Notation aus Abb. 1 die Korrektheit der Antworten bei 17,4% lag (was bei neun Symbolen nicht viel besser als zufälliges Raten ist), lagen die Prozentsätze für richtige Antworten bei der alternativen Notation (Abb. 3) bei 38,9%. Die Stereotypen (Abb. 5) erreichten den mit 67,4% höchsten Prozentsatz; die Symbol-Prototypen (Abb. 6) wurden zu 41,7% richtig interpretiert.

Neben Genon et al. gibt es zwei weitere Arbeiten, die das Verfahren der kollektiven Symbolerstellung für die Symbole einer graphischen Modellierungssprache nutzten. Arning und Zieffle [AZ09] wendeten das Verfahren auf die Geschäftsprozessmodellierungssprache C3 an. Abb. 7 zeigt diejenigen Symbolideen, die in [AZ09] mit einem Bild abgebildet wurden³. Kouhen et al. [eGD14, Ko15] erstellten alternative Symbole zu grundlegenden UML-Notationselementen. Auch Kouhen et al. stellten in Tests eine erheblich bessere Verständlichkeit der durch die Zielgruppe entworfenen Symbole im Vergleich zur Standard-UML-Notation fest. Die in [Ko15] gefundenen Vorschläge für UML-Modellelemente zeigt Abb. 8.

3 Recherche in Bilddatenbanken

Schritt 1 im beschriebenen Verfahren - das Zeichnen und Zusammentragen von Symbolvorschlägen - ist mit einigem Aufwand verbunden. Dieser Aufwand steigt nochmals

³ mit Ausnahme des trivialen Vorschlags „Rechteck“ zur Bezeichnung einer Aktivität












										
Aktivität	Werkzeug	Schwachstelle	Entscheidung (1) und (2)	Kontrollfluss	Informationsfluss	Information	synchrone Zusammenarbeit	gleichzeitige Ausführung (1) und (2)	Startbedingung	Endbedingung

Abb. 7: Symbolvorschläge für C3-Modellelemente (Quelle: [AZ09])








						
Aufzählung (Enumeration)	Komponente	Signal	Paket	Abhängigkeit	Zusammenfassen (merge)	Import

Abb. 8: Symbolvorschläge für UML-Modellelemente (Quelle: [Ko15])

beträchtlich, wenn Vorschläge von Probanden mit verschiedenem kulturellen und sprachlichem Hintergrund gesammelt werden sollen. In diesem Beitrag soll untersucht werden, ob eine Recherche in Bilddatenbanken diesen ersten Schritt ersetzen kann.

Hierzu wurden die in einer Modellierungssprache abzubildenden Konzepte, die auch in den drei Arbeiten [Ge12, AZ09, Ko15] untersucht wurden, als Abfragen für verschiedene Bilddatenbanken genutzt. Es wurde untersucht, ob unter den gelieferten Suchergebnissen auch die von den Probanden in den Experimenten vorgeschlagenen Symbole waren. Ist dies der Fall, kann der Aufwand für das manuelle Zeichnen von Symbolvorschlägen eingespart werden.

Da die Bildersuchen mitunter tausende Bilder als Ergebnis lieferten, wurden zur Vereinfachung nur die jeweils ersten 50 Suchtreffer ausgewertet. Alle Abfragen erfolgten im September 2017 von einem Desktop-PC in Deutschland aus. Mit Ausnahme von *Pinterest*, das nur mit Benutzeranmeldung nutzbar war, erfolgten alle Abfragen ohne Anmeldung an den Dienst und auch ohne Anmeldung an ein Google- oder Microsoft-Konto. Ein Filter für jugendgefährdende Darstellungen war nicht eingerichtet. Bei der Suche nach Begriffen, die aus mehreren Wörtern bestehen, wurde einerseits nach der gesamten Zeichenkette, andererseits auch nach einzelnen Wörtern gesucht, im Falle von „start condition“ also auch nach „start“.

Durchsucht wurden die folgenden Quellen:

Allgemeine Bild-Suchmaschinen: Genutzt wurden die Suche nach Bildern der Suchmaschinen *Google* und *Bing*, zunächst jeweils mit deren Standardeinstellungen, also ohne Filter auf Bildgröße und Bildart. Zusätzlich wurde die *Google*-Bildersuche mit einer Einschränkung auf den Bildertyp „Icon“ wiederholt. Schließlich wurde *Symbols.com* durchsucht, nach eigenen Angaben „die größte Ressource für Symbole, Zeichen und Flaggen im Web“.

Anbieter von Vektorgrafiken und Stockfotos: Um das Potential von (in der Regel gut mit Schlagworten versehenen) Datenbanken kommerzieller Anbieter von Bildern zu testen, wurden die Bilddatenbanken der Anbieter *Freepik* (www.freepik.com), *iStock* (www.istockphoto.com), *Fotolia* (www.fotolia.com) und *Shutterstock* (www.shutterstock.com) abgefragt.

Icon-Bibliotheken: Icon-Bibliotheken richten sich an Webdesigner sowie an Entwickler graphischer Benutzeroberflächen. Sie versprechen, sorgfältig gestaltete Symbole zur Darstellung typischer Begriffe anzubieten. Durchsucht wurden die Bestände der Anbieter *Iconfinder* (www.iconfinder.com), *Flaticon* (www.flaticon.com), *Icomoon* (icomoon.io/app/#/select) und *Findicons* (www.findicons.com).

Standards: Die International Organization for Standardization bietet auf www.iso.org/obp/ui die Möglichkeit für eine Suche nach Symbolen, die in einem ihrer Standards festgelegt sind. In den Suchergebnissen eingeschlossen sind somit die wichtigen Standards IEC 60417 und ISO 7000 (Graphische Symbole auf Einrichtungen), ISO 7001 (Graphische Symbole zur Information der Öffentlichkeit) sowie ISO 7010 (Sicherheitszeichen). Zur Suche nach diesen Symbolen muss die Suchabfrage auf der ISO-Webseite auf „graphical symbols“ eingeschränkt werden.

Visuelle Sprachen: Visuelle Sprachen werden heute vor allem zur Kommunikation mit Menschen mit kognitiven Schwierigkeiten eingesetzt. Sie stellen Piktogramme für mehrere Tausend Wörter zur Verfügung. Durchsucht wurden die Online-Abfragemöglichkeiten für die Sprachen *Pictogram* (www.pictogram.se/print/pictogram/view) und *Sclera* (<http://www.sclera.be/en/picto/>). Auf die Suche nach Symbolen in der Sprache Blissymbols wurde verzichtet, da diese nicht piktographisch aufgebaut ist, sondern Symbole nach einer speziellen Grammatik zusammenfügt. Für die Bildsprache BETA (www.betasymbols.com/) wurde keine kostenfrei zugängliche Symbolübersicht gefunden.

Soziale Plattformen: Unter den sozialen Plattformen wurden zwei Vertreter ausgewählt, bei denen das Veröffentlichen von Bildern eine vorrangige Bedeutung hat, nämlich *Flickr* und *Pinterest*. Bei der *Flickr*-Suche wurde nur nach Fotos (nicht nach Videos) gesucht; der Familienfilter von Flickr war bei der Suche eingeschaltet. *Pinterest* konnte nur nach einer Anmeldung genutzt werden. Diese erfolgte über ein *Google*-Konto, was *Flickr* Zugriff auf die Information zu Altersgruppe und Geschlecht des Nutzers erlaubte. Nach der Anmeldung müssen zwingend mindestens fünf Interessengebiete ausgewählt werden. Um dem Kontext „technische Modellierungssprache“ möglichst gut zu entsprechen, fiel die Wahl auf die Interessengebiete „Technologie und Technik“, „coole Produkte“, „Arbeitszimmer“, „Grafikdesign“ und „Bildung“.

ImageNet: *ImageNet* [De09] (www.image-net.org) ist ein Projekt mit dem Ziel, das semantische Netz WordNet [Fe98] um Bilder anzureichern. Die Organisation der Daten erfolgt dabei in Synsets, also Mengen von Begriffen, die synonym zueinander sind.

Emoji-Bibliotheken: Zum Finden von Emoji-Symbolen wurden *Emojifinder* (emojifinder.com) und *Emojipedia* (emojipedia.org) genutzt.

4 Auswertung

Tab. 1 zeigt, inwiefern die in den drei diskutierten Arbeiten per kollektiver Symbolerstellung gefundenen Symbole auch unter den Suchergebnissen der verschiedenen Bilddatenbanken zu finden sind. Dabei besagt ●, dass ein solches Symbol in den ersten 50 Suchtreffern enthalten war; ○ dass dies nicht der Fall war. Das Minuszeichen steht dafür, dass die Suche gar keine Bilder zum entsprechenden Suchwort lieferte.

Für die t^* -Modelle wurde sowohl für die Stereotypen (Abb. 5) als auch für die Prototypen (Abb. 6) ausgewertet, ob diese in den Treffermengen der Bildsuchen enthalten war. Sie sind in der Tabelle mit (S) bzw. (P) gekennzeichnet. Für „dependency“ wurden in [Ca13] zwei Prototypen mit der gleichen Zahl von Stimmen gewählt. Sie sind mit P1 und P2 gekennzeichnet. Das vierte Symbol für „dependency“ ist die in [Ko15] vorgeschlagene Kette aus Abb. 8. Für das Konzept „position“ wurde auf eine Unterscheidung zwischen Stereotyp und Prototyp verzichtet, da beide eine Markierung eines Feldes in einem Raster darstellen (vgl. Abb. 5 und 6).

Von den 42 Symbolen, die in den in [Ca13, AZ09, Ko15] beschriebenen Experimenten gefunden wurden, fanden sich 36 unter den ersten 50 Suchtreffern bei den nichtspezialisierten Suchmaschinen *Google* oder *Bing* oder bei der Suche mit *Google* eingeschränkt auf den Bildtyp „Clipart“. Nimmt man die Suchergebnisse der für unsere Zwecke ergiebigsten Icon-Bibliothek *Iconfinder* hinzu, erhöht sich die Zahl auf 38. Dieses Ergebnis belegt, dass die Suche in Bilddatenbanken eine effektive, aber zeit- und ressourcensparende Alternative zur Befragung menschlicher Probanden ist.

Die dennoch recht große Zahl von mit – oder ○ markierten Zellen in Tab. 1 ergibt sich vor allem aus der großen Zahl untersuchter Bilddatenbanken, unter denen sich einige als für unsere Zwecke offensichtlich ungeeignet herausgestellt haben. Im Folgenden soll die Eignung der verschiedenen Quellen kurz bewertet werden:

Generell gute Ergebnisse lieferten die allgemeinen Bildersuchen in *Google* und *Bing*. Es waren dort allerdings oft nicht die am häufigsten als Suchergebnis vorkommenden Bildmotive, die für die Darstellung eines Konzeptes in einer Modellierungssprache geeignet waren. Da die von diesen Suchmaschinen erfassten Inhalte keiner Einschränkung unterliegen, gab es unter den Suchergebnissen auch viele Bildmotive, die nicht für eine graphische

Modellierungssprache geeignet sind (vgl. Tab. 2). Diese ungeeigneten Motive können jedoch problemlos ignoriert werden. Als unbrauchbar erwies sich die Suche bei *symbols.com*. Es zeigte sich, dass der Schwerpunkt dieser Website auf Hoheitszeichen, Wappen, Flaggen und historischen Symbolen liegt. *Symbols.org* ist somit zwar für generelle semiotische Forschungen interessant, für unsere Problemstellung jedoch ungeeignet.

Gemischte Ergebnisse lieferten die kommerziellen Anbieter von Fotos und Grafiken. Diese Bildersammlungen sind dadurch gekennzeichnet, dass einiger Aufwand verwendet wurde, um die eingestellten Bilder mit Schlagworten zu versehen – schließlich sollen die Bilder bei der Suche gefunden werden, um Nutzungsrechte verkaufen zu können. Wie aus Tab. 1 ersichtlich, führte das jedoch zu keinen besseren Ergebnissen als bei der Nutzung der allgemeinen Suchmaschinen. Die Konzentration auf die Zielgruppe der Webdesigner machte sich gelegentlich auch durch eingeschränkte Suchergebnisse bemerkbar: Lieferten beispielsweise andere Quellen bei der Suche nach dem Stichwort „information“ gelegentlich auch das Symbol „Buch“, tauchten bei den kommerziellen Bildanbietern fast ausschließlich Bilder im Kontext Computer und Internet auf.

Als nützlicher erwiesen sich die Icon-Datenbanken (mit Ausnahme von *Icomoon*). Diese richten sich an Designer von graphischen Oberflächen für Computerprogramme. Die Ersteller der Icons haben in der Regel bereits großen Wert darauf gelegt, abstrakte Konzepte verständlich darzustellen. Wenn für die visuelle Darstellung eines Begriffs bereits allgemein bekannte Symbole existieren, ist davon auszugehen, dass diese bei der Suche in einer Icon-Datenbank gefunden werden. Ein Beispiel hierfür ist, dass zum Suchwort „Start“ das von CD- oder MP3-Spielern bekannte „Start“-Symbol ► gefunden wurde. Die Verwendung dieses Symbol zur Bezeichnung von Startereignissen in der Geschäftsprozess-Modellierungssprache YAWL wurde in [FMS09] wegen der intuitiven Verständlichkeit positiv hervorgehoben. Auffällig sind Schwächen der Funktionen zur Suche in den Icon-Sammlungen. So „berichtigte“ *icomoon.io*, die Schreibweise von „agent“ in „magnet“ und erlaubte keine Suche nach Zeichenketten aus mehreren Wörtern. Stattdessen wurden mehrere Suchwörter oder-verknüpft, was dann etwa bei der Suche nach „weak spot“ Bilder zum Thema „hotspot“ lieferte.

	Google	Google (Clipart)	Bing	symbols.com	Freepik	iStock	Fotolia	Shutterstock	Iconfinder	Flaticon	Iconmoon	Findicons	ISO Standards	Pictogram	Sclera	Flickr	Pinterest	ImageNet	Emojifinder	Emojipedia
actor	○	●	○	○	●	●	●	●	●	●	—	—	—	—	—	●	○	●	○	○
agent (S)	●	●	●	○	●	●	●	●	●	●	—	●	○	—	—	●	●	○	○	○
agent (P)	●	●	○	○	●	○	●	●	●	●	—	○	○	—	—	○	○	●	○	○
role (S)	○	●	○	○	○	○	○	●	●	○	—	—	—	—	—	—	—	○	—	—
role (P)	●	○	●	○	○	●	○	○	●	○	—	—	—	—	—	—	—	○	—	—
position	○	○	○	○	○	○	○	○	●	○	○	○	○	○	—	○	○	—	○	○
boundary (S)	●	●	●	○	●	●	○	●	●	●	—	—	○	—	—	●	●	●	○	○
boundary (P)	○	○	○	○	○	○	○	●	●	○	—	—	○	—	—	○	○	○	○	○
goal	●	●	●	○	●	●	●	●	●	●	●	●	—	○	○	●	○	○	○	○
softgoal	—	—	○	—	—	—	—	—	○	—	—	○	—	—	—	—	○	—	—	—
task	●	●	●	○	●	●	○	●	●	●	○	○	○	—	○	○	○	—	●	○
resource (S)	○	●	○	○	○	○	○	○	○	○	○	○	●	—	○	—	○	○	—	—
resource (P)	○	●	●	○	●	○	●	○	○	○	○	○	●	—	●	—	○	○	—	—
belief (S)	●	●	●	○	●	●	○	●	●	●	○	—	—	—	—	●	○	—	—	—
belief (P)	○	○	○	○	○	○	○	○	○	○	○	—	—	—	—	○	○	—	—	—
means-end (S)	○	○	○	—	—	—	—	○	○	○	—	—	—	—	—	○	○	○	○	○
means-end (P)	●	●	●	—	—	—	—	○	○	○	—	—	—	—	—	○	○	○	○	○
decomposition	○	●	○	○	○	○	○	○	●	—	—	—	—	—	—	—	○	○	—	○
dependency (S)	●	●	●	○	○	○	○	○	●	○	—	—	—	—	—	○	○	—	—	○
dependency (P1)	●	●	●	○	○	○	○	○	○	○	—	—	—	—	—	●	○	—	—	○
dependency (P2)	●	○	●	○	○	○	○	○	●	○	—	—	—	—	—	○	○	—	—	○
dependency	○	○	○	○	●	○	○	○	●	○	—	—	—	—	—	○	○	—	—	○
activity	●	●	○	○	●	●	●	○	○	○	○	○	○	○	○	○	○	—	—	○
tool	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
weak spot	●	○	○	—	—	●	●	●	○	—	○	●	—	—	—	●	○	—	—	—
decision (1)	●	●	●	○	○	●	●	●	○	●	—	●	—	○	○	●	○	—	—	—
decision (2)	●	○	●	○	○	●	○	○	○	●	—	●	—	○	○	○	○	—	—	—
control flow	●	●	●	—	○	—	○	○	○	—	○	○	—	—	—	○	○	○	○	○
information flow	●	●	●	—	○	○	—	●	○	—	○	○	—	—	—	○	○	○	○	○
information	●	●	●	○	●	●	○	●	●	●	●	●	●	●	●	○	○	○	●	●
synchronous	○	○	●	—	●	●	○	●	○	○	—	—	—	—	—	○	○	—	—	—
simultaneous (1)	○	○	●	—	—	○	○	●	○	—	—	—	○	○	○	●	○	○	○	○
simultaneous (2)	○	●	●	—	—	●	●	○	●	—	—	—	●	○	○	●	○	○	○	○
start (1)	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
start (2)	●	○	●	○	●	●	○	○	○	○	○	○	○	○	○	●	○	○	○	○
end (1)	○	○	○	○	○	○	○	○	○	○	○	○	●	—	—	○	○	○	○	○
end (2)	●	●	●	○	●	●	○	●	○	○	○	○	○	—	—	●	○	●	●	●
enumeration	●	●	○	○	●	●	●	●	○	○	○	—	—	—	—	○	○	—	—	—
component	●	●	○	○	●	○	○	○	●	○	○	○	●	—	—	—	○	○	—	—
signal	●	●	●	○	●	○	●	●	●	●	●	●	●	—	—	○	○	○	○	○
package	●	●	●	○	●	●	●	●	●	●	●	●	●	—	—	●	●	●	●	●
merge	●	●	●	○	●	●	○	●	●	●	●	●	—	—	—	○	○	—	○	○
import	○	●	●	○	○	○	○	●	●	●	○	●	—	—	—	○	○	—	○	○

Tab. 1: Vorhandensein der in [Ge12, AZ09, Ko15] vorgeschlagenen Symbole in den Suchergebnissen

Konzept	Häufigstes gefundenes Bildmotiv
activity	Kurve eines Aktivitätstrackers (Iconfinder)
end	Sonnenuntergang (Flickr)
position	Stellungen beim Geschlechtsverkehr (Google)
resource	Nahrungsergänzungsmittel (Bing)
task	Graffiti (Flickr)
tool	Bilder im Bezug zur Metal-Band <i>Tool</i> (Google, Bing)
weak spot	Schwachstellen von Panzern im Computerspiel <i>World of Tanks</i> (Google, Bing)

Tab. 2: „Unerwartete“ Ergebnisse der Bildersuche (Auswahl)

Entgegen der Erwartung, dass das Start-Symbol ► auch bei der Suche in den ISO-Standards gefunden wird, war dies nicht der Fall. Der Grund dafür ist, dass das Symbol dort nur unter den Stichworten „normal run; normal speed“ verschlagwortet ist. Zwar wurden bei der Suche nach Symbolen aus ISO-Standards nur 6 Symbole gefunden, die Benutzung dieser Datenbank scheint aber dennoch sehr ratsam. Das betrifft die Fälle, in denen es bereits ein standardisiertes und allgemein bekanntes Symbol für ein Konzept gibt (z.B. ⓘ für „Information“, das bereits 1975 von der International Union of Official Tourism Organizations als internationales Symbol vorgeschlagen wurde und weltweite Verbreitung fand). Interessant ist, dass das in [AZ09] für „Endbedingung“ vorgeschlagene Symbol → bei der Suche nach Symbolen zum Stichwort „end“ in den ISO-Standards gefunden wurde. Unter den Suchergebnissen der anderen Bilddatenbanken tauchte dieses Symbol jedoch nicht auf. Dies gibt Anlass zu der Vermutung, dass das Symbol nicht allgemein bekannt ist.

Als für unsere Zwecke unergiebig erwiesen sich die Suchergebnisse der Emoji-Sammlungen, der Bildsprachen *Pictogram* und *Sclera* sowie der sozialen Plattformen *Flickr* und *Pinterest*. Bei Letzteren steht die private Verwendung im Vordergrund. Daher wurden beispielsweise bei *Pinterest* bei der Suche nach „decision“ und „position“ vorwiegend Sinnsprüche und bei der Suche nach „activity“ vor allem Bilder vom Basteln mit Kindern gefunden.

Eine besondere Betrachtung verdient *ImageNet*. Da diese Ressource noch im Aufbau ist, wurde zu vielen Suchanfragen kein Ergebnis gefunden. Für die Suchworte, wo eine Suche bereits erfolgreich war, hat *ImageNet* aber einen großen Vorzug, der in keiner anderen Bilddatenbank zu finden ist: Da es sich bei *ImageNet* um eine Anreicherung der semantischen Datenbank *WordNet* um Bilder reichert, erfolgt nämlich die Suche wie von *WordNet* bekannt auf der Ebene von Synsets. Ein Synset bezeichnet eine Menge von Ausdrücken (Wörter bzw. Wortgruppen), die die selbe Bedeutung haben. Homonyme (Wörter mit mehreren Bedeutungen) finden sich folglich in mehreren Synsets wieder. Dies führt zum Beispiel bei der Suche nach dem Suchwort „actor“ zu der Erkenntnis, dass zwischen Film- bzw. Theaterschauspielern und den Akteuren unterschieden werden muss, die durch das Synset „actor, doer, worker (a person who acts and gets things done)“ beschrieben sind. Bei den manuell erstellten Vorschlägen wie auch bei den vorherrschenden Suchergebnissen der anderen Bilddatenbanken dominierte das Bild eines Schauspielers, was aber gerade *nicht* die

Bedeutung des entsprechenden Konzepts der Modellierungssprache t^* ist. Das Suchergebnis von *ImageNet* macht auf dieses potentielle Problem aufmerksam.

Tatsächlich erweist sich das „Schauspieler“-Symbol aus Abb. 5 als ungeeignet zur Beschreibung des Konzepts „Akteur“, sobald es von Benutzern aus verschiedenen Sprach- und Kulturkreisen verstanden werden soll. Bei dem in [Ge12, Ca13] beschriebenen Experiment erhielten die Probanden Beschreibungen der zu visualisierenden Konzepte in französischer und englischer Sprache. In beiden bezeichnet das Wort *actor* bzw. *acteur* sowohl den Akteur allgemein wie auch einen Schauspieler. Eine Wiederholung des Experiments mit deutschsprachigen Studenten führte erwartungsgemäß dazu, dass das Konzept „Akteur“ nicht durch einen Schauspieler bebildert wurde [LH13].

Hieraus ergibt sich ein weiterer Vorzug, den die Nutzung von Bilddatenbanken gegenüber herkömmlichen Ansätzen der kollektiven Symbolerstellung hat: Die Suchbegriffe können problemlos auch in anderen Sprachen eingegeben werden. Dadurch können Symbole bestimmt werden, die nur in manchen Sprachen verständlich sind. Auf Homonymen beruhende Symbolvorschläge wie ☉ für „Rolle“ können so ebenso identifiziert werden wie Symbole, die aufgrund kultureller Unterschiede nicht international verständlich sind (wie der nur in manchen Ländern übliche Hammer eines Richters als Symbol für „Entscheidung“). Sie sind als Symbole in einer Modellierungssprache, die international verständlich sein soll, schlecht geeignet. Wollte man solche Untersuchungen mit menschlichen Probanden durchführen, ergäbe sich selbst bei Verwendung von Crowdsourcing-Plattformen ein erheblicher Aufwand.

5 Vorgeschlagene Methode zur Symbolerstellung

In den Untersuchungen zeigte sich, dass bei der Recherche in Bilddatenbanken schnell Symbolvorschläge für die Darstellung typischer Konzepte in Modellierungssprachen ge-

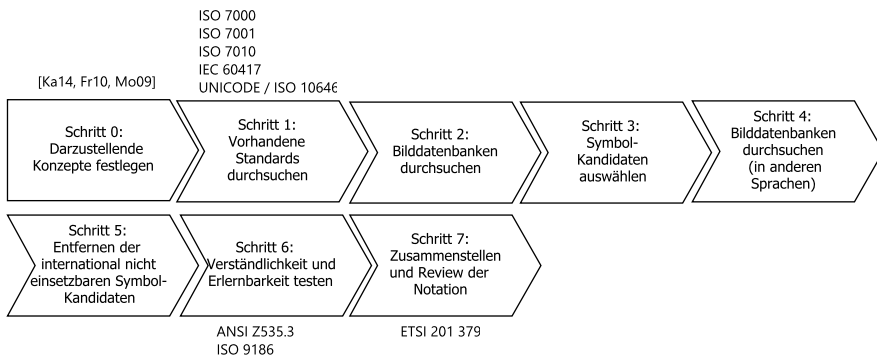


Abb. 9: Schritte der vorgeschlagenen Methode

funden werden können. Auf diese Beobachtung baut die in Abb. 9 skizzierte Methode zur Erstellung einer Notation auf. Sie berücksichtigt darüber hinaus Reviews und Tests einzelner Symbole sowie der entwickelten Notation als Ganzes. Auch dies ist ein Unterschied zum gegenwärtig üblichen Vorgehen, das in der Regel keine solchen Tests vorsieht [FB17]. An der Abbildung ist vermerkt, in welchen Schritten auf bestehende Richtlinien und Standards zurückgegriffen werden kann.

Schritt 0: Die Konzepte, die durch ein Symbol dargestellt werden sollen, werden festgelegt. Empfehlungen zu dieser (keineswegs trivialen) Aufgabe liefern [Ka14, Fr10, Mo09]. Es wird dann ein Metamodell der Modellierungssprache erstellt.

Schritt 1: Für jedes bildlich darzustellende Konzept wird untersucht, ob dafür bereits ein standardisiertes Symbol vorhanden ist. Standardisierte Symbole finden sich insbesondere in ISO-Standards sowie dem UNICODE-Block „Verschiedene Symbole“ (U+2600 bis U+26FF). Neben Symbolen, die unabhängig von Modellierungssprachen standardisiert wurden, können auch bereits vorhandene und in der Zielgruppe bekannte Modellierungssprachen wiederverwendet bzw. existierende Notationen für die eigenen Zwecke abgeändert werden (vgl. [Ka14]). Eine Hilfe hierfür könnte eine von van der Linden et al. [vdLHZ16] vorgeschlagene (aber bisher nicht realisierte) Bilddatenbank mit Icons für Modellierungssprachen sein.

Schritt 2: Für die Konzepte, für die nicht bereits in Schritt 1 ein zufriedenstellendes Symbol gefunden wurde, wird in Bilddatenbanken nach Bildvorschlägen gesucht.

Schritt 3: Aus den in Schritt 2 gefundenen Bildern wird eine Menge von Symbol-Kandidaten ausgewählt.

Schritt 4: Wenn die Modellierungssprache international verwendet werden soll, wird die Suche aus Schritt 2 wiederholt – diesmal unter Verwendung von Übersetzungen der Namen der Konzepte in andere Sprachen.

Schritt 5: Aus den in Schritt 3 ausgewählten Symbolen werden diejenigen entfernt, die bei der Suche in Schritt 4 nicht mehr gefunden wurden.

Schritt 6: Verständlichkeit und Erlernbarkeit der Symbol-Kandidaten werden mit Probanden getestet. Ausführliche Richtlinien zur Gestaltung solcher Tests finden sich in den

Standards ANSI Z535.3 und ISO 9186 [In14]. Das Ergebnis ist eine Rangfolge zwischen den Symbol-Kandidaten für jedes Konzept. Symbol-Kandidaten mit zu schlechter Verständlichkeit bzw. Erlernbarkeit scheidet aus.

Schritt 7: Aus den Symbol-Kandidaten wird eine Notation zusammengestellt. Hierbei werden die Symbole nicht mehr wie bisher separat betrachtet, sondern im Zusammenspiel miteinander. Insbesondere ist zu überprüfen, ob sich die Symbole hinreichend voneinander entscheiden. Symbole für die Konzepte, die nach dem in Schritt 0 erstellten Metamodell der Modellierungssprache Relationen sind, sollen zudem auch als *Kanten* in einem Graphen gezeichnet werden können. Ebenso ist zu beachten, dass die Symbole konsistent zueinander verwendet werden. Wenn beispielsweise „Informationsfluss“ als Pfeil mit einem \textcircled{I} dargestellt wird, sollte dieses Symbol auch für das Konzept „Information“ genutzt werden. In diesem Schritt unterstützen Moodys Prinzipien aus der *Physics of Notations* [Mo09, da16]. Weitere Punkte, die in diesem Schritt zu beachten sind, finden sich im ETSI-Standard EG 201 379 [Eu98]. Auch Checklisten für die Überarbeitung von Icons für graphische Benutzeroberflächen (vgl. [Ho94]) sind für das Review der Notation nützlich. Zusätzlich sind aber weitere Aspekte zu beachten, die so bei Icons nicht immer auftreten: Die Symbole sollen auch bei verkleinerter Darstellung sowie im schwarz/weiß-Druck noch erkennbar sein und sich auch leicht per Hand skizzieren lassen.

6 Diskussion und Zusammenfassung

Die Nutzung von Bilddatenbanken als Inspiration für Designer graphischer Benutzeroberflächen ist heute allgegenwärtig. Für die Gewinnung von Symbolen für graphische Notationen wurde diese Quelle aber bisher nicht systematisch genutzt. Im Beitrag wurde gezeigt, dass die Suche in Bilddatenbanken ähnliche Ergebnisse liefert, wie sie auch in aufwendigeren Experimenten gewonnen wurden, bei denen eine große Zahl von Probanden per Hand Symbolvorschläge zeichnet. Die in diesen Experimenten vorgeschlagenen Symbole konnten fast immer auch in Bilddatenbanken gefunden werden. Es bestätigte sich das Ergebnis von Xiao et al. [XAB10], dass Bilddatenbanken auch für abstrakte Begriffe passende Visualisierungen liefern. Eine Recherche in Bilddatenbanken kann daher den Prozess der Erstellung einer graphischen Notation sinnvoll unterstützen.

Vorzüge der Suche in Bilddatenbanken sind der erheblich geringere Aufwand und die Möglichkeit, durch die Verwendung von Suchworten in verschiedenen Sprachen bestimmte nicht international geeignete Symbole schnell auszuschließen. Es gibt aber einen Nachteil gegenüber der herkömmlichen kollektiven Symbolerstellung. Dort nämlich wissen die Probanden bei der Erstellung ihrer Symbolskizzen, in welchem Kontext die Symbole eingesetzt werden sollen. Nützlich war das beispielsweise in dem in [AZ09] beschriebenen Experiment, wo manche Probanden die „Schwachstelle“ (in einem Geschäftsprozess) mit dem Symbol „Flaschenhals“ darstellten. In den Studien zur Verbesserung der t^* -Notation

[Ge12, Ca13] zeigte sich allerdings, dass dieser Vorteil (zumindest bei wenig motivierten Probanden) nicht zum Tragen kommen muss. Ein Beleg dafür ist das am häufigsten für *belief* bzw. *croynance* vorgeschlagenen Symbol. Das religiöse Symbol eines Kreuzes ist nicht nur offensichtlich für einen kulturübergreifenden Einsatz ungeeignet, es stellt auch nicht die zu *i**passende Interpretation des Wortes *belief/croynance* dar. Hier wurde - wie auch bei *actor/lacteur* oder *agent* vorwiegend die Visualisierung vorgeschlagen, die mit dem Wort am ehesten verbunden ist. Wie gezeigt wurde, leistet dies auch die Abfrage in Bilddatenbanken.

Obwohl die Ergebnisse der durchgeführten Bildersuchen schon vielversprechend waren, sind noch Verbesserungen möglich. Die für diesen Beitrag verwendete Suchstrategie war nämlich ausgesprochen simpel. Das Beispiel der *Google*-Bildersuche mit der Einschränkung auf Cliparts zeigt, dass sich durch eine geschickte Nutzung von Suchfiltern bessere Suchergebnisse ergeben können. Gleiches gilt für die Nutzung von logischen Ausdrücken bei der Abfrage (Beispiel: *resource NOT human*) sowie bei der Verwendung zusätzlicher passender Suchbegriffe (*decomposition OR composition*).

Literaturverzeichnis

- [AZ09] Arning, Katrin; Ziefle, Martina: It's a bunch of shapes connected by lines. Evaluating the Graphical Notation System of Business Process Modelling Languages. In: 9th International Conference on Work With Computer Systems, Beijing, China. 2009.
- [Ca13] Caire, Patrice; Genon, Nicolas; Heymans, Patrick; Moody, Daniel L.: Visual notation design 2.0: Towards user comprehensible requirements engineering notations. In: 21st IEEE International Requirements Engineering Conference (RE). IEEE Computer Society, S. 115–124, 2013.
- [da16] da Silva Teixeira, Maria das Graças; Quirino, Glaice Kelly; Gailly, Frederik; de Almeida Falbo, Ricardo; Guizzardi, Giancarlo; Barcellos, Monalessa Perini: PoN-S: A Systematic Approach for Applying the Physics of Notation (PoN). In: BMMDS/EMMSAD. Jgg. 248 in LNBIP. Springer, S. 432–447, 2016.
- [De09] Deng, Jia; Dong, Wei; Socher, Richard; Li, Li-Jia; Li, Kai; Li, Fei-Fei: ImageNet: A large-scale hierarchical image database. In: 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, S. 248–255, 2009.
- [eGD14] el Kouhen, Amine; Gherbi, Abdelouahed; Dumoulin, Cédric: Improving Semantic Transparency of Committee-Designed Languages through Crowd-sourcing. In: 14th Workshop on Domain-Specific Modeling SPLASH. 2014.
- [Eu98] European Telecommunications Standards Institute: EG 201 379: Human Factors (HF); Framework for the development, evaluation and selection of graphical symbols. Bericht, 1998.
- [FB17] Fritsch, Andreas; Betz, Stefanie: Evaluation of Social Value Icons for a Domain-Specific Modeling Language. In: 47. Jahrestagung der Gesellschaft für Informatik e.V. (GI). Jgg. 275 in LNI. GI, S. 2323–2328, 2017.
- [Fe98] Fellbaum, Christiane, Hrsg. WordNet: An Electronic Lexical Database (Language, Speech, and Communication). The MIT Press, May 1998.

- [FMS09] Figl, K.; Mendling, J.; Strembeck, M.: Towards a Usability Assessment of Process Modeling Languages. In: 8. GI-Workshop EPK: Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. 2009.
- [Fr10] Frank, Ulrich: Outline of a method for designing domain-specific modelling languages. ICB Research Reports 42, University Duisburg-Essen, Institute for Computer Science and Business Information Systems (ICB), 2010.
- [Ge12] Genon, Nicolas; Caire, Patrice; Toussaint, Hubert; Heymans, Patrick; Moody, Daniel Laurence: Towards a More Semantically Transparent *i** Visual Syntax. In: REFSQ. Jgg. 7195 in LNCS. Springer, S. 140–146, 2012.
- [HF68] Howell, William C.; Fuchs, Alfred H.: Population stereotypy in code design. *Organizational Behavior and Human Performance*, 3(3):310 – 339, 1968.
- [Ho91] Howard, C.; O’Boyle, M.W.; Eastman, V.; Andre, T.; Motoyama, T.: The relative effectiveness of symbols and words to convey photocopier functions. *Applied Ergonomics*, 22(4):218 – 224, 1991.
- [Ho94] Horton, William: *Das Icon-Buch*. Addison Wesley, 1994.
- [In14] International Organization for Standardization: ISO 9186-1 Graphical Symbols – Test Methods. Bericht, 2014.
- [Jo83] Jones, Sheila: Stereotypy in pictograms of abstract concepts. *Ergonomics*, 26(6):605–611, 1983.
- [Ka14] Karsai, Gabor; Krahn, Holger; Pinkernell, Claas; Rumpe, Bernhard; Schindler, Martin; Völkel, Steven: Design Guidelines for Domain Specific Languages. The Computing Research Repository, abs/1409.2378, 2014.
- [Ko15] Kouhen, Amine El; Gherbi, Abdelouahed; Dumoulin, Cédric; Khendek, Ferhat: On the Semantic Transparency of Visual Notations: Experiments with UML. In: *SDL Forum*. Jgg. 9369 in *Lecture Notes in Computer Science*. Springer, S. 122–137, 2015.
- [LH13] Laue, Ralf; Hoglebe, Frank: Zur Verständlichkeit graphischer Symbole in Geschäftsprozessmodellierungssprachen. In: 43. Jahrestagung der Gesellschaft für Informatik e.V. (GI). Jgg. 220 in *LNI. GI*, S. 693–705, 2013.
- [MHM09] Moody, Daniel Laurence; Heymans, Patrick; Matulevicius, Raimundas: Improving the Effectiveness of Visual Representations in Requirements Engineering: An Evaluation of *i** Visual Syntax. In: 17th IEEE International Requirements Engineering Conference. S. 171–180, 2009.
- [MHM10] Moody, Daniel Laurence; Heymans, Patrick; Matulevicius, Raimundas: Visual syntax does matter: improving the cognitive effectiveness of the *i** visual notation. *Requir. Eng.*, 15(2):141–175, 2010.
- [Mo09] Moody, Daniel L.: The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Trans. Software Eng.*, 35(6):756–779, 2009.
- [Ob16] Object Management Group: Decision Model and Notation (DMN), Version 1.1. Bericht, 2016.

- [ST09] Siau, Keng; Tian, Yuhong: A semiotic analysis of unified modeling language graphical notations. *Requir. Eng.*, 14(1):15–26, 2009.
- [SZ08] Schröder, Sabine; Ziefle, Martina: Making a completely icon-based menu in mobile devices to become true: a user-centered design approach for its development. In: *Mobile HCI. ACM International Conference Proceeding Series. ACM*, S. 137–146, 2008.
- [vdLHZ16] van der Linden, Dirk; Hadar, Irit; Zamansky, Anna: Towards a Marketplace of Visual Elements for Notation Design. In: *RE. IEEE*, S. 353–358, 2016.
- [WGK13] Weitlaner, Doris; Guettinger, Annemarie; Kohlbacher, Markus: Intuitive Comprehensibility of Process Models. In: *S-BPM ONE. Communications in Computer and Information Science, Band 360. Springer*, S. 52–71, 2013.
- [XAB10] Xiao, Ping; Arroyo, Ernesto; Blat, Josep: Construct Connotation Dictionary of Visual Symbols. In (Huang, Mao Lin; Nguyen, Quang Vinh; Zhang, Kang, Hrsg.): *Visual Information Communication. Springer US, Boston, MA*, S. 119–134, 2010.
- [Yu11] Yu, Eric; Giorgini, Paolo; Maiden, Neil; Mylopoulos, John: *Social Modeling for Requirements Engineering. MIT Press*, 2011.
- [ZB83] Zwaga, H.J.; Boersema, T.: Evaluation of a set of graphic symbols. *Applied Ergonomics*, 14(1):43 – 54, 1983.

Exploiting Modular Language Extensions in Legacy C Code: An Automotive Case Study

Andreas Grosche¹, Burkhard Igel², Olaf Spinczyk³

Abstract: Model-driven software development using language workbenches like JetBrains MPS provide many advantages compared to traditional software development. Base languages can be incrementally extended to increase the abstractness up to domain-specific languages (DSLs). Changes can be performed more efficiently in problem-oriented language extensions or DSLs, than in a base language. In addition, formal analysis can be performed on abstract models. To benefit from the model-driven approach, non-model-based legacy code has to be reusable and transformable to language extensions and DSLs. For the development of embedded systems, mbeddr provides a C99-like base language and extensions for MPS, such as mathematical symbols and state machines. This paper presents a case study that shows how many legacy C code fragments of three automotive series projects could be replaced by mbeddr language extensions. Furthermore, a proof of concept shows the feasibility of fraction and foreach loop refactorings. This work is a first approach for future language extension refactorings.

Keywords: Case Study, MPS, mbeddr, Automotive, Embedded Systems, Model-Driven Software Development, Legacy C Code, Refactoring, Restructuring, Reverse Engineering, Reengineering

1 Introduction

While the software architecture of automotive embedded systems is commonly modeled using UML, different implementation techniques exist. Code generation from UML models and/or graphical modeling tools, such as Matlab/Simulink, hides implementation details from the developer, and thus raises the level of abstraction. To gain transparency and control [Gr05] over registers, memory, runtime and synchronization, especially time-critical software and basic software modules are typically implemented using a low-level programming language, such as C. However, the implementation of architectural and domain-specific concepts is time-consuming and error-prone in such languages. Therefore, during the past few years there have been efforts to bridge the gap between abstract modeling and low-level programming using extensible languages that provide multiple levels of abstraction within

¹ Behr-Hella Thermocontrol GmbH, Hansastraße 40, 59557 Lippstadt, andreas.grosche@bhtc.com

² FH Dortmund, Sonnenstraße 96, 44139 Dortmund, igel@fh-dortmund.de

³ TU Dortmund, Otto-Hahn-Str. 16, 44221 Dortmund, olaf.spinczyk@tu-dortmund.de

the same program. An important step in this direction has been taken with the JetBrains Meta Programming System (MPS)⁴ and mbeddr⁵, which will be described in the following.

Traditional integrated development environments (IDEs) provide an editor to modify the source code of a program as plain text as shown in Fig. 1a [Ca16]. To support advanced IDE features, such as refactoring and navigation, the source code is divided into tokens that are further parsed to build up abstract syntax trees (ASTs) in a similar way as compilers work. A build system invokes a compiler to build the executable output.

In contrast to traditional IDEs, JetBrains MPS enables model-driven software development (MDSO). Instead of modifying text files, the user directly edits the ASTs of the program using a projectional editor (see Fig. 1b). An editor definition for each AST node defines, how the node is presented using a concrete syntax. Different editor definitions for the same AST nodes can provide textual or graphical views and reveal different information of the same model without the need of a tokenizer or parser.

MPS is a language workbench [Vo14] that provides a general purpose *base language*. This language can be incrementally extended by user-defined language extensions that abstract common and domain-specific code fragments. Different abstraction levels right up to domain-specific languages (DSLs) can be realized within the same program. During the build process, a generator transforms the DSLs and language extensions into the base language that is further transformed to text, such as Java code or XML. A build system can invoke a compiler to build executable code.

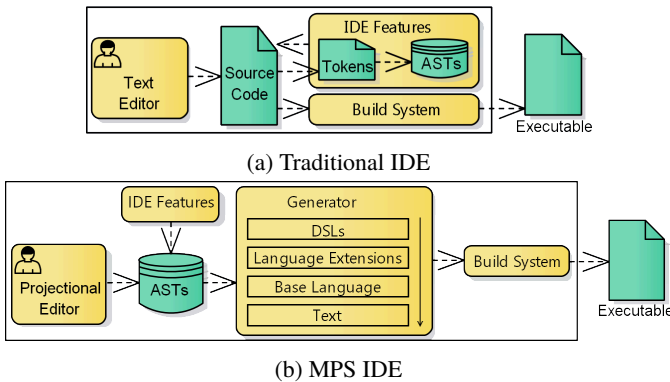


Fig. 1: Workflows of traditional IDEs and the MPS IDE.

The open source project mbeddr [Vo12] (primarily developed by itemis and fortiss) is based on MPS and provides a base language, which is similar to C99, for the development of embedded systems. The language extensions supplied with mbeddr, such as mathematical symbols and physical units, can be further extended by user-defined languages and DSLs

⁴ <https://www.jetbrains.com/mps/>

⁵ <http://mbeddr.com/>

to close the gap between high-level modeling and low-level programming languages. For instance, the language extension for state machines of *mbeddr* uses the full power of projectional editing. It now serves as an introductory example:

State machines can be modeled graphically, textually or as a table. The graphical projection of a state machine is shown in Fig. 2a. On reception of the *evtWrite* event, a transition to the *Writing* state is performed. On entry of that state, a function is called that writes the passed byte 12u via a universal asynchronous receiver/transmitter (UART). The state machine rests in the *Writing* state until the event *evtTxISR* is received. Fig. 2b shows the textual projection of this state machine.

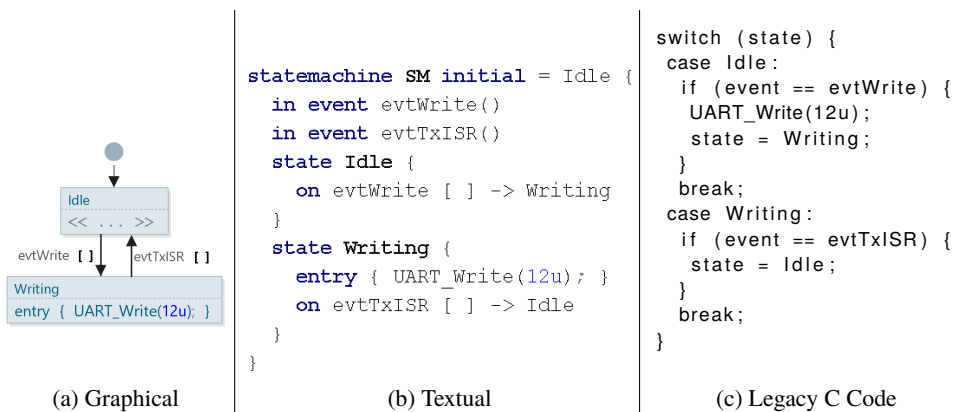


Fig. 2: The graphical and textual projections of an *mbeddr* state machine focus on the problem to be solved and hide implementation details.

Introducing *mbeddr* into the embedded software domain offers several advantages compared to traditional software development processes [Vo13b]:

- Projectional editors allow syntax that cannot easily be parsed by a traditional text-based IDE, such as mathematical symbols, tables and diagrams.
- The language extensions supplied with *mbeddr* can be further extended. The modular approach allows the combination of languages from different abstraction levels within the same program and even translation unit. For example, the C-like base language can be used for the definition of state machine actions (see Fig. 2a).
- Changing the structure of a software is simpler using higher-level abstractions than changing low-level C code, because implementation details are hidden.
- Automated domain-specific C verification [MVR14] closes the gap between C verification tools, such as CBMC [CKL04], and domain-specific language extensions, such as state machines and decision tables.

- Specifying requirements and unit tests using `mbeddr` allows the creation of links to achieve traceability between requirements, code fragments and verifications.

For an automotive company like Behr-Hella Thermocontrol GmbH (BHTC), it is common practice to reuse legacy C code in new projects, because *redevelopment* would be too expensive and time-consuming. The company `itemis`⁶ developed a commercial importer based on TypeChef [Käl1] that allows the import of legacy C code into `mbeddr`. However, the imported code makes primarily use of the C99-like base language. To benefit from the advantages of the model-driven approach, legacy C code fragments have to be replaced by corresponding language extensions after the import. For example, the legacy C code shown in Fig. 2c should be replaced by an `mbeddr` state machine (see Fig. 2a). This would improve maintainability because the states, transitions and actions for different events could be modified without thinking about how state transitions and event handling are realized. In addition, automated verifications could be applied to the state machine.

To avoid time-consuming and error-prone manual replacements of base language code with language extensions, refactorings have to be developed. The complexity of such refactorings depends on the gap between the base language and the corresponding language extension in terms of abstraction levels. Regarding state machines, several implementation techniques exist, such as using `switch` (see Fig. 2c) statements or complex frameworks [Sa09]. The refactoring of language extensions with a small abstraction gap may be fully automated, while approaches with user interaction or machine learning may be required for extensions with larger abstraction gaps, such as state machines.

Language extension refactorings could also help developers to learn how and when to use language extensions. For example, when a developer writes a code fragment and the IDE would propose to replace that fragment with more abstract code using a language extension, the developer would learn when to use that language extension without the need of reading the language manual.

As languages evolve over time, it is likely that new language extensions will be added to `mbeddr` in the future. In addition, new language extensions can be developed by the user. Semi-automatic language extension refactorings could help to modify existing `mbeddr` code to use the new language extensions.

This paper presents a case study that shows the amount of code fragments in automotive legacy C code that could be replaced by language extensions supplied with `mbeddr` to assess the importance of language extension refactorings. We also provide a proof of concept to show that, at least in many cases, fully automated refactorings are possible.

The case study and the examined language extensions are introduced in Sect. 2. The proof of concept is shown in Sect. 3. After the presentation of related work in Sect. 4, we conclude and show future work in Sect. 5.

⁶ <https://www.itemis.com/>

2 Case Study

The goal of this case study is to investigate the actual number of code fragments where our automotive series projects could benefit from the language extensions provided by mbeddr. Only handwritten C code in .c and .h files is part of this study. Generated code, such as from Matlab/Simulink models, is out of the scope of this case study, because refactoring such code should be done in Matlab/Simulink instead of the generated code. We introduce the analyzed projects in Sect. 2.1 before we present the applied method in Sect. 2.2. The examined language extensions are introduced in Sect. 2.3 and the results are discussed in Sect. 2.4.

2.1 Context

Our case study comprises three automotive projects with different characteristics as shown in Tab. 1. Project *A* contains the software for the main controller of a human-machine interface (HMI) control panel featuring a touch screen, capacitive buttons and acoustic feedback. Project *B* contains the software for the main controller of a center infotainment display (CID) featuring a touch screen, force sense and acoustic feedback. Project *C* contains the software for a CID slave controller that processes a tactile feedback.

Tab. 1: Automotive series projects analyzed in this case study.

Project	Status	Microcontroller	Standard	Files	Analyzed Files	Analyzed SLOC
A	Series	32 bit, 80 MHz	AUTOSAR	1,189	707 (289 .c and 418 .h)	238,651
B	Pre-Series	32 bit, 80 MHz	AUTOSAR	1,194	734 (283 .c and 451 .h)	222,203
C	Development	16 bit, 32 MHz	non-AUTOSAR	287	253 (101 .c and 152 .h)	15,390

Projects *A* and *B* are developed according to AUTOSAR [AU14]. The development phase of project *A* is finished, whereas project *B* is in the pre-series development phase. Both target a 32 bit microcontroller running at 80 MHz. For project *A*, we analyzed 707 handwritten .c and .h files containing 238,651 source lines of code⁷ (SLOC). For project *B*, we analyzed 734 handwritten .c and .h files containing 222,203 SLOC. The software for project *C* is not developed according to AUTOSAR. The target is a 16 bit microcontroller running at 32 MHz. This project is under rapid development and currently comprises 253 handwritten .c and .h files containing 15,390 SLOC that we analyzed.

2.2 Method

To find code fragments that could be replaced by mbeddr language extensions in the source code of the analyzed projects, a semi-automatic approach has been chosen. In the first step, we had to select all handwritten .c and .h files from the projects to be analyzed. We did

⁷ According to Count Lines of Code (CLOC), <http://cloc.sourceforge.net/>

this by scanning the files for specific keywords in the comment header of each file that are typical for handwritten and generated files. Only a few files had to be classified manually. The second step was to define code patterns for each mbeddr language extension. A code fragment is considered to be replaceable with a language extension, if the corresponding pattern matches. We used Coccinelle [Pa08] to match the patterns against the examined legacy C code. We could not specify exact patterns for all language extensions because of their complexity. That is why we had to verify and post-filter some of the results manually to avoid false positives.

List. 1 shows an excerpt of a Coccinelle pattern that finds all `for` loops in the analyzed code that fulfill the following requirements: In the initialization, an identifier has to be initialized with an arbitrary expression. This identifier has to be less than an arbitrary expression to enter the loop body. The same identifier has to be incremented on each iteration. The loop body may contain arbitrary statements. As `i` is a meta variable of Coccinelle the identifier can have an arbitrary name.

```
@@ identifier i; @@
* for (i = ...; i < ...; i++)
{
  ...
}
```

List. 1: Excerpt of the Coccinelle pattern that we used to find code fragments that could be replaced by `for` range loops.

2.3 Language Extensions

An overview of the examined language extensions is presented in Tab. 2. We give only a small introduction while Voelter et al. [Vo13a] present more details. The second column shows the language extensions as they can be edited in MPS. The third column shows examples of legacy C code in MPS that could be replaced by the corresponding extension. Except for error handling, which is explained later, the examples shown in the legacy C code column are conceptually similar to the code generated by MPS. We have subdivided the extensions into three categories:

Syntactic Sugar: The projectional editor allows mbeddr to provide language extensions for graphical mathematical symbols. We grouped them to fractions, mathematical functions as well as products (of sequences) and sums. **Fractions** can be used as an alternative syntax for divisions. To find code fragments replaceable by fractions, we used a Coccinelle pattern that looks for expressions that are divided by another expression.

Symbols for **mathematical functions** encapsulate calls to functions such as `abs`, `log`, `pow` and `sqrt` for calculating absolute values, logarithms, powers and square roots. To find code

Tab. 2: Screenshots of mbeddr language extensions and corresponding legacy C code in the MPS IDE.

Lang. Ext.	mbeddr Extension Code	Legacy C Code
Fraction	$z = \frac{x}{y + 1};$	<code>z = x / (y + 1);</code>
Math. Function	$z = x ; z = \log_y x;$ $z = x^y; z = \sqrt{x + y};$	<code>z = abs(x); z = log(x) / log(y);</code> <code>z = pow(x, y); z = sqrt(x + y);</code>
For Range	<code>for (i ++ in [5..20]) { array[i] = i; }</code>	<code>for (uint8 i = 5; i < 20; i++) { array[i] = i; }</code>
Foreach	<code>foreach (frame sized FRAME_SIZE as it) { checksum += it; }</code>	<code>for (uint8 i = 0; i < FRAME_SIZE; i++) { checksum += frame[i]; }</code>
Product and Sum	$z = \prod_{i=1}^{10} x + 1; z = \sum_{i=1}^{10} x + 1;$	<code>z = 1;</code> <code>for (int32 i = 1; i <= 10; i++) { z *= x + 1; }</code>
Phys. Unit	<code>uint8/N/ f = 10 kg * $\frac{10 \text{ m}}{5 \text{ s}^2};$</code> <small>Error: type (uint8 int8)/kg/ is not a subtype of uint8 /N/</small> <code>f = 5 kg;</code>	<code>uint8 fInNewton = 10 * (10 / 5);</code> <code>fInNewton = 5;</code>
Error Handling	<code>@errors E_UNINITIALIZED</code> <code>uint8 GetBrightness2() {</code> <code>if (_state == Uninitialized) {</code> <code>error E_UNINITIALIZED;</code> <code>}</code> <code>return _brightness;</code> <code>}</code> ----- <code>try {</code> <code>uint8 b = GetBrightness2();</code> <code>// do something with b</code> <code>}</code> <code>when E_UNINITIALIZED {</code> <code>// error handling</code> <code>}</code>	<code>Std_ReturnType GetBrightness1(uint8* B) {</code> <code>if (_state == Uninitialized) {</code> <code>return E_NOT_OK;</code> <code>} else if (B == NULL) {</code> <code>return E_NOT_OK;</code> <code>}</code> <code>*B = _brightness;</code> <code>return E_OK;</code> <code>}</code> ----- <code>uint8 b;</code> <code>if (GetBrightness1(&b) == E_OK) {</code> <code>// do something with b</code> <code>} else {</code> <code>// error handling</code> <code>}</code>
State Machine	See Fig. 2a	See Fig. 2c

fragments that could be replaced by such symbols, we looked for calls to functions with identifiers that contain `abs`, `log`, `pow` and `sqrt`. This also includes variants such as `labs` and user-defined implementations with a similar naming.

Enriched Syntax: To simplify `for` loops that are incremented or decremented by I on each iteration, `mbeddr` provides **for range** loops as shown in Tab. 2. Only the counter variable name, the minimum and the maximum (exclusive) have to be specified. The type of the counter variable, the compare operator in the condition as well as the increment of the counter variable are omitted. The `++` can be replaced by `--` to iterate backwards over the specified range. To find code fragments replaceable by `for range` loops, we looked for `for` loops that assign an arbitrary expression to a counter variable. This variable has to be used in the condition with an appropriate greater-than or less-than comparison. In addition, it has to be incremented or decremented in the iteration by I . Since we analyzed projects using C90 that requires the definition of the counter variable at the start of a block (e.g., a function body) instead of in the `for` loop itself, we further had to analyze the usage of the variable before and after the `for` loop.

The **foreach** language extension of `mbeddr` can be used to iterate through arrays. The `it` expression can be used to read or write the current array element. Tab. 2 shows an example of a `foreach` loop that iterates over the `frame` array with an array length of `FRAME_SIZE`. In the body, the current element `it` of the array is added to the `checksum` variable. To find code fragments that could be replaced by `foreach` loops, we looked for `for` loops where a counter variable is set to 0, compared within an appropriate condition and incremented by I . In addition, the counter variable has to be used in the body as an index to an array. It can be used multiple times, but only for the same array. Furthermore, calculations such as adding an offset to the counter variable must not be performed. As for `for range` loops, we had to analyze the context of the `for` loop, because the analyzed projects use C90.

Products and sums are expanded to `for` loops during the code generation. In the analyzed code, we looked for `for` and `while` loops that add or multiply an expression to an identifier using statements, such as `x = x + ...;` or the short form `x += ...;`.

Enriched Semantic: The **physical units** extension allows the annotation of types and literals with unit information. The force F in newton (N) is defined as $F = m \cdot a$. Annotating the type of the variable `f` with the unit N requires assigned expressions to evaluate to the corresponding unit. Trying to assign an expression that evaluates to another unit, e.g., `kg`, leads to an error message in the IDE as shown in Tab. 2. Units are evaluated in the model and do not add any overhead to the generated C code. As shown in the legacy C code column, developers sometimes append a suffix containing the unit to the identifiers. To find code fragments that could be annotated with physical units, we looked for macro and variable definitions containing identifiers that contain `us`, `ms`, `clk`, `Hz`, `freq`, etc. and further analyzed the context.

The common way in AUTOSAR to inform the caller of a function about an error is to return a value of the type `Std_ReturnType` [AU15] as shown in the simplified example in the legacy C code column of Tab. 2. The standard values are `E_OK` for success and `E_NOT_OK` for errors. These can be extended by user-defined values. The caller has to evaluate the return value and perform an error handling. As a consequence, call-by-reference has to be used to get values from functions like getter functions.

As shown in Tab. 2, mbeddr provides sophisticated **error handling** in the style of Java or C++ exceptions. The example shows a getter function that is annotated with the errors that are thrown in the function body using the `error` statement. The getter function is called in a `try` block. The execution of the statements in the `try` block is aborted as soon as a function throws an error, which is caught in the corresponding `when` block. During the code generation, this error handling is expanded to `goto` statements and an error function argument (call by reference). Therefore, the overhead is comparable to the AUTOSAR approach. To identify code fragments replaceable by mbeddr error handling, we looked for functions that return a value of type `Std_ReturnType`.

State machines are also examined but not shown in the table, because they have already been introduced in Sect. 1. To find state machines in the analyzed C code, we focused on `switch` and `if` statements. We further examined these constructs manually with typical implementations of state machines [Sa09] in mind, such as using a variable that holds the current state (see Fig. 2c).

The presented language extensions have been ordered by their level of abstraction. *Syntactic sugar* language extensions are simple alternative syntactic representations compared to the respective legacy C code. They do not provide notable abstractions but still improve readability. *Enriched syntax* language extensions provide in-place substitutions to simplify compound expressions or statements. They provide simple abstractions that hide implementation details and possibly provide declarative flavor. *Enriched semantic* language extensions extend the type system, provide meta information, abstract data or control flow or have a cross-cutting character. Members of all categories can make use of graphical representations to further improve readability and maintainability.

2.4 Results

The results of this case study are presented in Tab. 3. Each row contains the number of code fragments that could be replaced by the different mbeddr language extensions of one analyzed project. Except for the mathematical functions, products and sums, we got two-digit and three-digit numbers of possible replacements. The larger projects *A* and *B* generally contain more replaceable code fragments than the smaller project *C*. The details for the different language extensions are discussed in the following.

Tab. 3: Number of possible replacements of C code fragments with mbeddr language extensions in three automotive series projects.

Project	Fraction	Math. Func.	For Range	ForEach	Product	Sum	Physical Unit	Error Handling	State Machine
A	150	0	483	112	6	2	54	323	58
B	243	0	462	86	6	2	45	191	75
C	16	0	40	24	0	6	69	87	21

The MISRA C:2012⁸ rule 12.1 advises to use parentheses to make the operator precedences of C expressions explicit [MI13]. In practice, this leads to extensive use of functionally unnecessary parentheses to reduce possible human mistakes with the precedence rules of C. An example of the analyzed code is shown in Fig. 3. The first line shows a simplified version of a statement of the analyzed code with parentheses around the divisions to meet the MISRA C:2012 rule 12.1. The second line shows the same statement using **fractions**. This statement is much more readable and, in our opinion, the parentheses around the fractions can be omitted, because the precedence is obvious due to the graphical notation. In the generated C code, parentheses are inserted to meet the MISRA rule.

```
max = (1732u / fCLK) + 36u + (((4351u / fCLK) + 7324u) * blk) + (((184u / fCLK) + 44u) * n);
max =  $\frac{1732u}{fCLK}$  + 36u + (( $\frac{4351u}{fCLK}$  + 7324u) * blk) + (( $\frac{184u}{fCLK}$  + 44u) * n);
```

Fig. 3: A statement without and with using fractions in MPS.

The small amount of code fragments replaceable by symbols for **mathematical functions**, **products** and **sums** can be explained by the systematic use of Matlab/Simulink for signal processing in the analyzed projects. The code generated by Matlab/Simulink is not part of the analysis. The found code fragments replaceable by sums are used to calculate simple checksums for inter-processor-communications. The calculations of products are used to convert raw bus signals to SI values and vice versa.

For projects A and B, the **for range** loop is the most usable language extension. The abstraction gap of for range loops is pretty small and the use of a for loop with a loop variable that is incremented or decremented by I is very common.

Some of the found for loops that could be replaced by **foreach** loops could also be replaced by sums. An example is the primitive checksum calculation over an array (e.g., bytes received via a serial communication) as shown in Tab. 2. As we further analyzed non-matching for loops, we found loops that iterate over multiple arrays at the same time as shown in a simplified version in Fig. 4a. These arrays store different information for the same entity. In this example, one array stores the information whether each UART peripheral is enabled and another array stores the current state of each UART peripheral. The counter variable of

⁸ *Guideline for the use of the C language in critical systems* published by the Motor Industry Software Reliability Association (MISRA).

the `for` loop is used as an index for both arrays. In addition, the counter variable is passed to the `SendByte` function that uses the argument to access other arrays that store information about the UART peripherals.

<pre>for (uint8 i = 0; i < UARTS; i++) { if (uartEnabled[i] && uartState[i] == 1u) { SendByte(i); } }</pre>	<pre>foreach (uarts sized UARTS as it) { if (it.Enabled && it.State == 1u) { SendByte(&it); } }</pre>
(a) Without OO	(b) With OO

Fig. 4: A `for` loop without and a `foreach` loop with object-oriented flavor in MPS.

The attributes of all UARTs could be restructured from multiple arrays of primitive data types to one array of structures. Each structure could hold all attributes of a UART in an object-oriented way. The result would be a `for` loop that iterates over one array. This loop could be replaced by a `foreach` loop as shown in Fig. 4b. The attributes of the iterated UARTs could then be accessed using the `it` expression and a reference to `it` could be passed to a function that could access the attributes of the UART in a convenient way.

We did not include this kind of `for` loops in the results, because more sophisticated analyses would be necessary. Further replacements by `foreach` loops would be possible looking for `while` and `do...while` loops with appropriate data flow analysis to match the preconditions for the counter variable incrementation.

More code fragments could be replaced by **physical units**, if signal processing would not be done in Matlab/Simulink. However, physical units do not only make sense in signal processing but also in basic software, e.g., for calculations of times and frequencies. It is notable that more physical units could be used in the small non-AUTOSAR project *C* than in the larger AUTOSAR projects *A* and *B*. This can be explained by the workflow of code generators that directly calculate register values to configure the hardware in the AUTOSAR projects. To allow a convenient hardware configuration in project *C* despite the lack of code generators, calculations of register values are done in C code to allow the specification of the hardware configuration parameters in physical units, such as Hz and ms.

A variety of approaches exist to implement **state machines** [Sa09]. In the analyzed code, we found primarily simple approaches, which use `switch` or `if` statements to determine the current state using a state variable. Depending on the state, corresponding actions are implemented and the state variable is reassigned to switch to the next state. We also found more sophisticated approaches using tables as well as code fragments that were not designed with a state machine in mind that could be restructured to get a well-designed state machine.

Although we used complex patterns to compensate implementation variations, such as using `switch` or `if` for state machines, the results shown in Tab. 3 are pessimistic. More relaxed patterns in combination with program and data flow analysis would reveal more refactoring possibilities. For example, we assumed that the power is calculated using the `pow` function.

However, programmers also use multiplications, such as $x = y * y$, that may even be split into multiple statements.

Tab. 4 shows the minimum and maximum SLOC of C code that we found in the examined projects that could be replaced with language extensions. To replace divisions by **fractions** and for loops by **for range** loops, only one line of code has to be modified. Especially for fractions, multiple divisions may occur in one line. The replacement of for loops with **foreach** loops requires the modification of the loop header and each array access using the loop variable in the body. The code fragments of the examined projects that could be replaced by **products** or **sums** are similar to the legacy C code outlined in Tab. 2.

The number of affected SLOC for replacements with physical units, error handling and state machines have been determined statistically using three exemplary translation units. For **physical units**, we considered the definition (e.g., of a variable) and the usages. For the context of each usage, we recursively analyzed which identifiers and literals are involved and how appropriate units can be applied to them. The SLOC for **error handling** is the sum of lines to be modified in the function that emits an error and the error handling of all calling functions. For **state machines**, we counted the lines of code required for the definition and implementation of the states and transitions.

Tab. 4: Minimum and maximum SLOC of C code replaceable with mbeddr language extensions in three automotive series projects.


Project	Fraction	Math. Func.	For Range	For-each	Product	Sum	Physical Unit	Error Handling	State Machine
A	1	0	1	2 ... 10	4	4	1 ... 16	2 ... 9	36 ... 215
B	1	0	1	2 ... 10	4	4	1 ... 22	2 ... 30	18 ... 187
C	1	0	1	2 ... 4	0	4	1 ... 28	2 ... 9	67 ... 161

In general, larger abstraction gaps between language extensions and the base language require more complex patterns to find possible replacements. For example, the most complex Coccinelle pattern for the examined *syntactic sugar* language extensions took seven lines of code and no manual post-filtering was needed. In contrast to that, the *enriched semantic* language extensions required up to 234 lines of Coccinelle pattern and manual post-filtering.

3 Proof of Concept

As a proof of concept, we implemented refactorings that transform divisions to fractions and for loops to foreach loops in MPS. We used a simple approach that relies on the MPS base language with predefined extensions for model queries and transformations. The applicability of a refactoring is determined recursively using a set of preconditions. If all preconditions match, a model transformation is performed to replace divisions with fractions or for with foreach loops keeping the required properties and child elements.

Fig. 5 shows a statement for the calculation of the USART baudrate register of an Atmel ATmega8 microcontroller. Performing the refactoring for fractions on this statement replaces the division with a fraction bar. In addition, the parentheses around the fraction and around the denominator are removed because they are not required anymore.

`ubrr = (Fosc / (16 * Baudrate)) - 1;` 
$$\text{ubrr} = \frac{\text{Fosc}}{16 * \text{Baudrate}} - 1;$$

Use Math Symbols for Fractions

Fig. 5: Screenshots of the ATmega8 USART baudrate register calculation before and after the fractions refactoring in MPS.

An example for the `foreach` refactoring is shown in Fig. 6. The `for` loop is replaced with a `foreach` loop. The upper bound `UARTS` of the condition is reused in the new loop and all occurrences of an array indexed by the loop variable are replaced by the `it` expression. The number of preconditions to be checked for this refactoring is very large compared to the fractions refactoring, because many variations can occur. For example, one precondition ensures that the loop index is incremented by `1` on each iteration. To check this single precondition, multiple variations of the incrementation, such as `i++`, `++i` and `i = i + 1`, have to be considered. Using data and control flow analysis could additionally enable the replacement of `while` and `do..while` loops.


`for (uint8 i = 0; i < UARTS; i++) {`
`if (uarts[i].State == Writing) {`
`HandleWriting(&uarts[i]);`
`}`
`}`  `foreach (uarts sized UARTS as it) {`
`if (it.State == Writing) {`
`HandleWriting(&it);`
`}`
`}`

Fig. 6: Screenshots of a `for` loop before and after the refactoring to a `foreach` loop in MPS.

4 Related Work

Several case studies exist regarding the use of model-driven software development with `mbeddr` for embedded software. One case study deals with programming Lego Mindstorms robots based on an OSEK operating system [VKa] using appropriate language extensions with relevance beyond the Lego use case. Another case study approaches the first real-world project using `mbeddr` [VKb]. Further case studies show how language extensions affect the complexity, testability and runtime overhead of embedded software using `mbeddr`. They also show the effort for engineering a new project [Vo15, Vo17]. Vinogradov et al. [VOR15] describe the experience of using `mbeddr` in the railway domain including the integration of the model-driven approach into the traditional product lifecycle. All of these case studies conclude that language extensions and DSLs simplify reviews and the implementation of changes. Our case study is the first approach to evaluate the extent of applicability of `mbeddr` language extensions in existing automotive series projects.

Refactorings have been a research topic for several years [MT04] and most modern IDEs support basic refactorings. These traditional refactorings primarily focus on the structure of a software and improve it by restructuring, e.g., by moving statements into a new function

[FB13]. In contrast, language extension refactorings focus on the behavior and purpose of code fragments (considering different implementation techniques) and the replacement by more abstract language extensions.

Like the presented approach, reverse engineering aims at the automated comprehension of software to enhance development efficiency and maintainability. While classic reverse engineering creates representations at a higher level of abstraction without modifying the software [CC90], language extension refactoring *transforms* software fragments to a higher level of abstraction retaining the external behavior. Reverse engineering and design recovery techniques are a promising approach for the realization of language extension refactorings. However, language extension refactoring should not only recover design but also support the developer in improving the design and implementation which may require additional semantic analysis and input of domain-specific knowledge by the developer.

5 Conclusion and Future Work

Model-driven software development using mbeddr closes the gap between the programming language C, which is close to the hardware, and modeling languages, such as UML. Several case studies show the advantages of using mbeddr for the development of embedded software. Siemens PLM Software made use of these advantages and released the commercial LMS Imagine.Lab Embedded Software Designer (ESD) [Si] that is based on mbeddr.

The case study presented in this paper reflects the usefulness of the language extensions supplied with mbeddr in the automotive domain. It evaluates the number of possible replacements of legacy C code fragments by mbeddr language extensions in three automotive series projects. The extensions for products, sums and mathematical functions could rarely or not at all be applied. All other examined language extensions (e.g., fractions, for range loops and error handling) could be applied with a moderate to high degree. We could not specify sufficient preconditions to cover all the language extensions shipped with mbeddr, because of the high complexity of some of the language extensions, namely decision tables, interfaces and components. The missing extensions could be part of future case studies.

Since a refactoring must not alter the external behavior of a program while improving the internal structure [FB13], two major challenges have to be addressed for future language extension refactorings. The first challenge is to determine the applicability of a refactoring on a specific code fragment to ensure that the refactoring does not change the external behavior. The second challenge is the model-to-model transformation that replaces code fragments with more abstract language extensions. As a proof of concept, we implemented refactorings that transform divisions to fractions and for loops to foreach loops. Our implementation uses a simple approach that is sufficient for small abstraction gaps between the base language and the language extensions. However, further research is needed for refactorings of more abstract language extensions, such as state machines.

References

- [AU14] AUTOSAR: AUTOSAR 4.2.1 – 054 – Main Requirements. AUTOSAR, Munich, Germany, 2014.
- [AU15] AUTOSAR: AUTOSAR 4.2.2 – 043 – General Requirements on Basic Software Modules. AUTOSAR, Munich, Germany, 2015.
- [Ca16] Campagne, Fabien: The MPS Language Workbench Volume I. Campagne Laboratory and CreateSpace Independent Publishing, New York, NY and North Charleston, South Carolina, third edition, version 1.5.1, march 2016 edition, 2016.
- [CC90] Chikofsky, E. J.; Cross, J. H.: Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, 7(1):13–17, 1990.
- [CKL04] Clarke, Edmund; Kroening, Daniel; Lerda, Flavio: A Tool for Checking ANSI-C Programs. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *Lecture Notes in Computer Science*, pp. 168–176. Springer, 2004.
- [FB13] Fowler, Martin; Beck, Kent: *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA, 2013.
- [Gr05] Grossman, Dan; Hicks, Michael; Jim, Trevor; Morrisett, Greg: Cyclone: A Type-Safe Dialect of C. *C/C++ Users Journal*, 23(1), 2005.
- [Kä11] Kästner, Christian; Giarrusso, Paolo G.; Rendel, Tillmann; Erdweg, Sebastian; Ostermann, Klaus; Berger, Thorsten: Variability-Aware Parsing in the Presence of Lexical Macros and Conditional Compilation. In: *Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications (OOPSLA '11)*. ACM, New York, NY, USA, pp. 805–824, 2011.
- [MI13] MIRA Limited: *MISRA C:2012*. MIRA Limited, Nuneaton, UK, 2013.
- [MT04] Mens, Tom; Tourwe, Tom: A Survey of Software Refactoring. *IEEE Transactions on Software Engineering*, 30(2):126–139, 2004.
- [MVR14] Molotnikov, Zaur; Voelter, Markus; Ratiu, Daniel: Automated Domain-Specific C Verification with mbeddr. In: *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*. ACM Press, New York, NY, USA, pp. 539–550, 2014.
- [Pa08] Padioleau, Yoann; Lawall, Julia; Hansen, René Rydhof; Muller, Gilles: Documenting and Automating Collateral Evolutions in Linux Device Drivers. In: *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008 (EuroSys '08)*. ACM Press, New York, NY, USA, pp. 247–260, 2008.
- [Sa09] Samek, Miro: *Practical UML Statecharts in C/C++*. CRC Press, Boca Raton, 2. ed. edition, 2009.
- [Si] Siemens PLM Software: *Delivering model-based software engineering for software-intensive systems: with LMS Imagine.Lab Embedded Software Designer*.
- [VKa] Voelter, Markus; Kolb, Bernd: *Lego Mindstorms: an mbeddr Case Study*. http://mbeddr.com/files/mbeddr_casestudy_mindstorms.pdf.

- [VKb] Voelter, Markus; Kolb, Bernd: Smart Meter: an mbeddr Case Study. http://mbeddr.com/files/mbeddr_casesstudy_smartmeter.pdf.
- [Vo12] Voelter, Markus; Ratiu, Daniel; Schaetz, Bernhard; Kolb, Bernd: mbeddr: an Extensible C-based Programming Language and IDE for Embedded Systems. In: Proceedings of the 2012 ACM Conference on Systems, Programming, and Applications: Software for Humanity (SPLASH'12). Association for Computing Machinery, New York, NY, USA, pp. 121–140, 2012. <https://doi.org/10.1145/2384716.2384767>.
- [Vo13a] Voelter, Markus: DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. dslbook.org, 2013.
- [Vo13b] Voelter, Markus; Ratiu, Daniel; Kolb, Bernd; Schaetz, Bernhard: mbeddr: Instantiating a Language Workbench in the Embedded Software Domain. *Automated Software Engineering*, 20(3):339–390, 2013.
- [Vo14] Voelter, Markus: Generic Tools, Specific Languages. Ph.D. dissertation, Delft University of Technology, 2014.
- [Vo15] Voelter, Markus; van Deursen, Arie; Kolb, Bernd; Eberle, Stephan: Using C Language Extensions for Developing Embedded Software: A Case Study. In: Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'15). SIGPLAN notices, ACM Press, New York, NY, USA, pp. 655–674, 2015.
- [Vo17] Voelter, Markus; Kolb, Bernd; Szabó, Tamás; Ratiu, Daniel; van Deursen, Arie: Lessons Learned from Developing mbeddr: A Case Study in Language Engineering with MPS. *Software & Systems Modeling (SoSyM)*, pp. 1–46, 2017.
- [VOR15] Vinogradov, Sergey; Ozhigin, Artem; Ratiu, Daniel: Modern model-based development approach for embedded systems: Practical Experience. In: Proceedings of the 2015 IEEE International Symposium on Systems Engineering (ISSE). pp. 56–59, 2015.

Optimal Product Line Architectures for the Automotive Industry

Tobias Wägemann¹ Ramin Tavakoli Kolagari² Klaus Schmid³

Abstract:

The creation of product line architectures is a difficult and complex task. The resulting architectures must support the required system variabilities as well as further quality attributes. In the automotive domain, product lines of software-intensive system models have a great diversity of products, which leads to vast design spaces. Finding optimal product line architectures as part of the system design process requires the consideration of a variety of trade-offs. In practice, this challenge cannot be solved manually for all but the smallest problems, therefore an automated solution is required. Our contribution is the generation of a sound mathematical formalization of the problem. This formalization makes the product line optimization problem accessible to various established multi-objective optimization techniques. The applicability of the chosen approach is shown by means of applying a commercial tool for multi-criteria decision making.

Keywords: architecture optimization; multi-objective; variability; software product lines; automotive

1 Introduction

An important activity in product line engineering is the development of an adequate product line architecture [CN01, Sc03, Ti12]. Due to the great diversity of products that must be taken into account in this process, finding an optimal product line architecture is a complex and error-prone task. This is particularly the case when performing optimization of product line architectures of software-intensive systems (as opposed to product line architectures of software systems) as this must also account for objectives like weight or production cost. Creating automated support for architecture optimization in such a context can be extremely helpful, as it supports architects in navigating the complex and vast design space which constitutes the basis for a multi-objective decision making problem.

In this paper, we present an approach for product line architecture optimization of software-intensive systems based on EAST-ADL [B113], a domain-specific architecture description

¹ Technische Hochschule Nürnberg, Keßlerplatz 12, 90489 Nuremberg, Germany tobias.waegemann@th-nuernberg.de

² Technische Hochschule Nürnberg, Keßlerplatz 12, 90489 Nuremberg, Germany ramin.tavakolikolagari@th-nuernberg.de

³ Universität Hildesheim, FB 4, Institut für Informatik, Universitätsplatz 1, 31141 Hildesheim, Germany schmid@sse.uni-hildesheim.de

language for the automotive industry. As opposed to other work in this area [RRV16], we provide a full mathematical formalization of the optimization problem. In a previous publication we presented a concept for formalizing product line variability [WW15] as a basis for exploring the design space. Here, we extend our previous work by including design goals and variable realization elements (system components) in the formalization. We also created a prototypical implementation of our approach that uses an off-the-shelf optimization tool for solving the formalized problems. Since our approach builds on EAST-ADL system models as the basis for optimization, we must also note that creating such a model in the first place (including the aggregation of all relevant data) is a significant challenge in itself. However, the language and respective system models are already in industrial use and there are concerted efforts to further promote the industrial application of EAST-ADL by the automotive industry.

While the ideas behind our approach are in principle generic and can be transferred to other layered approaches for representing product line architectures, our implementation focuses on the use of models defined in EAST-ADL. The EAST-ADL language was not developed for (product line) architecture optimization in particular, but provides modeling techniques for automotive systems, including techniques for variation modeling. As a consequence, it contains constructs for variability representation, but not for describing an optimization space. We therefore have to explicitly differentiate between actual *product line variability* and the *architectural degrees of freedom* (i.e., the design space of the product line architecture, cf. Section 3.3). Since both are represented using the same language constructs in EAST-ADL, we introduce a method to differentiate between the two kinds of variation.

The paper is structured as follows: Section 2 gives an overview of the related work and shows the differences between our approach and existing work. Section 3 describes the domain-specific architecture description language (EAST-ADL) used in our work. Section 4 discusses the encoding of the architecture optimization problem as a search problem. The method used for deriving solutions is described in Section 5. Section 6 illustrates our approach by means of a small case study. Finally, Section 7 concludes the paper and describes our plans and ideas for future work.

2 RELATED WORK

There is a range of other work being conducted on the multi-objective optimization of product line system architectures and search-based system design in general. In regard to optimization approaches based on the EAST-ADL language specifically, to our knowledge only one other approach has been realized and published. Walker et al. [Wa13] present an optimization approach based on multi-objective genetic algorithms which considers system dependability, timing concerns and a simple cost metric. The approach uses HiP-HOPS⁴ for fault tree analysis and MAST⁵ for response time computation and is tightly coupled to these

⁴ <http://hip-hops.eu>

⁵ <http://mast.unican.es>

external solutions for the evaluation of objectives. A similar optimization approach for cost and dependability is presented by Mian et al. [Mi15] for the AADL⁶ language, also using HiP-HOPS for fault tree analysis.

Thüm et al. [Th12] present a classification framework for product line analysis strategies to provide systematic access and guidance to the research in this particular field. They divide the analysis strategies into four different categories: product-based, family-based and feature-based analysis, as well as techniques that use combinations of these three. Our approach operates solely on domain artifacts of the product line and can be classified as a family-based analysis strategy in regard to the classifications introduced by this work.

Colanzi et al. present a number of publications in the field of search-based product line design by means of multi-objective evolutionary algorithms. Their work includes an exploratory study of applying search-based design methods to the SPL-context [CV12], the introduction of a novel search-based approach for PLA-design based on PLA-specific metrics by the name of MOA4PLA [Co14], as well as the introduction of a feature-driven crossover operator for PLA optimization using genetic algorithms [CV16].

Aleti et al. [Al13] give a broad overview of common architecture optimization methods used in published work and present a taxonomy for the classification of existing research, based on the three categories Problem, Solution, and Validation. Lopez-Herrejon et al. [LHLE15] present a systematic mapping study on research regarding the application of search-based software engineering methods to the realm of software product lines. The study focuses on the type of employed SBSE techniques, the stage of the affected SPL life-cycle, commonly used validation methods and the specific forums for publication. Ramírez et al. [RRV16] present a comparative study of multi-objective evolutionary algorithms for software architecture optimization. This publication therefore centers on the internals of evolutionary architecture optimization, including an empirical exploration of the behavior of a set of selected multi- and many-objective algorithms in regard to predefined optimization problems with up to nine objectives.

Our approach integrates aspects from three different research fields: software product lines, system architecture modeling and mathematical optimization. In light of related research in this area, our approach is distinct by a combination of the following characteristics: (a) The result of our optimization is not an optimal product but a product line architecture with optimal architectural decisions. (b) The use of multi-objective integer linear programming (MOILP) as a rigorous mathematical formulation of the optimization problem. (c) Adaptability towards tools for optimization and multi-criteria decision making (MCDM) (cf. Section 5).

⁶ <http://www.aadl.info>

3 ARCHITECTURE MODELING APPROACH

EAST-ADL is a domain-specific architecture description language with a focus on capturing all relevant information to represent variant-rich software-intensive systems in a standardized way. The language was developed in a series of European research projects with strong participation of the automotive industry⁷ and applied/enhanced by a number of national and international research projects⁸. Today the EAST-ADL is managed by the EAST-ADL association⁹. The language has been tailored towards compatibility with the well-established AUTOSAR standard¹⁰, which in turn serves as an integral part of the EAST-ADL language by realizing one of its abstraction layers.

This section gives an overview of the EAST-ADL; details about the language can be found in the EAST-ADL white paper [B113] and in the language specification¹¹. Section 3.1 provides an overview of EAST-ADL language; Section 3.2 examines the EAST-ADL variability modeling approach, and Section 3.3 explains the distinction between product line variability and architectural degrees of freedom.

3.1 EAST-ADL—an Architecture Description Language

EAST-ADL defines a language for modeling automotive systems; the implementation of these systems is then managed by AUTOSAR in the aforementioned tight coupling with EAST-ADL. A major advantage of the EAST-ADL language is the organization of the system model along predefined abstraction levels (cf. Figure 1): Abstraction is not only supported in principle—as is often the case for ADLs—but is enforced by the system model with defined semantics for each level of abstraction. As a consequence, engineering information is structured in accordance to a reference methodology based on the V-Model that is widely used in the automotive domain. On each level of abstraction the system is complete from a given perspective: from a very abstract representation at higher levels to increasingly detailed representations at lower levels.

The system model is complemented by several extension packages that allow for modeling requirements, variability, timing, and dependability (cf. Figure 1). Most of the extensions are applicable on all levels of abstraction and are adapted to specific modeling needs of the automotive domain in general and EAST-ADL in particular; while the extensions heavily rely on the “core” system model, the system model itself is independent of the extensions,

⁷ ITEA EAST-EEA (<http://www.itea3.org/project/east-eea.html>), ATESSST, ATESSST2 (<http://www.atesst.org>), MAENAD (<http://www.maenad.eu>)

⁸ Artemis CESAR (<http://www.cesarproject.eu>), ITEA2 SAFE (<http://www.safe-project.eu>), Artemis MBAT (<http://www.mbat-artermis.eu>), to name a few

⁹ <http://www.east-adl.info>

¹⁰ <http://www.autosar.org>

¹¹ The EAST-ADL meta-model is published by the EAST-ADL Association: <http://east-adl.info/Specification/V2.1.12/html/>

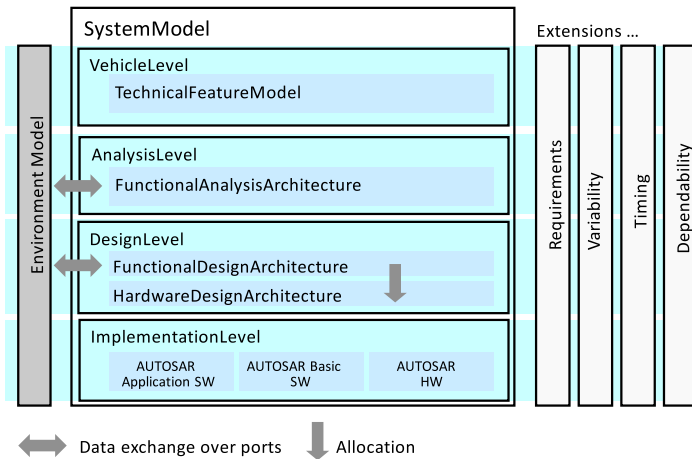


Fig. 1: EAST-ADL System Model [B113, p. 4]

such that extension modeling may be plugged in and out as required. Traceability with full support of SysML semantics is supported for all modeling entities, i.e., between different levels of abstraction as well as within a given abstraction level and to the extensions.

3.2 Variability Modeling Concepts in EAST-ADL

Managing variability is at the core of software product line engineering, it occurs because some aspect of a system may change from one variant of a system to another. EAST-ADL, as a decidedly automotive-specific language, takes these challenges into account and offers variability modeling techniques applicable for the car manufacturer and for the supplier. Its variability management starts on Vehicle Level (cf. Figure 1), where the external (i.e., user-centric/abstract) variability, as well as the product-line-strategy variability is described by cardinality-based feature models [CHE05]. The impact of the described variability is then defined on the Analysis and Design Levels, respectively. Traceability links the abstract/root view on Vehicle Level to the variability impact on lower levels of abstraction, i.e., the artifact levels. While the details of how variability is actually realized in the system are largely suppressed on the Vehicle Level, they are the focus of attention when managing variability on the artifact levels. The artifact levels are Analysis Level, Design Level and the extensions, where variability may occur as well. For example, one may think about the different requirements for a rain sensor with and without a rain light sensor. Variability is described in two ways on the artifact levels:

Feature models used on Analysis and Design Level get a much more concrete meaning as compared to the feature model on Vehicle Level in order to reflect detailed internal

variability as concrete implementation of the variability on the Vehicle Level. Configuration decisions [RTW09] link feature models to one another with the purpose of defining configuration, both within one abstraction level and across abstraction levels. As a consequence of linking artifact level variation to Vehicle Level feature models by means of configuration decisions, all major variability configuration is essentially controlled by the Vehicle Level.

Explicit variation is used to denote that modeling entities may be optional, i.e., be deleted from the system model. Dependencies among variable entities are captured in terms of variation groups. This integrated way of modeling variability can also be linked to configuration decisions such that the (partial) configuration of this variability is guided by feature models on a hierarchy level one step higher and ultimately by the feature models on Vehicle Level.

3.3 Product Line Variability versus Architectural Degrees of Freedom

For the purpose of this paper it is essential to understand the distinction between two kinds of (architectural) variabilities: 1. *Product line variability* [MP07, CN01] describes the variations regarding components (or modules) of proper products that are well-formed with respect to the product line design space. 2. *Architectural degrees of freedom* refer to potential alternatives for designing the product (line) architecture. In other words, the result of configuring all architectural degrees of freedom would be a product line architecture (PLA), whereas the result of configuring all product line variability would be a product. The architectural degrees of freedom are the basis for our optimization process, which intends to produce an optimal product line architecture with respect to (multiple) criteria chosen by an architect. Therefore, at the end of our optimization process, no architectural degrees of freedom remain in the system model and the remaining architectural variability is governed only by the product line design space.

Although product line variability and architectural degrees of freedom indeed both describe variability, their purpose and their role, e.g., in an architecture optimization process, differ considerably. It would therefore be useful for an architecture description language to manage these variabilities differently. As the EAST-ADL does not support this distinction (similar to all other established modeling languages we are familiar with), we use the variability modeling concepts of EAST-ADL for both and differentiate them as follows: we define all artifact variability that can be traced up to the Vehicle Level by means of configuration decisions as product line variability (i.e., all variability that is rooted on Vehicle Level is product line variability). Artifact variability without traceability to the Vehicle Level (i.e., variability that is introduced only at artifact levels) stands for architectural degrees of freedom; such variability is not configured as part of a product line configuration, but is instead decided at the time of system design. Identifying optimal design decisions for the architectural degrees of freedom is the primary objective of our optimization approach.

The chosen approach of distinguishing between product line variability and architectural degrees of freedom is advantageous in that it can be applied to any (variability) modeling approach that supports abstraction, which is indeed common. In the context of the EAST-ADL the chosen approach is especially elegant because of the predefined root abstraction level, i.e., the Vehicle Level.

4 TRANSLATION INTO AN OPTIMIZATION PROBLEM

A general definition of an optimization problem is the problem of finding the best solution in regard to specific criteria from all feasible solutions. Mapped to our problem domain, this definition translates to finding the best product line architectures within the optimization space defined by the architectural degrees of freedom as described by the EAST-ADL system models. The use of EAST-ADL models for architecture optimization requires to formalize all optimization-relevant system information in a way that is sufficient for optimization purposes.

In this section we present our formalization approach for variant-rich EAST-ADL system models, which involves (a) the identification of all model elements relevant for the optimization process, (b) an evaluation of the characteristics of these elements in regard to the intended optimization goals and (c) a generation of a mathematical formulation, which constitutes the basis of our optimization process. We also describe how we handle optimization problems with multiple objectives as part of our approach. It is not the goal of our approach to fully automate architecture definition.

4.1 Multiple Design Objectives

When considering multiple design objectives in a non-trivial optimization process, there is usually no solution that is truly optimal for each of the objectives simultaneously. This is caused by conflicts among the objectives, e.g., making the system more lightweight will likely increase costs, etc. There are two different ways of handling this issue, resulting in two different approaches to multi-objective optimization.

One possibility is to turn the initial multi-objective problem into a pseudo-single-objective problem, by aggregating all considered objectives into a single weighted objective function. The resulting single solution of this approach is optimal in regard to the predefined weights used for the design objective aggregate, which have to be determined before the optimization. This kind of approach is called scalarization or weighted normalization [GR06].

The other possibility, and the one we use in our approach, is to consider all design objectives simultaneously in a specialized optimization process called *Pareto* optimization. The result of a Pareto optimization is not a single solution, but a set of Pareto-optimal solutions, called the *Pareto front*. Pareto optimality is based on the concept of dominance; a solution is called

non-dominated—and is thus part of the Pareto front—if there are no other solutions that are better in at least one objective without degrading one or more of the other objectives [BK05, p. 414ff].

All solutions that are part of the Pareto front, i.e., all dominant solutions, are in principle equally optimal. Selecting the most suitable solution from the Pareto set is therefore subject to a trade-off analysis. This step is dependent on the expertise and the end goals of the user, typically an architect intending to find the architecture that best suits a specific set of requirements. From the perspective of an architect, our approach is therefore a means to efficiently explore the design space of system models, which guarantees that a chosen architecture is Pareto-optimal in regard to the considered design objectives.

4.2 Formalization Approach

In order to establish a sound basis for the exploration of an architecture optimization space, all optimization-relevant information of a given variant-rich system model must be formalized into a rigorous mathematical form. Our optimization problems have binary decision variables and multiple design objectives. Therefore, our problem domain is that of multi-objective integer linear programming (MOILP) with all variables $\in \{0, 1\}$. In order to translate our optimization problems into MOILP form, we first assign all relevant variable elements to numbered decision variables $x_1 \dots x_n$. We can then formulate the program as follows:

$$\begin{aligned} & \text{Minimize} && Cx \\ & \text{subject to} && Ax \geq a_0 \\ & && x \in \{0, 1\}^n \end{aligned} \tag{1}$$

where C is a (m, n) -Matrix of design objective values, A and a_0 are a (p, n) -matrix and a p -vector representing a set of constraints which maps the optimization space and x is an n -vector of binary decisions variables; with m being the number of design objectives, n being the number of decision variables and p being the number of program constraints. The matrix Cx translates to a set of linear objective functions $F(x) = (f_1(x), f_2(x), \dots, f_m(x))^T$, which represent the pursued design objectives.

First of all, a formalization approach must be able to distinguish between product line variability and the system's architectural degrees of freedom (cf. Section 3.3). The intended output of our optimization approach is a product line with Pareto-optimal architectural decisions, not a Pareto-optimal configuration of the product line, i.e., not a product. Therefore, it must be possible to omit all product-line-related variability from the constraint formalization, so that it doesn't get resolved as part of the optimization process. Our approach accomplishes this distinction between product line variability and architectural degrees of freedom by an evaluation of the language traceability across the abstraction levels of the EAST-ADL model. If a variation point is part of the system's product line variability, it must be possible to trace its origin up to the model's Vehicle Level, where the product line design

space is defined by means of feature models. If however the trace ends below the Vehicle Level, the variation point must necessarily be part of the architectural degrees of freedom instead. Using this distinction, we assign decision variables to the variable elements of the architectural degrees of freedom.

In order to generate the objective functions from our variant-rich EAST-ADL models, we parse a type of native EAST-ADL language annotations called *GenericConstraints*. Using predefined *GenericConstraintKinds* like *weight* or *piece cost*, *GenericConstraints* can be used to annotate quantifiable quality attribute information to elements—including variable elements—of the model. The set of possible objectives is therefore defined by the available *GenericConstraintKinds* in the EAST-ADL language specification. To generate the objective functions for these annotations, we allocate the numeric values of the *GenericConstraints* (of one specific *GenericConstraintKind*) to the previously introduced decision variables. In doing so, we produce linear objective functions in the form of $f(x) = c^T x$, where c^T is the transposed vector of the numeric values of the *GenericConstraints* for the variable elements associated with the decision variables x . Performing this step for all pursued design objectives produces a set of linear objective functions.

Next, we formalize the variability information into program constraints. Having established a way of filtering out the (for the formalization process) unwanted product line variability, the formalization of desired variability (i.e., architectural degrees of freedom) into constraints for our MOILP is done by applying a set of transformation rules based on an intermediate conversion into propositional logic. These transformation rules were presented in detail in one of our former publications [WW15], which was focused solely on a method for generating propositional constraints from variability descriptions. In this paper we incorporate this method into a fully-fledged multi-objective optimization approach. Table 1 gives a summary of the rules and Section 6.2 demonstrates their application by means of a case study. With the formalization of (a) quality attributes into design objectives and (b) variability information into program constraints in place, we can now assemble full MOILP representations of optimization problems for variant-rich EAST-ADL models. Our implementation generates these MOILPs in the standard formats of OPL¹² and AMPL¹³.

5 SOLVING OUR OPTIMIZATION PROBLEM

We use our formalization of the optimization problem as input for third-party optimization tooling. Our tool of choice is the commercial optimization software FINNOPT¹⁴, which provides a human decision maker with an interactive process for finding the most preferred compromise among all Pareto-optimal solutions of a multi-objective optimization problem. The FINNOPT approach is inherently iterative and allows the user to guide the process towards preferred solutions as part of a trade-off analysis. FINNOPT is based on the

¹² <https://www-01.ibm.com/software/commerce/optimization/modeling>

¹³ <http://ampl.com>

¹⁴ <http://www.finnopt.com>

Variability		Propositional Logic	Program Constraints
Feature Tree	Feature has parent	$f \rightarrow f_{parent}$	$f_{parent} - f \geq 0$
	Feature is excluded	$\neg f$	$(1 - f) = 1$
	Feature Group	for all m : $f_{parent} \rightarrow M_m(f_1, \dots, f_n)$	for all m : $M_m(f_1, \dots, f_n) - f_{parent} \geq 0$
Feature Link	needs	$f_{start} \rightarrow f_{end}$	$f_{end} - f_{start} \geq 0$
	optional alternative	$\neg(f_{start} \wedge f_{end})$	$f_{end} + f_{start} \leq 1$
	mandatory alternative	$f_{start} \oplus f_{end}$	$f_{start} + f_{end} = 1$
Variation Group	needs	$f_1 \rightarrow (f_2 \wedge f_3 \wedge \dots \wedge f_n)$	$\bigwedge_{k=2}^n (f_k - f_1 \geq 0)$
	optional alternative	for all m : $M_m(f_1, \dots, f_n)$	$f_1 + f_2 + \dots + f_n \leq 1$
	mandatory alternative	$f_1 \oplus f_2 \oplus \dots \oplus f_n$	$f_1 + f_2 + \dots + f_n = 1$
Configuration Decisions		$criteria \rightarrow effect$	$effect - criteria \geq 0$

Tab. 1: Overview of our transformation rules for EAST-ADL system variability [WW15].

IND-NIMBUS [Mi06] software that was developed by the Industrial Optimization Group of the University of Jyväskylä, Finland¹⁵. The tool integrates an external ILP-solver and utilizes it as part of its process. For this purpose we use the commercial solver CPLEX that is part of the IBM ILOG CPLEX Optimization Studio¹⁶ for mathematical optimization.

FINNOPT is well-suited to the task of identifying solutions that are both Pareto-optimal in regard to a preferred emphasis on specific design objectives and useful for a system architect. The software is able to handle large and complex optimization problems and has a user interface that is well suited to analyzing trade-offs for system architectures. However, since we generate our problem formalization in standardized formats (OPL, AMPL), the FINNOPT-based tool setup can in principle quite easily be exchanged with alternative optimization tools for multi-criteria optimization and decision making.

¹⁵ <http://www.mit.jyu.fi/optgroup>

¹⁶ <http://www.ibm.com/software/products/en/ibmilogcplexoptstud>

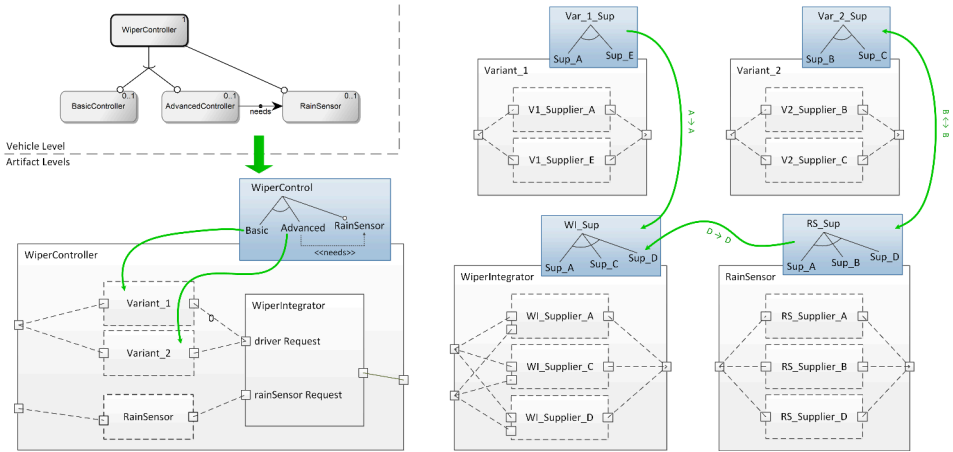


Fig. 2: The wiper control system demonstration model.

6 CASE EXAMPLE

We demonstrate the application of our architecture optimization approach by means of an example. The demonstration model is an extended version of an existing EAST-ADL example, which was previously used for showcasing the variability language concepts of the language in the MAENAD project¹⁷. While the demonstration model is smaller than real-world system models, it adequately serves the purpose of demonstrating our approach in the context of this paper. We will first give an overview of the model’s structure and its contained variabilities. We then demonstrate our MOILP-based formalization approach, before the formalized program is used to explore the optimization space. The resulting Pareto-optimal product line architectures are used for a discussion of the correctness and consistency of the solutions and the adequacy of our optimization approach in general.

6.1 The Demonstration Model

The model used for this demonstration is illustrated in Figure 2. The system shows a product line architecture for the control electronics of a windshield wiper. To keep the example concise, the demonstration model does not show the full EAST-ADL realization, but gives an overview of all elements that are relevant to the optimization process and makes a clear distinction between elements on the Vehicle Level and those on the artifact levels of the model. The level of abstraction used here also coincides with the level of abstraction used in the source document of the model. In this representation, variable elements are indicated by dashed lines, public feature models of containers are shown at the top right of the containers,

¹⁷ MAENAD Concept Presentation on EAST-ADL Variability: http://www.maenad.eu/public/conceptpresentations/6_Variability_EAST-ADL_Introduction_2013.pdf

	<i>weight / g</i>	<i>cost / €</i>	<i>power consumption / W</i>
V1_Supplier_A	122	3.30	2.80
V1_Supplier_E	131	3.25	2.75
V2_Supplier_B	154	5.32	3.10
V2_Supplier_C	154	5.90	3.15
WI_Supplier_A	155	3.54	2.60
WI_Supplier_C	167	3.51	2.55
WI_Supplier_D	158	3.58	2.50
RS_Supplier_A	143	10.65	5.50
RS_Supplier_B	150	10.82	5.50
RS_Supplier_D	126	11.46	5.45

Tab. 2: (Excerpt of) quality attribute values for system components from different suppliers.

and (non-obvious) configuration decisions are represented by arrows from public feature models to configuration targets.

The system can be configured in either a basic or an advanced configuration, which toggles between two different system variants V1 and V2. The system further includes an optional rain sensor that is mandatory for the advanced configuration. These variation points are part of the product line design space.

In addition, the demonstration model also contains variabilities that describe degrees of freedom for the architecture. In this example, these variability descriptions represent alternative—but functionally identical—components from different suppliers and the interdependencies among them. The basic variant V1 and the advanced variant V2 each have two alternative components, the rain sensor and the wiper integrator each have three. The alternative components from different suppliers have varying weight, cost and power consumption (cf. Table 2). The following constraints exist:

- The basic variant V1 from supplier A needs the wiper integrator from supplier A; e.g., because it is a built-in feature of the wiper integrator.
- The advanced variant V2 from supplier B and the rain sensor from supplier B need each other; e.g., because they are sold as a set and not as separate modules.
- The rain sensor from supplier D needs the wiper integrator from supplier D; e.g., because supplier D uses a non-standard proprietary connector.

6.2 Problem Formalization

In order to conduct an automated optimization of the architecture, we first need to formalize all optimization-relevant data of the demonstration model. The first step of

formalization is to assign ordered decision variables to all variable components of the model: $x_1 = V1_Supplier_A$, $x_2 = V1_Supplier_E$, $x_3 = V2_Supplier_B$, $x_4 = V2_Supplier_C$, $x_5 = WI_Supplier_A$, $x_6 = WI_Supplier_C$, $x_7 = WI_Supplier_D$, $x_8 = RS_Supplier_A$, $x_9 = RS_Supplier_B$, and $x_{10} = RS_Supplier_D$.

Next, we formalize the variability annotations of the model. The formalization follows the general process described in Section 4, more specifically the transformation rules shown in Table 1. This step produces the set of equality and inequality constraints for the integer linear program, i.e., the optimization space. The formalization of the degrees of freedom of the demonstration model results in the following set of constraints (with all decision variables $x_i \in \{0, 1\}$):

$$\begin{array}{ll}
 x_1 + x_2 = 1 & x_5 - x_1 \geq 0 \\
 x_3 + x_4 = 1 & x_7 - x_{10} \geq 0 \\
 x_5 + x_6 + x_7 = 1 & x_3 - x_9 = 0 \\
 x_8 + x_9 + x_{10} = 1 &
 \end{array} \tag{2}$$

Our goal is to find a Pareto-optimal product line architecture, not a Pareto-optimal product. Therefore, we only formalize the architectural degrees of freedom, not the product-line-related variability (which will still be present in the resulting architectures). In our demonstration model, the product line variability from the Vehicle Level traces to the WiperControl variation point on the artifact levels, which was thus omitted from the formalization. All other variabilities (i.e., supplier choices and dependencies) have no traceability to the Vehicle Level; they are thus architectural degrees of freedom and are included in the formalization.

The formalization of quality attributes into objective functions naturally use the same assignment of decision variables. The quality attribute values defined in Table 2 result in the following linear objective functions for weight, cost and power consumption, in this order:

$$\begin{array}{ll}
 \text{MIN} & 122 * x_1 + 131 * x_2 + 154 * x_3 + 154 * x_4 + \\
 & 155 * x_5 + 167 * x_6 + 158 * x_7 + 143 * x_8 + \\
 & 150 * x_9 + 126 * x_{10} \\
 \text{MIN} & 3.30 * x_1 + 3.25 * x_2 + 5.32 * x_3 + 5.90 * x_4 + \\
 & 3.54 * x_5 + 3.51 * x_6 + 3.58 * x_7 + 10.65 * x_8 + \\
 & 10.82 * x_9 + 11.46 * x_{10} \\
 \text{MIN} & 2.80 * x_1 + 2.75 * x_2 + 3.10 * x_3 + 3.15 * x_4 + \\
 & 2.60 * x_5 + 2.55 * x_6 + 2.50 * x_7 + 5.50 * x_8 + \\
 & 5.50 * x_9 + 5.45 * x_{10}
 \end{array} \tag{3}$$

Having generated both objective functions and program constraints means that we now have a proper multi-objective integer linear program (cf. Equation 1). This enables us to

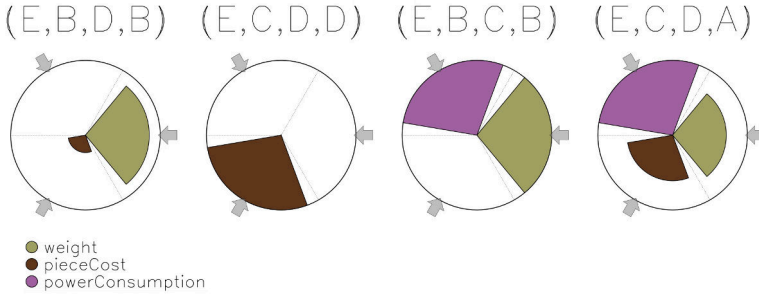


Fig. 3: Petal diagram of the Pareto-optimal solutions.

explore the optimization space and to identify Pareto-optimal solutions for the given quality attributes.

6.3 Discussion of Optimization Results

Even for the small optimization space of our demonstration model, finding Pareto-optimal solutions by hand can already be a tedious exercise. When scaling up the size of the problem to real-world models, finding solutions by hand quickly goes from tedious to impractical to outright impossible. In order to demonstrate our approach, we apply commercial tooling (cf. Section 5) for solving the formalization of our optimization problem and thereby identify four Pareto-optimal solutions. Regarding the consistency of our results, it should be noted that each solution directly corresponds to a real product line architecture for the given system, since all product line variability is still present in the resulting models. In other words, the product line variability space remains intact.

The optimization solutions can be described using a notation of 4-tuples in the form of $(V1, V2, WI, RS)$; e.g., (A, C, A, A) is a solution where $V1$, the wiper integrator and the rain sensor use components from supplier A, and $V2$ from supplier C. Using our optimization tooling, we identified the Pareto solutions (E, B, D, B) , (E, C, D, D) , (E, B, C, B) and (E, C, D, A) . Figure 3 shows a Petal diagram of these solutions, where the petals depict the relative quality of criteria per solution, thereby giving a rough overview of the characteristics of each solution. Petal diagrams are one of the visualization options of the FINNOPT tool (cf. Section 5). Such visualizations can be used (among other methods) to conduct a trade-off analysis of the Pareto-optimal alternatives for a system architecture. Pareto analyses for large and complex real-world industrial models are a non-trivial task that requires considerable system expertise from the conducting architect or engineer.

The Pareto set for our case study example has a particularly interesting characteristic: out of all products that can be configured from the given product line design space, the most lightweight possible product (277g, $V1$ from supplier A, WI from supplier A, no rain

sensor) is not part of the most lightweight product line architecture (E, C, D, D) (its most lightweight product has 289g). Effects like this are a consequence of family-based analyses like our product-line-aware optimization and must be taken into account when considering the adequacy of Pareto-optimal product line architectures for specific use cases.

7 CONCLUSIONS

In this paper we introduced a novel approach for product line architecture optimization and demonstrated its application by means of a case study. The approach defines a complete mathematical formalization as a multi-objective integer linear program that comprises the architectural degrees of freedom, the implementation components and the design objectives (cf. Section 4). We demonstrated that our approach has characteristics that differentiate it both from our previous work and from other research in this area. Furthermore, our approach is not merely theoretical, but has been implemented with the help of off-the-shelf optimization tooling (cf. Section 5).

As future work, we plan to compare our approach to other optimization frameworks in terms of efficiency and optimality of the results. We also plan to address variants of the given problem domain like an (optional) inclusion of specific border conditions, e.g., optimal product variants that shall always be part of a resulting product line architecture. Furthermore, we plan to identify useful application scenarios for the so far unused formalization of product line variability in addition to our current approach of optimizing only the architectural degrees of freedom. We plan to continuously evaluate our efforts with the aid of industry experts from the automotive domain. A first round of evaluations has already been concluded and yielded positive feedback.

References

- [AI13] Aleti, A.; Buhnova, B.; Grunske, L.; Koziolok, A.; Meedeniya, I.: Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 39(5):658–683, 2013.
- [BK05] Burke, E. K.; Kendall, G.: *Search methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 2005.
- [B113] Blom, H.; Lönn, H.; Hagl, F.; Papadopoulos, Y.; Reiser, M.-O.; Sjostedt, C.-J.; Chen, D.-J.; Tavakoli Kolagari, R.: , White Paper Version 2.1.12: EAST-ADL - An Architecture Description Language for Automotive Software-Intensive Systems, 2013.
- [CHE05] Czarnecki, K.; Helsen, S.; Eisenecker, U.: Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.
- [CN01] Clements, P.; Northrop, L.: *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, 2001.

- [Co14] Colanzi, T. E.; Vergilio, S. R.; Gimenes, I.; Oizumi, W. N.: A search-based approach for software product line design. In: Proceedings of the 18th International Software Product Line Conference-Volume 1. ACM, pp. 237–241, 2014.
- [CV12] Colanzi, T. E.; Vergilio, S. R.: Applying search based optimization to software product line architectures: Lessons learned. In: International Symposium on Search Based Software Engineering. Springer, pp. 259–266, 2012.
- [CV16] Colanzi, T. E.; Vergilio, S. R.: A feature-driven crossover operator for multi-objective and evolutionary optimization of product line architectures. *Journal of Systems and Software*, 121:126–143, 2016.
- [GR06] Grodzevich, O.; Romanko, O.: Normalization and other topics in multi-objective optimization. In: Proceedings of the Fields–MITACS Industrial Problems Workshop. The Fields Institute, pp. 89–102, 2006.
- [LHLE15] Lopez-Herrejon, R. E.; Linsbauer, L.; Egyed, A.: A systematic mapping study of search-based software engineering for software product lines. *Information and software technology*, 61:33–51, 2015.
- [Mi06] Miettinen, K.: IND-NIMBUS for demanding interactive multiobjective optimization. *Multiple Criteria Decision Making '05*, 1:137–150, 2006.
- [Mi15] Mian, Z.; Bottaci, L.; Papadopoulos, Y.; Sharvia, S.; Mahmud, N.: Model transformation for multi-objective architecture optimisation of dependable systems. In: *Dependability Problems of Complex Information Systems*, pp. 91–110. Springer, 2015.
- [MP07] Metzger, A.; Pohl, K.: Variability management in software product line engineering. In: *Companion to the proceedings of the 29th International Conference on Software Engineering*. IEEE Computer Society, pp. 186–187, 2007.
- [RRV16] Ramírez, A.; Romero, J.; Ventura, S.: A comparative study of many-objective evolutionary algorithms for the discovery of software architectures. *Empirical Software Engineering*, 21(6):2546–2600, 2016.
- [RTW09] Reiser, M.-O.; Tavakoli Kolagari, R.; Weber, M.: Compositional Variability: Concepts and Patterns. In: *42nd Hawaii International Conference on System Sciences*. pp. 1–10, 2009.
- [Sc03] Schmid, K.: A Quantitative Model of the Value of Architecture in Product Line Adoption. In: *Proceedings in the 5th International Workshop on Product Family Engineering*. 2003.
- [Th12] Thüm, T.; Apel, S.; Kästner, C.; Kuhlemann, M.; Schaefer, I.; Saake, G.: Analysis strategies for software product lines. *School of Computer Science, University of Magdeburg, Tech. Rep. FIN-004-2012*, 2012.
- [Ti12] Tischer, Christian; Boss, Birgit; Müller, Andreas; Thums, Andreas; Acharya, Rajneesh; Schmid, Klaus: Developing Long-Term Stable Product Line Architectures. In (de Almeida, E. Santana; Schwanninger, C.; Benavides, D., eds): *Proceedings of the 16th International Software Product Line Conference (SPLC '12)*. volume 1. ACM, pp. 86–95, 2012.
- [Wa13] Walker, M.; Reiser, M.-O.; Tucci-Piergiovanni, S.; Papadopoulos, Y.; Lönn, H.; Mraidha, C.; Parker, D.; Chen, D.-J.; Servat, D.: Automatic optimisation of system architectures using EAST-ADL. *Journal of Systems and Software*, 86(10):2467–2487, 2013.
- [WW15] Wägemann, T.; Werner, A.: Generating Multi-objective Programs from Variant-rich EAST-ADL Product Line Architectures. In: *GI-Jahrestagung*. pp. 1673–1685, 2015.

Extending a UML and OCL Tool for Meta-Modeling: Applications towards Model Quality Assessment

Khanh-Hoang Doan¹, Martin Gogolla¹

Abstract:

For developing software in a model-driven style, meta- and multi-level modeling is currently gaining more and more attention. In this contribution, we propose an approach to extend a two-level modeling tool to three-level modeling by adding a meta-model at the topmost level. Standard OCL does not support reflective constraints, i.e., constraints concerning properties of the model like the depth of inheritance. By adding an auto-generated instance of the topmost level to the middle level, we can offer an option for writing reflective constraints and queries. We apply the extension to demonstrate the usefulness of meta-modeling for model querying and model quality assessment. A first proposal towards level-crossing constraints is also put forward.

Keywords: UML; OCL; Meta-modeling; Reflective constraints; Model querying; Model quality assessment.

1 Introduction

Within software development, Model-Driven Engineering (MDE) is playing now a more and more important role. MDE considers models as central development artifacts, for example by combining the UML (Unified Modeling Language) [Ob15b], and the OCL (Object Constraint Language) [CG12]. Meta-models play a crucial role in modeling as they define the structure of models, and meta-modeling [AK03, Bé05] and multi-level modeling [At14, At15, AGC16] has become a major research topic. Meta-modeling is closely connected to multi-level modeling because a UML and OCL model, which we call a *user model*, can be regarded an *instance model*, i.e., instantiation of a metamodel. The user model in turn may act as a *type model*, which may be instantiated again to a run-time instance model. Following this process, one obtains a modeling architecture with at least three levels.

Our starting point is a version of the tool USE (Uml-based Specification Environment) [GBR07, GH16] that supports two-level modeling. In this contribution we show how to extend the tool to three levels of modeling by adding the OMG (Object Management Group) UML meta-model to the topmost level. In order to make the work with different modeling levels easier, we provide the option to automatically take a simplified view on

¹ University of Bremen, Computer Science Department, E-Mail: {doanhk.gogolla}@informatik.uni-bremen.de

the meta-model based on the elements of the current input user model. To offer access to the meta level, a meta-model instance corresponding to the user model is automatically generated and added to the middle level. By doing that, the user model turns into a instance model, and therefore modelers can write OCL constraints and queries about the model itself. We call this type of OCL expressions *reflective* constraints and queries. Thus the work in this paper supports meta-models and reflective OCL constraints and queries, i.e., expressions that treat the user model itself as data. Reflective querying helps to explore, understand and validate models, especially when we model large, complicated systems. For example, we can utilize reflective constraints to analyze a model and check for internal quality of a class diagram. In this paper, we also show the application of reflective constraints for *model quality* evaluation. The quality properties that we deal with in this paper are the problems on the class diagram might relate to general design issues, e.g., the naming of elements, or to questions regarding model metrics. Some similar approaches in this field have been presented, for example in the works [Ag11, AGO12, LGdL14]. Developers can use an appropriate method to evaluate their model and find drawbacks and problems in the model. The quality assessment method introduced in this contribution uses standard OCL in our three-layer modeling approach. Our three-layer modeling approach also offers an extension of standard OCL for level-crossing constraints, which are across all three levels, since the meta objects and run-time objects are accessible at the same time. An idea of extending OCL for level-crossing constraints is also presented in the later part of this paper.

The rest of the paper is organized as follows. Section 2 presents the general idea to extend our two-level tool to a three-level modeling tool. In Section 3 we show how to write reflective queries in the extended tool based on the available meta-model. The approach using reflective OCL constraints for quality assessment is introduced in Sect. 4. Section 5 presents a first proposal towards level-crossing constraints. The paper ends with some concluding remarks and future work in Section 7.

2 Tool-supported Meta Modeling

2.1 Meta-modeling in UML

The OMG has defined the Meta-Object-Facility (MOF) [Ob15a] as a fundamental standard for modeling. MOF provides a four-layer architecture for system modeling (three of them are shown in Fig. 1). Generally speaking, adjacent layers in the architecture are related by the instance-of relationship. This means that a lower layer is used for instantiating the next upper layer. One could also say that the same entities at a middle layer M_i can be (A) objects for the next upper layer M_{i+1} and (B) classes for the entities at the next lower layer M_{i-1} .

The top layer in the MOF architecture, named M3, is a meta-meta model. This meta-meta model is the language used to build the metamodels at the lower layer, called M2. The UML metamodel, which is used to describe the UML, is the most well-known example of a model

at M2 layer. The models at layer M2 describe the elements and the structure of the models at layer M1. Models at layer M1 can be, for instance, models written in UML. The last and bottom layer in the MOF architecture is the layer M0 (also called data layer). Models at this layer describe run-time instances (representations of real-world objects).

2.2 Basics of Meta Modeling in USE

In previous works, we have introduced USE (UML-based Specification Environment) [GBR07, GH16] as a two-level modeling tool, in which a user UML and OCL model at level M1 (class diagram and constraints) and run-time instances at layer M0 (object diagrams) are provided. In [Go15], ideas for experimenting with multi-level models in the two-level tool USE are presented. In the current contribution, we introduce an approach, in which the MOF architecture is integrated into USE for meta modeling. Roughly speaking, we now make the third OMG layer M2 explicitly available. Fig. 1 shows the general schema for our three-level modeling approach in the new version of USE. The model at layer M2 is the UML meta-model (the OMG superstructure) [Ob11]. This meta-model itself is an explicit UML class model formulated in USE with (currently) 63 classes and 99 associations. It is preloaded as a type model for all user UML models at layer M1 and is fixed during the modeling process. In the middle of our three-layer modeling approach there is a user UML and OCL model at layer M1, which is highlighted by the dashed rectangle in Fig. 1. The key point that makes our approach available for writing reflective constraints is an auto-generated meta-instance of the meta-model added to M1 level. Each meta-object is an instance of a meta-class and the number of generated meta-objects and links is based on the input user model. For example, if there are two classes in the user model, then two instances of meta class `Class` are generated. Our approach visits all user model elements (e.g., classes, attributes, operations, associations) and generates the corresponding meta-objects and links. Table 1 shows the mapping between the user model elements and the related meta-model classes and associations, which are the type elements for the generated meta-objects and links. In this work, we use the USE specific language SOIL (Simple OCL-based Imperative Language) [BG11] to create these meta-objects and links.

2.3 Three-layer Model Representation in USE

As discussed before, in the M1 layer we provide two views on the user model. The first view is a class diagram view, which can be seen as a type model for the object model at the lower layer M0. The second view is an object diagram view that is an instantiation of the UML meta-model at level M2 and that corresponds to the loaded UML user model. These two views represent the same information and are always in sync.

Fig. 2 contains an example of a three-level modeling representation in USE. The user model in this example is a simple model, with two classes, `Employee` and `Department`, and an

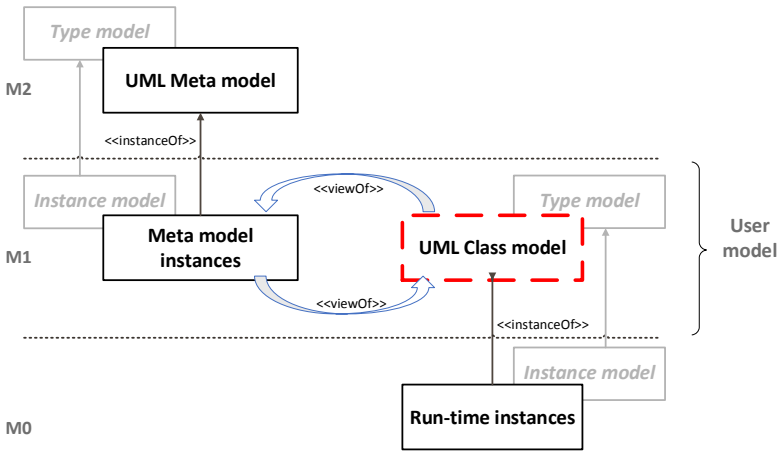


Fig. 1: General schema for three-layer model representation.

association WorksIn as shown in the right middle part of Fig. 2. As mentioned above, the full meta-model includes 63 classes and 99 associations. Therefore, viewing the full meta-model is not practical and sometimes not necessary, because many of the meta-model elements might not be used to describe the current user model. For example, the meta-model class Operation is not used to describe any element in the Employee-Department model. Starting from these observations, we provide a simplified view for the meta-model, as shown in the upper part of Fig. 2. To construct the simplified view from the full meta-model, we drop all unnecessary classes and associations, which are not needed for any element of the current user model. In the simplified view, we only show the meta-model elements, i.e., classes and associations, that the user model needs as type elements to instantiate model elements. Table 1 shows the mapping between the user model elements and the related meta-model classes and associations.

In each row, the item in the first column is a user model element, and the items in the second column are the classes that directly relate to the user model element, i.e., a typing model element. The third column contains the related meta-model associations (the subscript text includes the names of association ends corresponding to the classes). Based on this mapping, we can detect which meta-model elements will be displayed in the simplified view. For example, if a class in the user model contains attributes then the Property metaclass and two associations, i.e., $Class_{class} - Property_{ownedAttribute}$ and $DataType_{DataType} - Property_{ownedAttribute}$ will be shown. Concerning the example and as the result of the described mapping, only three classes, Class, Property, Association, and the corresponding associations are shown in the simplified meta-model view for the Department-Employee user model. Additionally, we still provide a full meta-model view, in case the developer wants to explore it.

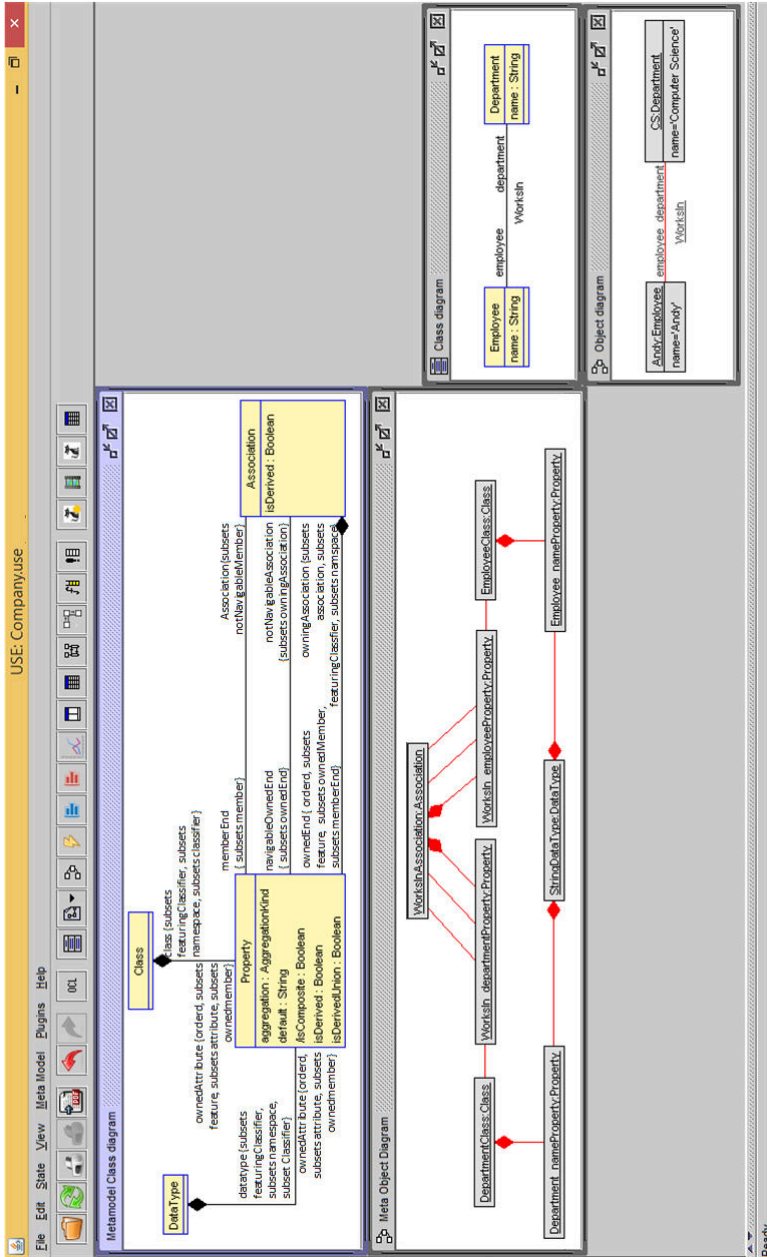


Fig. 2: Three-layer model representation in USE.

The left middle part in Fig. 2 is the user model represented as an instance of the meta-model. As can be seen, every element of the user model is an instance of a class from the meta model. Each instance is named as a combination of the name of the corresponding element from the user model and the meta-model class from which it is instantiated. For example, the object `WorksInAssociation` is an instance of metaclass `Association` and its name combines ‘WorksIn’, the name of the association from user model, and ‘Association’, the name of the meta-model class. The object diagram shown in the lower part of Fig. 2 is a run-time instance of the user model. It is the model at layer M0. ‘CS’ and ‘Andy’ are instances of the `Department` class and the `Employee` class, respectively.

There is a number of derived links between objects in the meta-instance view. However, to make the meta-instance view at layer M1 more focussing on the `instanceOf` aspect, we only show the direct links and do not show these derived links. The two views in layer M1 describe one model. They are equivalent and kept in sync. Each element in the user model class diagram view, e.g., a class, an attribute or an association, is presented as a meta-instance in the meta-instance view. If there is any change in the user model, e.g., a name change, an addition or a deletion of an element, the object diagram in meta-instance view will be updated. An example of a synchronous change on the views at layers M1 and M2 is presented and highlighted in Fig. 3. The change on the user model is made by

Tab. 1: Relationship between user model elements and meta-model elements.

User model elements	Related UML meta-model classes	Related UML meta-model associations
Class	Class	
Attribute	Property	<code>Class_{class} – Property_{ownedAttribute}</code> <code>DataType_{dataType} – Property_{ownedAttribute}</code>
Association	Association	<code>Class_{endType} – Association_{association}</code>
Association End	Property	<code>Association_{association} – Property_{memberEnd}</code> <code>Association_{association} – Property_{navigableOwnedEnd}</code> <code>Association_{association} – Property_{ownedEnd}</code>
Operation	Operation	<code>Class_{class} – Operation_{ownedOperation}</code> <code>DataType_{dataType} – Operation_{ownedOperation}</code>
Parameter	Parameter	<code>Operation_{operation} – Parameter_{ownedParameter}</code>
AssociationClass	AssociationClass	
Generalization	Generalization	<code>Class_{class} – Class_{superClass}</code> <code>Generalization_{Generalization} – Class_{specific}</code> <code>Generalization_{Generalization} – Class_{general}</code>
Redefined Attribute/ Redefined Association End		<code>Property_{property} – Property_{redefinedProperty}</code>
Subsetting Attribute/ Subsetting Association End		<code>Property_{property} – Property_{subsettingProperty}</code>

(A) adding the operation ‘numberOfEmp(): Integer’ into class Department. Consequently, two corresponding meta-instances are synchronously (B) added to the meta-instance view, i.e., the Department_numberOfEmp:Operation instance and the IntegerDataType:DataType instance for the return data type ‘Integer’. And synchronously, the metaclass Operation and related associations are (C) added to the simplified meta-model view (at layer M2).

3 Tool-based Reflective Querying

The access to the meta-level supported by standard OCL [Ob06] is limited, therefore writing reflective queries, e.g., “find the classes related to a given class *c* and their relevant roles”, is impossible. In this section, we will introduce how our approach supports more meta-level access capabilities for writing reflective queries within the extended tool.

3.1 Meta-level Accessibility in OCL

Standard OCL is a formal language for writing constraints and queries on UML models. OCL expressions are formulated on the level of classes (M1) and their semantics is applied on the level of objects (M0). Given a meta object *t*: OclType, the following table shows the list of supported OCL meta-level access capabilities.

Tab. 2: OCL built-in meta-level access

Expression	Semantics
<code>t.name() : String</code>	Get the name of the type <i>t</i>
<code>t.attributes() : Set(String)</code>	Get the set of names of all attributes of <i>t</i>
<code>t.operations() : Set(String)</code>	Get the set of names of all operations of <i>t</i>
<code>t.associationEnds() : Set(String)</code>	Get the set of names of all association ends navigable from <i>t</i>
<code>t.supertypes() : Set(OclType)</code>	Get the set of all direct supertypes of <i>t</i>
<code>t.allSupertypes() : Set(OclType)</code>	Get the transitive closure of the set of all supertypes of <i>t</i>

As we can see, with these limited meta-level access capabilities, standard OCL cannot express a number of reflective queries and constraints. The following list presents several reflective queries that cannot be expressed with the standard OCL.

1. Find all classes related to a given class
2. Find names of all subclasses of a given class
3. Find all abstract classes
4. Find all classes that have more than 10 attributes

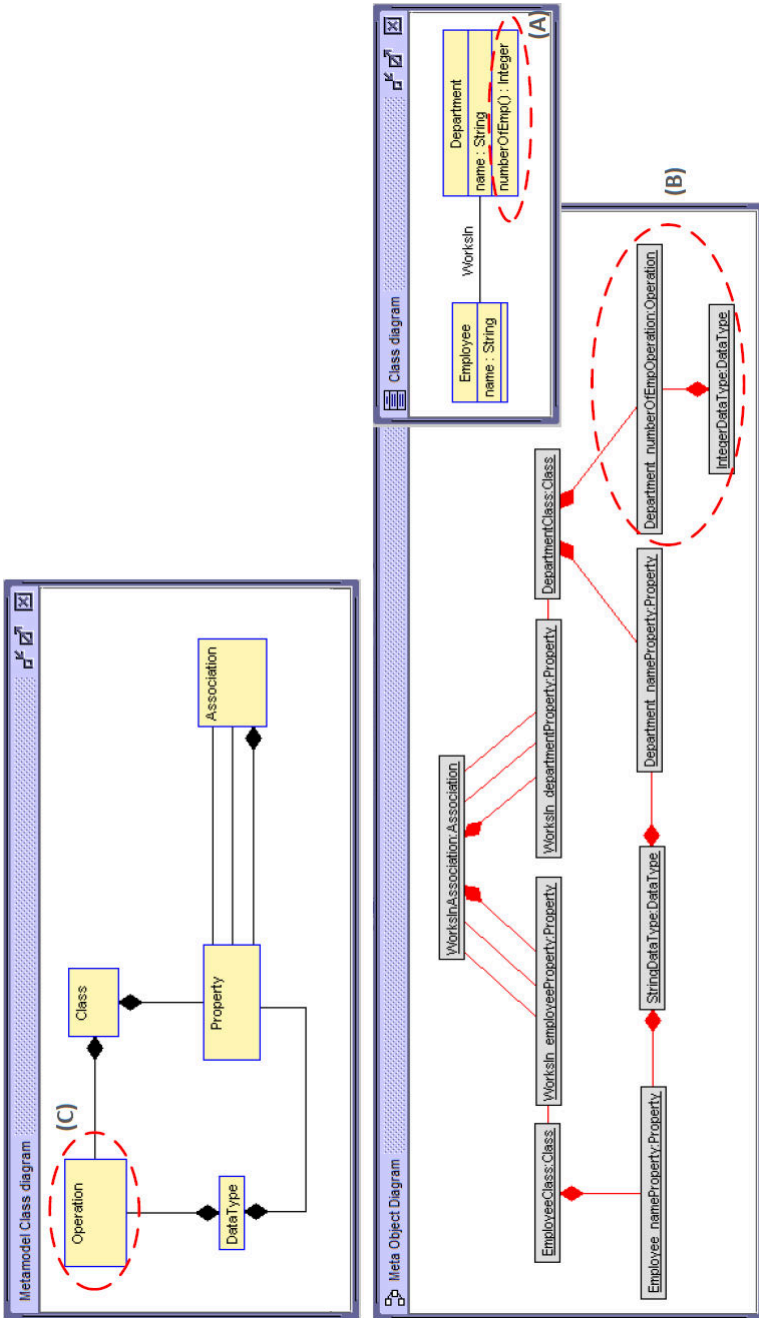


Fig. 3: Synchronous changes on the views of layer M1 and M2.

5. Find all classes that have more than 5 subclasses
6. Calculate the number of classes in a user model
7. Check for the setter and getter methods of all attributes
8. Find all classes of a user model that have no subclass

These queries, however, can be expressed with our three-level modeling approach introduced in the previous section. In the next section, we will show how to formulate and execute reflective queries in the extended tool.

3.2 Writing Reflective OCL Querying in Tool USE

As introduced in the previous section, our approach supports a three-layer UML and OCL specification: instances, model, and meta-model. Through the tool support, one can access the meta-model and create OCL queries for the user model by considering it as an instance of the meta-model. Model querying using the meta-model approach provides possibilities for considering the elements contained in a model, for example, by accessing the attributes, operations, and referenced elements of a given model element, by executing comprehensions and quantified expressions. A query is an OCL expression on the meta-model layer, and the result is a Boolean value or a set of user model elements in form of instances of a meta-model type element. Model queries such as “find the classes related to a class and their relevant roles” cannot be formulated in OCL directly. However, our meta-modeling approach can deal with this kind of model query. For example, the query “find classes related to class Department via an association” on the Employee-Department example can be formulated by the following OCL expression and executed by our tool as shown in Fig. 4.

The Association meta class is the type element for associations in the user model and endType is an end of a derived association that can be used to navigate from the Association meta class to the Class metaclass. DepartmentClass is the meta-instance of the Class metaclass; its name is a combination of the user model element and the corresponding metaclass. The result of executing this query, i.e., Bag{EmployeeClass}, is also shown in Fig. 4.

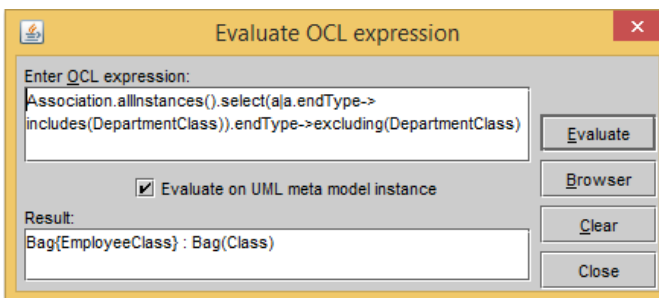


Fig. 4: Model query example: Find related classes.

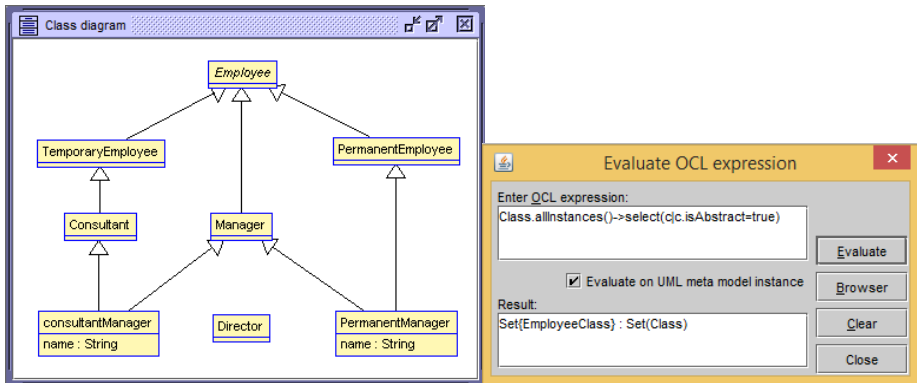


Fig. 5: Model query example: Find abstract classes.

Another example for model querying is presented in Fig. 5. The example there is an Employee hierarchy model, a typical subclass-superclass generalization model with a four-level inheritance structure. For example, one might want to find all abstract classes within this model. The OCL query to perform this task is stated in Fig. 5. In particular, `isAbstract` is a Boolean attribute of the metaclass `Class` in order to define whether a class at level M1 (in a user model) is abstract or not.

4 Model Quality Assessment with Reflective Constraints

Writing reflective constraints is now possible with our meta-modeling approach. Reflective constraints can be exploited for many applications, one of them is model quality assessment. Model quality assessment helps modelers to detect errors or mistakes on their models, to fix bugs and to improve the models. These assessment properties might include design properties: absence of isolated classes, respecting naming conventions (e.g., the name of every element must obey the camelCase convention) or metrics properties (e.g., a generalization hierarchy is not too deep). By visual inspection, we can identify several quality problems in the example model in Fig. 5:

1. There is one isolated class, i.e., `Director`. An isolated class is a class which is not involved in an association or in the inheritance hierarchy.
2. The name of the class `consultantManager` does not start with a capital letter (assuming the class names should obey the camelCase convention).
3. The attribute `name : String` is repeated in all subclasses of the class `Manager`. It should be defined in the superclass.

However, evaluating and detecting these kinds of quality problems on large and complicated models might take time and might even be impractical. In this section, we introduce a

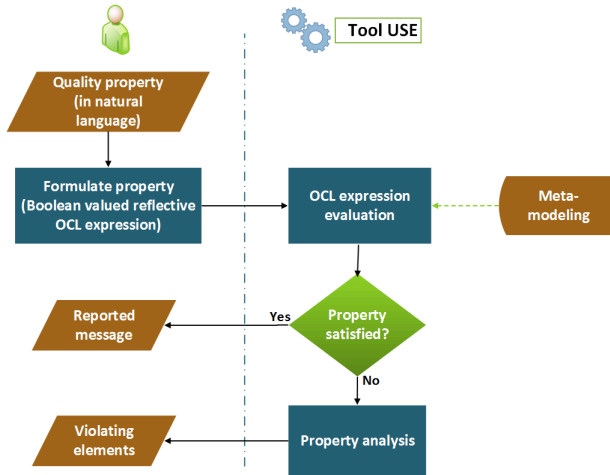


Fig. 6: The workflow of model quality assessment process

proposal that employs OCL utilizing the meta-level modeling approach as presented before. Thus, we can automatically evaluate quality properties of a user model.

Fig 6 shows the workflow of the model quality evaluation process. Firstly, the property must be formulated by the developer as a Boolean-valued reflective OCL expression. The next evaluation and analysis steps will be performed by tool USE. The reflective OCL expression will be evaluated. If the evaluation yields True, the property is satisfied and the model respects this property. On the contrary, if the property fails, the developer might be interested in the parts of the model that violate the property. Returning to the example in Fig. 5, we want to check the first problem mentioned in the section beginning, i.e., whether there are isolated classes in the user model. To achieve this, we formulate an OCL expression for the property.

```

Class.allInstances()->select(c | c.typeElement->isEmpty() and
c.superClass->isEmpty() and c.subClass->isEmpty())->isEmpty()
    
```

In this example, we navigate from a metaclass *c* to related associations through the *typeElement* role name. The *superClass* and *subClass* role names are used to navigate from the metaclass *c* to its superclass and subclass, respectively. The assessment result is False as shown in Fig. 7. That means the property is violated. It can be seen from the user class diagram in Fig. 5 that there is one isolated class, i.e., the class *Director*.

In the case of simple models, one can manually figure out the elements that cause the violation of an assumed property. However, with a large, complicated model, this can be hard work and can sometimes be impossible. Our tool supports designers to analyze such properties and to look for the reason for the unsatisfiability. Particularly the USE evaluation

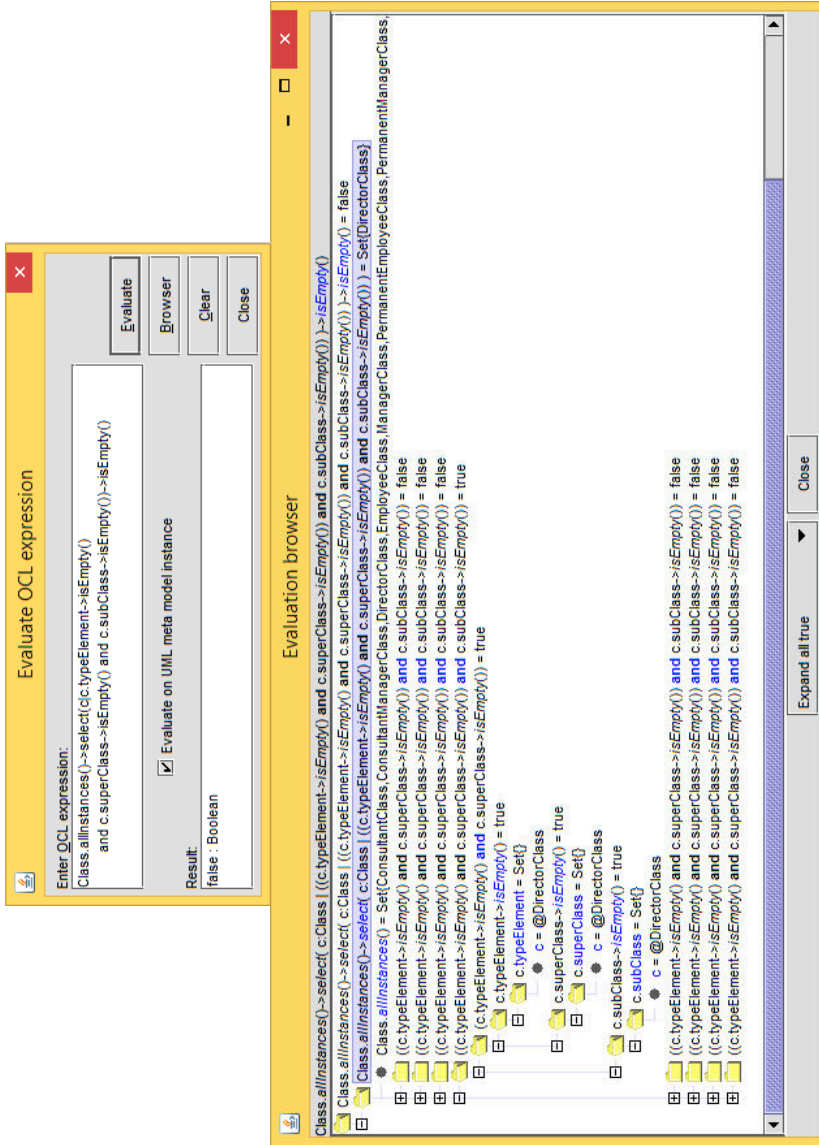


Fig. 7: Example of model assessment and analysis.

browser allows developers to dive into the details of the formula evaluation and identify the spots in the object diagram that contribute to the fact that the formula is not satisfied.

The USE evaluation browser in the lower part of Fig. 7 can be obtained by clicking the ‘Browser’ button on the right hand side of the OCL expression evaluation window in Fig. 7. The browser window decomposes the expression into sub-parts in a hierarchical structure, and every part is evaluated. From the evaluation browser, we can see that there is only one violating element, i.e., one isolated class, which is the Director class. For further analysis, one can expand sub-expressions and explore the evaluation of other sub-parts of the formula as shown in Fig. 7.

5 Towards an Approach for Level-Crossing Constraints

Level-crossing plays an important role in multi-level modeling. Standard OCL, however, only supports formulating constraints on the level of classes (M1) and their semantics concerns the level of objects (M0). That means it is impossible to write expressions on objects at different linguistic levels, e.g., the expression “Class.allInstances().allInstances()” is syntactically invalid in OCL. Looking back to our approach on three-level modeling, we can see that developers now can access meta objects as well as run-time objects at the same time. Therefore, we can say that our three-level modeling approach offers enough semantics for extending OCL for multi-level-crossing constraints. Let us consider the following level-crossing constraint, which is an invalid, ill-typed constraint in standard OCL.

```
Class.allInstances().allInstances()->forAll(age>18)
```

Assuming every class in the model has an age attribute with Integer type, this constraint ensures that the value of the age attribute of all instances of all classes is over 18. As we can see, the semantics of the first part of the constraint, i.e., Class.allInstances(), concerns the meta-level, and the semantics of the second part of the expression, i.e., allInstances()->forAll(age > 18), concerns the model level. The result of the meta-level part has type Set(Class). Unfortunately, the allInstances() operation (in standard OCL [Ob06]) is applicable only on type Class.

To overcome this issue, one could work with the forAll collection operation between the meta-level expression and the model-level expression. If we handle a term of type Set(Class) with the forAll iterator this gives the option to access a single Class, on which the model-level expression can be applied.

```
Class.allInstances()->forAll(c | # c.allInstances()->forAll(age > 18) # )
```

Naturally, we have to distinguish meta-level sub-expressions and model-level sub-expressions. In other words, we have to indicate in OCL, which sub-expression belongs to which level (meta-level or model-level). To achieve this, we introduce an additional notation in OCL expressions, i.e., #...#. This indicates that the expression within #...# is a model-level

expression. As the result, we propose a formula template for a level-crossing constraint as shown below.

```
<meta-level OCL expression>->forAll(c | # <model-level OCL expression> #)
```

Generally speaking, using this formula template, one can write constraints that go from the M2 to the M0 level through the M1 level. This capability supports writing more powerful and flexible constraints. Instead of using the universal quantification it would also be possible to use an existential quantification. Other OCL collection operations, e.g. one, are feasible as well. Working out details is subject to future work. Our proposed level-crossing formula template has a few restrictions: (a) the meta-level expression must return the type `Set(Class)`, (b) the model-level expression must be a Boolean-valued expression, and (c) the result of the overall level-crossing expression is always a Boolean value.

6 Related Work

There is a number of other proposals, which are related and similar to our work, that have been introduced in recent years. The tool Melanee [AG12] is designed as an Eclipse plug-in, supports strict multi-level metamodeling and facilitates general purpose languages as well as domain specific languages. Another tool is MetaDepth [dLG10] allowing linguistic as well as ontological instantiation with an arbitrary number of meta-levels supporting the potency concept. The framework Modelverse introduced in [Mi14] can be used to model a four-level language hierarchy. The work in [BKK16, IGS14] uses F-Logic as an implementation basis for multi-level models including constraints. In contrast to these approaches, our contribution deals with traditional two-level UML/OCL modeling approaches by extending them for meta-modeling and exploits added meta-data for writing reflective constraints and level-crossing constraints with OCL. One commonality between our work and these above mentioned approaches is the introduction of elements in the middle level that have both type and instance facets.

The idea of copying the M2-model instance to lower levels in the MOF meta-model architecture and exploiting it for reflective constraint writing is also presented in [Dr16]. In that paper, the meta instance is added to the M0 level, together with the run-time instances, through instantiation and reification processes. Adding elements to the M0 level is a major difference between the work in [Dr16] and our approach, because in our work, the meta instance is generated and added to the M1 level. Therefore, in order to write a reflective constraint or query with our approach, we only need to go from the M2 level to the M1 level. This means we do not need to extend the OCL for writing reflective constraints or queries.

7 Conclusion

This contribution has proposed an extension of the tool USE that supports three-level modeling where the middle level can be seen at the same time as an object diagram, i.e.,

the instantiation of the upper level model, and as a class diagram, i.e., the type model for the lower level. Based on these ideas, we present an approach for reflective constraints and queries within the extended tool and the application of this approach to model quality assessment. A first proposal towards level-crossing constraints was also introduced: a proposal that offers writing more powerful and flexible constraints.

Future work includes the following topics. First of all, we would like to work out within our approach formal definitions for notions like potency or strictness. Developer support for these notions should then be explored. The user interface in our tool USE for model querying and quality evaluation can be strengthened as well. For instance, one option might be to highlight the result of meta-level queries in the user class diagram. Another open item would be to implement a library of pre-defined quality assessment properties. With the integration of three-level modeling in the tool USE, more work on model metrics seems to be a promising direction for a USE extension. The proposal for level-crossing constraints must be implemented and extended to cover other formula templates. Last but not least, complex examples and case studies, especially case studies from large applications, must check the practicability of the proposal.

References

- [Ag11] Aguilera, David; García-Ranea, Raúl; Gómez, Cristina; Olivé, Antoni: An Eclipse Plugin for Validating Names in UML Conceptual Schemas. In: Proc. ER 2011 Workshops FP-UML, MoRE-BI, Onto-CoM, SeCoGIS, Variability@ER, WISM. pp. 323–327, 2011.
- [AG12] Atkinson, Colin; Gerbig, Ralph; Melanie: Multi-level Modeling and Ontology Engineering Environment. In: Proc. 2nd Int. Master Class MDE: Modeling Wizards, co-located with MODELS 2012. MW'12, pp. 7:1–7:2, 2012.
- [AGC16] Atkinson, Colin; Grossmann, Georg; Clark, Tony, eds. Proc. 3rd Int. Workshop Multi-Level Modelling co-located with MoDELS 2016, volume 1722 of CEUR Workshop Proceedings. CEUR-WS.org, 2016.
- [AGO12] Aguilera, David; Gómez, Cristina; Olivé, Antoni: A Method for the Definition and Treatment of Conceptual Schema Quality Issues. In: Proc. 31st Int. Conf. ER 2012. pp. 501–514, 2012.
- [AK03] Atkinson, C.; Kuhne, T.: Model-Driven Development: A Metamodeling Foundation. IEEE Software, 20(5):36–41, 2003.
- [At14] Atkinson, Colin; Grossmann, Georg; Kühne, Thomas; de Lara, Juan, eds. Proc. 1st Int. Workshop Multi-Level Modelling co-located with MoDELS 2014, volume 1286 of CEUR Workshop Proceedings. CEUR-WS.org, 2014.
- [At15] Atkinson, Colin; Grossmann, Georg; Kühne, Thomas; de Lara, Juan, eds. Proc. 2nd Int. Workshop Multi-Level Modelling co-located with MoDELS 2015, volume 1505 of CEUR Workshop Proceedings. CEUR-WS.org, 2015.
- [Bé05] Bézivin, Jean: On the Unification Power of Models. Software & Systems Modeling, 4(2):171–188, 2005.

- [BG11] Büttner, Fabian; Gogolla, Martin: Modular Embedding of the Object Constraint Language into a Programming Language. In: Formal Methods, Foundations and Applications: 14th Brazilian Symposium. Springer Berlin Heidelberg, pp. 124–139, 2011.
- [BKK16] Balaban, Mira; Khitron, Igal; Kifer, Michael: Multilevel Modeling and Reasoning with FOML. In: IEEE Int. Conf. SWSTE. pp. 61–70, 2016.
- [CG12] Cabot, Jordi; Gogolla, Martin: Object Constraint Language (OCL): A Definitive Guide. In (Bernardo, Marco; Cortellessa, Vittorio; Pierantonio, Alfonso, eds): Formal Methods for Model-Driven Engineering, LNCS 7320, pp. 58–90. Springer, 2012.
- [dLG10] de Lara, Juan; Guerra, Esther: Deep Meta-modelling with MetaDepth. In: Proc. 48th Int. Conf. TOOLS 2010. pp. 1–20, 2010.
- [Dr16] Draheim, Dirk: Reflective Constraint Writing. In: Special Issue on Database- and Expert-Systems Applications on Transactions on Large-Scale Data- and Knowledge-Centered Systems XXIV - Volume 9510. Springer-Verlag New York, Inc., pp. 1–60, 2016.
- [GBR07] Gogolla, Martin; Büttner, Fabian; Richters, Mark: USE: A UML-based Specification Environment for Validating UML and OCL. *Sci. Comput. Program.*, 69(1-3):27–34, 2007.
- [GH16] Gogolla, Martin; Hilken, Frank: Model Validation and Verification Options in a Contemporary UML and OCL Analysis Tool. In (Oberweis, Andreas; Reussner, Ralf, eds): Proc. Modellierung (MODELLIERUNG'2016). GI, LNI 254, pp. 203–218, 2016.
- [Go15] Gogolla, Martin: Experimenting with Multi-Level Models in a Two-Level Modeling Tool. In: Proc. 2nd Int. Workshop Multi-Level Modelling co-located with MoDELS 2015. pp. 3–12, 2015.
- [IGS14] Igamberdiev, Muzaffar; Grossmann, Georg; Stumptner, Markus: An Implementation of Multi-Level Modelling in F-Logic. In: Proc. Workshop Multi-Level Modelling co-located with MoDELS 2014. pp. 33–42, 2014.
- [LGdL14] López-Fernández, Jesús J.; Guerra, Esther; de Lara, Juan: Assessing the Quality of Meta-Models. In: Proc. 11th Workshop MoDeVva@MODELS 2014. pp. 3–12, 2014.
- [Mi14] Mierlo, Simon Van; Barroca, Bruno; Vangheluwe, Hans; Syriani, Eugene; Kühne, Thomas: Multi-Level Modelling in the Modelverse. In: Proc. Workshop Multi-Level Modelling co-located with MoDELS 2014. pp. 83–92, 2014.
- [Ob06] Object Management Group – OMG: . OMG: Object Constraint Language, version 2.0, 2006.
- [Ob11] Object Management Group – OMG: . OMG Unified Modeling Language(OMG UML), Superstructure, version 2.4.1, 2011.
- [Ob15a] Object Management Group – OMG: . OMG Meta Object Facility (MOF) Core Specification, version 2.5, 2015.
- [Ob15b] Object Management Group – OMG: . Unified Modeling Language Specification, version 2.5, 2015.

Measuring the Quality of System Specifications in Use Case Driven Approaches

Alexander Rauh¹, Wolfgang Golubski², Stefan Queins³

Abstract: One of the biggest challenges of a requirements analyst is to generate and provide a high-quality system specification in order to support other disciplines during system development. Today, there are only few mechanisms to measure the quality of requirements with less effort for the analyst. The following paper describes a meta-modeling and model-to-model transformation approach to formally evaluate different quality characteristics of system specifications like consistency and completeness in use case driven requirements analysis processes with less effort for the requirements analyst. Therefore, the mentioned concept integrates the information contained in different representations of requirements into a common requirements model and analyzes quality characteristics of the specification in two steps. In the first step, every representation within the specification will be evaluated separately according to predefined representation specific rules. In the second step after requirements integration, algorithms analyze the quality of the integrated information and calculate the overall characteristics of the specification.

Keywords: Requirements Quality, Specification Quality, Requirements Modeling, Meta-Modeling, Model-to-Model Transformation

1 Introduction

In systems engineering the system specifications are foundations for nearly every kind of discipline during development and after sales [Wa15]. Design, architecture and implementation transform the requirements of the specification into a set of components to realize the system and satisfy the needs of the customers. Testing and verification check the developed system against the system specification and ensure that the quality of the system fits the expectations. Additionally, during after sales the requirements of the system support the maintenance discipline to understand and improve the system's realization. In the context of this paper the term system addresses software systems as well as more technical systems consisting of software, mechanic and electric parts like cars.

For every purpose mentioned above high-quality requirements according to the characteristics listed in IEEE29148:2011 [IE11] have to be collected during

¹ University of Applied Sciences Zwickau, Dr.-Friedrichs-Ring 2a, 08056 Zwickau, alexander.rauh@fh-zwickau.de

² University of Applied Sciences Zwickau, Dr.-Friedrichs-Ring 2a, 08056 Zwickau, wolfgang.golubski@fh-zwickau.de

³ SOPHIST GmbH, Vordere Cramergasse 13, 90478 Nuremberg, stefan.queins@sophist.de

requirements elicitation. Especially, in case that these requirements will be used as direct input for source code generation as mentioned in [Sm15] or if simulating the system under consideration before development as explained in [Po12]. At the moment, there are only a few approaches which aim at the assurance of the quality of system specifications. Furthermore, these approaches do not provide any mechanisms to evaluate this quality by numbers, need high effort to calculate some numbers or only evaluate some samples of a specification instead of the overall quality.

The concept discussed in this paper explains how to formally measure different quality characteristics of system specifications for a use case driven requirements analysis process using common notations for requirements documentation. In addition to the measurements this approach provides the sources of the defects in the documented requirements.

Following this introduction, there is a section to discuss some related works and already known approaches to measure the quality of requirements. The third section describes a concept and a process for requirements integration in order to measure the quality of system specifications. After that, there is a discussion of already existing quality characteristics of requirements and requirements specifications. Furthermore, interdependencies between the characteristics are explained. The fifth section describes the quality measurement process during requirements integration which checks a specification against defined requirements modeling and documentation rules. After an example to show the results of measuring the quality of a specification, the benefits of this approach will be explained. In the last section, there are some open issues which may be relevant for further researches.

2 Related Works

There are different approaches which provide to measure the quality of a system specification or support the requirements analyst to document a consistent and ideally complete set of requirements.

The first related approach described in [Go11] analyzes consistency and completeness of a specification via trace relations between the requirements. Therefore, the requirements analyst has to identify interdependency between the requirements and has to trace them manually. A tool evaluates the types of the relations and reports contradictions. To manage the trace relations defines [Go11] a meta-model for requirements which is similar to the meta-model of the Requirements Interchange Format (ReqIF) [OM16]. The approach mentioned in [Go11] evaluates only the quality of the trace relations between requirements but does not formally analyze the quality of the requirements content.

Another tool [Fa01] evaluates the quality of requirements in natural language. This tool analyzes the textual requirements for keywords and assigns quality attributes to these

keywords. The quality attributes differ from the quality characteristics defined by IEEE 29148:2011 [IE11] and aim at defects in the language of the requirement sentence. For example, undefined multiplicities and vague terms are reported as defects. The tool in [Fa01] only supports textual requirements. Other representation types for requirements documentation like UML cannot be used for analysis purposes.

A fourth similar approach defined by [Da93] also evaluates the quality of textual requirements according to predefined attributes but does not provide an algorithm to measure these attributes formally. The concept only explains techniques for requirements analyst to evaluate the quality attributes manually. Disadvantage of the approach described in [Da93] is the very high effort for the analyst. Additionally, only textual requirements can be used for quality analysis.

Some other approaches consider less common representation types to evaluate the quality system specifications. Furthermore, these approaches do not provide possibilities to add more common representations.

For example, [Kr09] analyzes the consistency of requirements by capturing textually requirements, creating a UML use case model from these requirements manually and converting these models into a problem ontology. The consistency of this problem ontology is evaluated via reasoning and is matched to a domain ontology representing the domain knowledge related to the system's domain to discover further contradictions. UML use case models support only abstract views onto a system. For the detailed view onto the system's functionality other UML diagram types like activity diagrams or state charts have to be used, but are not supported by [Kr09].

A last similar approach described in [He96] applies algorithms for consistency checks to the formal Software Cost Reduction (SCR) tabular notation, but does not support more common requirements representation types like UML or textual requirements. Thereby, before applying this approach onto a specification, the requirements analyst has to transform a common system specification into the SCR notation and has to spend a lot of additional effort.

Approaches which use formal representation types of requirements like VHDL, MATLAB Simulink or different temporal logics are not discussed in this paper. These representation types are used in very specific domains but are usually not used for common system specifications.

3 Requirements Integration Concept for Quality Measurements

The idea to measure the quality of a system specification is to integrate information contained in the requirements into a common database and to evaluate the overall quality of this information. Thereby, representation specific information is encapsulated and the quality of the content described by requirements will be evaluated. Although today's requirements management tools provide mechanisms to store different views as

described by [Kr95] or [Cz15] onto the system under consideration in a common model are the information within these views only loosely coupled. Interrelations are often simple references which the requirements analyst has to create and manage manually. Additionally, the tools do not check whether there is a relation in between the content of the referenced parts or not. Hence, the idea is to extract the information of these several views and to integrate it in a common requirements model which includes formal relations between the information. Once integrated, algorithms can evaluate this requirements model according to predefined rules which address several quality issues of the overall system specification like consistency and completeness.

This mentioned integration is realized via a meta-modeling and model-to-model transformation concept. Therefore, the concept is divided into a representation layer, an interpretation layer and an integration layer. Every layer contains at least one meta-model and one or more instances of these meta-models. In between these layers model-to-model transformations are applied.

The first layer is the representation layer which contains the system specification use for requirements integration purposes. In the context of this approach such a system specification is a set of different models which store the requirements for the system under consideration. These models are instances of common meta-models used for requirements documentation like the UML meta-model and its several diagram types or template-based textual requirements. Notation or representation type of requirements may be used as synonym for the meta-models of the representation layer.

The interpretation layer consists of representation type specific interpretation meta-models and one instance of each of these meta-models. In contrast to the UML there is one dedicated interpretation meta-model for the common UML diagram types for requirements documentation. The idea of the interpretation layer is to evaluate the quality, especially the syntactic correctness, of each view onto the system under consideration separately before requirements integration. Thereby, the requirements analyst gets feedback to the quality of each view and has the possibility to adjust the requirements before an overall quality evaluation. For example, this layer allows evaluating the use case view onto the system independent of the system's information model where the terms, used in the use cases, are defined.

The third layer is the integration layer consisting of a function-oriented meta-model for requirements and one instance of this meta-model which contains the integrated information of the different views onto the system under consideration. Algorithms evaluate the overall quality of the system specification according to this meta-model, to the representation specific and the comprehensive modeling rules. Function-oriented means that this meta-model only defines the structure of information and terms of functional requirements including the quality of service requirements of these functions. Organizational requirements related to the development process or project constraints like time and budget as mentioned in [Dö11] are not part of this meta-model. The function-oriented meta-model of the integration layer was already described and

published in [Ka1/] and is not explained in the context of this paper. Fig. 1 shows these layers and their interrelations including the most significant terms.

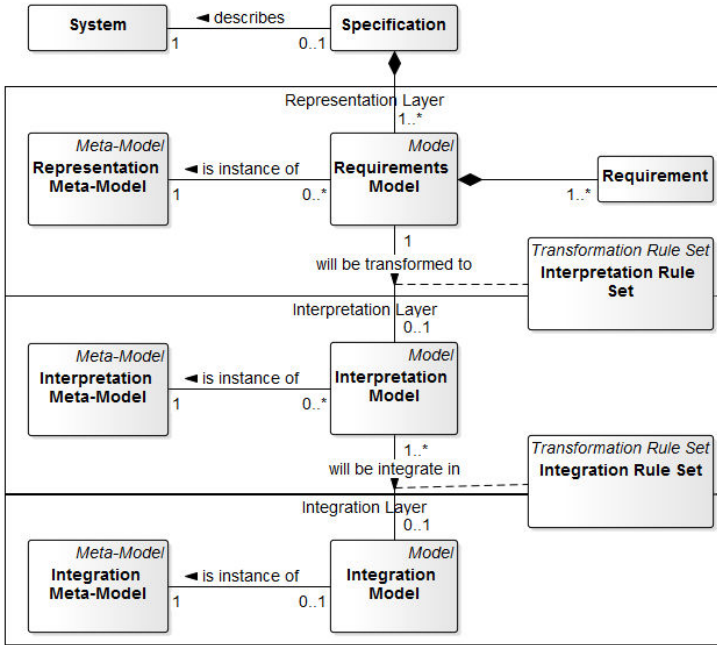


Fig. 1: Layer concept for requirements integration

In order to formally calculate the quality of the system specification several rules define how to use the different notations for requirements documentation. On the one hand, there are representation specific rules which address the usage of model elements within one view e.g. how to name a use case. Violations against these rules will be checked during transformation of the models in the representation layer into the models of the interpretation layer when applying the Interpretation Rule Set. For each violated rule, a so-called defect will be created. These representation specific defects are used to calculate quality characteristics for each view separately. On the other hand, there are representation comprehensive rules that aim at the interrelations of information in between the different views. After requirements integration algorithms check the integrated information according to these comprehensive rules and generate also defects in case of rule violations. Furthermore, the overall quality characteristics of the system specification will be measured. Fig. 2 gives an overview on the single steps of the requirements integration process which was described within this section. For each step, the stereotypes show the actor who performs this action.

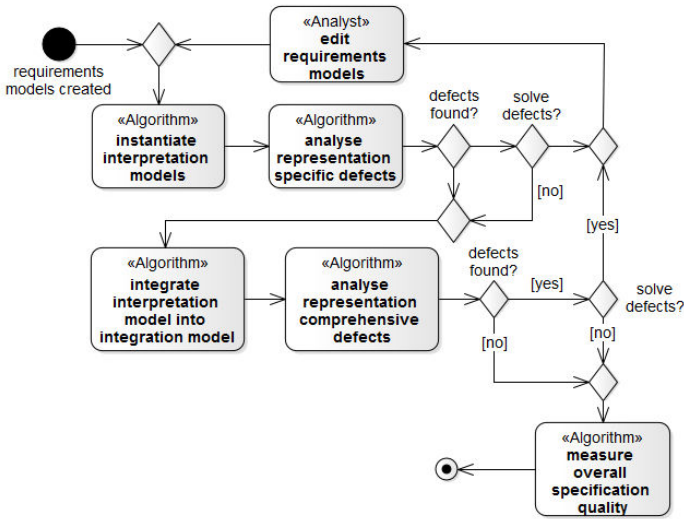


Fig. 2: Process for requirements integration and quality measurement

Depending on the requirements analysis process and the guideline for requirements documentation in a specific development project it may be necessary to adjust the meta-models of the interpretation layer and the predefined requirements documentation rules in order to apply this concept. The fifth section shows some examples for representation specific and comprehensive documentation rules and shows a related interpretation meta-model derived from the use case driven requirements analysis process according to [Cz15].

4 Characteristics of Requirements and System Specifications

In order to define the term quality in a more precise way, this section discusses several characteristics of requirements and sets of requirements. The standard IEEE 29148:2011 serves as foundation for the discussion. First of all, [IE11] differs between characteristics for single requirements and requirements documents. Single requirements have to be:

- Necessary
- Implementation Free
- Unambiguous
- Consistent
- Complete
- Singular

- Feasible
- Traceable
- Verifiable

Some definitions of the characteristics listed above refer not only a single requirement but also consider the context of this requirement. For example, consistency means that the requirement is free of conflicts to other requirements [IE11]. Similar to consistency states the definition of completeness that the requirement does not need further refinements [IE11]. In the context of a system specification it is impossible to determine completeness without analyzing the requirements context which means other requirements addressing the same subject. In addition to the quality characteristics for single requirements IEEE 29148:2011 defines the following criteria for sets of requirements:

- Complete
- Consistent
- Affordable
- Bounded

In order measure the quality of a system specification, a separation between single requirements and a set of requirements is not necessary. Therefore, quality characteristics in the context of this approach always refer to a set of requirements and defects within the requirements have influence onto one or more of these characteristics.

One goal of the concept mentioned in this paper is to evaluate the quality of a system specification automatically using algorithms. But a few of the previously listed characteristic cannot be checked by a tool but only by the requirements analyst himself. For example, it is not possible to determine feasibility and affordability of requirements without knowledge and experience from similar development projects. Additionally, one essential characteristic is missing which provides the foundation for automatic quality checks by algorithms. To apply algorithms onto requirements these requirements have to be syntactically correct. IEEE 29148:2011 only defines semantic correctness as a task to be established during requirements analysis and maintenance. Semantic correctness means that the requirements express the intentions of the stakeholders [IE11].

In addition to this mentioned dependency, there are further interrelations between the quality characteristics. Inconsistency leads to the issue that completeness and necessity cannot be determined by algorithms. For example, if there is an actor with no associations to any use case within the use case view onto the system under consideration this actor might be not necessary or the use case view is incomplete due to a missing association. Whether this defect addresses necessity or completeness depends on the solution of the analyst to solve this defect. Algorithms are not able to make this decision.

Unambiguity has interrelations to consistency, necessity and completeness. As defined by IEEE 29148:2011 means unambiguity that there are no possibilities for different interpretations of information. Inconsistency, incompleteness and violations of necessity cause such possibilities for interpretation.

The concept described in this paper supports the measurement of correctness, completeness, consistency, unambiguity and necessity.

5 Rule-based Quality Measurement of Requirements

The quality of a system specification is measured according to predefined requirements documentation rules. Each documentation rule has assigned at least one of the quality characteristics listed in the previous section. Rule violations lead to defects which decrease the assigned characteristics.

For the application of this concept onto a system specification for validation purpose the rules for requirements documentation, the related interpretation meta-models and the required transformations were defined for the use case driven requirements analysis process according to [Cz15]. The snippets below show two different rules for requirements documentation including the assigned quality characteristics to give an idea of these rules. The first rule is representation specific and addresses the documentation of use cases. The second rule is comprehensive and addresses and interrelation between use cases and the information model of the system's domain.

Rule 1: Names of UML Use Cases follow the structure <verb> [adjective] <noun>.

Assigned quality characteristics: Correctness

Criticality: high

Rule 1 addresses the naming of use case. The requirements analysis process in [Cz15] recommends that the name of a use case should consist of a process and an object of the system's domain. UML does not even constrain the naming [OM15]. The rule above provides the option to add an adjective between the verb for the process and the noun for the object. This adjective could be used to add further information to the object like a state of this object. For example, a use case could be named like "archive existing user profile". In order to simplify the implementation of this rule, a name of a use case could only consist of two or three tokens. The first token is the verb which defines the process. If the name contains three tokens, the second token is a state and third is the object of the domain. Domain objects which consist of more than one token will be named in camel case e.g. "UserProfile". For further researches, this simplification could be eliminated by the integration of a natural language processing (NLP) tool like [Bj10]. This tool would also provide mechanisms to categorize the types of the words and to analyze the grammar of natural language parts in the representation models.

As mentioned before, there is at least one quality characteristic of the previous section assigned to each of the requirements documentation rules. If such a documentation rule is violated, the related quality characteristic will be decreased. For example, in case that the name of a use case violates rule 1, the correctness of system specification is affected.

The criticality of the modeling rules classifies the impact of a rule violation onto the further requirements integration process. High criticality means that the related representation model element cannot be integrated into the integration model. Low criticality violations have no effects onto the requirements integration but cause also a loss in the requirements quality.

Rule 2: The noun in the name of a use case is defined as a class within the information model of the system specification.

Related representations: UML Use Case Diagram, UML Class Diagram

Assigned quality characteristics: Consistency, Completeness

Criticality: low

Rule 2 is a comprehensive modeling rule which addresses the usage of interrelations between UML use case diagrams and UML class diagrams for information modeling. The nouns in the names of use cases represent objects of the system's domain and thereby should be part of the information model. Consistency and completeness will be decreased, if rule 2 is violated. The impact of such a violation onto the requirements integration process is low.

In addition to these kinds of modeling rules, the mentioned requirements integration concept provides interpretation meta-models for the representation types required for a use case driven requirements analysis process according to [Cz15]. Fig. 3 shows the interpretation meta-model for UML use case diagrams. This meta-model is derived from eight use case specific requirements documentation rules.

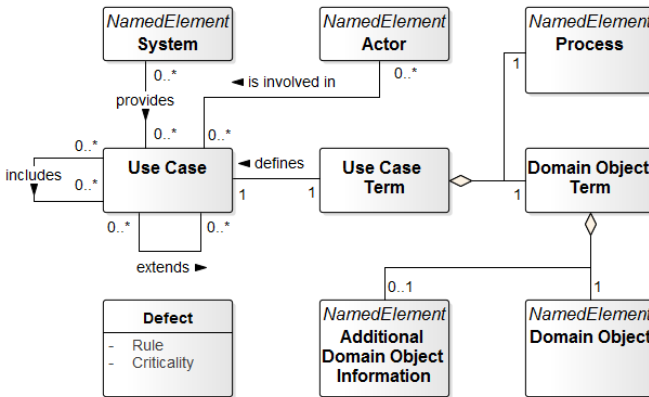


Fig. 3: Interpretation meta-model for UML use case diagrams

The core element of this meta-model is the *Use Case*, which represents functionality of the assigned *System* at high level of abstraction. In order to provide fault tolerance during instantiation of the use case interpretation model, the cardinalities of the relations between *Use Case* and *System* and *Use Case* and *Actor* according to the definition of use cases in [Ja92] are loosened from obligatory to optional.

In contrast to the UML, which defines the name of a use case as a non-empty string [OM15], the meta-model shown in Fig. 3 encapsulates the name of a use case as the separate class *Use Case Term*. Furthermore, the use case term and its parts depicts documentation rule 1 which defines the structure of the names of use cases.

Includes- and *extends-*associations of the use cases are simplified in the interpretation meta-model compared to the UML meta-model. The details of include- and extend-associations between use cases have to be documented in the control flows of the dedicated use cases. These control flows are parts of the activity of the system specification. Hence, there is no need to store further details for include- and extend-associations of use case diagrams within the interpretation model. The *Defect* will be used to store violations of use case specific requirements documentation rules. Such a defect persists the rule which was violated and the criticality of the violation in order to give the requirements analyst advises for editing the use case model.

Further interpretation meta-models for UML class diagrams, state charts and activity diagrams as well as the interpretation meta-models for glossary entries, functional and non-functional textual requirements will be presented later during researches due to the limited space in this paper.

Overall to measure the quality of system specifications there are 27 representation specific documentation rules, 13 comprehensive documentation rules and interpretation meta-models for:

- UML Use case diagrams
- UML Activity diagrams
- UML State Charts
- UML Class diagrams and glossary entries
- Template-based textual functional requirements
- Template-based textual non-functional requirements

The representation specific documentation rules are included in the Interpretation Rule Sets shown in Fig. 1 and the Integration Rule Set contains the comprehensive documentation rules. Tab. 1 lists the count of representation specific and comprehensive rules including the count of related quality characteristics.

Addressed Representation Type	Rule Count	Affected Quality Characteristics				
		Completeness	Correctness	Consistency	Unambiguity	Necessity
Use Case Diagrams	8	1	7	1	4	2
Activity Diagrams	4	3	4	0	2	0
Class Diagrams	2	0	2	0	1	0
Glossary Entries	4	1	1	2	2	0
State Charts	3	0	3	0	0	0
Textual functional requirements	3	0	3	1	1	0
Textual non-functional requirements	3	0	3	1	0	0
Representation Comprehensive	13	10	13	13	0	0

Tab. 1: Representation specific and comprehensive modeling rules and quality characteristics

These meta-models are implemented using Eclipse EMF. The model-to-model transformations are realized using a combination of the ATLAS Transformation Language (ATL) and Java. The transformation rule sets are implemented in ATL. Java was used to coordinate the transformation steps and to analyze the defects within the interpretation layer and the integration layer in order to calculate the numbers for the quality characteristics.

6 Example for Integration Results

In order to explain the possibilities and results of the approach mentioned in the previous sections, the integration concept was applied onto the following two simple diagrams of a system specification of an online shop. The representation specific defects and the comprehensive defects of the specification are explained. Fig. 4 shows a use case diagram on the left and an information diagrams on the right which were integrated. Additionally, there are two activity diagrams for “create CustomerAccount” and “buy

Article” within the specification which will be referenced to explain the integration results.

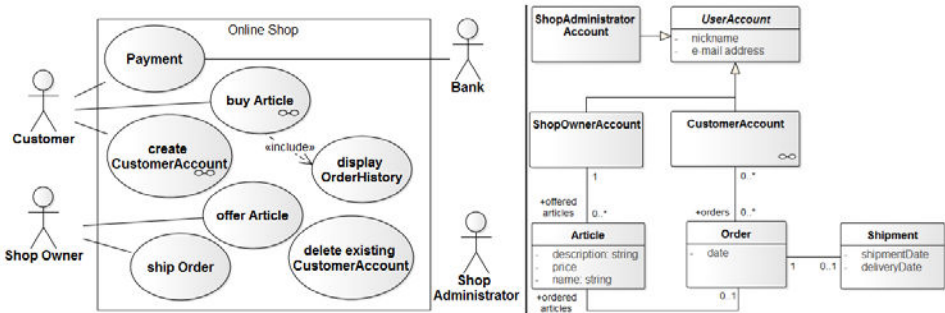


Fig. 4: Use cases and information model of an online shop

Tab. 2 shows the defects which were generated during instantiation of the interpretation model for the use case diagram in Fig. 4.

ID	Element	Defect	Related Quality Characteristics	Criticality
1	Payment	Incorrect name of use case	Correctness	high
2	display OrderHistory	missing association to any actor	Correctness, unambiguity	low
3	delete existing CustomerAccount	missing association to any actor	Correctness, unambiguity	low
4	Shop Administrator	missing association to any use case	Correctness, unambiguity	low

Tab. 2: Defects in the use case interpretation model

The first defect is the result of violated rule 1 which is explained in the previous section in detail. The name of the use case “Payment” does not consists of a verb and a noun but only contains a nominalization. This syntactic incorrectness leads to a high criticality of the first defect because the related use case cannot be instantiated within the interpretation model and, thereby, will be ignored in the further integration process.

The second and the third row address the violation of a documentation rule derived from the definition of use cases according to [Ja92]. Both use cases do not have an association to any actor and, thereby, may not be of value. Besides the impact onto the correctness of the use cases lead both defects to an ambiguity. The defect related use cases may be documented incomplete or they may be unnecessary. The requirements integration could be performed for both use cases. Hence, the criticality of the defects in the second and third row is evaluated as low.

The defect in the fourth row addresses the “Shop Administrator” without an association to any use case. The violation is similar to the previously explained missing associations. It has also an impact onto the correctness and the unambiguity of the specification but the requirements integration could be performed.

The class diagram on the left in Fig. 4 does not violate any representation specific documentation rules. After the instantiation of the interpretation models for the use cases and the information model the requirements analyst could edit the defects listed in Tab. 2 or the requirements integration could proceed. Tab. 3 shows the representation comprehensive defects after integration of the diagrams in Fig. 4.

ID	Element	Defect	Related Quality Characteristics	Criticality
1	display OrderHistory	noun OrderHistory is not part of the information model	Consistency, Completeness	low
2	display OrderHistory	activity "display OrderHistory" is missing	Consistency, Completeness	low
3	ship Order	activity "ship Order" is missing	Consistency, Completeness	low
4	offer Article	activity "offer article" is missing	Consistency, Completeness	low
5	delete existing CustomerAccount	activity "delete existing CustomerAccount" is missing	Consistency, Completeness	low
6	buy Article	control flow of activity "buy Article" does not contain call-behavior of activity "display OrderHistory"	Correctness, Consistency	low

Tab. 3: Representation comprehensive defects of the integrated requirements

The first defect is the result of a violation of rule two of the previous section. It aims at the interrelation between the nouns in the names of use cases and the classes in the information model. As shown in Fig. 4 exists no class “OrderHistory” within the information model. Consistency is decreased because of the contradictions between the different views and completeness is impacted due to obvious missing information.

The defects in rows two to five address the missing refinements of the use cases. Each use case has to be refined by one activity which defines the single steps of the control flow when executing the related use case. These three defects have also lead to inconsistencies as well as incompleteness of the specification.

The last row shows a violation of a documentation rule which aims at the semantic of includes-associations between use cases and their influence onto the control flows of these use cases. The activity of the included use case has to be part in the activity of the including use case as a call-behavior action. Due to the missing activity of “display OrderHistory” shown in the second defect, there is no possibility to add such a call-behavior action within the control flow of “buy Article”. The contradiction between the use case view and the activity view of these use cases is obviously an inconsistency. The correctness of the specification is also affected because of the wrong documentation of the includes-association. The six defects in Tab. 3 have a low criticality. They do not prevent the requirements integration.

7 Benefits

This approach provides mechanisms to formally measure the quality of systems specifications created during use case driven analysis process according to [Cz15] and reports the defects to the requirements analyst. There are two different kinds of

measurements. On the one hand, the quality of each view onto the system is calculated separately. Thereby, the requirements analyst can focus and improve one dedicated view. On the other hand, the requirements integration provides to check the overall quality of the requirements in all views. Especially, the interrelations between the dedicated views onto system are hard to be checked manually and take a very high effort. The mentioned concept supports to check these interrelations automatically with less effort.

Regarding the identified defects of a system specification, the requirements analyst can decide which of these defects he wants to fix. Thereby, the level of the quality of a specification can be adjusted to the projects needs and constraints. For example, the level of quality for a specification for safety critical systems (e.g. cars or airplanes) has to be quite higher than for less safety critical systems (e.g. business software).

The process to measure the quality of a system specification is very lightweight, which allows the requirements analyst to apply the concept in different kinds of projects. For example, the concept can be applied iterative in agile development processes as well as in more traditional processes when reaching milestones.

8 Conclusion & further Researches

The concept mentioned in this paper measures the quality of system specifications according to predefined requirements modeling rules. For each rule, there is at least one quality characteristic assigned, which will be decreased if the related rule is violated. The modeling rules are classified in representation specific and representation comprehensive rules. The representation specific rules are checked during instantiation of the interpretation models when applying the Interpretation Rule Set onto the representation models. After that, the requirements analyst has the possibility to fix violations before the interpretation models will be integrated. After requirements integration, the representation comprehensive rules including the interrelations between the different views onto the systems are analyzed and the overall quality of the system specification is calculated.

For further researches, the definition of metrics for specifications might be interesting. These metrics could be calculated from the numbers of rule violations per type of model element in the requirements models. Thereby, the maturity of a system specification could be determined periodically during the requirements analysis process.

In order to eliminate simplifications of the natural language parts in the representation models, the integration of natural language processing tools to analyze parts of the natural language in the requirements models seems to be necessary. Thereby, the modeling rules for naming model elements like use case and activities could be more flexible and would provide an additional benefit.

The example section lists the result when applying the described concept onto a small

system specification which consists of four diagrams with some interrelations. For further validation purposes, the application of this approach onto larger specifications or even during a requirements analysis process is necessary. Thereby, the integration results can be verified and compared to each other and issues of the current concept could become transparent.

References

- [Bj10] Björkelund, Anders; Bohnet, Bernd; Hafdell, Love; Nugues, Pierre (2010): A High-performance Syntactic and Semantic Dependency Parser. In: Proceedings of the 23rd International Conference on Computational Linguistics: Demonstrations. Stroudsburg, PA, USA: Association for Computational Linguistics (COLING '10), S. 33–36. Online available <http://dl.acm.org/citation.cfm?id=1944284.1944293>.
- [Cz15] Cziharz, Thorsten; Hruschka, Peter; Queins, Stefan; Weyer, Thorsten (2015): Handbook of Requirements Modeling IREB Standard. Version 1.1. Online available https://www.ireb.org/content/downloads/17-handbook-cpre-advanced-level-requirements-modeling/ireb_cpre_handbook_requirements-modeling_advanced-level-v1.1.pdf.
- [Da93] Davis, A.; Overmyer, S.; Jordan, K.; Caruso, J.; Dandashi, F.; Dinh, A. et al. (1993): Identifying and measuring quality in a software requirements specification. In: [1993] First International Software Metrics Symposium. Baltimore, MD, USA, 21–22 May 1993, S. 141–152.
- [Dö11] Dörr, Jörg (2011): Elicitation of a complete set of non-functional requirements. Stuttgart: Fraunhofer-Verl (PhD theses in experimental software engineering, 34).
- [Fa01] Fabbrini, Fabrizio; Fusani, Mario; Gnesi, Stefania; Lami, Giuseppe (2001): An automatic quality evaluation for natural language requirements. In: Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ, Bd. 1, S. 4–5. Online available <http://fmt.isti.cnr.it/WEBPAPER/P11RESFQ01.pdf>.
- [Go11] Goknil, Arda; Kurtev, Ivan; van den Berg, Klaas; Veldhuis, Jan-Willem (2011): Semantics of trace relations in requirements models for consistency checking and inferencing. In: *Softw Syst Model* 10 (1), S. 31–54.
- [He96] Heitmeyer, Constance L.; Jeffords, Ralph D.; Labaw, Bruce G. (1996): Automated consistency checking of requirements specifications. In: *ACM Trans. Softw. Eng. Methodol.* 5 (3), S. 231–261.
- [IE11] Institute of Electrical and Electronics Engineers. 2011: Systems and software engineering -- Life cycle processes --Requirements engineering.
- [Ja92] Jacobson, Ivar (1992): Object-oriented software engineering: A use case driven approach. [New York] and Wokingham and Eng and Reading and Mass: ACM Press and Addison-Wesley Pub.
- [Kr09] Kroha, Petr; Janetzko, Robert; Labra, José Emilio (2009): Ontologies in Checking for Inconsistency of Requirements Specification. In: Third International Conference on Advances in Semantic Processing (SEMAPRO). Sliema, Malta, S. 32–37.

- [Kr95] Kruchten, Philippe (1995): The 4+1 View Model of Architecture: IEEE Software.
- [OM15] Object Management Group, Inc. (2015): Unified Modeling Language. Version 2.5. Online available <http://www.omg.org/spec/UML/>.
- [OM16] Object Management Group, Inc. (2016): Requirements Interchange Format™ (ReqIF™). Version 1.2. Online available <http://www.omg.org/spec/ReqIF/1.2>.
- [Po12] Pohl, Klaus; Achatz, Reinhold; Hönninger, Harald; Broy, Manfred (2012): Model-based engineering of embedded systems: The SPES 2020 methodology. Berlin and New York: Springer.
- [Ra17] Rauh, Alexander; Golubski, Wolfgang; Queins, Stefan (2017): A requirements meta-model to integrate information for the definition of system services. In: 2017 IEEE Symposium on Service-Oriented System Engineering: IEEE / Institute of Electrical and Electronics Engineers Incorporated.
- [Sm15] Śmiałek, Michał; Nowakowski, Wiktor (2015): From Requirements to Java in a Snap. Model-Driven Requirements Engineering in Practice. Cham: Springer International Publishing (EBL-Schweitzer).
- [Wa15] Walden, David D.; Roedler, Garry J.; Forsberg, Kevin; Hamelin, R. Douglas; Shortell, Thomas M. (2015): Systems engineering handbook: A guide for system life cycle processes and activities. 4th edition.

Synthesis of Cost-optimized Controllers from Scenario-based GR(1) Specifications ¹

Daniel Gritzner,² Joel Greenyer²

Abstract: Modern systems often consist of many software-controlled components which must cooperate to fulfill difficult to achieve goals. Trying to reduce the cost of running such a system, e.g., by minimizing total energy consumption, adds additional complexity. To support engineers in the difficult design of such systems we developed a scenario-based specification approach enabling the intuitive modeling of goals and assumptions using short scenarios. These formal specifications allow defects to be detected and fixed early in development. In this paper we present and evaluate an extension to our approach which enables engineers to model costs of processes and thus to synthesize controllers which guarantee that the specified goals are fulfilled in a cost-optimized manner. Our approach even considers the transfer of energy between components to enable the design of systems in which, e.g., the braking energy of moving components can be leveraged to reduce the cost of running a system.

1 Introduction

Modern systems in several domains, e.g., manufacturing, transportation, or health care, often consist of many software-controlled components which must cooperate to fulfill their goals. As users of these systems demand increasingly complex functionality, the processes which need to be performed by the cooperating components become increasingly complex as well. This in turn means engineers face difficult challenges when designing and implementing the behavior required of each component in the system. Processes require the cooperation of multiple components and every component is involved in the implementation of multiple processes, often several processes at the same time. Each component must properly react to external events, e.g., user inputs, as well as internal events, i.e., actions of other components. Determining and implementing the correct behavior for every component is a difficult and error-prone task. This task becomes even more difficult when optimization is a concern, i.e., finding behavior which not only fulfills all requirements but is also optimal according to some well-defined criterion. Typical criteria are reducing the cost of operating a system or reducing its environmental impact, e.g., through reducing the amount of energy or raw materials required for the production of a physical item. A good example of such a system is an automated manufacturing facility in which many robots cooperate to produce cars.

¹ This research is funded by the DFG project EffiSynth.

² Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, D-30167 Hannover, Germany
daniel.gritzner|greenyer@inf.uni-hannover.de

To support engineers in the difficult design process of such systems we developed a formal, yet still intuitive scenario-based specification approach. In our approach, short scenarios are used to model *guarantees* (goals, requirements, desired behavior) and *assumptions* made about the environment. Scenarios are sequences of events, similar to how engineers describe requirements to each other, e.g., “When A and B happen, then component C_1 must do D , followed by C_2 doing E .” These sequences are used to describe, in an intuitive way [A114, GMMS12], when events or actions may, must, or must not occur. The formal nature of scenario-based specifications enables the use of powerful analysis techniques early in the design process. Through simulation and controller synthesis, which, if successful, can prove that the requirements defined in the specification are consistent, defects can be found and fixed early during development. Additionally, these same techniques can be used to directly execute a specification at runtime or to automatically generate executable code [Gr15, GG]. Doing so significantly reduces manual implementation effort, mitigating some of the cost of writing a formal specification and potentially even reducing overall development costs.

In this paper we present and evaluate an extension of our scenario-based specification approach enabling engineers to design systems which behave in a cost-optimized manner. Previously, when synthesizing a controller/strategy, our goal was to find a behavior strategy which allows the system to fulfill all guarantees infinitely often as long as all environment assumptions hold. The extension discussed in this paper enables two things, a) the modeling of costs associated with the behavior defined in a formal specification, and b) the synthesis of controllers which minimize the costs of executing the specified system. In some domains, e.g., manufacturing, negative costs occur: when robots decelerate, their kinetic energy can, through smart electrical design, be leveraged to drive concurrent processes [Gr14]. Usually, this energy is turned into heat and thus is lost to the system. Our proposed extension supports the transfer of energy between processes in order to find a system behavior in which positive and negative costs significantly overlap. Thus the overall energy consumption when running a system can be reduced, i.e., the system’s operational costs are reduced. The work in this paper is part of a realization of our proposed vision of generating code for the energy-efficient control of production systems [GG16].

Sect. 2 of this paper introduces a running example. Sect. 3 explains some preliminaries. Then, Sect. 4 discusses the main contribution, cost-optimized synthesis, followed by an evaluation in Sect. 5. The paper finishes with related work and a conclusion in Sect. 6 and 7.

2 Example

As a running example we use a production system as depicted in Fig. 1. While the figure only shows two robot arms, one welding and the other performing a pick-and-place operation, this example can easily be extended to additional robots which perform tasks such as drilling or cutting.

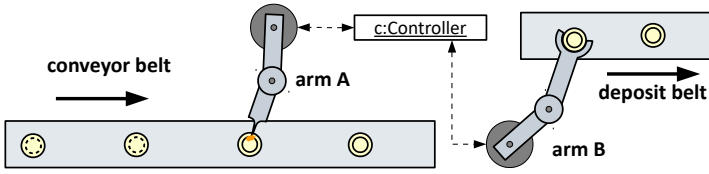


Fig. 1: A typical production system consisting of two robot arms and two conveyor belts. The first arm welds two pieces of a work item firmly together while the second arm picks up the finished items and places them on a different conveyor belt.

This is a typical production process and can be found in many domains, e.g., the manufacturing of cars. Partially built cars move along a conveyor system and once they are in place for the next step, robot arms move in and perform some action, e.g. tightening screws, welding, etc., on the car. Then the robots move back into a holding position and prepare themselves for the next car. These actions are then repeated again and again.

The optimization goal in our example is to find a strategy for controlling the robots such that their movement becomes synchronized in a way that minimizes the overall energy consumption of the system. However, the robots must still fulfill all required tasks. For this purpose, we include positive costs for acceleration and movement at constant velocity for each arm (these must be minimized) and negative costs for deceleration in our specification. Braking energy that cannot be used immediately is turned into heat, i.e., it is lost.

3 Preliminaries

In this section we explain two-player games with a General Reactivity of rank 1 (GR(1)) condition, a formalism to which we map our formal, scenario-based specification. The two players are the specified system and its environment. To synthesize a controller from a formal specification, we try to find a strategy for the system in a GR(1) game such that the system is able to fulfill all guarantees as long as the environment fulfills the assumptions.

3.1 GR(1) Games

A *game* is a directed graph, called *game graph*, $GG = (V, E, v_0)$ with a finite set of vertices V , a set of edges $E \subseteq V \times V$, and an initial state $v_0 \in V$. The set V is partitioned into two disjoint sets V_1 and V_2 . Each vertex $v \in V_i$ is controlled by player i , i.e., this player can choose which edge $e = (v, v') \in E$ with $v' \in V$ is traversed to progress the game. Since the players in our case are the system and its environment, we also use V_{sys} and V_{env} instead of V_1 and V_2 respectively. E is left-total, i.e., there are no dead ends in GG . A *play* or *run* is an infinite sequence of vertices and edges $r = v_0 e_0 v_1 e_1 v_2 e_2 v_3 \dots$ such that $\forall i : e_i = (v_i, v_{i+1}) \in E$.

A *GR(1) game* is a game with an additional winning condition for player *sys*. To help understanding a GR(1) condition, we start with the simpler *Büchi* condition. A Büchi condition consists of a set of goal states $G \subseteq V$ at least one of which must be visited infinitely often by any run which fulfills this condition. The equivalent Linear Temporal Logic (LTL) [Pn77] formula is $\Box \Diamond g$ with $g = \text{“a state } v \in G \text{ is visited”}$. This condition can be generalized into a *Generalized Büchi* condition with multiple, but finitely many, goal state sets G_i , all of which must be visited independently, i.e., $\bigwedge_i \Box \Diamond g_i$ in LTL. A GR(1) condition consists of two Generalized Büchi conditions, the *assumptions* $A = (A_1, A_2, \dots, A_n)$ and the *guarantees* $G = (G_1, G_2, \dots, G_m)$ with $n, m \in \mathbb{N}$ such that the formula

$$\left(\bigwedge_i \Box \Diamond a_i \right) \implies \left(\bigwedge_j \Box \Diamond g_j \right) \quad (1)$$

holds. This condition is fulfilled iff, whenever all the assumptions hold, i.e., their goal states are visited infinitely often, all the guarantees hold as well.

A *strategy* for player i is a mapping from any possible finite prefix $p = v_0 \dots v_j$ of any possible run r , with v_j controllable by i , to an edge $e = (v_j, v') \in E$. A strategy is *memoryless* if the mapping to edges e only depends on the current state v_j of a game. Player *sys* wins a run r iff r fulfills the GR(1) condition of the GR(1) game. Player *sys* wins a GR(1) game iff that player has a strategy such that *sys* wins every possible run r .

3.2 Scenario-based Modeling

We developed a DSL, the *Scenario Modeling Language* (SML), and an associated tool suite, SCENARIOTOOLS, for creating and analyzing formal, scenario-based specifications [Gr15, Gr16, GG]. SML is a textual variant of Live Sequence Charts [DH01, HM03a] which we extended with additional flow control features. Listing 1 shows an excerpt of a specification written in SML. In SML, short scenarios describe in an intuitive way how objects may, must, or must not behave. The excerpt shows an assumption of how robot arm movement works: when the robot is instructed to move, it will immediately accelerate, after that it will eventually move at a constant velocity, eventually decelerate and eventually arrive at its destination. At every step it notifies the system’s controller of its current state. *Roles* (lines 2&3) in SML serve the same purpose as lifelines in Message Sequence Charts. Different objects can be bound at runtime to *dynamic roles*, even concurrently. As an example, if two robots move concurrently there will be two instances of RobotMovement (lines 4-10) active concurrently with each instance having different, unique role bindings. Classes of objects can be defined as controllable by the system (line 1). All other objects are uncontrollable, or controllable by the environment, by default.

An SML specification consists of sets of scenarios and the play out algorithm [HM03a, HM03b] defines how they can be interwoven into a play. Basically, the algorithm waits for

```

1  controllable { Controller }
2  dynamic role Controller controller
3  dynamic role Robot robot
4  assumption scenario RobotMovement {
5      controller -> robot.move()
6      committed robot -> controller.accelerate()
7      eventually robot -> controller.move()
8      eventually robot -> controller.decelerate()
9      eventually robot -> controller.arrived()
10 }

```

List. 1: Excerpt from an SML specification of the example shown in Fig. 1

the environment to choose an event and then activates and progresses scenarios accordingly. Whenever at least one event, sent by a system object, is enabled which is flagged by a liveness condition (e.g., committed), play out will execute one of these events, unless it is blocked by another scenario.

The play out algorithm is generally non-deterministic meaning that multiple events may be possible according to the algorithm given a specific set of active scenarios and an object instance model. This induces a state space, in particular a game graph $GG = (V, E, v_0)$. Each vertex represents a system state, i.e., a set of active scenarios and an object model. The initial vertex has no active scenarios but still has an object model. Each edge is labeled with an event that corresponds to the event that caused the changes in the system state between the two connected vertices. Each vertex is controllable by either the system or the environment, i.e., all outgoing edges, also called outgoing transitions, correspond to events in which the senders are all controllable by the same player. To implement the keyword eventually properly, system controlled vertices might have a system-controllable outgoing edge explicitly representing “wait for the environment”.

To turn this game into a GR(1) game, we also extract a GR(1) condition from the specification. Each active scenario with unique role bindings is turned into one assumption or guarantee condition depending on the scenario’s type (assumption or guarantee). The goal states for such an active scenario are all states in which this scenario has no liveness condition to fulfill (in particular, this is true if the scenario is not even active in a given state) and no safety violation of that scenario (tracked via a flag) has occurred in previous states. Additionally, we add a guarantee that contains all environment-controlled states as goal states. This is to ensure that the system will eventually react to external inputs again, assuming the specification is well-separated, which is a desirable property of a good specification [MR16]. In a well-separated specification, the system cannot force a violation of the assumptions by any of its choices, i.e., the system must focus on fulfilling the guarantees.

Active scenarios AS represent subgraphs of this game graph. These subgraphs are characterized by initializing edges, terminating edges, and a set of states $V' \subset V$ in which AS is active. *Initializing edges* represent events which activate AS , i.e., which represent the first event in the associated scenario S . *Terminating edges* represent events which

terminate AS , usually the last event in S . The smallest possible such subgraph is just a single edge which is both initializing and terminating an active scenario which merely checks a condition when a certain event occurs. However, for simplicity, we will later only consider subgraphs in which the sets of initializing and terminating edges are disjoint, implying $V' \neq \emptyset$. These subgraphs represent *activities* which last for an extended duration and are characterized by events which indicate the beginning, progress, or end of an activity. The scenario `RobotMovement` in Listing 1 is an example of such an activity. The set V' of an activity can be partitioned into a set of weakly connected components with exactly one component for each instance of the activity within the game graph.

3.3 Controller Synthesis

Our controller synthesis is an implementation of Chatterjee's attractor-based GR(1) game solving algorithm [Ch16]. It uses the assumptions' and guarantees' goal states to calculate *attractors*. An attractor is dependent on a player and a condition, e.g., a system attractor of guarantee G_j is a state from which the system can ensure to visit a goal state of G_j regardless of the environment's behavior. Chatterjee's algorithm iteratively calculates and removes environment dominions from the game graph. An environment dominion is a set of states all of which are not a system attractor of at least one guarantee. Furthermore, these states are all environment attractors of all assumptions. The states retained after no further environment dominion can be found are known to contain a strategy which allows the system to either fulfill all of its guarantees or to ensure the violation of at least one assumption. If the initial state of the game graph is in this set, the specification is *realizable*, i.e., the system wins the game. We call the set of retained states the *winning states*.

To extract a strategy from the winning states we start with calculating system attractors strategies for all guarantees. This is done via a reverse search starting from the goal states of a guarantee. First, all goal states are marked as attractors. Then, iteratively, all non-attractor states which are directly reachable via traversing an edge from an attractor in reverse direction, are tested whether they are attractors or not, i.e., are they system controllable with an edge leading to an attractor or are they uncontrollable with all edges leading to attractors. All new attractors found this way are also marked as attractors and for each new attractor that is controllable, the edge used to reach it (via reverse direction traversal) is stored as the system's move in that particular state. These mappings of states to edges are a cycle-free strategy to reach a goal state, i.e., this mapping is a system attractor strategy.

Next, for each guarantee, we calculate system dominions, analogously to the environment dominions used in the initial game solving algorithm, in the non-attractor states of that guarantee. We store the strategy to reach and stay in such a dominion as we did before for fulfilling the guarantee. Then we remove it and repeat the process until no further dominion can be found and removed. This way we end up with a strategy for each guarantee to either fulfill it or end up in a subgraph in which at least one assumption is violated.

Finally, a memoryless strategy can be calculated by creating a new game graph which consists of m copies of the original graph's vertices ($m =$ number of guarantees). The states in this new graph are labeled with the label of the state they are a copy of and the label of the guarantee G_j which should be fulfilled next. An arbitrary state, which is a copy of the original initial state, can be chosen as the new initial state. From this state the strategy for G_j is followed (all outgoing edges are added for uncontrollable states) until a goal state is reached. The outgoing edges of this goal state are labeled with a new $G_{j'}$ to fulfill and its strategy is then followed. One after another, all guarantees are fulfilled this way. Most of the states created by this approach will never be reached, thus creating the necessary states on-the-fly while following the current strategy greatly reduces the resulting graph's size. Memoryless strategies make subsequent processes, e.g., code generation, easier as only the system state needs to be tracked at runtime as opposed to the entire event history.

4 Cost-optimized Synthesis

In this section we describe our technique for synthesizing a cost-optimized controller. It is built on top of the GR(1) game solving algorithm described in the previous section, reusing as much of the existing functionality as possible. We followed a divide-and-conquer approach in our design. First, we execute the regular attractor-based game solving algorithm. If the system is realizable, we run our cost optimization algorithm on the winning states. Finally, we perform a modified strategy extraction to extract a cost-optimized memoryless strategy from the winning states, or in other words, synthesize a cost-optimized controller.

4.1 Optimization Goals, Costs and Gains

GR(1) games are infinite games, i.e., the systems represented by these games are intended to be run for an indefinite amount of time. While this is a good analogy for how systems in many domains, e.g., manufacturing, are actually run, this poses a problem for optimization: if every event or activity (cf. end of Sect. 3.2) can, in theory, occur an infinite number of times, how do we model costs in a well-defined, preferably finite, way that can be computed with reasonable effort? We observed that components repeat finite activities an infinite number of times in these games. An analogous pattern in a programming language would be a while-loop whose loop condition is always true and which, of course, has a finite body. Often even the whole system follows such a pattern. Thus, to optimize an infinite game, we need to identify its finite "body", which is a subgraph with a clearly defined beginning and end like that of an activity, and optimize the behavior while in that body.

To identify finite subgraphs during which the system's behavior shall be optimized we added the ability to annotate scenarios in SML as *optimization goals*. The activity represented by the scenario then defines a set of weakly connected components during which optimization shall be performed. A specification may already contain suitable scenarios which merely

must be annotated by an expert. But even if not, SML allows writing appropriate scenarios which do not alter the game graph or the GR(1) condition induced by the specification. These can be used to define optimization goals in a clear, separate manner. Through annotations engineers can also choose which parts of the system's behavior are optimized while others, which need not be optimized, e.g., a one time initialization, or which must not be optimized, e.g., a safe shutdown procedure in case of an error or emergency, remain untouched.

We use the same notion of activities we used for defining optimization goals to define *costs* (scenarios annotated with a cost $\in \mathbb{R}^+$) and *gains* (scenarios annotated with a cost $\in \mathbb{R}^-$). By using activities instead of merely assigning cost values to vertices or edges we are able to identify when costs and gains overlap to model effects like the transfer of energy between components mentioned in the introduction. Costs may model whichever property shall be optimized such as the usage of energy, raw materials, or money. Concrete cost values have to be provided by engineers at design time. Measuring or estimating these costs is outside the scope of this paper as it depends on the domain of the system and the type of the cost.

Without loss of generality, for the remainder of this paper we only consider optimization goals, costs, and gains which consist of a single weakly connected component. In practice, we simply split subgraphs consisting of several such components into multiple subgraphs. For optimization goals, we further assume that they do not overlap. Optimization goal instances which, after splitting into single weakly connected components, overlap are merged into one single instance of an optimization goal. This opens up a new issue: circular dependencies of goals. If, e.g., two activities which shall both be optimized overlap which each other such that the end of the first activity overlaps with the beginning of the second activity and vice versa, a simple trick can resolve this issue. The circle becomes a finite subgraph if only one of the two activities is defined as optimization goal. If the impact of the activity, which is not optimized, is sufficiently small or the overlap of the activities is sufficiently large, then the effect on the resulting optimized behavior can be negligible. Assuming the robot arms in Fig. 1 work in such a timing that one arm always starts processing the next work item while the other just finishes its work on its current work item, then only optimizing the movement of one arm yields good results. It will synchronize itself such to the movement of the other arm that the overall system behaves in an optimal way.

4.2 Optimization Objective

Our optimization objective is to find optimal paths through every optimization goal OG of a game. For every system-controllable $v \in V_{OG}$ we want to find the optimal edge to choose for the player sys such that the path $p = ve_0v_1e_1 \dots e_n$ with e_n being a terminating transition incurs the least possible cost. We optimize the worst-case cost, i.e., player env will always choose the edge that leads to the highest overall cost.

The cost of a path $p = ve_0v_1e_1 \dots e_n$ is defined as

$$\text{cost}(p) = \left(\sum_{i=0}^n \text{cost}(e_i) \right) + \text{cost}(\text{targetState}(e_n)) \quad (2)$$

with the cost of an edge e_i being the sum of all costs of activities which terminate in e_i and the cost of a state v' being the sum of all costs of activities active in v' . The costs of the target state v' of e_n are included, despite OG already being terminated, because following the path p makes performing the activities active in v' unavoidable. However, to take effects such as transfer of energy into account, we consider the effective cost of a path p , which is

$$\text{effectiveCost}(p) = \text{cost}(p) - \text{optimalGain}(p). \quad (3)$$

The optimal gain is the maximum amount of cost which can be mitigated by gains. Assuming we have a function $\text{costTransfer} : \text{Costs} \times \text{Gains} \rightarrow \mathbb{R}_0^+$ which maps pairs of costs $c \in \text{Costs}$ and gains $g \in \text{Gains}$ to the amount of c mitigated by g , the optimal gain is defined as

$$\text{optimalGain}(p) = \max_{\text{costTransfer}} \left(\sum_{c \in \text{Costs}(p), g \in \text{Gains}(p)} \text{costTransfer}(c, g) \right), \quad (4)$$

i.e., it is the result of applying the best possible cost transfer function to all relevant cost and gain pairs. A cost, and analogously gain, is only relevant to a path p iff it affects $\text{cost}(p)$, i.e., it overlaps p . While every path p may have a different optimal cost transfer function, all such functions must fulfill the following constraints:

- $\forall c \in \text{Costs}, g \in \text{Gains} : (V_c \cap V_g = \emptyset) \Rightarrow \text{costTransfer}(c, g) = 0$, with V_c, V_g being states in which c, g are active, i.e., gains can only mitigate concurrently active costs.
- $\forall c \in \text{Costs} : \left(\sum_{g \in \text{Gains}} \text{costTransfer}(c, g) \right) \leq \text{cost}(c)$, with $\text{cost}(c)$ being the cost value assigned to c . This constraint ensures that no cost is “overcompensated”.
- $\forall g \in \text{Gains} : \left(\sum_{c \in \text{Costs}} \text{costTransfer}(c, g) \right) \leq |\text{cost}(g)|$, i.e., no gain can be “overused” to mitigate concurrent costs.

A single cost can be mitigated by multiple gains and a single gain can mitigate multiple costs as long as the constraints above are fulfilled. Gains, which do not overlap with sufficiently high costs, are (partially) lost.

4.3 Cost Propagation

Algorithm 1 shows how we apply dynamic programming and depth-first search to build a map M of paths, which minimize the effective cost, through an optimization goal OG .

Algorithm 1 Cost propagation for optimization goal OG

Input: terminating transitions E_T of OG
Output: map M of cost-optimized paths

- 1: initialize empty map M , stack S , set $Done$, and set $Improved$
- 2: $S.pushAll(e \in E_T)$
- 3: **while** S is not empty **do**
- 4: $e \leftarrow S.pop()$, $ss \leftarrow e.sourceState$, $ts \leftarrow e.targetState$
- 5: **if** $M.containsKey(ts) \wedge ts$ is system-controllable **then**
- 6: $p \leftarrow M[ts].first + e$
- 7: **else if** $M.containsKey(ts) \wedge ts$ is environment-controllable **then**
- 8: $p \leftarrow M[ts].last + e$
- 9: **else**
- 10: $p \leftarrow e$
- 11: $effCost \leftarrow calculateEffectiveCost(p)$
- 12: **if** $M[ss] = null$ **then**
- 13: $M[ss] \leftarrow \{p\}$, $Improved \leftarrow Improved \cup \{ss\}$
- 14: **else**
- 15: $M[ss].addOrdered(p)$
- 16: **if** $mappingImproved(M, ss, p)$ **then**
- 17: $Improved \leftarrow Improved \cup \{ss\}$
- 18: $Done \leftarrow Done \cup \{e\}$
- 19: $pushImprovedTransitions(S, ss, Done, Improved)$
- 20: **return** M

The algorithm starts at the terminating transitions of OG and searches optimal paths in the reverse direction. Calculating the effective cost of paths in that order is easier than in a forward search.

For every edge e , the algorithm takes the best path p from e 's target state to the end of OG (stored in M), adds e to the front of p (lines 5-10), calculates the effective cost of the new p (line 11) and then finally updates M (lines 12-17), which will eventually store one entry for every outgoing transition of every state in OG . The mappings in M are stored in order from least effective cost to highest effective cost. Mappings with equal cost are stored in the order they are found, as this will cause strategy extraction to favor edges which do not lead to cycles within OG .

To calculate the effective cost (line 11), we maintain an additional map of paths $p' = v'e_0v_1e_1v_2e_2 \dots e_n$ to a data structure storing the path's (non-effective) cost, which gain beginning in e_0 or later compensates which cost and the degree of cost transfer, and how much cost transfer from gains active in v' could potentially currently occur. From this information the effective cost of p can be easily computed.

To calculate the cost of $p = vev'e_0v_1e_1 \dots e_n$ (concatenation of e and p'), we add the values of costs terminating in e to the cost of p' , determine the optimal cost transfer from gains beginning in e (thus turning potential cost transfer into actual cost transfer), and calculate the new potential cost transfer of gains active in v . When turning potential cost

transfers to actual cost transfers we approximate the optimal cost transfer function via a heuristic to avoid expensive combinatorial explosion. Our heuristic is based on the number $\text{numOverlaps}(g)$ of costs a gain g could potentially compensate. The idea is to use gains with less opportunities to compensate costs first. When multiple gains begin in the same edge, we process them in the order of their $\text{numOverlaps}(g)$ from lowest to highest. Similarly, during the processing of each gain we iterate over costs c based on $\text{numOverlaps}(c)$. While doing so, we make sure that the constraints defined in Sect. 4.2 are still satisfied. Computing the potential cost transfer from still active gains follows the same approach.

Algorithm 1 uses the sets *Improved* and *Done* to determine which edges to push onto the search stack S (line 19). Whenever S is empty, all incoming edges of states in *Improved* are pushed onto S and then *Improved* is cleared. As an optimization, the incoming edges of a state may be pushed onto S if all of the states outgoing edges are in *Done*, even though S is not yet empty. This reduces the number of loop iterations necessary to fully construct M .

The algorithm will eventually terminate because the terminating edges of OG are only pushed onto S once and the incoming edges of every vertex v in OG are pushed onto S at most as often as the number of outgoing edges of v . After termination, M maps every vertex in OG to a list of its outgoing edges ordered from least to highest effective cost.

4.4 Modified Strategy Extraction

Extracting a cost-optimized memoryless strategy works similar to extracting a regular memoryless strategy, though requires some modification to system attractor strategy calculation. Other steps, e.g., merging the strategies for each guarantees into a single strategy, work the same as before. This implies that parts of the game graph, in which the system must ensure that the assumptions are violated, are not optimized. We do not consider this to be a goal of a well-defined system.

The modified system attractor calculation for each guarantee works by iterating over all its attractor vertices v after discarding the system moves determined by the regular system attractor calculation (cf. Sect. 3.3). A forward depth-first search is performed, iterating over outgoing edges of each controllable vertex in the order stored in $M[v]$ (a random order is tried when there is no such $M[v]$). The search starts at an arbitrary controllable attractor vertex v . When a vertex v_{Goal} , which is either a goal state or for which a move is already known, is found, the outgoing transitions used to construct the path from v to v_{Goal} are stored as the systems' moves for each controllable vertex on said path. Then a new search is started from a controllable vertex for which no move has been determined so far. If the search detects a cycle in the current path or encounters a non-attractor state it will backtrack. This way the system will take the most cost-efficient path to fulfill each guarantee. There will be no paths with a lower effective cost that lead to a goal state. The approach terminates as all options are eventually considered, assuming the least cost-effective path is still the best option which still fulfills a given guarantee.

5 Evaluation

In this section we discuss benchmark results of our approach for the synthesis of cost-optimized controllers. We measured the time required to execute the algorithms as well as the size of the game graphs and the synthesized controllers.

5.1 Benchmark Setup

All benchmarks were run on a laptop with an Intel Core i7-5500U, 8 GB RAM, and Windows 10 64-bit. `SCENARIOTOOLS`, the implementation of our DSL and tool-suite, was run using Eclipse Modeling Tools Oxygen and Java 8. During benchmarking no other processes other than system services were running on this system and it was not connected to any network.

We ran benchmarks using the example described in Sect. 2. We used different instances with a varying number of robots for benchmarking. The specification consisted of seven scenarios, one of which was annotated as optimization goal, one was annotated as a cost activity (a robot accelerates and moves) and one was annotated as a gain activity (a robot decelerates; potentially losing kinetic energy as heat). The specified situation was that of new work items, e.g., a car, arriving at each robot's work station simultaneously, followed by each robot moving in from a holding position, performing a task (these were not modeled in detail but kept abstract) and then moving back into the initial holding position waiting for the next work item. The optimization goal was to minimize the loss of kinetic energy as heat during each work item processing cycle, i.e., minimize the total energy cost of the system.

Each example was benchmarked 10 times. Before running the actual benchmarks, five warm up iterations were performed, to prevent the results from becoming tainted by on-demand resource loading or an undefined cache state.

5.2 Benchmark Results

Tables 1 and 2 show our benchmark results. The game graph and controller size comparison shows that only a fraction of the states and transitions from the game graph actually end up being part of the synthesized controller, despite the controller potentially containing multiple copies of each state of the game graph. Also, the state space explosion problem is apparent, as the size of the game graphs increases exponentially with the number of robots. Controllers for instances with five or more robots could not be synthesized as `SCENARIOTOOLS` ran out of memory during game graph creation. Furthermore, there was a single optimization goal instance which spanned nearly the entire game graph.

Table 2 shows that the majority of time was spent creating the game graph, consuming about 95% of the time required to synthesize a controller. The other steps require a comparatively insignificant amount of time, indicating that the algorithms for performing these other steps,

number of robots	1	2	3	4
states in game graph	16	279	4010	52877
transitions in game graph	16	379	6310	91477
states in controller	16.0	62.7	1462.7	18954.7
transitions in controller	16.0	67.7	1818.3	26681.3
states in optimization goal	14	275	4002	52861

Tab. 1: Size of the game graphs, synthesized controllers, and the optimization goal subgraph; controller metrics are averaged over 10 iterations.

number of robots	1	2	3	4
game graph creation	21.8 ms	326.3 ms	6109.5 ms	117181.8 ms
GR(1) game solving	0.1 ms	1.7 ms	44.5 ms	2193.4 ms
strategy extraction (just GR(1))	0.4 ms	1.5 ms	41.6 ms	1974.1 ms
cost optimization	0.1 ms	1.7 ms	18.8 ms	468.7 ms
strategy extraction (optimized)	1.6 ms	2.7 ms	57.9 ms	2739.1 ms
total (extracting opt. strategy)	23.6 ms	332.4 ms	6230.7 ms	122583.0 ms

Tab. 2: Performance measurements; all measurements are times in milliseconds and averaged over 10 iterations.

including the cost optimization presented in this paper, scale sufficiently well to larger systems. Game graph creating becomes a problem long before these other steps start to run slowly. In particular, the cost optimization, i.e., extracting optimization goals, costs, and gains and propagating effective costs through optimization goals, scales well. For larger systems, it performs significantly better than every other step. Overall, for the instance with four robots, cost optimization and modified strategy extraction added only 1233.7 ms to the process compared to synthesizing a non-optimized controller. This means for that instance the overall synthesis time only increased by 1%.

The modified strategy extraction appears to require significantly more time than the regular strategy extraction for smaller systems (a factor 4 difference for a system with a single robot), but this difference decreases as the size of the system increases (a factor 1.39 difference for a system with four robots). While the effect on the overall synthesis process is already negligible, Table 2 also indicates that the performance difference between the two strategy extraction approaches themselves becomes negligible for sufficiently large game graphs.

6 Related Work

There are many approaches for synthesizing optimized controllers from a formal specification, e.g., by Smith et al. [Sm10], Karaman et al. [KF11, KSF08], Wolff et al. [WTM12], and Jing et al. [JEKG13]. The last one even offers an extensible GR(1) synthesis tool [ER16]. These approaches use temporal logic, usually LTL, for their formal specifications and

transition-based costs. Our approach, instead, uses intuitive scenario-based specifications and optimizes based on activities. Activity-based costs are necessary to leverage effects such as the transfer of braking energy to drive concurrently active components.

Also related is the work done on energy games. Introduced by Bouyer et al. [Bo08], there are synthesis approaches proposed by Chatterjee et al. [Ch10, CRR14], Maoz et al. [MPR16], and Brim et al. [Br11]. Actions performed by the players, i.e., transitions traversed in the game graph, cause an energy level to rise and fall throughout the game, thus modeling costs and gains. The first player's objective is to keep this level non-negative. Again, transition-based costs and gains are insufficient for our optimization objective (cf. Sect. 4.2).

Mean pay-off games offer the ability to model vertex-based rewards [CHJ05]. Maximizing rewards can be interpreted as minimizing costs. However, vertex-based costs/rewards have the same limitations as transition-based rewards.

Pellicciari et al. [Pe13] try to utilize idle times of individual robot trajectories. Wigström et al. [Wi13] use dynamic programming to determine an optimal task schedule for a multi robot cell. However, neither approach is able to automatically find optimization opportunities outside of isolated time windows or pre-defined energy exchange opportunities between components. Our approach is able to consider all optimization opportunities.

7 Conclusion

In this paper we presented an approach for synthesizing cost-optimized controllers from scenario-based specifications, an intuitive method for creating formal specifications. We evaluated the performance of this optimization technique and found that it works very fast, especially for larger systems. It is able to handle optimization goals which leverage the transfer of energy between components to reduce the overall energy consumption of a system, a goal difficult or even impossible to model in many existing approaches.

In future work, we want to extend SML with the ability to model time-based constraints and costs. This enables not only the modeling of safety properties in which timing plays an important role but also opens up new avenues for optimization. In robotics, moving a component the same distance at different speeds causes different energy consumptions. To take advantage of this, we need to know how much time a component has to fulfill its goal without slowing down the overall system. Furthermore, knowing how long processes take would also allow for a more accurate model of cost compensation. If we know by how much the deceleration and movement of different components overlap we can more accurately determine how much of the braking energy can actually be recouped.

References

- [Al14] Alexandron, G.; Armoni, M.; Gordon, M.; Harel, D.: Scenario-Based Programming: Reducing the Cognitive Load, Fostering Abstract Thinking. In: Proc. 36th Int. Conf. on

- Software Engineering (ICSE). pp. 311–320, 2014.
- [Bo08] Bouyer, Patricia; Fahrenberg, Uli; Larsen, Kim G; Markey, Nicolas; Srba, Jiří: Infinite runs in weighted timed automata with energy constraints. In: International Conference on Formal Modeling and Analysis of Timed Systems. Springer, pp. 33–47, 2008.
- [Br11] Brim, Lubos; Chaloupka, Jakub; Doyen, Laurent; Gentilini, Raffaella; Raskin, Jean-François: Faster algorithms for mean-payoff games. *Formal methods in system design*, 38(2):97–118, 2011.
- [Ch10] Chatterjee, Krishnendu; Doyen, Laurent; Henzinger, Thomas A; Raskin, Jean-François: Generalized mean-payoff and energy games. arXiv preprint arXiv:1007.1669, 2010.
- [Ch16] Chatterjee, Krishnendu; Dvorák, Wolfgang; Henzinger, Monika; Loitzenbauer, Veronika: Conditionally Optimal Algorithms for Generalized Büchi Games. In (Faliszewski, Piotr; Muscholl, Anca; Niedermeier, Rolf, eds): 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016). volume 58 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 25:1–25:15, 2016.
- [CHJ05] Chatterjee, Krishnendu; Henzinger, Thomas A; Jurdzinski, Marcin: Mean-payoff parity games. In: *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*. IEEE, pp. 178–187, 2005.
- [CRR14] Chatterjee, Krishnendu; Randour, Mickael; Raskin, Jean-François: Strategy synthesis for multi-dimensional quantitative objectives. *Acta Informatica*, 51(3-4):129–163, 2014.
- [DH01] Damm, Werner; Harel, David: LSCs: Breathing Life into Message Sequence Charts. In: *Formal Methods in System Design*. volume 19, pp. 45–80, 2001.
- [ER16] Ehlers, Rüdiger; Raman, Vasumathi: Slugs: Extensible gr (1) synthesis. In: International Conference on Computer Aided Verification. Springer, pp. 333–339, 2016.
- [GG] Gritzner, Daniel; Greenyer, Joel: Controller Synthesis and PCL Code Generation from Scenario-based GR (1) Robot Specifications. In: *Proceedings of the 4th Workshop on Model-Driven Robot Software Engineering (MORSE 2017), co-located with Software Technologies: Applications and Foundations (STAF 2017) (to appear)*.
- [GG16] Greenyer, Joel; Gritzner, Daniel: An Approach for Synthesizing Energy-Efficient Controllers for Production Systems from Scenario-Based Specifications. In: *Proc. of the MoDELS 2016 Demo and Poster Sessions, co-located with ACM/IEEE 19th Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS 2016)*. volume 1725. CEUR Workshop Proceedings, pp. 87–93, 2016.
- [GMMS12] Gordon, M.; Marron, A.; Meerbaum-Salant, O.: Spaghetti for the Main Course? Observations on the Naturalness of Scenario-Based Programming. In: *Proc. 17th Conf. on Innovation and Technology in Computer Science Education (ITICSE)*. pp. 198–203, 2012.
- [Gr14] Greenyer, Joel; Hansen, Christian; Kotlarski, Jens; Ortmaier, Tobias: Towards Synthesizing Energy-efficient Controllers for Modern Production Systems from Scenario-based Specifications. *Procedia Technology (Proceedings of the 2nd International Conference on System-Integrated Intelligence (SysInt 2014))*, 15(0):388–397, 2014.

- [Gr15] Greenyer, Joel; Gritzner, Daniel; Gutjahr, Timo; Duente, Tim; Dulle, Stefan; Deppe, Falk-David; Glade, Nils; Hilbich, Marius; Koenig, Florian; Luennemann, Jannis; Prenner, Nils; Raetz, Kevin; Schnelle, Thilo; Singer, Martin; Tempelmeier, Nicolas; Voges, Raphael: Scenarios@run.time – Distributed Execution of Specifications on IoT-Connected Robots. In: Proceedings of the 10th International Workshop on Models@Run.Time (MRT 2015), co-located with MODELS 2015. CEUR Workshop Proceedings, 2015.
- [Gr16] Greenyer, Joel; Gritzner, Daniel; Katz, Guy; Marron, Assaf: Scenario-Based Modeling and Synthesis for Reactive Systems with Dynamic System Structure in ScenarioTools. In: Proc. of the MoDELS 2016 Demo and Poster Sessions, co-located with ACM/IEEE 19th Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS 2016), volume 1725. CEUR, pp. 16–32, 2016.
- [HM03a] Harel, D.; Marelly, R.: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer, 2003.
- [HM03b] Harel, David; Marelly, Rami: Specifying and Executing Behavioral Requirements: The Play-In/Play-Out Approach. *SoSyM*, 2:82–107, 2003.
- [JEKG13] Jing, Gangyuan; Ehlers, Rüdiger; Kress-Gazit, Hadas: Shortcut through an evil door: Optimality of correct-by-construction controllers in adversarial environments. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on. IEEE, pp. 4796–4802, 2013.
- [KF11] Karaman, Sertac; Frazzoli, Emilio: Linear temporal logic vehicle routing with applications to multi-UAV mission planning. *International Journal of Robust and Nonlinear Control*, 21(12):1372–1395, 2011.
- [KSF08] Karaman, Sertac; Sanfelice, Ricardo G; Frazzoli, Emilio: Optimal control of mixed logical dynamical systems with linear temporal logic specifications. In: Decision and Control, 2008. CDC 2008. 47th IEEE Conference on. IEEE, pp. 2117–2122, 2008.
- [MPR16] Maoz, Shahar; Pistiner, Or; Ringert, Jan Oliver: Symbolic BDD and ADD Algorithms for Energy Games. *arXiv preprint arXiv:1611.07622*, 2016.
- [MR16] Maoz, Shahar; Ringert, Jan Oliver: On well-separation of GR (1) specifications. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, pp. 362–372, 2016.
- [Pe13] Pellicciari, M; Berselli, G; Leali, F; Vergnano, A: A method for reducing the energy consumption of pick-and-place industrial robots. *Mechatronics*, 23(3), 2013.
- [Pn77] Pnueli, Amir: The temporal logic of programs. In: Foundations of Computer Science, 1977., 18th Annual Symposium on. IEEE, pp. 46–57, 1977.
- [Sm10] Smith, Stephen L; Tůmová, Jana; Belta, Calin; Rus, Daniela: Optimal path planning under temporal logic constraints. In: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on. IEEE, pp. 3288–3293, 2010.
- [Wi13] Wigstrom, Oskar; Lennartson, Bengt; Vergnano, Alberto; Breitholtz, Claes: High-level scheduling of energy optimal trajectories. *IEEE Transactions on Automation Science and Engineering*, 10(1):57–64, 2013.
- [WTM12] Wolff, Eric M; Topcu, Ufuk; Murray, Richard M: Optimal control with weighted average costs and temporal logic specifications. *Proc. of Robotics: Science and Systems VIII*, 2012.

Collaborative Modeling Enabled By Version Control

Dilshod Kuryazov¹, Andreas Winter¹, Ralf Reussner²

Abstract: Model-Driven Software Development is a key field in software development activities which is well-suited to design and develop large-scale software systems. Developing and maintaining large-scale model-driven software systems entail a need for collaborative modeling by a large number of software designers and developers. As long as software models are constantly changed during development and maintenance, collaborative modeling requires frequently sharing of model changes between collaborators. Thereby, a solid change representation support for model changes plays an essential role for collaborative modeling systems. This paper focuses on the problem of model change representation for collaborative modeling. It introduces a meta-model generic, operation-based and textual difference language to represent model changes in collaborative modeling. This paper also demonstrates a collaborative modeling application *Kotelett*.

Keywords: Model-driven Software Development and Evolution; Collaborative Modeling; Modeling Deltas; Model Difference Representation

1 Motivation

As a software engineering paradigm, Model-Driven Software Development (MDSD) is the modern day style of software development which supports well-suited abstraction concepts to software development activities. It intends to improve the productivity of the software development, maintenance activities, and communication among various team members and stakeholders. In MDSD, software models which also comprise source code are the central artifacts. MDSD brings several main benefits such as a productivity boost, models become a single point of truth, and they are reusable and automatically kept up-to-date with the code they represent [KWB03, pp. 9ff].

Software models (e.g. in UML [RJB04]) are the key artifacts in MDSD activities. They are well-suited for designing, developing and producing large-scale software projects. In order to cope with constantly growing amounts of software artifacts and their complexity, software systems to be developed and maintained are initially shifted to abstract forms using modeling concepts. Software models are the documentation and implementation of software systems being developed and evolved [KWB03].

¹ University of Oldenburg, Software Engineering Group, Uhlhornsweg 84, 26111 Oldenburg, Germany, {kuryazov, winter}@se.uni-oldenburg.de

² Karlsruhe Institute of Technology, Institute for Program Structures and Data Organization, Postfach 6980, D-76128 Karlsruhe, Germany, ralf.reussner@kit.edu

Software models are constantly changed during their development and evolutionary life-cycle. They are constantly evolved and maintained undergoing diverse changes such as extensions, corrections, optimization, adaptations and other improvements. All development and maintenance activities contribute to the evolution of software models resulting in several subsequent revisions. During software evolution, models become large and complex raising a need for collaboration of several developers, designers and stakeholders (i.e. collaborators) on shared models i.e. *Collaborative modeling*.

Depending on the nature of interaction, collaborative modeling systems can be divided into two main forms, namely *sequential* and *concurrent* collaborative modeling [ESG91]:

- *Sequential Collaboration*. In sequential collaboration, collaborators design a shared model by checking out it into their distributed environment. After making changes, they merge their local changes into the main model. This scenario results in several subsequent revisions of the same shared central model. After each change, differences between subsequent model revisions are identified and represented in model repositories as *modeling deltas*. Modeling deltas serve as information resources in further manipulations and analysis of models. *Modeling deltas* representing significant changes between subsequent revisions of models are first-class entities in storing the histories of model changes in model repositories. The sequential version control is referred to as *macro-versioning* in this paper.
- *Concurrent Collaboration*. Concurrent collaboration is usually dedicated to instantly creating, modifying and maintaining huge, shared and centralized models in real-time by a team of collaborators. Thus, the changes made by collaborators have to be continually detected and synchronized among several concurrent instances of that model. As long as synchronization has to occur in real-time, performance of interaction matters. Thus, model changes have to be represented and synchronized using very simple notations. Concurrent model instances can be differentiated by changes represented in small *modeling deltas*. The required performance of synchronization in real-time can be achieved by exchanging small modeling deltas [KW15]. Real-time synchronization of modeling deltas is referred to as *micro-versioning*.

Sequential and concurrent version control are the key activities of sequential and concurrent collaborative modeling, respectively. In both activities, *modeling deltas* are first-class entities and play an essential role in storing, exchanging and synchronizing the changes between the subsequent and parallel revisions of evolving models. Thus, the efficient representation of modeling deltas is crucial.

An *efficient change representation notation* is needed for both micro-versioning and macro-versioning. Both versioning techniques might rely on the same base-technology to deal with modeling deltas. To that end, this paper introduces a difference language (DL) to the problem of model difference presentation in modeling deltas. The proposed DL is meta-model generic, operation-based, modeling tool generic, reusable, applicable, and extensible. Associated technical support also focuses on providing a catalog of supplementary services which

allow for reusing and exploiting modeling deltas represented by DL. These supplementary services extend the application areas of DL.

The remainder of this paper is structured as follows: Section 2 investigates existing related approaches in the research domain. Several requirements for DL and its collaborative modeling application are defined in Section 3. The core concepts behind the DL-based collaborative modeling are depicted in Section 4 and the same section explains some DL services. Section 5 explains collaborative modeling applications. Section 6 discusses the evaluation of the collaborative modeling application in various modeling languages. This paper ends up in Section 7 by drawing some conclusions.

2 Related Work

The problem of collaborative modeling and its change representation is the actively discussed and extensively addressed topic among the research community of software engineering. There is a large number of research papers addressing to collaborative modeling (Section 2.1) and model difference representation (Section 2.2).

2.1 Collaborative Systems

Collaborative approaches are widely investigated in collaborative document writing and code-driven software development. Thus, this section also reviews some collaborative source code development and document editing systems in order to derive some general concepts and principals.

Sequential collaborative systems such as Git [Sw08], Mercurial [Ma10], Subversion [CSFP04] are used for sequential revision control in source code-driven software development. There are also sequential collaborative modeling systems such as EMF Store [HK13], SMOVer [Al07], AMOR [Br10] and [Ta12] that are discussed below.

Concurrent collaborative text editing systems record and synchronize change notifications during concurrent collaborative development. Collaborative documents writing systems like Google Docs [Go17] and Etherpad [Ap17] are widely used systems in concurrent document creation and editing. There are also several browser-based web modeling tools like GenMyModel [DMA13], Creately [Ci15] which exchange changes over WebSockets. Their core ideas and underlying implementation technologies are not explicitly documented. Their modeling notations and other services are not accessible making them difficult to study and extend.

For differentiating revisions and calculating differences, the most collaborative systems for source code-driven software development use Myer's LCS [My86] algorithm which is based on recursively finding the longest sequence of common lines in the list of lines of compared revisions. The core idea behind this algorithm is to represent differences documents by additions, removals, and matches of textual lines. Software models can also be represented in textual formats using XMI exchange formats, but it is commonly agreed

that the collaborative approaches for source code cannot sufficiently fit to MDSD because of the paradigm shift between source code- and model-driven concepts [CRP07], [St08]. Differentiating the textual lines of the XMI-based software models can not provide sufficient information about the changes in associated and composite data structures of software models. As there are already outstanding collaborative systems for textual documents and the source code of software systems, MDSD also requires support for generic, solid, configurable and extensible collaborative modeling applications for their development, maintenance, and evolution.

2.2 Modeling Delta Representation Approaches.

The existing collaborative modeling approaches employ various techniques for model difference representation. Below, the existing approaches are classified and discussed.

Model-based approaches represent modeling deltas by differences models. Cicchetti et al. [CRP07] introduced a meta-model independent approach. The approach uses software models for representing model differences conforming to a differences meta-model. Cicchetti et al. also provides a service to apply differences models to differentiated models in order to transform them from one revision to another. A fundamental approach to sequential model version control based on graph modifications introduced by Taentzer et. al. [Ta12] is also used to represent model differences by differences models. The approach is validated in Adaptable Model Versioning System (AMOR) [Br10] for EMF models [St08]. The major focus of the approach is differentiation and merging of software models that serve as the main foundations for sequential model version control.

Graph-based approaches represent model differences using internal graph-like structures. It is usually similar to model-based representation, but relying on the low-level graph-like structures. A generic approach to model difference calculation and representation using edit scripts is introduced by the SiDiff approach [Ke13]. SiDiff consists of a chain of model differencing processes which include correspondence matching, difference derivation, and semantic lifting [Ke12]. SiDiff does not rely on a specific modeling language.

Standard ER Database represents model differences in standard entity-relational (ER) databases. Likely, SMOVER [Al07] is a sequential revision control system for software models using ER database.

Text-based approaches represent model differences by a sequence of edit operations in the textual forms embedding change-related difference information. An early approach is introduced by Alanen and Porres [AP03] in the text-based difference representation area. DeltaEcore [SSA14] is a delta language generation framework and addresses the problem of generating delta modeling languages for software product lines and software ecosystems. EMF Store [HK13] is a model and data repository for EMF-based software models. The framework enables collaborative work of several modelers directly via peer-to-peer connection providing semantic version control of models.

Operation-based. All of these approaches identify themselves as the operation-based difference representation. They usually use basic edit operations such as *create*, *delete* and *change* (or similar and more) which is a general concept being relevant to many difference representation approaches. Regardless of their difference representation techniques, they employ these basic operations only for recognizing the type of model changes, but information about model changes is generally stored in various forms as discussed above.

Lessons Learned.

This section discusses the existing related approaches based on the criteria whether representation by these approaches can provide small modeling deltas for both sequential and concurrent collaborative modeling. The most approaches focus on only some aspects of (or only one of) these collaborative modeling scenarios.

Software models themselves are too complex and structured for modeling tool developers and users. Modeling deltas represented by models or graphs usually consist of additional conceptual information for representing its modeling or graph concepts alongside actual difference/change information. In this case, model- or graph-based modeling deltas might not be as small as text-based modeling deltas. Moreover, if model differences are again represented by models or graphs, further extension of this class of approaches might require more knowledge and effort in developing further services on the top. *Model- and graph-based difference representations* are unlikely to show high performance in concurrent collaborative modeling because of their complex data structures which causes difficulties in rapid synchronization in real-time. During the evolutionary life-cycle, if all difference information is stored in a database, the database becomes very complex and huge with an associated and entity-relational data set. In *database-based representation*, modeling deltas may not easily be identified and reused.

Modeling deltas represented in *textual forms* are the most likely to be small, efficient and well-suited for collaborative modeling for many reasons: (1) directly executable descriptions of model differences; (2) easy to implement; (3) fully expressive, yet unambiguous providing necessary knowledge; (4) easy to synchronize with high performance; (5) minimalistic in comparison to model-, graph- and database-based representations.

Literature reviews show that research on difference representation for collaborative modeling is still in its infancy. Considering the aforementioned discussion, collaborative modeling requests a textual *difference language (DL)* to represent modeling deltas in sequential and concurrent collaborative modeling.

3 Requirements

As long as there are several modeling languages and model designing tools, collaborative modeling approaches should not rely on a specific modeling language or modeling tool. Collaborative modeling has to support both sequential (macro-versioning) and concurrent (micro-versioning) collaboration forms. Since difference representation lies at the core of

collaborative modeling, this section defines several requirements that a proposed DL-based difference representation has to fulfill.

Awareness of Content and Layout. The content of software models is recognized by looking at the meta-models they conform to. The graphical design of models is aligned by their layout data in the model editors of model designing tools. There are several graphical modeling editors which display the modeling content with their layout representation information. Like models conform to their meta-models, layout information is represented by and conform to their graphical notation. In order to provide collaborative modeling on such graphical modeling tools, DL has to be aware of layout notation together with the meta-models (content) of modeling languages. For instance, if a designer changes the position and size of a modeling artifact in a graphical editor, the same changes have to simultaneously occur in the modeling editors of other tool instances as well.

Genericness. There are several modeling languages and graphical notations following diverse formal specifications and concepts. DL has to be generic with respect to the meta-models of modeling languages and their layout without restricting itself to a particular modeling language or tool. If DL is generic, its collaborative modeling support can then be tailored to the wide range of modeling languages and tools.

Supportiveness. As defined in Section 1, collaborative modeling forms two scenarios such as concurrent *micro-versioning* and sequential *macro-versioning*, whereas difference representation is a common and fundamental concern for both. The most approaches investigated in Section 2 focus on only some aspects of these collaborative modeling scenarios. DL has to support both scenarios of collaborative modeling by being applicable, persistent, implementable and expressive.

These requirements are also partly addressed to in [CRP07], [HK10], [SBG08]. They are proper characteristics for emerging an appropriate difference representation approach for collaborative modeling in MDS. They are the efficient design of the data structure for representing model differences in modeling deltas. These significant principles are also the design foundations for DL that contribute to empowering the qualification and solidity of difference representation. DL aims at fulfilling these requirements throughout this paper.

4 Modeling Deltas

In collaborative modeling, changed modeling artifacts have to be properly identified and represented in modeling deltas for further processing of software models. A *model change* defines any kind of change made to a modeling artifact during its evolutionary life-cycle. Modeling deltas including model changes are the core entities in representing model changes and building collaborative modeling on top. DL focuses on representing model changes in modeling deltas. Section 4.1 demonstrates the overall conceptual idea of DL. Section 4.2 gives a very simplified example of DL-based change representation. Section 4.3 explains a subset of supplementary DL services required for collaborative modeling.

4.1 Conceptual Idea

In order to use the collaborative modeling environment, specific DLs have to be generated from the meta-models of modeling languages. DL is a generic approach with respect to the meta-models of modeling languages. It is conceptually a family of domain-specific languages. As long as the modeling concepts of any modeling language can be recognized by looking at the meta-model of that language, a specific DL for a particular modeling language is generated by the DL generator service (explain in Section 4.3) importing its meta-model. Then, modeling deltas can be represented in terms of DL for instance models conforming to that modeling language.

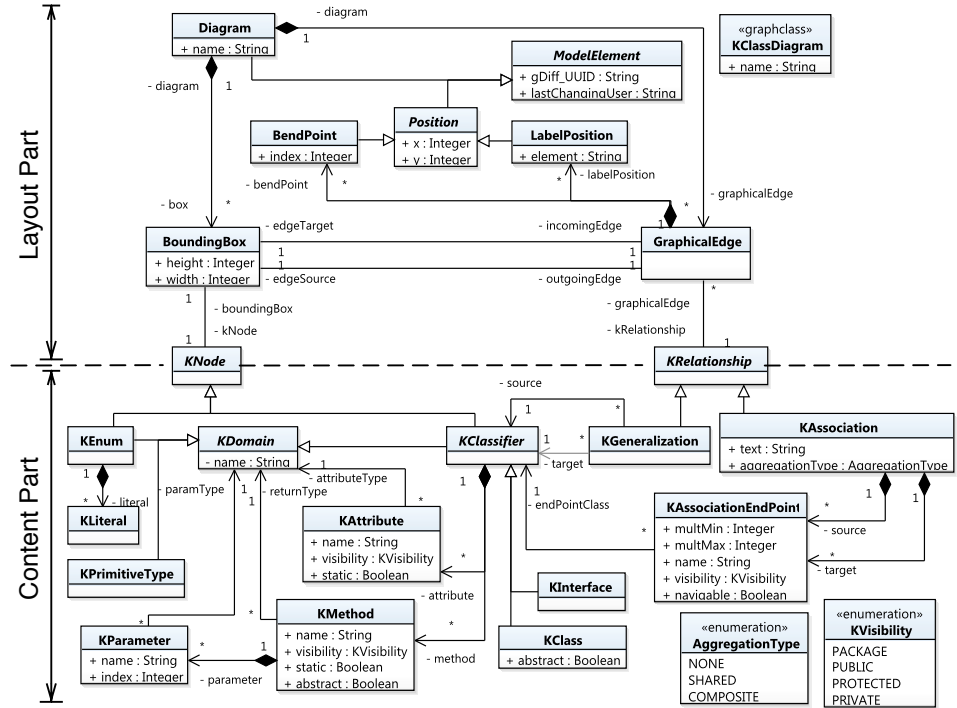


Fig. 1: UML Class Diagram Meta-model

Figure 1 depicts the meta-model of a subset of UML class diagrams which is used throughout this paper as an example. The meta-model is separated into two parts by a dashed line. Below of the line, it depicts the *content part* which is used to represent the subset of the modeling concepts of UML class diagram. In graphical modeling, every modeling object has design information such as color, size, and position, also called layout information. Above the dashed line, the figure consists of the *layout part* that is used to depict notation for layout information for the conceptual part. This way of designing meta-models is another advantage of the DL approach satisfying the *Awareness of Content and Layout* requirement. This allows for using the same collaborative modeling environment for different modeling

content. The complete meta-model is used for creating overall collaborative modeling application explained in Section 5.

4.2 Motivating Example

In order to explicitly explain how changes are represented in terms of DL, this section presents a simplified example of the DL-based change representation in modeling deltas. A very simple UML class diagram [RJB04, pp. 47ff] is chosen as a running example to apply the DL approach. Figure 2 depicts two subsequent revisions of the same class diagram namely Rev_1 and Rev_2, they conform to the meta-model depicted in Figure 1. Figure 2 further portrays two concurrent instances of the latest revision, in this case, Rev_2. Two designers, namely Designer_1 and Designer_2, are working on these concurrent instances.

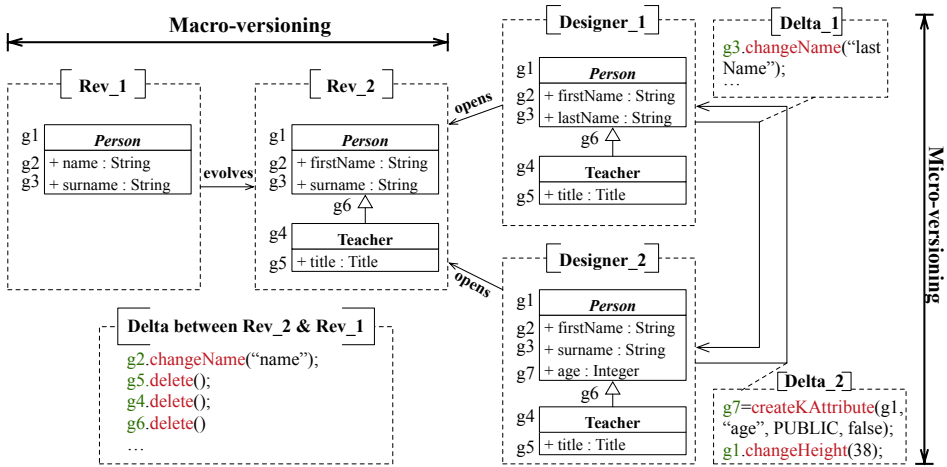


Fig. 2: Modeling Deltas in Collaborative Modeling

In the first revision, the model has only one class named Person. While evolving from the first revision to the second, the following changes are made: a class with the name Teacher is added and it is generalized to the existing class Person, the attribute name of the class Person is changed to `firstName`.

According to the `ModelElement` class of the meta-model in Figure 1, each model element has an attribute named `gDiff_UUID` which means all model elements are assigned to *universally unique identifiers (UUID)*. Therefore, each model element in this example also has a persistent identifier. Assigning UUID to modeling artifacts allows for identifying and keeping track of modeling artifacts over time. Any particular modeling artifact can be traced by detecting the predecessor and successor artifact of that modeling artifact (inter-delta references). The persistent identifiers are always used in the DL operations in order to preserve consistency of modeling deltas. They are also used as indicators to refer to modeling concepts from the DL operations in modeling deltas (delta-model references).

The DL operations representing modeling deltas in Figure 2 conform to a specific DL generated from the meta-model depicted in Figure 1. Each delta operation consists of a `Do` part (cf. `g2.changeName("name");`) which describes the *kind of change* by means of *operations* (one of create, change, delete) and an `Object` part (with attributes if required) (cf. `g2.changeName("name");`) which refers to the modeling concept. The modeling deltas between the subsequent revisions refer to macro-versioning and the modeling deltas between the two concurrent instances refer to micro-versioning.

Each object of any modeling language can be created, deleted or attributes of each object can be changed during the evolution process. DL defines model changes by three basic operations such as creation, deletion of modeling artifacts or change of the attributes of these artifacts. These basic operations are sufficient set of operations to represent any kind of model changes by DL.

Macro-versioning. In macro-versioning (cf. Subversion [CSFP04], Git [Sw08], Mercurial [Ma10]), the differences between subsequent revisions are usually identified and represented in reverse order i.e. they represent changes in the *backward deltas* [KW15]. Because these systems intend to store differences as directly executable forms which are more practical in retrieving the earlier revisions of software systems. They store the most recent revision of software systems and several differences deltas for tracing back to the initial revision. Hence, the latest revision is the most frequently accessed revision. DL-based difference representation also follows the similar art of delta representation. Thus, the modeling delta between the first and second revisions is represented in the backward delta (`Delta` between `Rev_2` and `Rev_1`). Modeling deltas are directly executable descriptions of model differences i.e. application of the modeling delta to the second revision results in the first revision. When the delta `Delta` between `Rev_2` and `Rev_1` is applied to the revision `Rev_2`, the first operation changes the attribute `firstName` of the class `Person` to `name`, and the following deletion operations delete the attribute of the class `Teacher`, the class `Teacher` itself and the generalization assigned to `g6`. For the sack of simplicity, the delta depicts the conditional full-stops (...) on the last line, meaning the delta consists of other DL operations for changing layout information.

Micro-versioning. In the second revision, the model is then being further developed by two designers concurrently. `Designer_1` changes the attribute of the class `Person` from `surname` to `lastName`. This change is then sent to the other instance as a `Delta`. On the other instance, `Designer_2` creates a new attribute with name `age` in the class `Person`. That change is sent to the instance, `Designer_1` is working on, as a `Delta`. When the new attribute is added, the height of the class `Person` is increased and this change of layout information is also described by DL operation (`g1.changeHeight(38);`) in the same delta. Both of these deltas are represented as *forward* deltas in this case. Because the latest changes have to be reflected on the parallel models, thus, changes have to be applied to models in forward order.

4.3 DL Services

In collaborative modeling, several services are involved in generating specific DLs, calculating and applying the DL-based modeling deltas. As all DL services are explained in [KW15], this section briefly revisits the services which are involved in collaborative modeling.

DL Generator. DL generator generates a specific DL for a particular modeling language by importing its meta-model. While generating the specific DL, it inspects all of the concrete (i.e., non-abstract) meta-classes of a given meta-model and the attributes of these meta-classes. For each meta-class, it generates interfaces with implementations for creation and deletion operations, as well as for change operations for each attribute. Modification of the attribute named `gDiff_UUID` (e.g. the class *ModelElement* in Figure 1 is not allowed by default as it tightly defines the identity of a modeling artifact, therefore, should not be changed as the part of model changes. A specific DL is always generated in the form of a Java Interface.

After generating a specific DL for the given meta-model, several DL services still have to be involved in collaborative modeling as depicted in Figure 3. It depicts an abstract architecture of collaborative modeling including *server* and *client* sides.

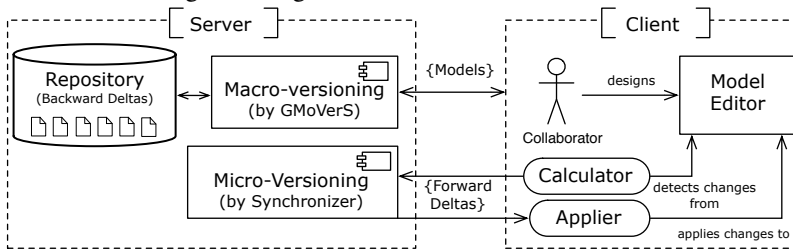


Fig. 3: Overall Architecture of Collaborative Modeling

The server-side consists of the concurrent synchronization service *micro-versioning* and sequential version control service *macro-versioning* which uses the DL-based modeling delta repository. On the client side, while collaborators are designing models using a model editor, their changes are constantly detected by the *calculator* service and sent to other clients as modeling deltas. Once these deltas arrive at other clients, they are applied to models by the *applier* service. During collaboration, designers may store a particular state of their model whenever their model is complete and correct. They are able to manage models and load any revisions of their model which they have saved earlier. The macro-versioning feature is provided by the DL application GMoVerS (Generic Model Versioning System) [KW15].

Calculator. Calculation of modeling deltas depends on the form of collaboration whether it is sequential (macro-versioning) or concurrent (micro-versioning) collaboration. In micro-versioning, they are calculated by listening for changes in models by *change listeners* [HK10]. Because changes have to be synchronized in real-time providing sufficient performance. In

macro-versioning, modeling deltas between subsequent model revisions are calculated using *state-based* comparison of subsequent revisions [Ke13]. The state-based comparison and change listener are two different features of the DL delta *calculator* service. For instance, all modeling deltas in Figure 2 are calculated by this service. The modeling delta *Delta* between *Rev_2* and *Rev_1* is computed by the *state-based* comparison (implemented using SiDiff algorithm [Ke13]), whereas the modeling deltas on both instances of *Designer_1* and *Designer_2* is computed by the *change listener* in real-time. The state-based comparison feature can calculate both, *forward* and *backward* deltas, whereas change listener can calculate only *forward* deltas.

Applier. In collaborative modeling, modeling deltas are applied to the base model in order to transform them from one revision to another. Application of modeling deltas to the base models is provided by the DL delta *applier* service [KW15]. In case of loss or damage of information on the initial copy of a model, designers might feel a need to revert their model for obtaining earlier versions or undoing recent changes they made. For instance, in Figure 2, the applier applies the *Delta* between *Rev_2* and *Rev_1* to *Rev_2* to revert it to *Rev_1*. The applier is also employed in micro-versioning in order to propagate model changes on the concurrent instances of a shared model. In Figure 2, the applier is used to apply both *Delta_1* and *Delta_2* to the instances of *Designer_1* and *Designer_2*, respectively.

The both, DL calculator and applier, services follow the general principles and do the same core operating tasks of difference calculator and applier for textual version control systems. But, their realization follows modeling concepts i.e. the DL calculator and applier operate on the composite and graph-like software models, whereas the latter operate on the textual documents.

5 Application: Kotelett

The collaborative modeling applications are established by the specific orchestrations of the DL services and on the top of the DL-based difference presentation. This section explains the collaborative modeling application *Kotelett* and other ongoing work.

The collaborative modeling application entitled *Kotelett tool* was initially developed by a students project group in the Software Engineering Group at the Carl von Ossietzky University of Oldenburg. *Kotelett* takes advantage of the DL-based modeling deltas for exchanging changes among various collaborators of the shared model. Figure 4 depicts the overall user interface of *Kotelett*. It displays two independent tool instances working on the same model concurrently. Each *Kotelett* instance consists of several windows as explained below. When the tool is launched, it shows the list of models which are currently available in the repository and asks the user which model to join as a collaborator. But, the users can open multiple models during collaboration.

In micro-versioning, modeling deltas are not stored for reverting changes. However, reversion of changes in micro-versioning happens on the client side of the editor and provided by the

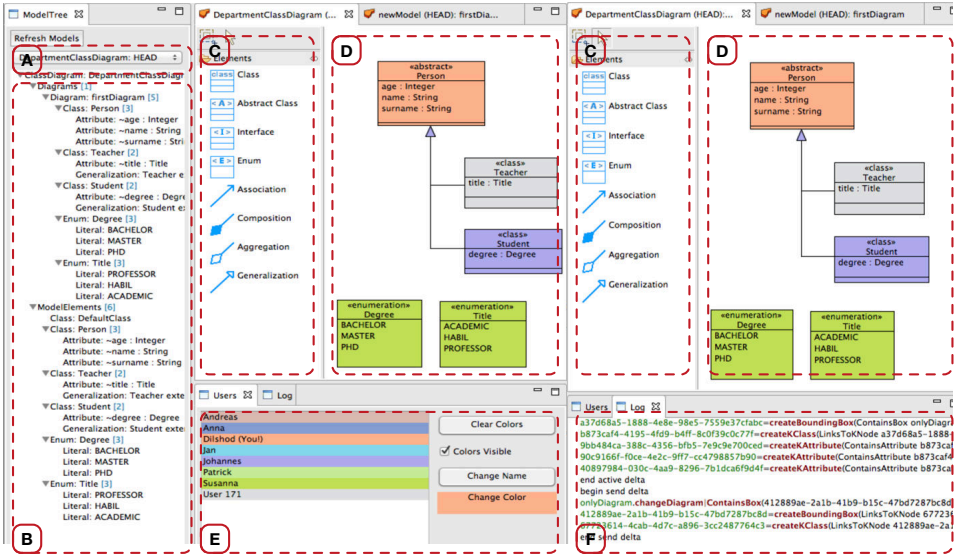


Fig. 4: Kotelett Screenshot

Redo/Undo features of the editor. Kotelett can save the model at a particular time and by clicking a save button whenever the model is complete and correct. When the tool is asked to save the model, it calculates the differences (backward deltas for macro-versioning) between the last and current revisions. It then stores these differences in the form of the DL-based modeling delta. On the left instance of the Kotelett screenshot, the pop-up menu *model browser* (A) allows for selecting and opening any version of the model which was saved previously. The menu lists all automatically and manually saved versions of the current model. Once any previous revision is selected, that revision is reverted by the *DL applicer* service by applying sequential modeling deltas to the base model.

The *model tree* (B) shows the list of diagrams the user is currently working on. Each diagram belongs to a specific model. It also shows the list of model elements that are created in the current diagram. The *model editor* (D) area is the main part which allows users for designing UML class diagrams. The *Elements* (C) part lists the most important notations of UML class diagram where the users can select and draw that element in the model editor. These notations of the UML class diagram are created based on the meta-model depicted in Figure 1. The correctness of the model on this editor is checked according to that meta-model, automatically. The *user list* (E) (left instance) window lists all users that are currently working on the initial diagram. These users are highlighted with different colors in order to show the clear distinction between them and to recognize which change is made by which user in the editor. The *log* (F) window (right instance) constantly displays the modeling deltas that are exchanged among collaborators after each user action. Creating one model element on the graphical modeling editor may result in one or several change

operations that are contained in one modeling delta and synchronized between collaborators. Modeling deltas are represented by a specific DL generated from the meta-model in Figure 1.

The JGraLab (Java Graph Laboratory) [DW98] technical space is used for realizing DL in Kotelett. TGraph [ERW08] is used to represent software models internally, whereas the TGraph schema used to define meta-models. JGraLab further provides generic features for defining in-place model manipulations to implement the DL-based operations. The architectural and implementation concepts of Kotelett is explained in [KW15] in detail.

DL in UML Designer.

As ongoing work, the collaborative modeling approach is being applied to UML designer which is an EMF- and Sirius-based domain-specific modeling tool [Ob17]. DL will be applied to UML designer using the EMF technical space. To represent model changes in UML designer, a specific DL will be derived from the EMF-based Ecore meta-model [St08]. The delta calculator and applier services will be extended by EMF transactional editing domain and command stack extensions. All other underlying technologies remain the same and unchanged.

6 Evaluation

The collaborative modeling approach in this paper is validated to achieve its intended goals by fulfilling the requirements identified in Section 3. More specifically, these requirements are revisited in this section for presenting their fulfillment by DL. All DL concepts, services and applications provide evidence to the fulfillment of each of these requirements. Finally, the extend-ability of the approach is shown in this section.

6.1 Validation

Kotelett is used in Software Engineering lectures for teaching purposes by a group of students including more than ten collaborators in parallel. The tool was also used experimentally by more than ten users located over long distance (Germany, Canada, Mozambique, and Uzbekistan), all connecting to the server located in Germany.

During these experiments, the tool has shown sufficiently high performance by synchronization of small DL-based modeling deltas. So far Kotelett did not face any change conflicts in micro-versioning. This probably is attributed to the rapid synchronization of small modeling deltas. Exchanging binary formats of deltas can be faster, but so far it was not necessary. It can be experimented for optimization purposes. But, it probably might not be expressive enough making it difficult for further extensions. For merging various model revisions in macro-versioning, Kotelett aims at employing the existing *merge* feature provided by the *JGraLab* technical space [DW98].

In order to accomplish a solid, appropriate, generic and efficient collaborative modeling approach, this paper has kept the requirements defined in Section 3 in its focus and thoroughly

attempted to satisfy them. Based on the layout notation part and modeling language notation part of meta-models (e.g. Figure 1), the collaborative modeling environment is aware of content and layout of models. The same collaboration environment can be used for modeling language and layout notations (*Awareness of Content and Layout*), simultaneously. The approach is generic with respect to the meta-models of modeling languages. The collaborative modeling environment can be applied to wide range of modeling languages and tools by generating specific DLs by the DL generator explained in Section 4.3. It then can import any meta-model defining modeling concepts and layout notation and generates specific DL for it (*Genericness*). If meta-models are changed, they can be re-imported and specific DLs can be regenerated newly for changed meta-models.

The collaborative modeling approach is applied to Kotelett and other applications under development. These applications support concurrent (by *micro-versioning*) and sequential (by *macro-versioning*) collaborative modeling (*Supportiveness*). They are developed on top of DL which provides more efficient ways of representing, exchanging and synchronizing modeling deltas improving the performance of data processing. DL is utilized as solid and common syntactic ground for representing modeling deltas in these applications.

6.2 Extendability

In the approach, existing functionality can be extended, new features can be added by extending underlying technologies and concepts such as the DL services. The collaborative modeling approach can represent software models under collaboration using internal graph-like structures [ERW08] and graphical editors can be separated from that graph representation. This allows for extending graphical editors without changing the underlying concepts and technologies of collaborative modeling. This enables tool developers to define further graphical notations, shapes, and views as they want and reuse existing collaborative modeling environment without any further development effort. As depicted in Figure 1, the layout notation part (above the dashed line) of the meta-models enables extendability of the approach for further modeling languages with the same layout notation. The modeling concept part (below the dashed line) of the meta-model should be replaced by the meta-model of another modeling language. Eventually, the same layout information can be reused for further modeling languages.

The catalog of the DL services can also be replaced by other implementations and extended with further services or features. Any service, component or plug-in required for collaborative modeling can easily be developed and registered in the service catalog which is practically useful for further service-oriented modeling tool development. The only prerequisite for these services is to recognize the syntax of DL. Eventually, these services can again be involved in service orchestrations for establishing the collaborative modeling applications.

7 Conclusions

The approach proposed in this paper is meta-model generic, aware of both content and layout of models and supports sequential and concurrent collaborative modeling. As *modeling deltas* are first-class entities and play an essential role in storing, exchanging and synchronizing changes between the subsequent and concurrent revisions of evolving models, representation of modeling deltas is very crucial in both forms of collaboration. Thus, the collaborative modeling approach is elaborated on top of the efficient difference language notation which is employed in both micro-versioning and macro-versioning. Both forms rely on the same base difference representation language for modeling deltas.

A proposed operation-based DL serves as a common change representation and exchange format for collaborative modeling, making software models commonly available to multiple users in different locations. Micro-versioning achieves high performance of synchronization by exchanging small DL-based forward deltas, whereas macro-versioning can effectively store the change histories of evolving modeling artifacts in model repositories as the DL-based backward deltas. These and further collaborative modeling scenarios can be developed by specific orchestrations of the DL services which can produce, store, exchange, synchronize and apply modeling deltas in collaborative modeling. Kotelett can be downloaded at <https://pg-kotelett.informatik.uni-oldenburg.de:8443/build/stable/> and presented at the conference as well.

References

- [Al07] Altmanninger, K.; Bergmayr, A.; Schwinger, W.; Kotsis, G.: Semantically enhanced conflict detection between model versions in SMOVer by example. In: Proc. of the Int. Workshop on Semantic-Based Software Development at OOPSLA. 2007.
- [AP03] Alanen, M.; Porres, I.: Difference and union of models. In P.Stevens, J.Whittle, and G. Booch, editors, Proc. 6th Int. Conf. on the UML, Springer, LNCS 2863:2–17, 2003.
- [Ap17] AppJet Inc.: , Etherpad. <http://www.etherpad.com>, visited on 01.02.2017.
- [Br10] Brosch, P.; Kappel, G.; Seidl, M.; Wieland, K.; Wimmer, M.; Kargl, H.; Langer, P.: Adaptable Model Versioning in Action. in: Proc. Modellierung 2010, Klagenfurt, Österreich, LNI 161:221–236, March 24-26 2010.
- [Ci15] Cinergix Pty.: , CreateLy. <http://www.creately.com>, visited on 01.06.2015.
- [CRP07] Cicchetti, A.; Ruscio, D.; Pierantonio, A.: A Metamodel independent approach to difference representation. Journal of Object Technology, 6:9:165–185, October 2007.
- [CSFP04] Collins-Sussman, B.; Fitzpatrick, B.; Pilato, M.: Version Control with Subversion. O’Reilly Media, June 2004.
- [DMA13] Dirix, M.; Muller, A.; Aranega, V.: GenMyModel: UML case tool. In: ECOOP. 2013.
- [DW98] Dahm, P.; Widmann, F.: Das Graphenlabor. Technical Report 11/98, Universität Koblenz-Landau, Koblenz, 1998. Fachberichte Informatik.

- [ERW08] Ebert, J.; Riediger, V.; Winter, A.: Graph technology in reverse engineering. The TGraph approach. In: 10th Workshop Software Reengineering. GI LNI. pp. 23–24, 2008.
- [ESG91] Ellis, C.; Simon, G.; Gail, R.: Groupware: Some Issues and Experiences. *ACM*, 34(1):39–58, 1991.
- [Go17] Google Inc.: , Google Docs. <http://docs.google.com>, February 2017. online.
- [HK10] Herrmannsdoerfer, M.; Koegel, M.: Towards a generic operation recorder for model evolution. In: Proceedings of the 1st International Workshop on Model Comparison in Practice. *ACM*, pp. 76–81, 2010.
- [HK13] Helming, J.; Koegel, M.: , EMFStore., 2013. <http://eclipse.org/emfstore>.
- [Ke12] Kehrer, T.; Kelter, U.; Ohrndorf, M.; Sollbach, T.: Understanding model evolution through semantically lifting model differences with SiLift. In: Software Maintenance (ICSM), 2012 28th IEEE International Conference on. *IEEE*, pp. 638–641, 2012.
- [Ke13] Kehrer, T.; Rindt, M.; Pietsch, P.; Kelter, U.: Generating Edit Operations for Profiled UML Models. In: MoDELS. pp. 30–39, 2013.
- [KW15] Kuryazov, D.; Winter, A.: Collaborative Modeling Empowered by Modeling Deltas. *ACM*, Lausanne, Switzerland, 09 2015.
- [KWB03] Kleppe, A.; Warmer, J.; Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston, USA, 2003.
- [Ma10] Mackall, M.: The mercurial scm. Internet Website, last accessed 05.07.2017, 2010.
- [My86] Myers, E. W.: An O (ND) difference algorithm and its variations. *Algorithmica*, 1(1):251–266, 1986.
- [Ob17] Obeo Network: , UML designer. <http://www.uml designer.org/>, visited on 02.10.2017.
- [RJB04] Rumbaugh, J.; Jacobson, I.; Booch, G.: Unified modeling language reference manual. Pearson Higher Education, 2004.
- [SBG08] Sriplakich, P.; Blanc, X.; Gervais, M.: Collaborative software engineering on large-scale models: requirements and experience in modelbus. In: Proceedings of the 2008 ACM symposium on Applied computing. *ACM*, pp. 674–681, 2008.
- [SSA14] Seidl, C.; Schaefer, I.; Abmann, U.: DeltaEcore-A Model-Based Delta Language Generation Framework. In: Modellierung. pp. 81–96, 2014.
- [St08] Steinberg, D.; Budinsky, F.; Merks, E.; Paternostro, M.: EMF: Eclipse Modeling Framework. Addison-Wesley Longman Publishing Co., Inc., 2008.
- [Sw08] Swicegood, T.: Pragmatic version control using Git. Pragmatic Bookshelf, 2008.
- [Ta12] Taentzer, G.; Ermel, C.; Langer, P.; Wimmer, M.: A fundamental approach to model versioning based on graph modifications: from theory to implementation. *journal: Software and Systems Modeling*, April 25 2012.

Enhancing MDWE with Collaborative Live Coding

Peter de Lange, Petru Nicolaescu, Thomas Winkler, Ralf Klamma¹

Abstract: Model-Driven Web Engineering (MDWE) methodologies enhance productivity and offer a high level view on software artifacts. Coming from classical software development processes, many existing approaches rather enforce a top-down structure instead of supporting a cyclic approach that integrates smoother with modern agile development. State-of-the-art MDWE should integrate established and emerging Web development features, such as (near real-time) collaborative modeling and shared editing on the generated code. The challenge when covering these requirements lies with synchronizing source code and models, an essential need to cope with regular changes in the software architecture and provide the flexibility needed for agile MDWE. In this paper, we present an approach that enables cyclic, collaborative development of Web applications by using traceability in model-to-text transformations to deal with the synchronization. We adopt a trace-based solution for collaborative live coding in order to merge manual code changes into Web application models and ensure that the open-source code repositories reflect both model and manual code refinements. Our evaluation shows a reliable code to model synchronization and investigates the usability in collaborative software development settings. With our approach we contribute to integrating agile development practices into MDWE.

Keywords: Model-Driven Web Engineering; Traceability; Model to Text Transformations; Collaborative Live Coding

1 Introduction

Mostly adopted in classical software development models, past MDWE research does not cope with the paradigm shift towards agile development [MR03], inclusion of end-users and various stakeholders into the development process and the increased communication and collaboration in (remote) teams on the Web. To adapt to this new intensive information exchange setting, a MDWE approach has to support development cycles with rapid changes in the architecture and code being simultaneously edited, all in a multi-user collaborative and *Near Real-Time* (NRT) scenario. Hence, traditional methods that enable the synchronization between model and code need to be adapted to the collaborative setting. Furthermore, they need to cope with powerful, collaborative frontend technologies and paradigms beyond simple website and client-server models.

In this paper, we present a cyclic MDWE development process in which model updates are synchronized with source code refinements. The authoring of models and code is

¹ RWTH Aachen University, Lehrstuhl Informatik 5, Advanced Information Systems Group (ACIS), Ahornstrasse 55, 52074 Aachen, Germany {lastname}@dbis.rwth-aachen.de

performed collaboratively in NRT on the Web, bringing together teams composed of various stakeholders. We apply related work on traceability [OO07] and synchronization [HLR08] from the model-driven engineering domain to formalize an agile collaborative MDWE method for the Web. In order to adapt the synchronization techniques to the NRT collaboration setting, we developed a trace model that provides linking information between model elements and source code artifacts. We apply this concept using a prototype that integrates a live collaborative code editor into an existing MDWE framework. All source code and traces are stored in a source code repository using the Git protocol.

In the following, we start with introducing the background and related work our paper is based on in Section 2, before Section 3 introduces our MDWE process. Section 4 presents the formalization of our traceability-based synchronization approach. Section 5 describes the integration of our proof-of-concept prototype into the CAE [La17], a Web-based MDWE framework. In Section 6, we describe and interpret the results of our user evaluation. Finally, Section 7 concludes this paper and provides an outlook on future work.

2 Background and Related Work

In the scope of MDWE, *Model to Text* (M2T) transformations are a special form of *Model to Model* (M2M) approaches, in which the target model consists of textual artifacts [Mv06], in this case the source code of the generated Web application. The target model is generated based on transformation rules, defined with respect to a models' metamodel [Me02]. Template-based approaches are (together with visitor-based approaches) the most prominent solution for M2T transformations [CH06]. Here, text fragments consisting of static and dynamic sections are used for code generation. While dynamic sections are replaced by code depending on the parsed model, static sections represent code fragments not being altered by the content of the parsed model [ORK14]. An important aspect of M2T transformations is *Model Synchronization*. It deals with the problem that upon regeneration, changes to the source model have to be integrated into the already generated (and possibly manually modified) source code. To achieve this, traces are used to identify manual source code changes during a M2T (re)transformation. In MDWE, managing traceability has evolved to one of the key challenges [ALC08]. Another challenge is the decision on the appropriate granularity of traces, as the more detailed the links are, the more error-prone they become [Go12, Va14]. In addition to model synchronization, *Round-Trip Engineering* (RTE) also considers changes in the source code which are propagated back into the model. Among others, formal definitions of model synchronization and RTE for M2M transformations have been proposed in [GW06] and [HLR08].

OOHDM was one of the first approaches towards the changing requirements in the development of Web applications in comparison to traditional applications [SR98]. It focuses on the hypertext structure of Web applications, as traditional software engineering concepts do not offer appropriate abstractions for those structures. Following the *separation of concerns* approach, OOHDM divides the development process into four tasks, i.e. the

conceptual design, the navigation design, the abstract interface design and the implementation. In *UML-based Web Engineering (UWE)*, the modeling of Web applications integrates the separation of concerns by separately modeling content, navigation, business processes and presentation [KKK07]. While stereotypes are used to define specific semantics of model elements, e.g. “navigationLink” for direct links, the *Object Constraint Language (OCL)* is used to define additional static semantics and constraints for a model element such as class invariants. *MagicUWE* is a plugin for the commercial CASE tool MagicDraw that supports the UWE notation [BK09]. Another prominent modeling language is *WebML*, in which Web applications are defined by high-level and platform-independent specifications [CFB00]. While a structural model describes the site content, a hypertext model is responsible for the composition of contents and the navigation between pages. In a presentation model, the layout and graphical representation of a page is defined independently of the displaying device and the language used for the visual representation. In addition, a personalization model is used to store user specific content [CFB00]. In 2013, an extension of WebML lead to the specification of the *Interaction Flow Modeling Language (IFML)*, a language that was adopted as a standard by the *Object Management Group (OMG)*. While especially UWE and WebML are the most prominent examples of recent developments in the domain of MDWE research, none of them are based on a (formalized) RTE approach that allows for an agile use of these approaches.

Medini QVT, developed by IKV++, is a commercial tool that implements the declarative part of the QVT specification, i.e. the QVT Relations. Thus, it supports incremental bidirectional M2M transformations and the generation of trace models during the transformation process. *UML Lab*, developed by Yatta, is a commercial modeling suite which is integrated into the Eclipse platform. It provides a graphical UML modeling editor and template-based M2T transformations, supporting reverse engineering and RTE. However, NRT collaboration facilities are not provided. Finally, *MOFScript* is a M2T transformation tool developed as an Eclipse plugin. As the MOFScript language does not depend on any actual metamodel, it can be used for code generation of arbitrary metamodels and their instances. In addition, it supports the generation of traces as well as the synchronization between models and source code. However, it does not provide any RTE facilities. Besides, the tool does neither offer NRT collaborative modeling nor coding functionalities. To our knowledge, there currently exists no agile NRT MDWE framework, that allows for a cyclic, collaborative development process with modeling phases followed by (live) coding phases and vice-versa.

3 Agile Collaborative MDWE

Our approach introduces an agile life-cycle for MDWE. From a general perspective on the modeling-coding cycle, changes in the architecture (i.e. changes happening in NRT as a team work result of multiple stakeholders with and without technical knowledge) are performed in the collaborative modeling phase. Detailed behavior is refined in the collaborative coding phase, using the automatically generated code from the model artifacts.

The cycle is achieved by synchronizing collaborative modeling phases and live source code editing. At any point in time, stakeholders can switch between one of the two phases. All changes of the model are immediately reflected in the generated source code. Changes to the source code are taken into account upon model-to-code regeneration, integrating them accordingly into the regenerated source code. Both modeling and coding is done on the Web in a NRT collaborative manner, meaning that changes in model and source code are directly visible to all stakeholders at all time.

Although our approach can be used for arbitrary MDWE frameworks and Web applications, in the scope of this work we consider Web applications composed of HTML5 and JavaScript frontends and RESTful microservice backends. Since especially frontend architectures can be highly unstructured, we propose to unify the architecture of applications developed with our approach through the usage of protected segments, that enforce a certain base architecture, facilitating both future service and frontend orchestration, maintenance and training efforts for new developers. Protected segments in the source code describe a functionality that is reflected by a modeling element. In order to encourage the reuse of software components, we allow changes which modify the architecture only in modeling phases. Since our approach offers a cyclic development process, this can be done instantly by switching to modeling, changing the corresponding element and returning to a new coding phase. To further enforce this methodology, before source code changes are persisted, a model violation detection is performed. This informs the user about source code violating its corresponding model, e.g. architecture elements manually added to the source code instead of being modeled. Concerning the synchronization between the code and the model, our collaborative MDWE process uses a trace-based approach. Changes in the code produce traces, which are used in the model-to-code (re)generation in order to keep the corresponding code synchronized to the model elements. This way, the process can be reflected without the need to implement a full RTE approach.

4 Model Synchronization Strategy

Although the conceptual idea of our model synchronization strategy can be applied to arbitrary Web application metamodels, for a better understanding we give simple examples of the appliance of our concept to the metamodel we use in our implementation at certain points in this section. Therefore, in Fig. 1 we illustrate an excerpt of our *Frontend Component* metamodel. It contains the three elements *HTML Element*, *Event* and *Function*, their attributes and their connections between each other.

Our general concept of model synchronization is depicted in Fig. 2. It is divided into two separate synchronizations: a synchronization between the source code and its trace model and a second synchronization between the source model and the source code. In the following, we explain our synchronization concept by using a simple formalization. We denote the source models by S_i , source code models by T_i and trace models by tr_i . The source- and source code-metamodels are denoted by M_S and M_T . We use the definition of

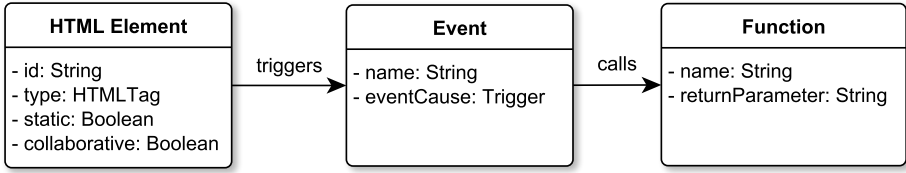


Fig. 1: Example model

synchronization expressed in [HLR08] as follows: two models A and B with corresponding metamodels M_A and M_B are *synchronized*, if

$$trans(A) = strip(B, trans) \quad (1)$$

holds for the transformation $trans : M_A \rightarrow M_B$ and a function $strip : M \times (M_A \rightarrow M_B) \rightarrow M$ that reduces a model of M with either $M = M_A$ or $M = M_B$ to only its elements relevant for the transformation. This definition uses the $trans$ and $strip$ functions [HLR08]. Intuitively, the $trans$ function expresses that applying a transformation to the source model yields the target model. The function $strip$ is used to remove any additional elements and map models to only the relevant source/target model. As an example, consider the *height* attribute of an HTML *img* tag. As it can be seen in Fig. 1, the *HTML Element* of our metamodel does not contain a *height* attribute, so this manually added attribute would not be part of the stripped model according to our transformation.

4.1 Synchronization of Source Code and Trace Model

Based on a first model S_1 , an initial generation of the source code T_1 and its trace model tr_1 is performed. As depicted in Fig. 2, the trace model tr_i is updated, once the source code changes. ΔT_{2i-1} are applied to the source code T_{2i-1} in the i -th code refinement phase. Formally, a single source code change can be denoted by one of the two functions $\delta_{M_T}^+ : M_T \times C \times \mathbb{N} \rightarrow M_T$ and $\delta_{M_T}^- : M_T \times C \times \mathbb{N} \rightarrow M_T$. While the former inserts a character $c \in C$ at position $n \in \mathbb{N}$, the latter deletes a character c from position n in the source code. Then, the result of applying the source code changes ΔT_{2i-1} on T_{2i-1} is defined by:

$$T_{2i-1} \Delta T_{2i-1} := \delta_{M_T}^\pm (\delta_{M_T}^\pm (\dots \delta_{M_T}^\pm (T_{2i-1}, c_1, n_1) \dots, c_{k-1}, n_{k-1}), c_k, n_k) =: T_{2i} \quad (2)$$

for $n_k, \in \mathbb{N}$, $c_k \in C$ and $k \in \mathbb{N}$.

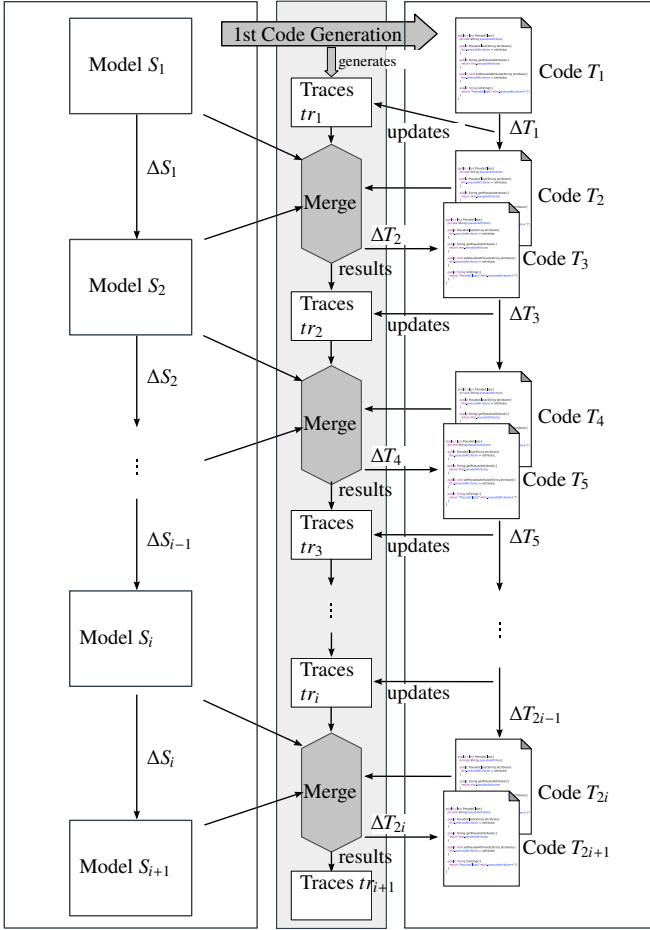


Fig. 2: Model synchronization

Considering Eq. 1, the condition $trans(T_{2i}) = strip(tr_i, trans)$ must hold for the synchronization between the updated source code T_{2i} and trace model tr_i :

$$trans(T_{2i}) = strip(tr_i, trans) \tag{3}$$

$$\iff trans(T_{2i-1}\Delta T_{2i-1}) = strip(tr_i, trans) \tag{4}$$

$$\iff trans(\delta_{M_T}^\pm(\delta_{M_T}^\pm(\dots\delta_{M_T}^\pm(T_{2i-1}, c_1, n_1)\dots, c_{k-1}, n_{k-1}), c_k, n_k)) = strip(tr_i, trans) \tag{5}$$

For the synchronization between source code and trace model, we only need to update the lengths of the segments of the trace model. Therefore, we assume $strip(tr_i, trans) = len(tr_i)$,

where $len(tr_i)$ is a tuple containing the segments' lengths. This leads to the following equation that must hold after the source code was updated:

$$trans(\delta_{M_T}^{\pm}(\delta_{M_T}^{\pm}(\cdots \delta_{M_T}^{\pm}(T_{2i-1}, c_1, n_1) \cdots, c_{k-1}, n_{k-1}), c_k, n_k)) = len(tr_i) \quad (6)$$

To satisfy this condition, each source code change needs to update the length of the segment that is affected by the deletion or insertion. Therefore, each $\delta_{M_T}^{\pm}$ is transformed to an update of the trace model tr_i :

$$\begin{aligned} &\delta_{M_T}^{\pm}(\delta_{M_T}^{\pm}(\cdots \delta_{M_T}^{\pm}(T_{2i-1}, c_1, n_1) \cdots, c_{k-1}, n_{k-1}), c_k, n_k) \rightarrow \\ &\delta_{len}^{\pm}(\delta_{len}^{\pm}(\cdots \delta_{len}^{\pm}(len(tr_i), n_1) \cdots, n_{k-1}), n_k) \end{aligned} \quad (7)$$

$$with \quad \delta_{len}^+((l_1, \cdots, l_m), n) := (l_1, \cdots, l_j + 1, \cdots, len_m) \quad (8)$$

$$\delta_{len}^-((l_1, \cdots, l_m), n) := (l_1, \cdots, l_j - 1, \cdots, len_m) \quad (9)$$

where $l_i \in \mathbb{N}$ for $i, m \in \mathbb{N}$, $1 \leq i \leq m$ is the length of the i -th segment and $l_j, j \in \mathbb{N}$ for $1 \leq j \leq m$ is the length of the segment that is affected by an insertion or deletion in the source code at position n .

4.2 Synchronization of Model and Source Code

In the model synchronization process, the last synchronized model S_i , the updated model S_{i+1} , the current trace model and the last synchronized source code T_{2i} are involved. By using the trace model of S_i , the applied model changes ΔS_i can be merged into the last synchronized source code T_{2i} without overwriting already implemented code refinements. As a result of the model synchronization, the updated source code T_{2i+1} and its trace model are obtained.

In general, model changes can be defined as functions of the form $\delta : M_S \rightarrow M_S$. More specifically, the model changes can be denoted by the following five functions, adapted from [HLR08]: δ_t^+ , δ_t^- : creating/deleting element of type t ; $\delta_{e,s1,s2}^+$, $\delta_{e,s1,s2}^-$: adding/deleting edge from element $s1$ to $s2$; and $\delta_{a,s1,v}^{attr}$: setting attribute a of element $s1$ to value v . As such, applying ΔS_i to S_i can be defined as a sequence of these changes:

$$S_i \Delta S_i := \delta_1 \circ \cdots \circ \delta_n(S_i) =: S_{i+1} \quad (10)$$

According to Eq. 1, the following equation must hold for the synchronization between model and source code:

$$trans(S_{i+1}) = strip(T_{2i+1}, trans) \quad (11)$$

$$\iff trans(S_i \Delta S_i) = strip(T_{2i+1}, trans) \quad (12)$$

$$\iff trans(\delta_1 \circ \cdots \circ \delta_n(S_i)) = strip(T_{2i+1}, trans) \quad (13)$$

Furthermore, as all parts of the source code that directly correspond to model elements are contained in protected segments, we assume $strip(T_{2i+1}, trans) = prot(T_{2i+1})$, where $prot(T_{2i+1})$ represents the source code that is reduced to the content of its protected segments. Finally, this leads to the following equation that must hold after the synchronization process:

$$trans(\delta_1 \circ \dots \circ \delta_n(S_i)) = prot(T_{2i+1}) \quad (14)$$

To satisfy this equation, each individual model change $\delta_i, i \in \mathbb{N}, 1 \leq i \leq n$ is transformed to its corresponding source code changes. Next, we first introduce formulas that are needed for the later transformations.

Attribute value: the value of the attribute labeled *name* of a model element *elm* is denoted by $attr_{name}(elm) := (c_1, \dots, c_k)$ with $c_i \in C$ for $i, k \in \mathbb{N}, 1 \leq i \leq k$.

Position and length of an element: the position of the first character of a model element *elm* within a file is defined by $pos_{seg}(elm)$. The length of *elm* is defined by $len_{seg}(elm)$.

Position and length of an attribute: the position of the first character of an attribute *a* of a model element *elm* is defined by $pos_{attr}(a, elm)$. The length of *a* is defined by $len_{attr}(a, elm)$.

Template: a template for an element *elm* of type *t* is denoted by

$$temp_t(attr_{name_1}(elm), \dots, attr_{name_n}(elm)) := (c_1, \dots, c_k)$$

with $c_i \in C$ for $k, i \in \mathbb{N}, 1 \leq i \leq k$. The attributes are used for the instantiation of the variables occurring in the template. We further define two functions that ease the formulas for deleting and inserting multiple characters:

$$\delta^{*+}(T, (c_1, \dots, c_k), n) := \delta_{MT}^+(\dots \delta_{MT}^+(T, c_k, n+k) \dots, c_1, n) \quad (15)$$

$$\delta^{*-}(T, n, k) := \delta_{MT}^-(\dots \delta_{MT}^-(T, c_{n+k}, n+k) \dots, c_n, n) \quad (16)$$

While the former inserts a tuple of characters starting from position *n* into a file, the later deletes the characters c_n, \dots, c_{n+k} at the positions $n, \dots, n+k$ from a file.

As the transformation of model to source code changes is highly dependent on the type of the updated model elements, the concept for synchronization is shown exemplary for *Events* of the example model described in Fig 1. A valid *Event* element has two edges *e* and *e'* that connect it to an *HTML Element* *h* and to a *Function* element *f*, respectively. Then, a newly created *Event* element is transformed to source code changes by

$$\delta_i^+(S_i) \rightarrow \delta^{*+}(T_{2i}, t_{event}, pos_{events}) \quad (17)$$

where pos_{events} references the position in the source code that contains all events and t_{event} is the following template:

$$t_{event} := temp_t(attr_{name}(event), attr_{cause}(event), attr_{name}(f), attr_{id}(h)) \quad (18)$$

Thereby, a new source code artifact representing the *Event* element is inserted into the source code. The deletion of an *Event* element is transformed as follows:

$$\delta_i^-(S_i) \rightarrow \delta^{*-}(T_{2i}, pos_{seg}(event), len_{seg}(event)) \quad (19)$$

The code artifact of the deleted *Event* element is removed from the source code. Finally a value of an attribute a is transformed to source code changes:

$$\delta_{a,event,v}^{attr}(S_i) \rightarrow \delta^{*+}(\delta^{*-}(T_{2i}, pos_{attr}(a, event), len_{attr}(a, event)), v, pos_{attr}(a, event)) \quad (20)$$

Thus, the old value of the attribute is first deleted from the source code and its new value is inserted at the same position. As we are only considering valid models and according to our model, an *Event* element needs two edges, we can assume that for each deletion of an edge there is also an insertion of a new edge. Therefore, we only transform edge updates. Since an *Event* element has two edges, we need to differentiate:

1. If the current *HTML Element* h connected to an *Event* is changed to another *HTML Element* h' , the transformation from model to source code is:

$$\delta_{e,event,h'}^+(\delta_{e,event,h}^-(S_i)) \rightarrow \delta^{*+}(\delta^{*-}(T_{2i}, pos_{attr}('id', h), len_{attr}('id', h)), attr_{id}(h'), pos_{attr}('id', h))$$

2. If the current *Function* element f of an *Event* is changed to another *Function* element f' , the source code is modified according to:

$$\delta_{e,event,f'}^+(\delta_{e,event,f}^-(S_i)) \rightarrow \delta^{*+}(\delta^{*-}(T_{2i}, pos_{attr}('name', f), len_{attr}('name', f)), attr_{name}(f'), pos_{attr}('name', f))$$

5 Realization

We integrated our model synchronization strategy into a Web-based collaborative modeling environment called *Community Application Editor* (CAE) [La16]. CAE uses a template-based MDWE approach to support NRT collaboration between developers and other involved stakeholders. In the scope of this paper's contribution, we extended the frontend with a *Live Code Editor* widget based on the *ACE editor*², which realizes the collaborative NRT editing of (generated) source code directly in the browser. The NRT collaboration and shared editing features are realized using the Yjs [Ni16] framework. Available as an open source Javascript library, Yjs enables shared editing for arbitrary data types and formats (e.g. lists, maps, objects, text, JSON) on the Web. It uses protocols such as WebRTC and WebSockets for message propagation in NRT. Integrated with the live code editor widget, our framework only needs to manage the Yjs *Collaboration Spaces* (similar to a chat room, with all involved users being able to collaborate on one application model and code). We integrated the model synchronization and trace generation functionality into the Java-based backend of the CAE and extended it with means to manage local Git repositories used by the live code editor widget to update the source code.

² <https://ace.c9.io>

Trace Generation and Model Synchronization The template engine, implemented in the backend of the CAE, forms the main component for trace generation and model synchronization. It is used for both the initial code generation, as well as for further model synchronization processes. Except for some special cases, like renaming or deleting files, applying the *strategy design pattern* allows us to use the same methods for both the initial code generation and model synchronization.

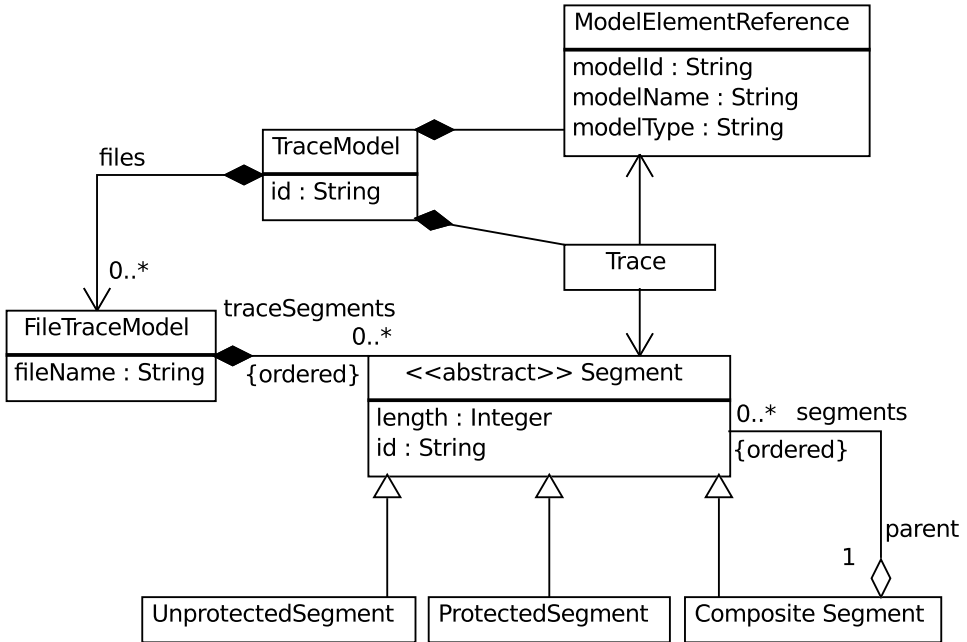


Fig. 3: Trace model

Fig. 3 depicts our trace model, adapted from the metamodel of traces presented in [OO07]. For each *FileTraceModel*, and thus for each file, we instantiate a template engine class, which can hold several template objects. A template object is a composition of *Segments*, generated by parsing a template file. A template file contains the basic structure of an element of the Web application’s metamodel. In such a file, variables are defined to be used as placeholders, which are later replaced with their final values from the model. Additionally, the template file contains information about which part of the generated source code is protected or unprotected. Based on the template syntax for variables and unprotected parts, a template file is parsed and transformed into a composition of segments of the trace model. For each variable a protected segment is added, and for each unprotected part, an unprotected segment is added to the composition. The parts of a template file that are neither variables nor unprotected parts are also added to the composition as protected segments. According to the definition of model synchronization for M2T transformations, Eq. 1 must hold for the model synchronization. Thus, we need to update the content of each variable for all templates of all model elements. However, maintaining a trace and a model element

reference for all of these variables is not feasible due to the large size of such a file trace model. Instead, traces are only explicitly maintained for the composition of segments of a template. Linking a segment of a variable to its model element is done implicitly by using the element's id as a prefix for its segment id. When templates are appended to a variable, the type of its linked segment is changed to a composition.

Following the strategy design pattern, we implemented an *Initial Generation Strategy* and a *Synchronization Strategy*, which are used by our template engine. Each synchronization strategy instance holds a reference to the file trace model of the last synchronized source code to detect new model elements as well as to find source code artifacts of updated model elements. As in some cases source code artifacts of model elements can be located in different files, a synchronization strategy can also hold multiple file trace models in order to find code artifacts across files. After a template engine and its template strategy were properly initiated, the template engine is passed as an argument to the code generators. These create template instances for the model elements based on the template engine. The engine checks if a segment of the model element is contained in the trace models file by recursively traversing its segments. If a corresponding segment for the model element was found, a template reusing this segment is returned. Otherwise, a new composition of segments, obtained by parsing the template file of the model element, is used for the returned template. For new model elements, new source code artifacts are generated. For updated elements, their corresponding artifacts are reused and updated. As templates can contain other templates in their variables, these nested templates need to be synchronized as well. In the generated final files, source code artifacts of model elements that were deleted in the updated model must be removed from the source code. Therefore, the nested templates, more specifically their segment compositions, are replaced with special segments by the synchronization strategy. By following the *proxy design pattern*, these special segments are used as proxies for the original compositions and ensure that templates of deleted model elements are removed from the final source code.

To ensure that source code artifacts that directly correspond to a model element are not manually added by users (and thus hold no corresponding modeling element, making the model an inaccurate representation of the source code), we implemented a model violation detection. For each detected violation a feedback note containing the position of the violation, as well as a message describing it, is provided to the user. Model violations are defined in terms of violation rules. A violation rule consists of a model element type, a regular expression that is used to find the violation, a group number that can be used to reference a specific group of the regular expression and a message that describes the violation.

Live Code Editor The live code editor allows multiple users to collaboratively work on the same file at the same time. As it can be seen in Fig. 4, the live code editor widget is divided into three parts. On the left side of the widget, a list of currently active users as well as a file list is displayed. The actual editor is located in the center of the code editor widget. The cursors of remote users are displayed in different colors to all users participating in this live coding phase. For highlighting protected segments in the viewport of the Ace



Fig. 4: Screenshot of the live code editor widget

editor, we use a gray background color. The depicted screenshot shows the development of a frontend component. Here, a tree containing the widget's *HTML elements* is shown on the right side of the editor. The synchronization of the file content among all users, the synchronization between the source code and its traces and the concept of (un)protected segments are integrated into the code editor. While the content of an unprotected segment can be edited, a protected segment is immutable. In order to synchronize the file content among all users, each unprotected segment is individually synchronized on the frontend, using the previously mentioned Yjs library. As protected segments are not editable, they are not synchronized. Because every unprotected segment is synchronized individually, the length of each segment is also synchronized. Thereby, the transformation of source code changes to updates of the trace model defined in Eq. 7 is implicitly performed for all users. Thus, the trace model and source code are implicitly synchronized for all users. The concept of (un)protected segments is implemented by replacing the default command handler of the Ace editor with a *CommandDecorator* component. This decorator handles code changes of the local user and is called before the command is actually executed by the Ace editor and the corresponding Yjs instance of the edited segment is updated. Based on the type of command, it is decided which actual decorator will be executed. We distinguish between navigation, deletion, insertion and other allowed operations. In order to decide to which segment a source code change belongs, a reference to the current active segment is updated in a navigation decorator, every time the local user's text cursor changes. Based on the active segment, it is decided if a source code change is allowed or forbidden, i.e. the operation is performed in an unprotected or protected segment. A deletion operation is performed if the current active segment is not protected and the dimension of the selected text that should be deleted is not out of the bounds of the active segment. Similar checks are performed for insertion operations. The last group of operations consists of commands that do not change the source code and that can be executed without side effects towards the trace model.

On the backend, we extended the CAE's REST API with means for maintaining local Git repositories. When a request for storing a file is received, its content and its file traces are stored and committed to a local repository. Before that, a check is performed to determine

if storing the file is allowed. To prevent conflicts with files that are artifacts of an earlier model synchronization process, the id of the trace model that is updated in each model synchronization process is also included in each file trace model. Thus, each file is assigned to the id of the model synchronization process in which it was created. Even if our frontend is designed to reload files after a model synchronization process, this protection mechanism additionally ensures that the synchronization between files and their models does not accidentally break. As the content and traces of updated files are (first) only stored in local repositories, the GitHub proxy service provides means to push locally committed changes to a remote repository. Possible conflicts with the remote repository are automatically resolved by using the Git *Theirs* merging strategy. This strategy ensures that in case of merging conflicts, changes of the branch that is merged are used for resolving the conflicts.

6 Evaluation

We performed a usability study with student developers to assess how our collaborative MDWE method is received in practice. We carried out eight user evaluation sessions, each consisting of two participants. After receiving a short introduction into the CAE and filling out a pre-survey to assess their experiences in Web development, the participants were seated in the same room and asked to extend an existing application, which consisted of two frontend components and two corresponding microservices. Each evaluation session took about half an hour of development time. At the end of each session, we let the participants fill out a five Likert scale questionnaire containing questions about their Web development experience and gathered their feedback regarding the cyclic development process and the live code editor.

Results and Observations. As expected, the (pre-survey) rating of the familiarity with Web technologies (4.00) and RESTful Web services (4.07) was rather high. However, only a minority of our participants were familiar with MDWE (2.67) or had used collaborative coding for creating Web applications before (2.40). Fig. 5 shows the main results regarding our development paradigm. As it can be observed, the participants rated connections between our two collaborative phases, namely the access to the code editor from the model (4.67) and the reverse process with the synchronization enabled (4.40) very high. The same also holds for the awareness introduced into the live code editor, as the participants could easily see where other developers were working (4.33). They were able to successfully collaborate on a shared part of the application by using the live code editor (4.47). These two rather high ratings show that the developed live code editor fulfills the requirements for live collaborative code editing of model-based applications. Compared to the other ratings, the general idea of a collaborative code editor for development and the need for collaboration during the code refinements phase were rated lower (both 3.47). One explanation for this is that developers are familiar with using version control systems and therefore do not see a high demand for a live collaborative code editor when working together with other developers. Even though the chosen application was, due to the time constraints of a live evaluation setting, quite

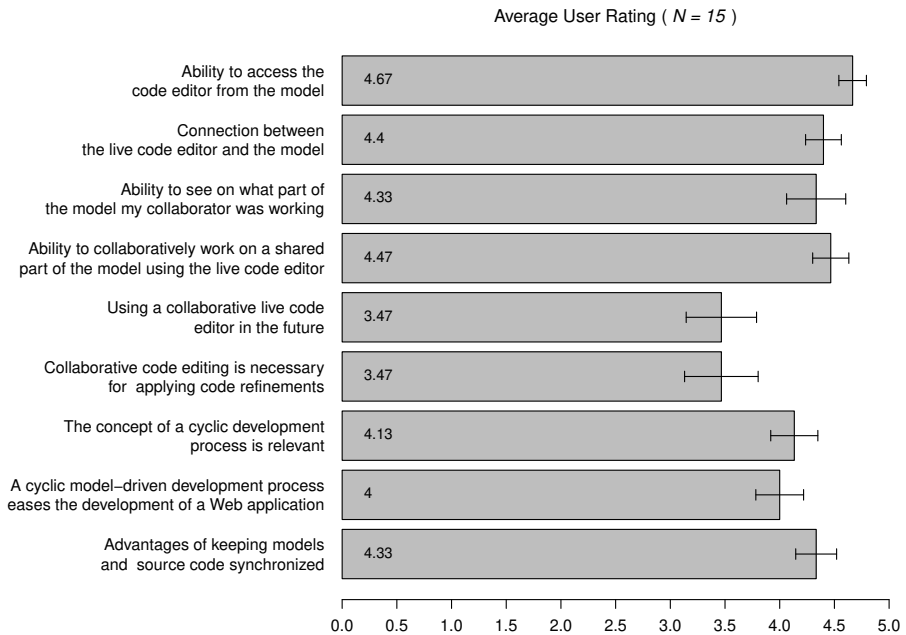


Fig. 5: Results of the user evaluation

simple, the evaluation participants mostly saw cyclic development in general as relevant (4.13) and also rated the benefits of a cyclic MDWE process high (4.00). Moreover, all participants could identify the advantages of code and model synchronization (4.33). All in all, the perception of participants towards our approach was very positive and we consider it as a successful step towards evaluating our prototype using a more complex application in a real-world development scenario.

In order to evaluate the influence of trace generation on performance and space requirements over time, we measured the code generation time during the complete evaluation and analyzed one frontend component Git repository of the final resulting application. Here, we consider that even though it is isolated, due to our template-based approach, this case gives a good approximation about the behavior of our prototype in various scenarios. Since we used one repository for performing all sessions, we were able to consider data from 490 commits. The exemplary frontend component had a final total size of 634 KB, which contained 9 KB of generated and refined code. The static JavaScript libraries, images and other files added up to 173 KB. The final trace information occupied 42 KB. This leaves 410 KB of Git history, of which the trace history occupied 338 KB. All in all, the trace information, even though it occupies considerably more space than the code output, does not negatively impact the space requirements in a usual Web development setting. Especially, since it only scales with source code changes, which usually don't make up the larger part of an Web

application in comparison to media assets and data. Moreover, the time to store, commit, push and retrieve code with trace information – considering our evaluation scenario data and participant’s subjective opinion – does not introduce observable delays in the development process.

7 Conclusions and Future Work

This paper presents a concept for synchronizing models and source code, in the context of an agile MDWE scenario on the Web. A trace model providing linking information between model elements and source code artifacts, as well as a prototype of a live collaborative code editor that supports our concept of traceability and model synchronization have been developed and integrated into an existing MDWE approach. The evaluation of our prototype showed that our method is relevant and a valuable enhancement for introducing agile development practices into MDWE.

As future work, we plan to extend our evaluation on larger scale projects, to gain deeper insights on the effects of using our agile MDWE method. Since the developed prototype is integrated into a larger framework with which we want to integrate complete professional communities into the development process, we want to further investigate the impact which the interplay between live coding, live preview of Web application changes and collaborative modeling has on the communication between end-users and developers.

References

- [ALC08] Angyal, László; Lengyel, László; Charaf, Hassan: A Synchronizing Technique for Syntactic Model-Code Round-Trip Engineering. In: Proceedings of the 15th International Conference and Workshop on the Engineering of Computer-Based Systems. IEEE Computer Society, pp. 463–472, 2008.
- [BK09] Busch, Marianne; Koch, Nora: MagicUWE – A CASE Tool Plugin for Modeling Web Applications. In: Proceedings of the 9th International Conference on Web Engineering. Springer, pp. 505–508, 2009.
- [CFB00] Ceri, Stefano; Fraternali, Piero; Bongio, Aldo: Web Modeling Language (WebML): A Modeling Language for Designing Web sites. *Computer Networks*, 33(1):137–157, 2000.
- [CH06] Czarnecki, Krzysztof; Helsen, Simon: Feature-Based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3):621–645, 2006.
- [Go12] Gotel, Orlena; Cleland-Huang, Jane; Hayes, Jane Huffman; Zisman, Andrea; Egyed, Alexander; Grünbacher, Paul; Dekhtyar, Alex; Antoniol, Giuliano; Maletic, Jonathan: The Grand Challenge of Traceability (v1.0). In: *Software and Systems Traceability*, pp. 343–409. Springer, 2012.
- [GW06] Giese, Holger; Wagner, Robert: Incremental Model Synchronization with Triple Graph Grammars. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems. Springer, pp. 543–557, 2006.

- [HLR08] Hettel, Thomas; Lawley, Michael; Raymond, Kerry: Model Synchronisation: Definitions for Round-Trip Engineering. In: Proceedings of the First International Conference on Model Transformations. Springer, pp. 31–45, 2008.
- [KKK07] Kraus, Andreas; Knapp, Alexander; Koch, Nora: Model-Driven Generation of Web Applications in UWE. In: 3rd International Workshop on Model-Driven Web Engineering. CEUR-WS, 2007.
- [La16] de Lange, Peter; Nicolaescu, Petru; Derntl, Michael; Jarke, Matthias; Klamma, Ralf: Community Application Editor: Collaborative Near Real-Time Modeling and Composition of Microservice-based Web Applications. In: Modellierung 2016 Workshopband, pp. 123–127. 2016.
- [La17] de Lange, Peter; Nicolaescu, Petru; Klamma, Ralf; Jarke, Matthias: Engineering Web Applications Using Real-Time Collaborative Modeling. In: Proceedings of the 23rd International Conference on Collaboration and Technology (CRIWG 2017). Springer, pp. 213–228, 2017.
- [Me02] Mellor, Stephen J.; Scott, Kendall; Uhl, Axel; Weise, Dirk: Model-Driven Architecture. In: Proceedings of the 8th International Conference on Object-Oriented Information Systems. Springer, pp. 290–297, 2002.
- [MR03] Martin, Robert Cecil: Agile Software Development: Principles, Patterns, and Practices. Prentice Hall PTR, New Jersey, USA, 2003.
- [Mv06] Mens, Tom; van Gorp, Pieter: A Taxonomy of Model Transformation. Electronic Notes in Theoretical Computer Science, 152:125–142, 2006.
- [Ni16] Nicolaescu, Petru; Jahns, Kevin; Derntl, Michael; Klamma, Ralf: Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types. In: Proceedings of the 19th International Conference on Supporting Group Work. ACM, pp. 39–49, 2016.
- [OO07] Olsen, Gøran K.; Oldevik, Jon: Scenarios of Traceability in Model to Text Transformations. In: Proceedings of the Third European Conference on Modelling Foundations and Applications. Springer, pp. 144–156, 2007.
- [ORK14] Ogunyomi, Babajide; Rose, Louis M.; Kolovos, Dimitrios S.: On the Use of Signatures for Source Incremental Model-to-text Transformation. In: 17th International Conference on Model Driven Engineering Languages and Systems. Springer International Publishing, pp. 84–98, 2014.
- [SR98] Schwabe, Daniel; Rossi, Gustavo: An Object Oriented Approach to Web-based Applications Design. TAPOS, 4(4):207–225, 1998.
- [Va14] Vara, Juan Manuel; Bollati, Veronica A.; Jimenez, Alvaro; Marcos, Esperanza: Dealing with Traceability in the MDD of Model Transformations. IEEE Transactions on Software Engineering, 40(6):555–583, 2014.

Modellierung plattformübergreifender Quellcode-Entsprechungen für die koordinierte Co-Evolution portierter Software-Systeme

Tilman Stehle Matthias Riebisch¹

Abstract: Wird Software auf eine neue Plattform portiert, so entsteht häufig eine zusätzliche Quellcode-Basis für die neue Plattform, die gemeinsam mit dem ursprünglichen Quellcode weiterentwickelt werden muss. Doppelte Arbeit kann dabei vermieden werden, indem die portierte Implementation die Entwurfsentscheidungen sowie Terminologie und Lösungsmuster der ursprünglichen Implementation übernimmt, sodass eine vereinheitlichte Weiterentwicklung gleichartiger Codeteile ermöglicht wird. Modelle können Entsprechungen zwischen konkreten Code-Elementen wie Klassen und Methoden beider Implementationen explizit und formal erfassen, um eine solche gemeinsame Weiterentwicklung zu vereinfachen und partiell zu automatisieren. Bisherige Ansätze zur Suche nach Entsprechungen befassen sich mit der Verknüpfung von Softwareartefakten auf unterschiedlichen Ebenen, aber erlauben keinen Vergleich zwischen gleichartigen Softwareartefakten unterschiedlicher Sprachen. In diesem Paper beschreiben wir ein Verfahren zur Erhebung sprachübergreifender Entsprechungen und zeigen, wie die resultierenden Modelle zur Koordination der gemeinsamen Evolution von ursprünglicher und portierter Implementation genutzt werden können. Zur Verwirklichung des Nutzungspotentials wurden öffentlich zugängliche Erweiterungen für Entwicklungsumgebungen implementiert. Das beschriebene Verfahren zur Erhebung der Modelle wurde anhand zweier quelloffener Portierungsprojekte evaluiert.

Keywords: Modellierung; Portierung; Quellcode-Entsprechungen; Abhängigkeiten

1 Einleitung

Häufig muss eine bestehende Software auf eine neue Plattform portiert werden, um sie einem größeren Nutzerkreis zugänglich zu machen, oder neue Anwendungen zu ermöglichen. Insbesondere im Feld der Softwareentwicklung für mobile Endgeräte ist dies eine regelmäßige Herausforderung [JMK13]. Aber auch Software-Bibliotheken werden portiert, um sie auf anderen Plattformen nutzbar zu machen. Beispiele dafür sind Apache Lucene² oder JGit³, deren ursprüngliche Implementationen für die Java Virtual Machine auf die .Net-Plattform übertragen wurden.

Es existieren Frameworks wie Apache Cordova [Th17a], Xamarin [Xa17] oder Unity [Un17], die es erlauben, Software für mehrere Plattformen auf einer gemeinsamen Codebasis

¹ Universität Hamburg, {stehle,riebisch}@informatik.uni-hamburg.de

² Website zur .Net-Implementation von Lucene: <https://lucenenet.apache.org/>

³ Github-Seite der .Net-Implementation von JGit: <https://github.com/mono/ngit>

zu entwickeln. Diese können für Neuentwicklungen sinnvoll genutzt werden. Existiert aber bereits eine reife Implementation der Software für eine Plattform, so kann diese nicht einfach an die Entwurfsvorgaben eines solchen Frameworks angepasst werden. Die Übertragung hätte eine Neuentwicklung auf Basis des Frameworks zur Folge, die die ursprüngliche Implementation durch eine unreife Implementation ersetzen würde. Zudem müssen etwaige plattformspezifische Teile der Implementation über komplexe Mechanismen wie bedingtes Kompilieren realisiert und mit der gemeinsamen Codebasis verbunden werden, was die Lesbarkeit des Quelltextes vermindert. Entwickler, die vor der Aufgabe einer Portierung stehen, entscheiden sich daher regelmäßig dafür, die bestehende Implementation beizubehalten und erneuten Entwicklungsaufwand zu betreiben, um eine zusätzliche Implementation für die Zielplattform zu erstellen [JMK13]. Aus der Portierung durch erneutes Implementieren entstehen zwei parallele Entwicklungsstränge, die mit doppeltem Aufwand gepflegt werden müssen. Es erfolgt eine gemeinsame Weiterentwicklung der beiden Implementationen, die auch als Co-Evolution bezeichnet wird. Bei Änderungen in funktional gleichen Teilen der Implementationen sollten Synergieeffekte angestrebt werden, um diese doppelte Arbeit zu verhindern und die Änderungen konsistent durchzuführen.

Problemstellung: Plattformübergreifende Abhängigkeiten in der Co-Evolution portierter Software. Ursprüngliche und portierte Implementation sollen in der Regel funktional konsistent zueinander sein und gleichartig strukturiert sein. Wird eine Software beispielsweise für mobile Geräte mit unterschiedlichen Betriebssystemen bereitgestellt, so erwarten die Nutzer, dass sie auf allen Plattformen die gleichen Arbeitsabläufe mit gleichem Ergebnis durchführen können und dass ggf. ein Austausch von Daten zwischen den plattformspezifischen Implementationen möglich ist. Die Weiterentwicklung der Software muss für beide Plattformen erfolgen. Folgen die Implementationen auf beiden Plattformen einem einheitlichen Entwurf, so können Änderungen zusammengelegt werden und doppelter Aufwand kann vermieden werden. Entwickler können den Erhalt funktionaler und struktureller Konsistenzen erleichtern, indem sie für alle Plattformen dieselben Entwurfsentscheidungen und Lösungsmuster sowie eine einheitliche Terminologie einsetzen. Um die genannten Konsistenzen aufrechtzuerhalten, muss bewusst Aufwand investiert werden. Ohne diesen gezielten Aufwand würden Inkonsistenzen entstehen. Eine Ursache dafür ist, dass ursprüngliche und portierte Implementation regelmäßig von unterschiedlichen Teams weiterentwickelt werden⁴. Bei hoher Fluktuation im Entwicklungsteam sowie bei großen Systemen ist zudem die Wahrscheinlichkeit hoch, dass nicht allen Entwicklern alle Entwurfsentscheidungen bewusst sind und deswegen bei Änderungen Inkonsistenzen eingeführt werden. In der Folge verursachen die eingeführten Unterschiede zwischen den Implementationen doppelte Arbeit bei der Konzeption weiterer Änderungen für jede Plattform und die funktionale Konsistenz nimmt ab. Es ist folglich notwendig, die Gemeinsamkeiten der Implementationen explizit zu beschreiben.

⁴ Diese Vermutung legt der Vergleich der Autoren der ursprünglichen Implementation mit den Autoren der portierten Implementation sowohl bei JGit (<https://github.com/eclipse/jgit/graphs/contributors> bzw. <https://github.com/mono/ngit/graphs/contributors>), als auch bei Lucene nahe (<https://github.com/apache/lucene-solr/graphs/contributors> bzw. <https://github.com/apache/lucenenet/graphs/contributors>)

Beitrag dieser Arbeit: Repräsentation, Erhebung und Nutzungspotentiale plattformübergreifender Abhängigkeitsmodelle. Um die Synchronisation zwischen den Implementationen zu unterstützen, ist es erforderlich, die Entsprechungen zwischen den Quellcode-Elementen beider Implementationen explizit zu modellieren. Unser Beitrag umfasst drei wesentliche Aspekte: Wir beschreiben eine Modellierungssprache zur formalen Erfassung plattformübergreifender Entsprechungen. Zweitens beschreiben wir ein Verfahren zur automatischen Erhebung dieser Entsprechungen. Drittens zeigen wir konkrete Nutzungspotentiale der Modelle für die koordinierte plattformübergreifende Co-Evolution auf und zeigen deren Realisierbarkeit durch Werkzeugimplementationen.

2 Verwandte Arbeiten

[Di11] gibt einen Überblick über formale Methoden zur Konsistenzerhaltung bei Änderungen gekoppelter Modelle. Manche dieser Techniken können auf die parallele plattformübergreifende Entwicklung übertragen werden. Deren Anwendung setzt allerdings eine formale und korrekte Beschreibung der Modellbeziehungen und Konsistenzbedingungen voraus. Dies ist insbesondere bei der Pflege manuell portierter Software nicht gegeben, sodass eine voll automatisierte Konsistenzerhaltung im Rahmen dieser Arbeit zunächst nicht angestrebt wird.

Als *Trace Link* bezeichnet man die explizite Verknüpfung zweier Softwareartefakte [CHGZ12]. In diesem Sinne sind auch die hier diskutierten expliziten Entsprechungen zwischen Code-Elementen der ursprünglichen Implementation und ihren portierten Pendants als Trace Links zu bezeichnen. Das Erstellen und Nutzen von Trace Links zwischen verschiedenartigen Softwareartefakten wie Anforderungsbeschreibungen und zugehörigen Quellcode-Dateien ist Gegenstand diverser Forschungsarbeiten [De07] [AAT10] [OI10]. Diese setzen Techniken des *Information Retrieval* ein, die die Struktur des Quellcodes außer Acht lassen, weshalb sie nicht in unsere Arbeit einfließen. [Mi03] schlägt zur Erkennung von Code-Duplikaten einen Ansatz basierend auf formaler Konzeptanalyse vor, der sowohl Strukturen als auch Bezeichner in die Ähnlichkeitsanalyse einbezieht. Dieser ist jedoch hoch komplex und nimmt laut der Autoren bereits bei kleinen Code-Basen extrem viel Rechenzeit in Anspruch. Er ist damit nicht zum Auffinden von Entsprechungen zwischen laufend weiterentwickelten Code-Basen einsetzbar. Wertvoll ist allerdings die Idee der Gewichtung von Bezeichnern nach ihrer Hierarchie im Syntaxbaum, die wir in ähnlicher Weise zum Vergleich von Quellcode-Elementen nutzen.

Das Auffinden von Quellcode-Duplikaten auf Basis textueller oder struktureller Ähnlichkeiten ist ebenfalls gut erforscht [Ud13]. Die erforschten Techniken sind jedoch hauptsächlich auf Ähnlichkeiten zwischen Code-Elementen derselben Sprache anwendbar, weshalb wir sie zur Erhebung von Entsprechungen zwischen ursprünglichen Code-Elementen und ihren portierten Pendants nicht einsetzen. Das Problem der Entwicklung plattformübergreifender Software mit separater Quellcode-Basis je Plattform ist wissenschaftlich kaum beleuchtet. Gleichwohl gibt es Arbeiten, die sich mit sprachübergreifender und sprachunabhängiger

Softwareentwicklung befassen. [MS12] führt sogenannte *Semantic Links* ein, die Entwicklern sprachübergreifende Abhängigkeiten innerhalb einer Implementation aufzeigen. Sie setzen allerdings eine gegebene explizite Verbindung zwischen den Code-Elementen voraus, weshalb sie im Portierungs-Kontext nicht genutzt werden können.

[RCPO14] schlägt eine Technik zum Vergleich des Verhaltens von Anwendungen anhand ihrer Netzwerk-Kommunikation vor. Da es sich dabei um einen Black-Box-Ansatz handelt, ist aus dem Vergleich kein Rückschluss auf die ursächlichen Unterschiede und Gleichheiten im Quellcode zu ziehen. Der hier vorgestellte Ansatz bietet im Gegensatz dazu eine Basis für den Vergleich der Implementationen auf Ebene des Quellcodes.

[SKL06] und [Ti01] zeigen modellbasierte Ansätze auf, mit denen objektorientierte Strukturen und Refactorings sprachunabhängig beschrieben werden können, sodass ihre Anwendung auf Code in verschiedenen Sprachen eine einheitliche Strukturveränderung bewirkt. Unser Mechanismus für plattformübergreifende Refactorings setzt diese Äquivalenz für Refactorings in verschiedenen Sprachen voraus, um die strukturellen Entsprechungen zwischen synchron restrukturierten Code-Elemente zu wahren.

In [SR15] haben wir einen Ansatz vorgestellt, der während der initialen Portierung systematisch Entsprechungen im Design und in der Terminologie der ursprünglichen und portierten Implementation herstellt. Diese Entsprechungen können in den hier eingeführten Entsprechungsmodellen erfasst werden.

3 Repräsentation plattformübergreifender Quellcode-Entsprechungen

Um die benötigten plattformübergreifenden Quellcode-Entsprechungen zur Koordination der Co-Evolution zweier Implementationen nutzen zu können, müssen sie explizit in Modellen dargestellt werden, die wir im Weiteren als Entsprechungsmodelle bezeichnen. Die Modellierungstheorie [Th13] definiert wichtige Eigenschaften von Modellen für die Metamodell-Definition. *Purpose*: Der Zweck eines Entsprechungsmodells ist es, Aufwand bei der Co-Evolution der verknüpften Implementationen einzusparen. Dazu soll es folgende Aktivitäten unterstützen: (1)Die Suche nach korrespondierenden Code-Elementen, (2)die Navigation zwischen korrespondierenden Code-Elementen über die Grenzen verschiedener Entwicklungsumgebungen hinweg, (3)die Übertragung automatisierbarer Änderungen wie Refactorings, sowie (4)die Koordination händischer Änderungen sich entsprechender Code-Elemente. Diesen Zwecken trägt das Metamodell dadurch Rechnung, dass es Paare sich entsprechender Quellcode-Elemente einander zuordnet. Das Metamodell ist formal, sodass die Entsprechungsmodelle zur Reduktion von Aufwand *automatisiert* von Werkzeugen verarbeitet werden können, die den Zwecken 1 bis 4 dienen. *Impact*: Teil der Purpose-Eigenschaft ist der angestrebte Einfluss der Modelle, konkret die Reduktion des Aufwands für die parallele Weiterentwicklung durch Koordination und Zusammenlegung der Änderungen sich entsprechender Code-Elemente. Dazu gehören die Unterstützung von Verständnis und Navigation, das erleichterte Auffinden von korrespondierenden Code-Elementen sowie der Erhalt von Ähnlichkeiten bezüglich des Entwurfs und der eingesetzten Terminologie bei möglichst geringem Modellierungsaufwand. Aus der Forderung nach geringem Modellierungsaufwand

leitet sich die Forderung ab, die Modelle automatisiert zu erheben. Beim automatisierten Erheben der Entsprechungen herrscht Unsicherheit über die Korrektheit der gefundenen Entsprechungen. Diese Unsicherheit muss in den Entsprechungsmodellen erfasst werden, was im Metamodell vorgesehen ist. *Restrictions*: Aus dem Zweck der Modelle ergeben sich Einschränkungen für das Metamodell. Die Entsprechungsmodelle sind beispielsweise nicht dazu gedacht, über den Entwurf einer einzelnen Implementation zu diskutieren und enthalten folglich über die Entsprechungen hinaus keine anderen Beziehungen zwischen Code-Elementen. *Pragmatism*: Es gibt eine intendierte Nutzergruppe, die die Modelle auf eine bestimmte Weise einsetzen soll. Im konkreten Fall sollen die Modelle teil-automatisiert mit Werkzeugen für die oben genannten Zwecke während der Weiterentwicklung portierter Software genutzt werden. Dem entspricht das Metamodell durch die formale Definition erlaubter Modellelemente und Beziehungen, sodass entsprechende Modelle automatisiert mit Werkzeugen für Softwareentwickler verarbeitet werden können. *Amplification*: Ein Modell kann zusätzliche Informationen enthalten, die das Original nicht enthält. Die hier eingeführten Modelle enthalten explizite Entsprechungen zwischen Quellcode-Elementen, die im Quellcode nicht explizit sind. *Truncation*: Modelle abstrahieren vom Original und lassen dabei Informationen aus, die im Sinne des Zwecks irrelevant sind. Relevant für die Beschreibung der Entsprechungen sind lediglich Ursprung und Ziel sowie ein Konfidenzwert, der angibt, mit welcher Wahrscheinlichkeit eine automatisch erhobene Entsprechung korrekt ist. *Mapping*: Modelle beziehen sich stets auf ein Original, dessen Elemente sich im Modell wiederfinden. Die im Entsprechungsmodell verknüpften Repräsentationen von Quellcode-Elementen beziehen sich auf die konkreten Quellcode-Elemente in der ursprünglichen und portierten Implementation. *Idealisation*: Modelle nehmen im Allgemeinen eine zweckdienliche Vereinfachung vor. Im Fall der Entsprechungsmodelle werden lediglich 1:1-Beziehungen der Ist-Situation abgebildet. Es wird nicht berücksichtigt, dass eine Klasse beispielsweise auch mehr als einen Zweck haben kann und eventuell nicht vollständig in ihr Pendant in der Zielimplementation übertragen worden ist.

Um Quell- und Zielelement unabhängig von ihrer Art eindeutig zu identifizieren, gibt es im Metamodell (Abbildung 1) die abstrakte Klasse Code-Element, die ein Code-Element repräsentiert und Informationen zum Auffinden des verknüpften Elements hält.

Spezifische Klassen für Code-Elemente wie Methoden oder Typen erben von ihr. Die Klasse Methode speichert neben dem Namen der repräsentierten Methode eine Referenz auf den Typ, in dem die Methode definiert ist. Typ speichert als identifizierendes Attribut den vollständigen Bezeichner des Typs und erbt selbst von Code-Element, da Typen wie Klassen oder Interfaces ebenfalls Code-Elemente sind und Entsprechungen haben können. Die

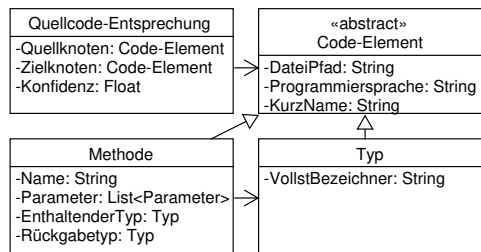


Abb. 1: Ausgewählte Klassen des Metamodells für plattformübergreifende Quellcode-Entsprechungen im UML-Klassendiagramm

Quellcode-Entsprechungen werden durch Instanzen der Klasse `Quellcode-Entsprechung` repräsentiert. Sie halten jeweils einen Verweis auf das ursprüngliche und das portierte Code-Element sowie eine Bewertung der Ähnlichkeit zwischen den verknüpften Elementen. Diese Bewertung dient als Indikator dafür, mit welcher Wahrscheinlichkeit die identifizierte Entsprechung korrekt ist und ist dementsprechend im `Konfidenz`-Attribut einer Quellcode-Entsprechung modelliert. Die Bewertung ist notwendig, da automatisch erhobene Quellcode-Entsprechungen nicht immer mit Sicherheit zutreffen. Damit Entwickler anhand eines Entsprechungs-Modells systematisch nach dem portierten Pendant zu einem gegebenen Code-Element suchen können, werden die potentiellen Entsprechungen nach dieser Bewertung sortiert.

4 Automatische Erhebung plattformübergreifender Quellcode-Entsprechungen und Erfassung in Entsprechungsmodellen

Um den Aufwand für die Erstellung von Entsprechungsmodellen gering zu halten, schlagen wir ein Verfahren zu deren automatischen Erhebung vor. Es ermittelt Entsprechungen von Code-Elementen wie Klassen oder Methoden zwischen Quellcode der ursprünglichen und der portierten Implementation unabhängig von den eingesetzten Programmiersprachen. Die gefundenen Entsprechungen werden in Modellen gemäß des in Abschnitt 3 eingeführten Metamodells erfasst. Voraussetzung dafür ist, dass die ursprüngliche Programmiersprache und die Programmiersprache der Zielplattform demselben Paradigma folgen, sodass ein gleicher Schnitt der Quellcode-Elemente nach Zuständigkeiten möglich ist. Das wäre beispielsweise bei der Portierung einer ursprünglich funktional programmierten Software in eine objektorientierte Zeilsprache nicht der Fall.

Das Verfahren kann in zwei Teilprozesse unterteilt werden: Der erste indexiert die Code-Elemente der ursprünglichen und der portierten Implementation. Der zweite Teilprozess durchsucht den erstellten Index anhand eines gegebenen Code-Elements nach terminologisch und strukturell ähnlichen Elementen.

Der erste Teilprozess ist in Abbildung 2 dargestellt. Im ersten Schritt werden die Quellcodes sowohl der ursprünglichen als auch der portierten Implementation eingelesen und abstrakte Syntaxbäume für sie erstellt. Aus den abstrakten Syntaxbäumen wird zu jedem Quellcode-Element eine Multimenge erstellt, die die Bezeichner des Elements beinhaltet. Da diese Multimengen indexiert und später im Prozess mit einer Suchanfrage verglichen werden, bezeichnen wir sie im Sinne des Information Retrieval als *Dokument* [MRS08]. Beispielsweise werden in das Dokument zu einer Klasse neben dem Klassennamen auch sämtliche Namen von Methoden, Feldern und Variablen, eingefügt, die innerhalb der Klasse verwendet werden. So wird von der Syntax der Programmiersprache abstrahiert. Dabei soll berücksichtigt werden, dass Bezeichner auf verschiedenen Ebenen des Syntaxbaumes unterschiedlich stark zur Bedeutung eines Code-Elements beitragen. Beispielsweise ist der Klassenname für die Beschreibung einer Klasse aussagekräftiger als der Name einer lokalen Variable. Diese Strukturinformation soll erhalten bleiben. Zu diesem Zweck haben

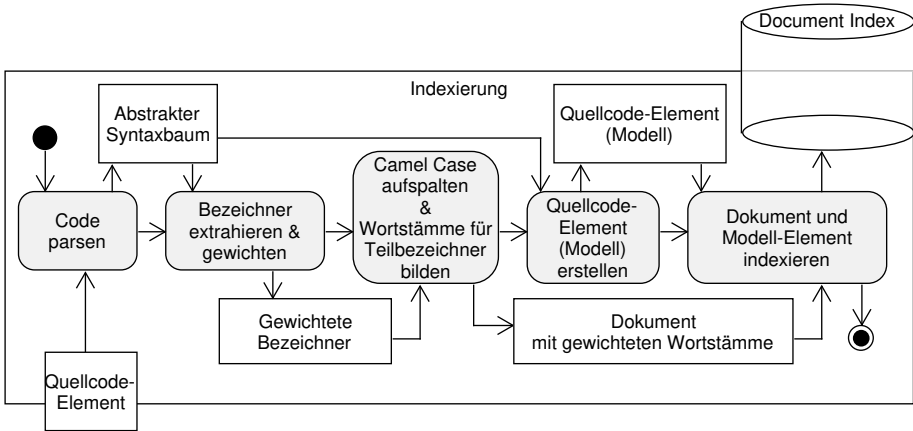


Abb. 2: Parsen und Indexieren der Quellcode-Elemente in der ursprünglichen und portierten Implementation

wir entsprechend der Grammatiken objektorientierter Sprachen wie Java, C# oder Swift eine Hierarchie abgeleitet, die festlegt, wie oft der Bezeichner eines Sprachkonstrukts in das Dokument eines Code-Elements aufgenommen wird. In das Dokument zu einer Klasse fließt beispielsweise der Bezeichner einer enthaltenen lokalen Variablen einfach ein, der Name der umschließenden Methode doppelt und der Name der Klasse vierfach. Zwischen dem Bezeichner eines enthaltenen Elements und seinem umschließenden Element liegt also der Faktor 2.

Die in das Dokument aufgenommenen gewichteten Bezeichner werden im nächsten Schritt in ihre Bestandteile zerlegt, auf ihren Wortstamm reduziert und Großbuchstaben durch Kleinbuchstaben ersetzt, sodass die spätere Suche auch Ähnlichkeiten erkennt, wenn Bezeichner sich nur in Teilen gleichen. Der Bezeichner *DateConverter* würde beispielsweise in *date* und *convert* zerlegt. Das Kompositum bleibt zusätzlich im Dokument enthalten, um zwischen Bezeichnern wie *ServiceLocation* und *LocationService* unterscheiden zu können. Das Dokument wird gemeinsam mit einer Instanz der Klasse *Quellcode-Element* (siehe Abb. 1) indexiert, die das Quellcode-Element eindeutig identifiziert.

Der zweite Teilprozess ist in Abbildung 3 dargestellt. Er sucht für ein gegebenes Code-Element nach einer Entsprechung im Index. Dazu wird zu einem gewählten Code-Element die identifizierende Instanz von *Quellcode-Element* (siehe Abb. 1) erzeugt und im Index das zugehörige Dokument Q mit den Teilbezeichnern q_i nachgeschlagen. Es wird mit jedem anderen Dokument D des Index anhand der Ähnlichkeitsfunktion Okapi BM25 [RZ09] verglichen, die als eine der erfolgreichsten Text-Retrieval Algorithmen ohne Beachtung der

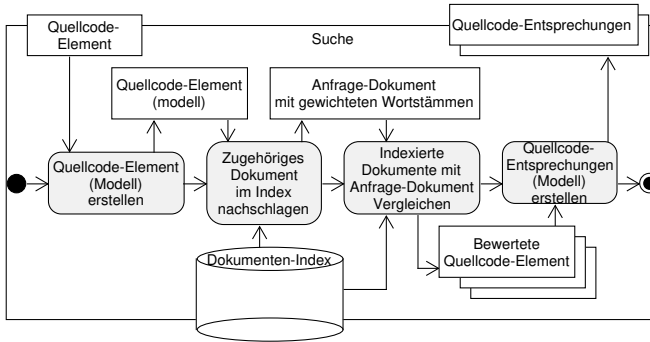


Abb. 3: Suche nach Trace Links zu einem gegebenen Quellcode-Element im erzeugten Index

Syntax gilt (ebd.). Okapi BM25 wird als numerischer Wert für die Ähnlichkeit wie folgt berechnet:

$$\sum_{i=1}^n IDF(q_i) \cdot \frac{f(q_i, D) \cdot (2.2)}{f(q_i, D) + 1.2 \cdot (0.25 + 0.75 \cdot \frac{|D|}{avgdl})}$$

avgdl ist dabei die durchschnittliche Anzahl von Bezeichnern in einem Dokument und $|D|$ ist die konkrete Anzahl der Bezeichner in D . $f(q_i, D)$ ist die Häufigkeit des Auftretens von q_i in D . Dieser Wert wird für wichtige Bezeichner wie Klassennamen durch die Gewichtung der Bezeichner bei der Indexierung erhöht. Die Funktion $IDF(q_i)$ in der Gleichung ist die umgekehrte Dokumentenhäufigkeit (**I**nverse **D**ocument **F**requency) des Bezeichners q_i . Sie berechnet einen Wert dafür, wie gut das Enthaltensein von q_i ein Dokument von anderen unterscheidet. Sie wird berechnet als:

$$IDF(q_i) = \log\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}\right)$$

wobei N die Zahl aller Dokumente im Index ist und $n(q_i)$ die Anzahl der Dokumente, die q_i enthalten. Die bei der Indexierung vorgenommene Gewichtung von Bezeichnern hat keinen Einfluss hierauf. Die rechte Seite der Formel kann für Dokumente durchschnittlicher Länge zu folgender Teilformel vereinfacht werden:

$$\frac{f(q_i, D) \cdot (2.2)}{f(q_i, D) + 2.2}$$

Sie berechnet einen Wert für die Bedeutung jedes Bezeichners q_i des Abfragedokuments für das zu vergleichende Dokument D . Die relative Dokumentlänge $\frac{|D|}{avgdl}$ im Nenner führt dazu, dass lange Dokumente niedriger bewertet werden. Dahinter verbirgt sich die Annahme, dass lange Dokumente typischerweise weniger spezifisch sind.

Okapi BM25 wird für jedes indizierte Dokument berechnet, sodass jeweils ein Wert für die Ähnlichkeit zum Anfrage-Dokument Q vorliegt. Die zum indizierten Dokument gehörige

Instanz von Code-Element wird jeweils mit dem Code-Element des Anfrage-Dokuments Q durch eine Quellcode-Entsprechung verknüpft, sofern der Ähnlichkeitswert nicht Null beträgt. Die Entsprechung erhält als Wert für das Konfidenz-Attribut den errechneten Ähnlichkeitswert. Die erzeugten Entsprechungen können durch Werkzeuge nach Konfidenz geordnet werden, sodass man systematisch nach korrekten Entsprechungen zu einem Code-Element suchen kann, indem man Entsprechungen mit hohem Konfidenzwert zuerst verfolgt.

5 Nutzung von Entsprechungsmodellen für die Co-Evolution von ursprünglicher und portierter Implementation

Die erzeugten plattformübergreifenden Entsprechungsmodelle können genutzt werden, um die koordinierte Co-Evolution mehrerer Implementationen für unterschiedliche Plattformen zu unterstützen. Konkret unterstützen wir drei Tätigkeiten: (1) Das Auffinden des implementierenden Quellcodes zu einem gegebenen Entwurfsbaustein oder einem gegebenen Verhalten, welches als *Feature Location* oder auch *Concept Location* [RW02] bezeichnet wird, (2) die Koordination von Änderungen verknüpfter Elemente und (3) die automatisierte Synchronisierung von Änderungen an äquivalenten Quellcode-Elementen.

Nutzung der Entsprechungsmodelle für Feature Location. [Si16] und [Fr14] stellen fest, dass Entwickler für das Verstehen des Codes und das Identifizieren der zu ändernden Code-Teile erheblichen Aufwand in das Navigieren entlang von Quellcode-Abhängigkeiten investieren. Sie stellen dadurch Bezüge zwischen Code-Elementen her und verstehen somit, wo im Quellcode welche Funktionalität implementiert ist.

Durch die Nutzung der eingeführten Entsprechungsmodelle kann dieses Vorgehen auf eine Implementation beschränkt werden: Ist das zu ändernde Code-Element in der ersten Implementation gefunden, kann das anzupassende korrespondierende Element in der zweiten Implementation anhand der Quellcode-Entsprechungen ohne doppelten Aufwand für Feature Location ermittelt werden.

Dazu muss die Navigation entlang der Entsprechungen in die eingesetzten Entwicklungsumgebungen integriert werden.

Koordination von Änderungen verknüpfter Elemente. Die Co-Evolution mehrerer Implementationen kann auf Basis der Entsprechungsmodelle koordiniert werden: Nach Änderung eines Code-Elements mit einer Entsprechung in der zweiten Implementation kann das Modell genutzt werden, um an dieser Entsprechung einen TODO-Kommentar zu verfassen, der die korrespondierende Angleichung fordert. TODO-Kommentare werden in der Softwareentwicklung häufig als halbformale Markierungen für ausstehende Aufgaben am Code hinterlegt. Der für das markierte Code-Element zuständige Entwickler wird so auf die Notwendigkeit hingewiesen, die beiden Implementationen durch eine entsprechende Änderung einander wieder anzugleichen.

Automatisierte Koordination von Änderungen an einander entsprechenden Quellcode-Elementen. Manche Änderungen an einander entsprechenden Quellcode-Elementen können auf Basis der Entsprechungsmodelle koordiniert und in beiden Implementationen synchronisiert durchgeführt werden. Dafür müssen zwei Voraussetzungen erfüllt sein: Erstens müssen sich die zu ändernden Quellcode-Elemente in der zu ändernden Eigenschaft (z.B. Name oder Sprachkonstrukt) gleichen; zweitens muss die Änderung in beiden Implementationen in gleicher Weise umsetzbar sein. Zum Beispiel können anstehende Refactorings gleichzeitig auf beide Implementationen angewendet werden, um strukturelle Entsprechungen zu bewahren. Das einfachste Beispiel ist die Umbenennung zweier sich entsprechender Klassen. Um die Konsistenz der Benennungen zu wahren, ist es sinnvoll, die Umbenennung gleichzeitig in beiden Implementationen durchzuführen. Dies kann auf Basis der Entsprechungsmodelle automatisiert werden.

6 Evaluierung

Um das in Abschnitt 4 beschriebene Verfahren zum Auffinden von Quellcode-Entsprechungen zu evaluieren, wurde es auf zwei Portierungsprojekte angewendet. Das erste Projekt namens Twidere ist ein quelloffener Twitter-Client, der ursprünglich für das Betriebssystem Android entwickelt wurde⁵ und aktuell manuell für das Betriebssystem iOS reimplementiert wird⁶. Bei der manuellen Übertragung aus den Programmiersprachen Java und Kotlin (Android) in die Sprache Swift (iOS) sind an vielen Stellen Ähnlichkeiten in Form gleicher Entwurfsbausteine und gleicher Bezeichner für übertragene Lösungsmuster zwischen den Implementationen entstanden. Trotzdem unterscheiden sich beide Implementationen strukturell und funktional stark. Beispielsweise wurden die Interfaces `FavoritesResources`, `FriendsFollowersResources`, `HelpResources` und 3 weitere Interfaces der Android-Implementation nicht explizit nach Swift übertragen. Die Operationen dieser Interfaces sind in der Klasse `MicroblogService` vereint, die kein Interface explizit implementiert. Die iOS-Implementation enthält zudem viele Funktionalitäten der Android-Version nicht. Wir halten Twidere damit für ein Portierungsprojekt mit typischen Schwächen bei der Konsistenz zwischen ursprünglicher und portierter Implementation, wenn kein Aufwand in die Konsistenz investiert wird.

Das zweite zur Evaluation betrachtete Projekt ist Apache Lucene.Net⁷. Dabei handelt es sich um die portierte Implementation des Such-Frameworks Apache Lucene, das teilweise mit Code-Convertoren in C# übersetzt und dann manuell angepasst wird. Dabei wird besonderes Augenmerk auf Konsistenz bei der Strukturierung des Quelltextes gelegt, obgleich .Net-typische Schnittstellen angestrebt werden [Th17b]. Durch die hohe Konsistenz und dadurch, dass die ursprüngliche und die portierte Implementation von unterschiedlichen Entwicklern bearbeitet wird, stellt es einen guten Kontrast zu Twidere dar. Mit diesen beiden Projekten kann die Leistungsfähigkeit des Verfahrens bei niedriger und hoher Konsistenz

⁵ Link zum Repository: <https://github.com/TwidereProject/Twidere-Android>

⁶ Link zum Repository: <https://github.com/TwidereProject/Twidere-iOS>

⁷ Website zur .Net-Implementation von Lucene: <https://lucenenet.apache.org/>

verglichen werden.

Wir haben 50 der 1128 Typen der Android-Implementation von Twidere ausgewählt und ihre Entsprechungen in den 132 Typen der iOS-Implementation identifiziert. Dazu haben wir systematisch die Ordnerstruktur des Projektes mittels Tiefensuche durchlaufen und Entsprechungen in der Swift-Implementation ausfindig gemacht. Konnte keine Entsprechung eindeutig identifiziert werden, wurde der Typ von der Evaluation ausgeschlossen. Daraus ergaben sich 62 Entsprechungen, die wir als Vergleichsdaten für die Evaluation nutzen⁸. Die identifizierten Links wurden von einer Person geprüft, die ansonsten nicht an dieser Arbeit beteiligt ist. Auf der Ebene von Methoden erwies es sich als schwierig, im Twidere-Projekt eindeutige Entsprechungen ausfindig zu machen, die andere Methoden als Getter und Setter mit ihren Pendants verknüpfen. Diese Methoden stellen das Verfahren angesichts ihrer Kürze und banalen Funktionalität nicht auf die Probe, sodass wir keine Evaluation auf Methodenebene in Twidere vorgenommen haben.

In gleicher Weise haben wir Entsprechungen für die Kernfunktionalität von Lucene identifiziert, die im Ordner `core` der Implementation zu finden ist. Darin sind 1871 Typen definiert. Zu den 50 ersten Typen haben wir die Pendants in der C#-Implementation identifiziert⁹. Zusätzlich haben wir zu jedem dieser 50 Typen das Pendant der ersten in ihm definierten Operation identifiziert¹⁰. Getter und Setter haben wir dabei übersprungen. Gab es in einem Typ ausschließlich Getter und Setter, so wurde der Typ übersprungen und eine Operation eines Typs gewählt, der weiter hinten in der Sortierung nach Ordnerstruktur liegt.

Die manuell gewonnenen Entsprechungen haben wir als Sollwerte genutzt, um unser Verfahren anhand der Metrik *Mean Average Precision* (MAP) [MRS08, s.159] zu bewerten. Bei MAP handelt es sich um eine Metrik zur Bestimmung der Qualität von Suchergebnissen, die im Gegensatz zu Precision und Recall die Reihenfolge der Ergebnisse berücksichtigt. Sie ist eine der gebräuchlichsten Metriken im Bereich des Information Retrieval [MRS08, s.159]. MAP berechnet den durchschnittlichen Precision-Wert bei zunehmendem Recall-Wert wie folgt:

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \left(\frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk}) \right)$$

wobei Q die Menge aller Informationsbedarfe q_j , also aller Anfragen nach Code-Entsprechungen ist. Eine perfekte Suche würde zu einem gegebenen Code-Element $e_j(OrigImpl)$ der Originalimplementation die Menge $\{e_1(ZielImpl), e_2(ZielImpl), \dots, e_{m_j}(ZielImpl)\}$ mit allen m_j korrespondierenden Code-Elementen der Zielimplementation liefern. MAP bildet für jede der m_j korrekten Entsprechungen Teillisten R_{jk} , die nur die gelieferten Ergebnisse bis zur k -ten korrekten Entsprechung enthalten. In der Teilliste R_{j1} sind also alle Suchergebnisse bis zur ersten

⁸ manche Code-Elemente in der Android-Implementation haben mehrere Entsprechungen in der iOS-Implementation. Die vollständige Liste der Entsprechungen ist unter <https://swk-www.informatik.uni-hamburg.de/~stehle/TwidereLinks.pdf> abrufbar.

⁹ Diese sind unter <https://swk-www.informatik.uni-hamburg.de/~stehle/LuceneTypeLinks.pdf> abrufbar.

¹⁰ Diese sind unter <https://swk-www.informatik.uni-hamburg.de/~stehle/LuceneMethodLinks.pdf> abrufbar.

korrekten Entsprechung enthalten; in R_{j2} alle Ergebnisse bis zur zweiten korrekten Entsprechung. Für alle Teil-Ergebnislisten wird der durchschnittliche Precision-Wert ermittelt. Dies wird für alle Suchanfragen wiederholt und abschließend der Durchschnitt über alle Suchanfragen gebildet.

Für die identifizierten Entsprechungen der Typen des Twidere-Projekts ergibt sich ein MAP von 0,75; für die identifizierten Entsprechungen der Typen von Lucene bzw. Lucene.Net ein MAP von 0,86 und für die verknüpften Methoden von Lucene ebenfalls ein MAP von 0,86. Nach unserer Kenntnis gibt es kein anderes Verfahren, das sprachübergreifende Entsprechungen von Code-Elementen identifiziert, sodass ein Vergleich nicht möglich ist. Im Vergleich mit modernen Verfahren zur Verknüpfung natürlichsprachlicher Dokumente mit dem zugehörigen Quellcode erreicht unser Verfahren sehr gute MAP-Werte. Diese erreichen einen MAP zwischen 0,7 und 0,76 [ZLL13]. Insgesamt befinden wir unser Verfahren damit für effektiv. Die Grenzen des Verfahrens werden bei der Suche nach Entsprechungen für plattformspezifisch benannte Code-Elemente sichtbar. Beispielsweise wird die Methode `close()` der Klasse `org.apache.lucene.analysis.CharFilter` nicht mit ihrem Pendant **Lucene.Net.Analysis.CharFilter.dispose()** verknüpft, da eine plattformspezifische Bezeichnung gewählt wurde und wenig Gemeinsamkeiten zwischen den verwendeten Bezeichnern innerhalb der Methoden bestehen.

Unser Verfahren zur Erhebung von Entsprechungen wurde im Rahmen zweier studentischer Arbeiten in ein Plugin für die Entwicklungsumgebung IntelliJ IDEA integriert und in [Gr17] hinsichtlich Benutzbarkeit evaluiert. Der zugehörige Quellcode steht auf GitHub zur Verfügung¹¹. Das Plugin bietet neben der Erkennung von Quellcode-Entsprechungen zwischen Java-, Kotlin- und Swift-Code auch die Navigation entlang der Entsprechungen und die Erzeugung von TODO-Kommentaren. Dazu wird ein Bezeichner, z.B. einer Methode oder einer Klasse markiert und über das Kontext-Menü eine Liste potentieller Entsprechungen aufgerufen, über die der Entwickler zur portierten Version navigieren, oder dort einen Kommentar hinterlegen kann. Entwickler können damit nach Änderungen an Java-Elementen einen entsprechenden Vermerk am zugehörigen Swift-Element erstellen, um die Angleichung der Swift-Implementation anzustoßen.

Ein zweites Plugin bietet für automatisch konvertierten Code das plattformübergreifende Umbenennen einer Methode und ihrer Entsprechungen in einer verknüpften Swift-Implementation¹². Der Entwickler findet nach Installation des Plugins den zusätzlichen Eintrag *rename method and linked elements* im Kontext-Menü für Refactorings. Er kann nach Auswahl dieser Option aus der Liste potenzieller Entsprechungen eine Swift-Methode auswählen und einen neuen Namen für beide sich entsprechenden Methoden inklusive ihrer Aufrufe automatisch vergeben lassen.

¹¹ Der Quellcode des Plugins sowie eine Anleitung zum Kompilieren und Nutzen des Plugins finden sich unter: <https://github.com/TilStehle/Java-Kotlin-Swift-Trace-Link-Recovery>

¹² Der Quellcode des Plugins sowie eine Anleitung zum Kompilieren und Nutzen des Plugins finden sich unter: <https://github.com/TilStehle/Cross-Platform-Traceability>

7 Fazit und Ausblick

In diesem Paper haben wir einen Ansatz zur Modellierung von Quellcodeentsprechungen in der plattformübergreifenden Entwicklung und ein sprachunabhängiges Verfahren zu deren Erhebung entwickelt. Es führt zur Vermeidung doppelter Arbeit bei der plattformübergreifenden Co-Evolution, weil Feature Location sowie automatisierbare Änderungen nur einmal durchgeführt werden müssen und nicht automatisierbare Änderungen koordiniert werden. Um das Verfahren zur Erhebung der Quellcode-Entsprechungen zu evaluieren, haben wir es erfolgreich auf zwei quelloffene Portierungsprojekte angewendet. Als Proof of Concept haben wir Werkzeuge entwickelt, die dieses Verfahren implementieren und die koordinierte Co-Evolution unterstützen.

Unsere zukünftigen Arbeiten werden die Nutzung der beschriebenen Entsprechungsmodelle für automatisierte Konsistenzprüfungen zwischen den plattformspezifischen Implementationen untersuchen. Darüber hinaus lohnt es sich, Bibliotheken von Entsprechungen auf der Ebene plattformtypischer Strukturen aufzubauen, um konkrete Quellcode-Entsprechungen trotz plattformspezifischer Strukturen aufzufinden. Zudem könnte die Qualität der Entsprechungsmodelle durch das Wissen und Nutzungsverhalten der Entwickler verbessert werden, was weitere Informationen im Modell erfordert.

Literaturverzeichnis

- [AAT10] Asuncion, Hazeline U.; Asuncion, Arthur U.; Taylor, Richard N.: Software Traceability with Topic Modeling. In: ICSE2010, Vol. 1. ACM, S. 95–104, 2010.
- [CHGZ12] Cleland-Huang, Jane; Gotel, Orlena; Zisman, Andrea: Software and Systems Traceability. Springer, 2012.
- [De07] De Lucia, Andrea; Fasano, Fausto; Oliveto, Rocco; Tortora, Genoveffa: Recovering Traceability Links in Software Artifact Management Systems Using Information Retrieval Methods. ACM TOSEM, 16(4), September 2007.
- [Di11] Diskin, Zinovy: Model Synchronization: Mappings, Tiles, and Categories. In: GTTSE 2009. Springer, S. 92–165, 2011.
- [Fr14] Fritz, Thomas; Shepherd, David C.; Kevic, Katja; Snipes, Will; Bräunlich, Christoph: Developers' code context models for change tasks. In: FSE2014. ACM, S. 7–18, 2014.
- [Gr17] Greiert, Gerrit: , Entwicklung eines Plugins in IntelliJ IDEA zum Auffinden von Quellcode-Entsprechungen. <https://github.com/TilStehle/Java-Kotlin-Swift-Trace-Link-Recovery/blob/master/Entwicklung%20eines%20Plugins%20in%20IntelliJIDEA%20zum%20Auffinden%20von%20Quellcode-Entsprechungen.pdf>, 2017. Studie.
- [JMK13] Joorabchi, M. E.; Mesbah, A.; Kruchten, P.: Real Challenges in Mobile App Development. In: 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement. S. 15–24, Oct 2013.
- [Mi03] Mishne, Gilad: Source Code Retrieval using Conceptual Graphs. Masterarbeit, University of Amsterdam, 2003.

- [MRS08] Manning, Christopher D.; Raghavan, Prabhakar; Schütze, Heinrich: Introduction to Information Retrieval. Cambridge University Press, 2008.
- [MS12] Mayer, P.; Schroeder, A.: Cross-Language Code Analysis and Refactoring. In: SCAM2012. IEEE, S. 94–103, Sept 2012.
- [OI10] Oliveto, R.; Gethers, M.; Poshyvanyk, D.; De Lucia, A.: On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery. In: ICPC2019. IEEE, S. 68–71, June 2010.
- [RCPO14] Roy Choudhary, Shauvik; Prasad, Mukul R.; Orso, Alessandro: Cross-platform Feature Matching for Web Applications. In: ISSTA2014. ACM, S. 82–92, 2014.
- [RW02] Rajlich, Václav; Wilde, Norman: The Role of Concepts in Program Comprehension. In: IWPC '02. IEEE, S. 271–278, 2002.
- [RZ09] Robertson, Stephen; Zaragoza, Hugo: The Probabilistic Relevance Framework: BM25 and Beyond. Found. Trends Inf. Retr., 3(4):333–389, April 2009.
- [Si16] Singh, Alka; Henley, Austin Z.; Flemming, Scott D.; Luong, Maria V.: An Empirical Evaluation of Models of Programmer Navigation. In: ICSME2016. IEEE, S. 9–19, 2016.
- [SKL06] Strein, D.; Kratz, H.; Lowe, W.: Cross-Language Program Analysis and Refactoring. In: SCAM'06. IEEE, S. 207–216, Sept 2006.
- [SR15] Stehle, Tilmann; Riebisch, Matthias: Establishing Common Architectures for Porting Mobile Applications to new Platforms. In: WSRE2015. GI-FG SRE, S. 26–27, 2015.
- [Th13] Thalheim, Bernhard: The Conception of the Model. In: BIS2013. S. 113–124, 2013.
- [Th17a] The Apache Software Foundation: , Apache Lucene Core, March 2017.
- [Th17b] The Apache Software Foundation: , Lucene.net, 2017.
- [Ti01] Tichelaar, Sander: Modeling object-oriented software for reverse engineering and refactoring. Dissertation, University of Berne, 2001.
- [Ud13] Udagawa, Yoshihisa: Source Code Retrieval Using Sequence Based Similarity. Journ. IJDKP, 3(4):57–74, July 2013.
- [Un17] Unity Technologies: , Unity - Game Engine. <https://unity3d.com/de/>, 2017.
- [Xa17] Xamarin Inc.: , Xamarin. <https://www.xamarin.com/>, 2017.
- [ZLL13] Zhou, J.; Lu, Y.; Lundqvist, K.: A Context-based Information Retrieval Technique for Recovering Use-Case-to-Source-Code Trace Links in Embedded Software Systems. In: Euromicro Conference on Software Engineering and Advanced Applications. S. 252–259, Sept 2013.

Praxisforum – Eingeladene Industriebeiträge

Praxisforum

Modellierung stellt eines der wichtigsten Hilfsmittel zur Beherrschung komplexer Systeme dar. Die Themenbereiche der Entwicklung, Nutzung, Kommunikation und Verarbeitung von Modellen sind dabei so vielfältig wie die Informatik mit all ihren Anwendungen selbst.

Die in zweijährigem Turnus durchgeführte Fachtagung MODELLIERUNG wird vom Querschnittsfachausschuss Modellierung der Gesellschaft für Informatik e.V. seit 1998 durchgeführt und hat sich als einschlägiges Forum für Grundlagen, Methoden, Techniken, Werkzeuge sowie Domänen und Anwendungen der Modellierung etabliert.

Schon immer lag ein Schwerpunkt der Tagung auf dem Austausch zwischen Praxis und Wissenschaft. Dies wird auch 2018 durch das Praxisforum weitergeführt.

Das Praxisforum bietet die Gelegenheit, über die Anwendung und Umsetzung von Konzepten, Techniken und Werkzeugen der Modellierung in der Praxis zu berichten. Dabei werden insbesondere gewonnene Erfahrungen sowie aktuelle Probleme und Lösungsansätze ausgetauscht.

Um die Qualität der Beiträge hochzuhalten, wurden für das Praxisforum dediziert Praktiker angesprochen, um umfassend über Erfolgsgeschichten und Fehlschläge im Umgang mit Modellen in der Praxis zu berichten. Dabei wurden insgesamt 6 Vorträge von Vertretern unterschiedlicher Firmen ausgewählt. Im Konferenzband enthalten sind schriftliche Ausarbeitungen zu 5 der Vorträge. Komplettiert wird das Praxisforum durch 5 weitere Vorträge von Vertretern der Sponsoren der Tagung. Entstanden ist dadurch ein ausgewogenes Programm mit Themen, die von der pragmatischen Nutzung von Modellen in agilen Softwareentwicklungsprojekten bis hin zur Nutzung von Modellen in sicherheitskritischen Anwendungen reichen.

Braunschweig, im Februar 2018

Andreas Vogelsang, Technische Universität Berlin

Daniel Méndez, Technische Universität München

Implementing Knowledge Management in Agile Projects by Pragmatic Modeling

Harald Störrle¹

Abstract: BACKGROUND: Team knowledge is diluted and destroyed through domain evolution and staff turnover. A challenge to any project, agile projects are particularly vulnerably as they rely more on tacit knowledge than plan-based approaches. Increasing project sizes and durations deteriorate this situation. Introducing documentation and modeling to turn tacit into explicit knowledge as exercised in traditional approaches is perceived as costly, and impeding with agility.

OBJECTIVE: We want to improve agile practices for large, long-running projects by adopting and adapting long-standing modeling practices, challenging these practices in the process. We aim to establish a more pragmatic view of what should be considered a model, and how complex system models could be organized to better support their usage.

METHOD: We propose several additions and changes to existing agile practices, and a new notion of model. We highlight how models are used in industry, and how existing modeling languages and tools might be improved to better support these usage modes.

RESULTS: We have successfully implemented our approach in a large project. A transfer to a smaller, more typical agile project in a different environment is under way.

CONCLUSIONS: Modeling can be a valuable and appreciated addition to agile development projects. However, this requires a pragmatic approach beyond the conventional wisdom of MDE and academic modeling practices. A broader view on what models and modeling are is useful in practice, and offers relevant new research questions.

1 Introduction

Over the past 20 years, agile approaches to software development have evolved from innovation to main-stream. The “*home ground*” [Bo02, p. 64] of agile approaches are smaller projects in dynamic environments with few external constraints. Pushing for the limits, though, practitioners have attempted to apply agile practices also to large and long-running projects in complex domains or highly regimented environments, facing substantial “*barriers*” [BT05, p. 30]. In this paper we argue that the commonality in all these barriers is the (un)availability of knowledge in a team.

¹ Dr. Harald Störrle, Principal IT Consultant, QAware GmbH, Aschauer Str. 32, 82152 München, hstorle@acm.org

Obviously, software engineering is a highly knowledge-driven activity, making software engineers prototypical knowledge workers [CM04], and underscoring the importance of knowledge sharing. Agile approaches acknowledged this from the start, aiming to “*reduce the cost of moving information between people*” [CH01, p. 131] by striving to “*replace documents with talking in person and at whiteboards*” (op cit.).

Therefore, agile approaches tend to “*focus on individual competency as a critical factor in project success*” [CH01, p. 131], relying critically on tacit knowledge (“in the heads of the team members”) [Bo02], and de-emphasizing the role of explicit knowledge as expressed in documents, models, and other artifacts.² Clearly, relying on tacit knowledge presents a critical challenge for agile approaches when faced with large teams, high staff turnover, or long running projects [NMM05, BT05, Hi03]. Consider these factors in how they affect knowledge in a development team.

- **Team Size:** With increasing team size, it is more and more unrealistic to assume that all team members are exchangeable, with the same level and profile of expertise, interests, strengths, and experience. Thus, larger teams will exhibit an increasing differentiation of roles filled within a team. Inevitably, team *structures* will emerge, be they formal or informal.
- **Staff turn over:** Clearly, when knowledge is primarily tacit (“in the heads”), removing experienced and knowledgeable members from the team also removes knowledge from the team. Explicit knowledge codified in documents, on the other hand, is much less affected by staff turn over.
- **Long-running projects:** Regardless of staff turnover, passing time alone brings about changes in the application domain, system structure, and technologies used. Therefore, knowledge decays over time: similar to technical debt accumulating with interest over time, technology and domain “inflation” depreciate the value of knowledge. Unlike staff turn over, however, these changes affect tacit and explicit knowledge in similar ways: both of them decay and need active effort to keep up to date.

In short: staff turn over amounts to knowledge loss and the inevitable differentiation of growing teams impedes with agile practices to compensate loss of knowledge. Conventional measures to turn tacit knowledge into explicit knowledge, i. e., documentation and modeling, are no solution: Simply transplanting such approaches and saddling developers with these tasks risks agility [BT05, p. 34], and does not solve the issue of knowledge decay, thus limiting their expected benefit. This leads to the research question we explore in this paper:

² Davenport and Prusak define knowledge as “*a fluid mix of framed experience, values, contextual information, and expert insight that provides a framework for evaluating and incorporating new experiences and information. [...] In organizations, it often becomes imbedded (sic.) [...] in documents or repositories [...]*” [DP98, p. 5]. See [Po66] for an explanation of the dichotomy of tacit and explicit knowledge.

RQ: How can we adapt existing methods to reduce loss or decay of knowledge in large, long-running agile projects, yet maintain agility as far as possible?

In this paper we report on a project that overcame this challenge by re-interpreting what it means to model, and how models can be used as an effective knowledge management medium. Our observations generalize to projects that start out highly agile, and evolve into a more moderate pace as they mature, addressing the needs of increasing business criticality.

2 Documentation in practice

In order to turn implicit knowledge into explicit knowledge, it must be committed to writing (or drawing) in one form or another. This inevitably results in (large) sets of documents in a multitude of more or less organized storage spaces. Typically, there would be a mixture of shared drives with a folder hierarchy of many documents in different formats, quite possibly with some “proper” models³ mixed in. Additionally, there might be wikis, paper documents, posters and drawings on walls and whiteboards, and the proverbial personal drawer. Clearly, it is difficult to find information in such a setting, and even more difficult be sure that a particular bit of information is missing. This makes adding and updating documents error-prone, increasing the burden of documentation further. As a consequence, information gaps, duplicates, and diverging variants of documents appear increasingly. In short, the cost-benefit proposition of such a form of documentation is not compelling.

Another way of documenting large scale systems is by creating comprehensive models. However, as such models get large, structuring them offers a challenge of its own [St10]. Also, whatever modeling language is used, it is not as universally (and proficiently) adopted as natural language, and improvised sketches. Then, all existing modeling languages have more or less severe shortcomings, and often are not a good match for the application domain. For instance, UML has been criticized repeatedly for the lack of perceptual support and conceptual clarity [FS07]. Finally, existing modeling tools fall short of users’ legitimate expectations regarding functionality and usability.

Among the many forms of documentation frequently found in real projects, two stick out. First, there are many informal diagrams which team members would use to illustrate a point, or supplement communication with one another, clients, or other stakeholders. In order to clearly separate them from the more regimented and restricted diagrams found in UML and similar languages, we will use the name

³ In the remainder, we use the term model to refer to such “proper” models, i. e., expressed in one of the common languages like UML, created by a specific tool, and offering diagrams along with semantic structures.

sketch for these artifacts. Sketches are very widely used [BD14], and often resemble well-known notations such as class diagrams.

Second, one often discovers documents with model-like content in a fairly rigid structure, with no attached visualizations but created mainly for human users. For instance, data models might be described as text documents with a section per entity containing a table of fields/columns and their respective properties. Likewise, one often sees spreadsheets with lists of REST-endpoints, use cases, or states and applicable triggers. So, following generic definitions of models in software development [St73, Lu03, Ma09], we maintain that for all intents and purposes, structured text or tables can act as models. In the remainder, we call such documents *modeloids*.

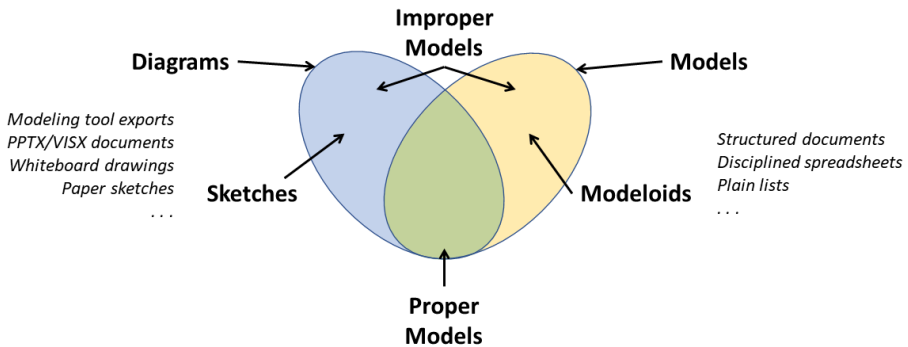


Fig. 1: Models, Diagrams, and Modeloids: differentiating the content from the presentation aspect

With sufficient effort, both sketches and modeloids could be transformed into “proper” models, at least to a substantial degree. Quite possibly, we might have to add either comments to capture the prose description of modeloids, or visual annotations and colors to mimic the visual appeal of sketches. Of course, it is questionable whether spending this effort is justified by the benefit, and whether it is realistic to expect this to happen in practice. In both cases, in our experience, the answer is no. This begs the question how we could use sketches and modeloids as elements in knowledge management of software development projects rather than “proper” models.

3 Pragmatic Agile Documentation

Our approach combines three major elements: a single unified repository, improved access paths, and content curation.

3.1 Single unified repository

Development projects typically use shared drives, version controlled repositories, wikis, ticket systems, chats, emails, various forms of dedicated collaboration software (e. g. Slack, Google Docs), along with conventional modeling tools, and analog media. The existence of multiple storage spaces is obviously an impediment to finding explicit knowledge, and keeping it consistent and up to date. So, the first goal is to integrate all existing sources of documentation into a single Project and System Knowledge Base (PSKB).

In practice, there will never be a single, unified storage for all documents. We can strive to reduce their number, though, and impose clearly defined usage roles. We suggest to use an enterprise strength wiki system as the information backbone. It offers several advantages over the classic documents-in-subversion approach.

- **comprehensive** Wikis allow all forms of documents, from simple wiki pages with embedded images and links, to attachments (including modeloids);
- **contextuality** Wikis easily allow to provide context to all documentation elements;
- **changeable** Grace to the simplicity of mark-down and the built-in straightforward version tracking on individual pages, it simplifies changing it and is forgiving in the face of errors;
- **collaborative** Wikis facilitate distributed collaboration across organizations.

In our case, we use the pre-existing Confluence wiki. Apart from already being in place and widely used, it offers rich functionality, including integration with Jira and HipChat.

3.2 Improved access paths

In order to cater for different capability levels and tastes of different users, we provide multiple alternative access methods to the information in the knowledge base. The Confluence wiki comes with a built-in conventional full-text search and sophisticated filtering. Information spaces are organized in a tree-structured page hierarchy, that allows users to navigate the tree very much the same way they would navigate a directory hierarchy.

Additionally, we also provide a visual path to knowledge through Visual Access Maps (VAM). VAMs are (large) diagrams representing important aspects or parts of the overall system or project, where diagram elements are instrumented with links to pages or attached documents that provide more detailed information to that element. For instance, the system architecture map affords links to the major neighboring systems, subsystems, interfaces, data items and use cases. It is irrelevant

whether these diagrams follow any particular well-defined syntax, whether they overlap, or what aspect they cover – organization charts or marketing can be as useful as ER diagrams or UML Assembly Diagrams⁴. Particularly, informal sketches can be used as well as “proper” diagrams. That way, existing diagrams can be reused with little additional effort.

Likewise, it does not matter what the elements link to. A link’s target may be another diagram (i. e., zooming into a refinement), a wiki page with some description, an attached slide set with a high-level description, or a lengthy table with detailed information. Or, in fact, a combination of the above. Additional elements that do not naturally fit into given model structures (e. g., a glossary) may be included by simply adding a document icon with a link.

Upgrading existing diagrams to VAMs allows user to more easily relate to the new structure. Also, reusing existing material makes for a faster bootstrap of the new documentation approach and thus supports overall project agility. Contrary to intuition, it is not cheaper, though, as most of the effort stems from stratifying the diagram (improving layout, closing gaps), and adding links to it. Obviously, the more abstract and high-level these diagrams are, the lower the change rate, and, thus, the smaller the maintenance effort. Either way, keeping such diagrams up to date is part of the ongoing curation process (see below).

Using all forms of improper models directly, we not only save the effort of transforming them into proper models, but also ensure that the stakeholders who created them can also maintain them, even if they do not speak whatever modeling language is “the right one”. Working with what we actually find, and changing as little as possible also allows us to progress much more quickly—no big up-front efforts are required. This way, it is possible to adopt a modeling approach in an agile project. By focusing on existing documents we ensure that the right pieces are documented: if people found it necessary to write the documents, they are obviously worthwhile. Also, there is no risk of impeding agility by burdening a team with unwanted documentation.

All in all, the PSKB created this way very much resembles a large, well-organized UML model with many diagrams in a proper modeling tool, except that it readily incorporates all kinds of diagrams, and all kinds of information. Syntactic and semantic restrictions embedded in UML (or any other language) no longer apply. This new-won freedom must be used with care, though, to ensure a prolonged usefulness of the overall structure.

⁴ Also known as Part-Port-Diagrams

3.3 Content curation

Content curation consists of two parts. First, building the knowledge base requires an initial, one-time effort to transform existing elements of documentation and consolidate them into an the PSKB. In our case, the pre-existing wiki-page hierarchy needed to be re-organized, cleansed, and consolidated. The pre-existing multiple large documents needed to be split up and inserted into the PSKB in suitable places.

Second, in order to maintain the quality level reached, ongoing curation is indispensable. We suggest to add the dedicated role of Knowledge Manager, supplemented by cyclic team efforts (“Documentation Day”) similar to refactoring of source code and system structure. Repeated reminders in sprint retrospectives serve to maintain a focus on the quality of documentation. We used a list of knowledge gaps for identified “undocumented important knowledge” and “known unknowns”, and a ticket system for larger issues with the existing documentation.

4 Case Study: RepairResearch

In this section we describe the RepairResearch (RR) project in which we developed our approach. RR has been launched by a major premium car manufacturer to support after sales activities, mainly maintenance, repair, and upgrade of vehicles. Each week, information on over 220,000 vehicles is delivered to approx. 5,000 parties world wide via RR.

The RR system consists of over 220 KLoC of source code (mostly Java) and employs substantial amounts of 3rd-party code. Launched in 2011, over 130 person-years have been spent creating RR (not counting client efforts). In total, over 60 different people have worked on this project. The staff size (including client-side personnel) ranged from 20 to 35 people, and is currently at 26 people (19.3 full time equivalents), with some more loosely attached.

Starting with hardly any documentation, the number of people involved and the (inevitable) staff turnover required more and more elements of explicit knowledge. Also, realizing the criticality of the application and the knowledge monopoly of particular individuals, the client demanded conventional documentation. Initially, there was a wiki with over 500 pages, more than 900 attachments (more than 700 images, 145 documents), and a total volume of almost 300MB. There were three shared drives with a combined volume of over 200GB, in almost 15,000 folders with approx. 100,000 files. The so called System Handbook alone comprised 400 printed pages, and was complemented by handbooks for architecture, maintenance, configuration management, and users. There were several overlapping glossaries,

and many versions of project plans, organization charts and similar diagrams. Many team members had (outdated) copies of subsets of these resources.

While a great deal of information was *technically* available, many team members (client side as well as supplier side) felt that actually very little information was practically available. This was particularly felt by new team members and those whose work required familiarity with multiple topics. Probing deeper, the following reasons for this perception appeared.

- **Location** Multiple storage locations contained overlapping or complementary information.
- **Access** The stores had different, and sometimes difficult to use access paths; full-text search covered only part of the storage.
- **Obsolescence** Some (parts of) information resources were outdated/obsolete.
- **Ambiguity** Information in different places was partially inconsistent.
- **Gaps** Even considering all information stores, there were many actual gaps in the documented knowledge.

Judging from past projects, we consider this size and state typical for a project of this scope and age, irrespective of whether they apply agile practices or not. However, agile practices with the emphasis on code rather than other artifacts are bound to suffer more from the impact of unregulated proliferation of documentation. This is when the RR project decided to meet the challenge by a dedicated knowledge management effort. Knowledge management has been an official role in RR since 2014, though the activities were low-key. In early 2016, a large number of experienced team members were shifted out, both client side and supplier side, leading to increased demand of the remaining experts, and requests for more explicit knowledge. By autumn 2016, knowledge management activities were ramped up resulting in the solution described in this paper. The projects adopted the approach in late 2017.

Our experience so far is promising: team members report improved quality and availability of system and project knowledge, which is felt almost immediately. However, it might be that the benefits we experience are due to the initiative as such (Hawthorne effect). Also, it appears that the investment up to this point was well-spent and yields a positive return on invest. However, it is too early to say whether our approach is truly sustainable and self-perpetuating. This can only be judged in hindsight, that is, in a few years. Finally, we have no hard evidence on the relative sizes of cleaning up obsolete and overlapping documents, filling actual gaps, improving access by visual access, and unifying storage locations. Thus, this case is but a first exploration into the opportunities.

We observed that implementing the PSKB did not in any way displace the existing system and domain experts—they were just as sought after for sharing personal insight as they were before. However, they reported ‘questions to become “more

interesting” as many simple questions can now be answered by other team members. The effort of building and maintaining the PKSB are so far smaller than expected. The initial set up effort amounted to roughly 40 person-days (8% of a project-month), and ongoing curation comes at 1-2 person days per month (2-3% of a project month). Periodic “documentation days” cost roughly a full person-day of a quarter of the team every three months (1-2% of a project month). Altogether, this amounts to 10% of a project month once, and 2-4% of the budget continuously, on top of whatever effort was spent by individuals for creating specific bits of knowledge, which did not change.

5 Related Work

Davenport and Prusak’s seminal work [DP98] defines knowledge in the sense used in this paper, and Polanyi supplements the distinction between tacit and explicit knowledge [Po66]. Schneider provides a comprehensive and accessible introduction to knowledge management in the context of software engineering [Sc09]. Scientific reviews of the field are provided in [BD08, RL02], though both are somewhat dated and shed little light on KM practices in agile projects.

Primary research about KM for SE in SMEs using agile practices goes back to [Di02], where post-mortems, pair programming, and team rotation are highlighted as effective knowledge sharing practices. As we have outlined in the introduction, these practices are limited in scope to small projects with stable teams and little role differentiation. Maurer [Ma02] examines limiting factors to scaling agile approaches, but considers only alternatives to co-location, sidestepping growing team size and project duration.

There are three popular flavors of applying agile principles in larger scale projects: Large Scale Scrum (LeSS)⁵, Disciplined Agile Delivery (DaD)⁶, and the Scaled Agile Framework (SAFe)⁷ (see [He] for a comparison of these approaches).

Chau et al. [CMM03] compared the knowledge management mechanism of agile and plan-based development projects. Dorairaj et al. [DNM12] acknowledge that knowledge management is an important part of agile development projects, and report on a qualitative study of practitioners experienced in large scale agile projects.

6 Conclusion

Agile approaches to software development live by the assumption that “*direct communication is more effective than documentation*” [Ma02, p. 14] which is a

⁵ <https://less.works/>

⁶ <http://www.disciplinedagiledelivery.com/agility-at-scale/large-agile-teams/>

⁷ <http://www.scaledagileframework.com/>

“*limiting factor for scalability*” (op cit.). As projects grow in size and lifetime, the mechanisms proposed to promote knowledge sharing in agile environments (stand ups, retrospectives, rotation, wiki) become more and more ineffective. Traditional formats of explicit knowledge (i. e., documentation) like models or prose documents, however, are not considered cost-effective. So, we propose a novel approach that amalgamates elements from both worlds into a wiki hypertext structure, integrating proper diagrams and sketches, as well as proper models and model-like documents (“modeloids”). Arbitrary elements may be attached to wiki pages. Visual access maps facilitate finding elements of knowledge. Our approach is practical, and has proven its viability in a real world, very large agile project.

We are currently applying our approach to a second project with a very different context. In particular, this new case study has a much smaller team size, more like the typical agile project. This project will teach us how to adapt our approach for smaller projects, and whether it is still economically (and practically) viable under those circumstances.

References

- [BD08] Bjørnson, Finn Olav; Dingsøy, Torgeir: Knowledge Management in Software Engineering: A Systematic Review of Studied Concepts, Findings and Research Methods Used. *Information and Software Technology*, 50(11):1055–1068, 2008.
- [BD14] Baltes, Sebastian; Diehl, Stephan: Sketches and Diagrams in Practice. In: *Proc. 22nd ACM SIGSOFT Intl. Symp. Foundations of Software Engineering (FSE)*. ACM, pp. 530–541, 2014.
- [Bo02] Boehm, Barry: Get ready for agile methods, with care. *IEEE Computer*, 35(1):64–69, 2002.
- [BT05] Boehm, Barry; Turner, Richard: Management challenges to implementing agile processes in traditional development organizations. *IEEE Software*, 22(5):30–39, 2005.
- [CH01] Cockburn, Alistair; Highsmith, Jim: *Agile Software Development: The People Factor*. Computer, pp. 131–133, 2001.
- [CM04] Chau, Thomas; Maurer, Frank: Knowledge Sharing in Agile Software Teams. *Logic versus approximation*, pp. 173–183, 2004.
- [CMM03] Chau, Thomas; Maurer, Frank; Melnik, Grigori: Knowledge sharing: Agile methods vs. tayloristic methods. In: *Proc. 12th IEEE Intl. Ws. Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE, pp. 302–307, 2003.
- [Di02] Dingsøy, Torgeir: Knowledge management in medium-sized software consulting companies. *Empirical Software Engineering*, 7(4):383–386, 2002.

- [DNM12] Dorairaj, Siva; Noble, James; Malik, Petra: Knowledge management in distributed agile software development. In: Proc. Agile Conf. (AGILE). IEEE, pp. 64–73, 2012.
- [DP98] Davenport, Thomas H.; Prusak, Laurence: Working Knowledge: How Organizations Manage What They Know. Harvard Business School Press, 1998.
- [FS07] Fish, Andrew; Störrle, Harald: Visual qualities of the Unified Modeling Language: Deficiencies and Improvements. In (Cox, Phil; Hosking, John, eds): Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE CS, pp. 41–49, 2007.
- [He] Heusser, Matt: Comparing scaling agile frameworks. CIO Magazine. originally published 2015-08-21, last accessed at 2017-12-07.
- [Hi03] Highsmith, Jim: Agile Project Management: Principles and Tools. Cutter Consortium Reports, 4(2), February 2003.
- [Lu03] Ludewig, Jochen: Models in Software Engineering – an introduction. J. Softw. Syst. Model., 2(1):5–14, 2003.
- [Ma02] Maurer, Frank: Supporting Distributed Extreme Programming. In (Wells, Don; Williams, Laurie, eds): Proc. 2nd XP Universe, Proc. 1st Agile Universe Conf. (XP/Agile Universe). Springer Verlag, pp. 13–22, 2002.
- [Ma09] Mahr, Bernd: Information Science and the logic of models. J. Softw. Syst. Model., pp. 365–383, 2009.
- [NMM05] Nerur, Sridhar; Mahapatra, Radha-Kanta; Mangalaraj, George: Challenges of Migrating to Agile Methodologies. CACM, 48(5):72–78, 2005.
- [Po66] Polanyi, Michael: The Tacit Dimension. Doubleday & Company, 1966.
- [RL02] Rus, Ioana; Lindvall, Mikael: Knowledge management in software engineering. IEEE Software, 19(3):26, 2002.
- [Sc09] Schneider, Kurt: Experience and Knowledge Management in Software Engineering. Springer Verlag, 2009.
- [St73] Stachowiak, Herbert: Allgemeine Modelltheorie. Springer Verlag, 1973.
- [St10] Störrle, Harald: Structuring very large domain models: experiences from industrial MDSD projects. In (Gorton, Ian; Cuesta, Carlos; Babar, Muhammad Ali, eds): Proc. 4th Eur. Conf. Sw. Architecture (ECSA): Companion Volume. ACM, pp. 49–54, 2010.

Systematic Refinement of CPS Requirements using SysML, Template Language and Contracts

Markus Grabowski¹, Bernhard Kaiser², Yu Bai³

Abstract: In these days, we encounter the transition from traditional closed and restricted-purpose embedded systems towards networked Cyber-Physical Systems. This applies to many industries, but in particular to the automotive industry, where assistance and automated driving functions are shaped out of complex combinations of functions and electronic control units, and even the car as a whole becomes part of a larger network of many vehicles plus infrastructure. Still, verifiable assertions must be available in the end to satisfy the safety case. The specification skills in industry often turn out to be insufficient. Even today, the mandatory V-model is hard to apply in practice and expressing appropriate requirements and refinements along with the evolution of the architecture is a hard thing to do. When development becomes agile and centered around component reuse, things become even more complex. We report about our experience with the application of contract-based development and explain keystones of our approach. We present a new template language called SSPL that allows the specification of requirements and assertions on every system architecture level and show how contract-based requirements refinement can go hand in hand with architecture refinement in SysML. We further present our Eclipse-based tool SAVONA that enables practical application of the approach.

Keywords: contract-based design; template language; system refinement; system verification; cyber physical systems

1 Introduction

Almost all technology-related industries are facing a rapid transition from formerly closed, local, restricted-purpose Embedded Systems to open, interconnected and jointly acting Cyber Physical Systems (CPS), uniting different physical domains with IT and networking technology. This affects existing industry branches, such as automotive and industrial automation, but also enables entirely new application fields, such as home automation, sensor networks, and the Internet of things. This transition can exemplarily be observed by the automotive industry which is forced into the most dramatic transformation since the invention of the car. Novel assistance functions have arisen, so that in many cases, sensors and actuators made in a variety of technologies serve for many different functions. Some of these components have been designed and released years before the functions they are later used in. Consequently, the original developers of these components had

¹ Assystem Germany GmbH, Berlin, Germany, mgrabowski@assystem.com

² Assystem Germany GmbH, Berlin, Germany, bkaiser@assystem.com

³ Assystem Germany GmbH, Berlin, Germany, ybai@assystem.com

no chances to overlook all future implications, and the architect wanting to reuse an existing component today does not know under which assumptions it had been developed. Still, systematic properties like correct behavior and safety have to be validated in the end. Highly automated driving (HAD) not only accelerates this trend, but also enforces open interconnections between vehicles and their environment (V2X), because the reach of on-board sensors is limited and therefore external information about road conditions, incidents etc. must be received and directly influences safety-relevant maneuver decisions. The single car is no longer the biggest conceivable system scope, but the whole traffic system may act as a System-of-Systems (SoS). As new suppliers and off-board services enter the automotive domain, technologies and development approaches from domains like IT are harshly contrasting the traditional automotive processes like the V-model. In the light of these tremendous changes, it can be expected that automotive industries will have to adapt their way of specifying, developing, and releasing their CPSs towards more agile and reuse/recombination-oriented approaches. It can be questioned if the traditional V-model will carry the industry into the future at all, but even in places where the V-model is officially mandatory, we have observed many flaws in its application in industrial practice. Foremost, these flaws affect the first and most important process phases, the requirements engineering and architecture specification. Requirements found in practice are often imprecise or badly put in words, although the use of template languages has been encouraged for many years [HJD04]. Specifications from car OEMs to suppliers often include many details about process aspects like manufacturing, environmental tests, standards compliance etc. but when it comes to the core function of the component (e.g. steering controller, radar sensor), there is not much of the component's behavior specified in detail. Additionally, when Safety Integrity Levels (ASILs) are assigned, it is sometimes not clear which property or service exactly is subject to the ASIL. Moreover, requirements are often stated on a wrong scope. For instance, a requirement towards a radar sensor, specifying the initialization of a braking action when detecting certain obstacles, is clearly a requirement on vehicle scope, but not on sensor scope; nevertheless, such kinds of requirements are often found in specifications towards the radar sensor supplier. On supplier level, requirements on system level are simply passed through to software component developers, without performing the architect's core duty: to decompose the requirement into sub-requirements for each component and verify that the whole shows the expected behavior. Requirement specifications are sometimes only exchanged at project setup and not further kept up-to-date so that their usefulness as a reference for verification and safety case creation can be doubted. Backward requirements from suppliers to the OEM or assumptions about the usage environment are usually not formally captured. Flaws in implementation or verification are often the consequence, in the worst case leading to actual safety risks. Contract-based development approaches have become more and more mature and popular in the last couple of years and actually have the potential to support better requirements refinement, in particular when combined with model-based development. Template-based assertion and contract specification languages brought up by recent research projects [CE10, Bo15] have contributed a lot, but still need to be extended to allow expressing all necessary kinds of requirements on system and software level.

In this paper, we present an advanced template language that goes another step in transferring contract-based development approaches into industrial practice, allowing the stepwise requirements refinement to go hand in hand with model-based system design. After collecting some requirements regarding a modern specification process that have guided us when developing our approach, we will briefly give a short introduction on contract-based design and template languages, as those are the key ingredients of our methodology. After that, we present our *System Specification Pattern Language (SSPL)* with which behavioral system properties can be described in a well readable and unambiguous way. Furthermore, we describe the method of utilizing contract-based design and SSPL in a stepwise system refinement process [Ka15]. Additionally, we shortly introduce our tool framework SAVONA, which implements the method and offers support for creating system models and template expressions and allows early system verification. At the end, we report on a first experience with our approach applied on a research case study.

2 Requirements towards an approach to address these issues

Based on our own industrial project experience, we have collected some requirements for a specification method that could help industrial companies mastering the aforementioned challenges. The approach should not be restricted on a Waterfall or V-Model, but also allow going bottom-up, i.e. assembling a new system from preexisting function blocks or practicing agile development methods. It should address the system level and multi-physics domains, thereby be based on industry-accepted standard languages such as SysML, Simulink, Modelica, AADL, and the like. This implies interfacing to standard tools and supporting co-working at different sites on black-box-level, which helps protecting intellectual property. Tools exploiting the specification technique should provide side-by-side development of architecture and requirements, thereby allowing the specification of requirements on every level of the architecture. A fixed order of requirements formulation and architecture modeling should not be imposed. All requirements or assertions shall be bound to architectural components and formulated on their scope by only referring to externally observable behavior at the component's ports without making any assumptions about the internal design and implementation. A central point is the guided and verifiable refinement of the requirements of the super-component onto the requirements of its sub-components. Regarding the specification language for requirements or properties, the key requirement is sufficient expressive power for most kinds of embedded systems and CPS, not stopping at standard phrases like 'The System shall <perform action>'. A highly extensible and adaptable language is needed which still has a precisely defined syntax in Backus-Naur-Form (BNF) or similar to enable automated parsing or bottom-up construction using a template editor. If architecture development and requirements specification go hand in hand, the architecture model including class hierarchy is a natural source for an ontology of terms to be used. To assure correct and safe operation of the highly integrated system and to assure safe reuse of components in the future, it is necessary to capture not only the requirements

towards a component, but also the assumptions towards the operational environment under which the component has been developed.

One approach that seems specifically suited to match these requirements is contract-based development (CBD), that will be at the core of the approach presented in this paper and will be explained in more details below. However, with its origins in formal software specification, existing approaches to CBD do not fulfill all of the listed requirements. We will complement CBD with approaches to specify assertions by templates that are close to natural English languages for better understandability. We will also tightly integrate it with architecture modeling, centered about SysML Internal Block Diagrams (IBD) as the modeling technique for the static system and software architecture on all levels before more specific modeling techniques like Simulink, UML, or hardware block diagrams take over.

3 Fundamentals and related work

In this section, we give a short introduction to contracts and contract-based design. Additionally, specification languages for requirements and contracts are discussed. Since these topics are quite extensive we will also refer to some fundamentals and only shortly outline related work.

3.1 Contract-based design

Initially intended as a verification method for sequential software, Bertrand Meyer [Me92] introduced Contracts using preconditions (which must hold at program entry), post-conditions (which must hold at program exit), and invariants (which must hold at every point in time). This idea has been adopted later to component-based software and system development. A system applying the '*contract-based*' design is represented by a system model containing components, ports and signals. The system itself is seen as a hierarchical composition of components, which can exchange information, energy or mass flow through their interfaces, called ports, that are connected to each other via signals. In the contract-based design paradigm, contracts are defined using assertions that allow black box specifications of components, which means while describing inputs and observable behavior from the outside, the inner (structure and) working remains unknown. Those assertions need to be distinguished between *assumptions* about conditions of their environment and the *guarantees* that can be provided given that the assumptions are fulfilled. This separation into assumptions and guarantees allows arguing about the functioning of a component composition, as for every contract can be verified whether the assumptions are met by the guarantees.

Applying contracts on a system and its components can lower the complexity of verifying the implementation against the specification. Fig. 1 shows an example of a contract specification for an airbag system. The system's contract specifies that the airbag expects a given value

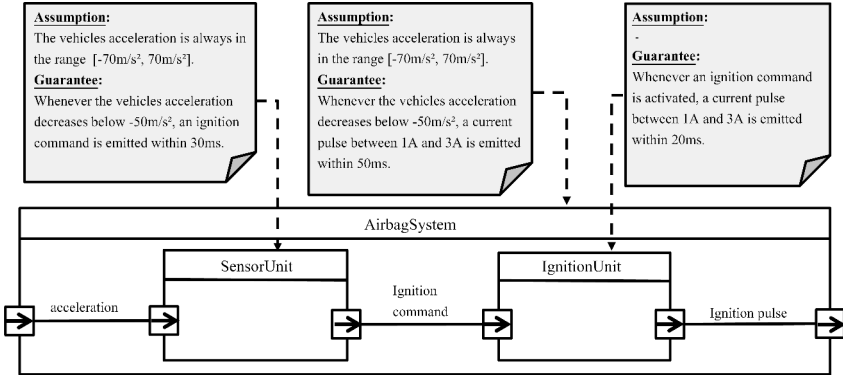


Fig. 1: Example of a contract specification

range on the input port *acceleration*. If that assumption is fulfilled the airbag guarantees that whenever the value decreases below a certain value it sends a current pulse within 50ms. The system is decomposed into two subsystems each having one contract. By assuming that the functionality on subsystem *IgnitionUnit* depends on the output of subsystem *sensorUnit* and that the subsystems would not be annotated with contracts, validating the contract of the overall *AirbagSystem* would be very complicated, as the composed behavior of both subsystems has to be computed, leading to large state spaces. By using contracts for the two subsystems, we can validate the sub-contracts locally and avoid the complex state machine composition [Ci12].

3.2 Template languages

Because formal expressions are hard to write and understand by non-experts, there is a huge suppression in using them. That turns out to be very unfortunate as they provide many striving characteristics, with which requirements engineering processes would benefit from. A well-defined syntax and semantics offer only one way to interpret statements, making things like automatic validation, tracing etc. possible. Expressions in natural language might be easier to read, but they have no constrains in syntax and semantics resulting in ambiguous statements which make automation nearly impossible without further work. In addition, they will likely always need a person with an appropriate domain knowledge to interpret and validate the expressions correctly.

Template Languages can close the gap between purely formal and unconstrained natural language. They provide a well chosen set of allowed sentence patterns, which results in a straight syntax to unify expressions making it easier to read for the recipient. Ideally, the template language also has unambiguous semantics, leaving only one way to interpret an expression. As an example, Hull et al. [HJD04] proposed to use *Requirement Boilerplates* like '*The <system> shall <function> <object> every <performance> <units>.*', where

<keyword> are placeholders to be replaced by the requirement engineer. Similar approaches have been done by [IKD09, Ma09, DSS12]. Having an expression written in a specified syntax and semantics, a machine, which has knowledge about the used grammar, can now parse and process it further allowing automatic verification. With this, template languages have the same advantages as formal languages. In addition, they feature a better readability as they are only constraining natural language instead of expecting formal expressions. The only drawback is the overhead of assigning keywords a meaning, so that a machine can interpret them correctly.

4 System Specification Pattern Language (SSPL)

In a previous work [Gr17], we analyzed existing template languages to find out why those are not currently used in practice yet. We focused on the semi-formal languages *Goal and Contract Specification Language* (GCSL)[Bo15], *Requirement Specification Language* (RSL)[CE10] and the *Property Specification Pattern* (PSP)[Au15] as these are able to describe system behavior and have influence on research and/or industry. Other than that, they featured a well-defined semantics in combination with the possibility of being translatable to a verifiable formal language. Language attributes like readability and expressiveness were evaluated by expressing real industrial system requirements with the respective languages. Others like formalization capabilities have been evaluated by the completeness of syntactic and semantic definition. We found that none of the languages features a highly readable and expressible syntax while providing unambiguous (and formal) semantics. That gave us reason to come up with an own template approach called *System Specification Pattern Language* (SSPL), which improves on the deficits of existing template languages.

4.1 Language Characteristics

To represent an applicable specification language, SSPL has been designed to feature **highly readable and well understandable expressions**. We ran a study [Gr17] on the acceptance of our template expressions from a readers' perspective and showed that SSPL performed strictly better than other template languages and is even received as less ambiguous than natural language.

SSPL enables the specification of **simple and complex system behavior** by allowing chains of basic expressions. We were able to achieve an overall translation rate of about 92%[Gr17] of all functional requirements from an automotive light system specification provided by Daimler within the project ASSUME⁴. A full description of the language as well as its syntax in Backus-Naur-Form and semantics in temporal logic can be found in [Gr17], where we also provide **supporting materials** such as a language manual for an easy application of

⁴ Affordable Safe and SecUre Mobility Evolution. <http://assume-project.eu/>

the language. Within this paper, we will shortly introduce the three general pattern types of SSPL that are used to express functional system behavior:

Global Invariant Patterns allow the definition of conditions that need to hold without any constraints. They have no restricted scope and need to be fulfilled at all points in time. An example for this type of pattern would be

supply_voltage is always less than or equal 14V.

Trigger-Reaction Patterns specify system behavior that stands in some trigger-reaction relation to each other. That is why this pattern type is the most important one when it comes to the specification of system behavior. In SSPL, the reaction must occur at some point in time *after* the trigger is fulfilled, even if the reaction time is instantiated with e.g. 0ms. If a simultaneous reaction is desired, please refer to the *Simultaneity Patterns*.

The Trigger-Reaction Patterns always feature the basic structure '**Whenever** <trigger> **then in response** <reaction> **within** <time>', where the *trigger* and *reaction* parts can be replaced by various expressions. A possible pattern instance would be

Whenever *sys_temp* **increases above** 120°C **while** *temp_warning* **is** 'OFF' **then in response** *temp_warning* **changes to** 'ON' **within** 50ms **and then** *sys_state* **changes to** 'CRITICAL' **within** 30ms.

Simultaneity Patterns describe the dependent fulfillment of two or more conditions at the same points in time. Equivalent to the logical implication, these patterns allow expressions like

While *voltage* **is less than** 3V, *start_up* **does not occur**.

Each general pattern type features a variety of possible instantiations to support a broad band of different system behaviors. Furthermore, SSPL is designed to use an existing **system architecture as ontology** to derive suitable keyword replacements such as interface names. Combining that with a **scope restriction** on the system component to be described results in an overall increased quality of specifications.

4.2 Introducing Macros

Sometimes it is unavoidable to use complex expressions within a pattern language, where a natural language expression would be much shorter or easier to read and understand. That is

why we introduce the concept of typed *macros*, which extends the expressiveness of our pattern language towards a DSL while maintaining readability. Other than the approach of the already existing template languages [Gr08, Au15], we oblige the user to define a meaning for each natural language phrase by specifying a corresponding pattern language expression. Macros are not merely text replacement, but are typed according to a class hierarchy. For instance, *event* is a built-in type of our language, and the domain engineer can derive a subclass *failure event* from it. This way we ensure that even with natural language elements, all built expressions within our pattern language have unambiguous semantics. Macros can only replace a non-terminal from the BNF, as the semantics is only guaranteed to be specified on that level. Terminals can have different meanings due to their context and can thereby not be used as a macro definition.

To demonstrate the possible advantages of macros, we first translate the following example without using them:

NL: Whenever any system critical error occurs the system must enter the safe mode within 30ms.

SSPL: Whenever any of the following events occur:

- *sys_err1* occurs
- *sys_err2* occurs
- *sys_temp* increases above 120°C

then in response *sys_mode* changes to 'SAFE' within 30ms.

We now want to simplify the pattern expression by replacing the event list with a macro. To do this, we need to look into the syntax definition of the pattern language and search for the corresponding non-terminal expression

<any_event_occurs>: **any of the following events occurs:** <event_list>

We found out that the corresponding non-terminal is <any_event_occurs>, which will be the type of our new macro we want to define next. To be able to identify a macro more easily when applied in a pattern expression we chose to underline it

'any system critical error occurs' is defined as:

any of the following events occur:

- *sys_err1* occurs
- *sys_err2* occurs
- *sys_temp* increases above 120°C

After defining the macro, we can now use it within our pattern language in every place, where the non-terminal <any_event_occurs> can be inserted. The resulting expression using the macro now looks very similar to the original natural language expression

NL: Whenever any system critical error occurs the system must enter the safe mode within 30ms.

SSPL: Whenever any system critical error occurs **then in response** *sys_mode* **changes to** 'SAFE' **within** 30ms.

To be even more similar to the original expression, we can define a macro of the type `<var_change>` for the change to safe mode. The resulting template expression looks nearly identical to the original one

'the system enters safe mode' is defined as:
sys_mode **changes to** 'SAFE'

NL: Whenever any system critical error occurs the system must enter the safe mode within 30ms.

SSPL: Whenever any system critical error occurs **then in response** the system enters safe mode **within** 30ms.

When using macros it must be considered that there exist some drawbacks in comparison to pattern expressions without them. Macros introduce a layer of abstraction to the textual representation of the expression, as actual interface names and values can be masked behind a natural language phrase. That given, the reader can not directly identify those key elements and must look for the definition of the macros. There is also the possibility that a bad macro name is chosen, which could lead the reader to a false interpretation. To be completely sure on the meaning of an expression that uses macros, studying their definitions is essential.

5 Method and Tool Integration

Only providing the raw templates without any further guidance may result in user despair. The large number of possible statements or the variety of expressions can easily confuse the user, making the work harder instead of bringing more ease to it. In addition, the user might even be unsure at which step during the development process the templates should be used. At some process steps, it might not be useful or even impossible to apply template expressions.

In the following section, we provide solutions to all of the problems above. We introduce a development process that combines the component- and contract-based design approach with our previously proposed pattern language. Furthermore, we present our prototypical Tool-Framework SAVONA, which implements the described development process and allows an easy application of our pattern language.

5.1 System Development Process Using Contract-Based Design

To exploit our template language for integrated architecture refinement, we suggest following the contract-based top-down process first presented in Kaiser et al. [Ka15]. The chosen

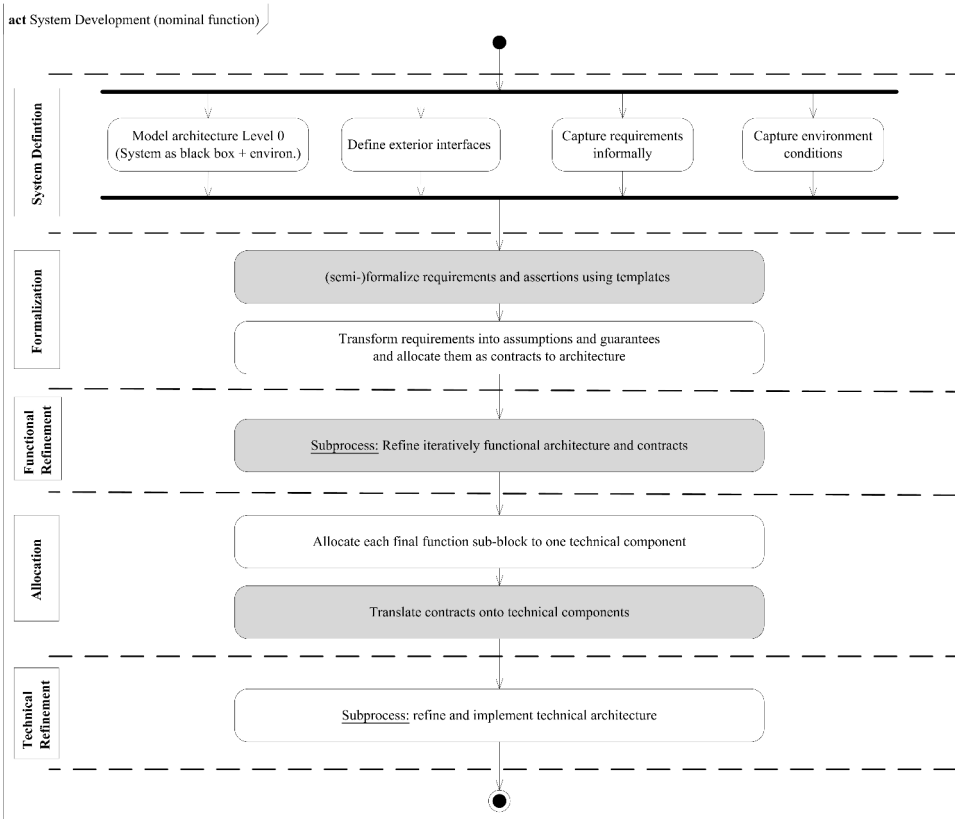


Fig. 2: Activity diagram showing the phases of the suggested contract-based development process of [Ka15]. Gray activities include the application of SSPL.

development process can be applied to the standard V-Model process and consists of following phases: *System Definition*, *Formalization*, *Functional Refinement*, *Allocation* and *Technical Refinement* (see Fig. 2). We describe each of them with regards to the interplay with our template language.

System Definition Phase At the beginning of the development process, the system engineer creates a static model of the system architecture. For that, the top-level interfaces of the system need to be defined, as they are essential for the interaction with the systems environment. The system to be developed is first modeled as a black box, because at the current development phase there is no information given on the inner workings of

the system. Aside, the top-level system requirements need to be specified. It is advised that they are captured in natural language at this development phase, as it is much faster and sufficient for the first attempt on gathering relevant information. In addition to the requirements on the system, environmental conditions do also need to be recorded. They are later used to form assumptions on inputs the system has to work with and to formulate guarantees the system has to assure in order to fulfill the environments' assumptions.

Formalization Phase Next, the requirements and assumptions are formalized using our SSPL language, which should be supported by a convenient tool, offering natural language typing with syntax highlighting and/or a template-wizard with pick lists. We use the existing *system architecture as ontology* to get information for filling out the templates, e.g. available interfaces serve as variable names to replace the corresponding placeholders and available components in the model are potential subjects to requirements. SSPL formulates assertions, which can be used in the role of an assumption or a guarantee and can be allocated to any component within the architecture (at first, the system, which is the top-level component). It is assured by the tool that only behavioral aspects that are visible on the current scope can be mentioned in the assertion, i.e. talking about internal variables or interfaces is not allowed, as well as talking about foreign components. The possibility to use macros and definitions keeps the assertions compact and readable.

Functional Refinement Phase Until now the system has only been modeled and specified at its top level. In the functional refinement phase, the system will no longer be considered as a black box, but is refined and modeled by using sub-components within the functional architecture.

This step involves design decisions by the engineer, as he or she comes up with suggestions on components, which realize the functionality of the current system level, which are also described on black-box level, addressing only the visible behavior at the ports. Each new sub-component is initially described by a natural language description and a feature list. The external interface (the ports) is defined and connected to other components via signals. In order to budget reaction times or value ranges more easily it is suggested first to assign assertion to the signals. This involves some arbitrary assignment of sub-functions to components and some arbitrary budgeting of reaction times, accuracy, ASIL etc., which is all part of the architect's design decisions and should be guided by experience what is feasible for the later technical implementation of the components. After this assignment and budgeting, new assumptions and guarantees are formed as template expressions based on the signal and super-block assertions but tailored to the scope of the component they are assigned to. This means that only interface names are available for usage in the template expressions if there exists the corresponding interface on that exact component. The formalized assumptions and guarantees are bundled as one or more contracts and assigned to the corresponding components. The contracts are then validated and verified against the architecture and the other assigned contracts. The validation includes a *compatibility check*, which verifies that only ports with compatibly types are connected, and a *consistency check*, which verifies that each constraint in a contract is satisfiable. The most important

verification step compares contracts on lower architecture level against contracts on the next higher level and checks whether the sub-components contracts allow the satisfaction the super-components contracts. This is called *refinement check* and must in most cases be performed manually today, by a review that is guided by the tool on detailed level. The more patterns of our language can be underpinned with formal semantics, the more of this type of verification can be performed formally. Detailed information about the formal refinement check of contracts can be found e.g. in [CT12]. The refinement is repeated iteratively.

Allocation Phase When the architecture has reached a level of details where actual technical components can take place of the lowest-level subcomponents, the allocation phase starts. Each black box component is replaced by one technical component with matching interfaces. The assigned contracts of the black box component become the requirements for the technical implementation and the verification obligations for the technical components. For each replacement, a final refinement check must be done to verify that the system contracts are still valid.

Technical Refinement Phase After specifying the system and modeling the static architecture, it now comes to the creation of dynamic models. Each technical component of the static architecture needs to be refined with a dynamic model (e.g. state diagrams) which represents the behavior based on its contracts' specification. The dynamic model is refined iteratively until a sufficient model depth is reached and verification is performed with existing means (e.g. testing, model checking, simulation) to show for each technical component that it fulfills its guarantees, provided that the assumptions hold.

As shown in [Ka15], the advantage of the contract-based model-integrated approach is that it works not only in a top-down manner, as suggested by the V- or Waterfall-Model, but also bottom-up. As long as pre-existing components are annotated with assumptions and guarantees, they can be stored in a library and reused later in a new context, and after re-executing the incremental verification, it becomes clear whether the resulting system is correct and safe, or the component has to be modified or replaced by another component due to detected inconsistency. As this plug-and-play approach works quite fast and contract violations are visible immediately, it supports agile development approaches to safety-critical automotive systems in an ideal way. In this case, the claim is usually not to write a complete specification, but only to fix the properties in contracts that absolutely must be guaranteed (e.g. to fulfill the safety case) and leave the rest as flexible design decisions to the development team.

5.2 Tool Support

Combining our methodological and template language approaches resulted in the prototypical tool-framework SAVONA. Based on Papyrus⁵, it features the **creation of system models**

⁵ <https://eclipse.org/papyrus/>

as Internal Block Diagram (IBD), which is provided by SysML. Due to its appearance, it fits perfectly into the component-based design paradigm and fulfills the property to model a static system architecture. In addition, Internal Block Diagrams can be used either in the system architecture, component architecture or in the hard- and software architecture by using the same model elements. As SysML is widely used as a modeling language for system engineers, the users of our tool-framework do not need to get used to any new modeling language. Additionally, various **verification mechanisms** have been implemented to ensure the validity of the modeled architecture, such as the detection on inconsistent port assignments, detection of invalid connectors, and the detection of cycles within the system architecture. We thought about different ways to **support the user at writing template**

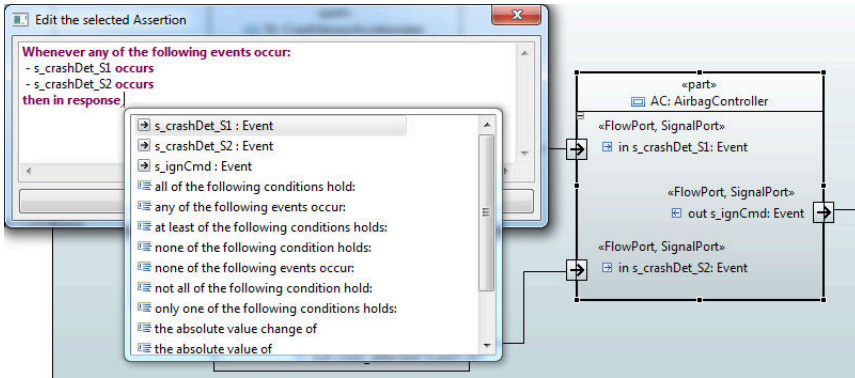


Fig. 3: *Assertion Editor*: Using the IBD's static system architecture model as ontology for SSPL pattern instantiation in SAVONA.

assertions and ended up providing two ways, which allow the user to specify assertions by using our patterns more easily. The first option the user has is to use an *Assertion Wizard*, which guides him or her through a preselected set of available pattern constructs together with examples. If the user has decided on a pattern, he or she just needs to adjust minor details such as variable names or conditional relations until the assertion is completed. The other option is to directly type assertions in a text editor called *Assertion Editor* (see Fig. 3), which features automatic syntax checks content assistance and auto-completion. A more detailed description of the SAVONA tool and its features can be found in [Gr17].

6 First experience with case study and industry projects

After the initial benchmark of our template approach by translating an automotive light system specification provided by Daimler, we are currently applying the method and tooling on two case studies. Both are provided by us within the AMASS⁶ project. One of them is a platoon of model cars with a 1:8 scale, where we want to cope with autonomous driving by

⁶ Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems, <https://www.amass-ecsel.eu/>

developing a cooperative & adaptive cruise control (CACC).

The other case study we are contributing is a standalone direct current (DC-)motor drive system, which is used in the model cars. This system is so simple that it allows an easy understanding and verification, while still exhibiting all relevant properties of a typical embedded system, in particular the combination of discrete-state logic with event typed input signals and continuous-value dynamics with continuous input and output signals. By applying our proposed model- and contract-based development approach we are facing various system properties to design and specify. To give an example, we present the following natural language requirement that describes a part of the DC-Drives control unit:

'If the measured rotational speed Spd_Act_Meas is less than 1 rpm for more than 20ms and the rotational speed target Adj_Spd_Tgt is equal 0 rpm then the voltage output V_Mot to the motor shall be reduced to a range of [0.1V,1V] within 10ms and stay within that range for at most 15ms. After that, the voltage output will be set to 0V and remain so until the rotational speed target greater than 1 rpm for a duration of at least 50ms.'

Fig. 4 shows a possible signal trace that fulfills the given requirement. An appropriate translation with SSPL results in the following:

**Whenever Spd_Act_Meas is less than 1rpm for more than 20ms
and (Adj_Spd_Tgt is 0rpm)
then in response
 V_Mot is always in the range from 0.1V to 1V for at most 15ms
starting after at most 10ms
and then (V_Mot is always 0V starting without any delay
until (Adj_Spd_Tgt is greater than 1rpm for at least 40ms)).**

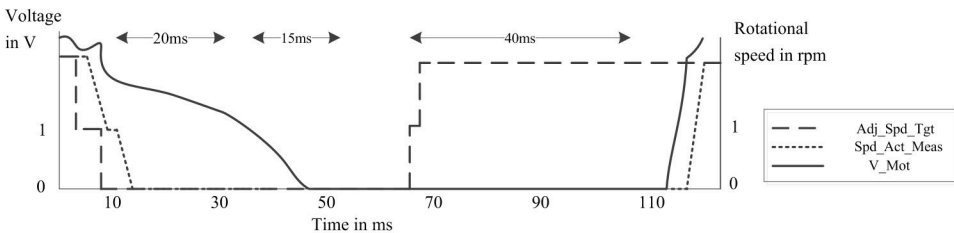


Fig. 4: Possible signal trace fulfilling the given requirement from section 6.

In the first 10ms, the target speed decreases to 0rpm and thereby stops the rotation of the motor as voltage on V_Mot is decreased. In the interval [10ms:30ms], the measured speed target speed are 0, resulting in the fulfillment of the trigger.

With a couple milliseconds delay, the motor voltage enters the given range and reaches 0V with a duration less than 15ms. After that, a new target speed is set and holds for 40ms, resulting in an increasing output voltage and measured rotation speed.

7 Conclusions and outlook

The presented approach and prototypical tool addresses a large part of the requirements listed in the introduction section. Furthermore, first experiences with applying the approach and tool onto a small DC motor drive system, as well as a more complex case study of an autonomous and networked model car, including its CACC / Platooning function have been made. Although even the latter case study is still much simpler than actual vehicle systems, we could gain a good impression about the applicability to real-world automotive systems. However, even these small case studies show that there is still a long way to go towards an industry-mature specification and modeling approach. One enhancement with high priority is to implement a formal check of the refinement for as many assertion patterns as possible, in order to unburden the developer with the manual review steps necessary today. Verification possibilities on lowest level should also be kept in mind, e.g. by enabling the automated generation of observers from the pattern expressions, which check the compliance of actual (model-in-the-loop) simulation runs or (hardware-in-the-loop) test runs. As the specification languages on system, component, hardware, and software level are quite different, it is desirable to extend the pattern language by a set of predefined macros towards a variety of domain specific languages to improve user acceptance. Regarding the extension of semantics, we are working especially on further patterns for continuous signal properties. Examples are properties like stability or bounded output signal range, or the settling time of a controller. A potential extension we are investigating is to provide patterns for frequency domain properties. Another extension of our approach would be directed towards structured data types at interfaces plus specification patterns for set, quantifier, and ordering information, in order to deal with object lists etc. For safety and reliability/availability assertions, but also in order to specify the nominal properties of environment-perceiving sensors, probabilistic patterns should be provided. Potential verification approaches for these assertions could be probabilistic model checking, but, more promisingly, Monte-Carlo simulation of the underlying behavioral models. To bridge the different levels of abstraction, a tool will have to offer different views with the option to hide details or aspects that are not of interest on a higher level of abstraction. This could on long term be complemented by a sort of 'approximate refinement' that relaxes the verification step from one level of abstraction toward the next lower level by allowing that the refined system not fully complies with the specification on higher level, but only 'well enough'.

Regarding the transition from ES to CPS and SoS, the technique should ideally be extensible towards runtime certification mechanisms, i.e. it should be possible to specify different sets of assumptions and guarantees as meta-information at runtime, so that partial systems willing to cooperate can check in which constellation assumptions and guarantees match, so that a template safety case prepared at development time is fulfilled at runtime, meaning that safe operation is assured.

References

- [Au15] Autili, Marco; Grunske, Lars; Lumpe, Markus; Tang, Antony: Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar. *IEEE Transactions on Software Engineering*, 41(7):1-1, 2015.
- [Bo15] Boyer, Benoît; Quilbeuf, Jean; Etzien, Christoph; Marazza, Marco; Senni, Valerio; Stramandinoli, Francesca; Peikenkamp, Thomas: GCSL syntax, semantics and meta-model. Technical report, DANSE Research Project, 2015. DANSE Deliverable 6.3.3.
- [CE10] CESAR: Definition and exemplification of RSL and RMM. Deliverable D_SP2_R2.1_M1, Costefficient methods and processes for safety relevant embedded systems. Technical report, CESAR, April 2010. Zugriff am 29.08.2016.
- [Ci12] Cimatti, Alessandro; Roveri, Marco; Susi, Angelo; Tonetta, Stefano: Validation of requirements for hybrid systems: A formal approach. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 21(4):22, 2012.
- [CT12] Cimatti, A.; Tonetta, S.: A Property-Based Proof System for Contract-Based Design. In: *Software Engineering and Advanced Applications (SEAA)*, 2012 38th EUROMICRO Conference on. pp. 21–28, Sept 2012.
- [DSS12] Daramola, Olawande; Sindre, Guttorm; Stalhane, Tor: Pattern-based security requirements specification using ontologies and boilerplates. In: *Requirements Patterns (RePa)*, 2012 IEEE Second International Workshop on. IEEE, pp. 54–59, 2012.
- [Gr08] Grunske, Lars: Specification patterns for probabilistic quality properties. In: *Software Engineering*, 2008. ICSE '08. ACM/IEEE 30th International Conference on. pp. 31–40, May 2008.
- [Gr17] Grabowski, Markus: Why Templates on System Behavior Are Not Used in Practice Yet: A Proposal for Enhancements, Application and Formalization. Master's thesis, Technische Universität Berlin, 2017.
- [HJD04] Hull, Elizabeth; Jackson, Ken; Dick, Jeremy: *Requirements Engineering*. Springer, 2004.
- [IKD09] Ibrahim, Noraini; Kadir, Wan MN Wan; Deris, Safaai: Propagating requirement change into software high level designs towards resilient software evolution. In: *Software Engineering Conference*, 2009. APSEC'09. Asia-Pacific. IEEE, pp. 347–354, 2009.
- [Ka15] Kaiser, Bernhard; Weber, Raphael; Oertel, Markus; Böde, Eckard; Monajemi Nejad, Behrang; Zander, Justyna: Contract-Based Design of Embedded Systems Integrating Nominal Behavior and Safety. *Complex Systems Informatics and Modeling Quarterly (CSIMQ)*, 2015 (4):66–91, Oct 2015.
- [Ma09] Mavin, Alistair; Wilkinson, Philip; Harwood, Adrian; Novak, Mark: Easy approach to requirements syntax (EARS). In: *Requirements Engineering Conference*, 2009. RE'09. 17th IEEE International. IEEE, pp. 317–322, 2009.
- [Me92] Meyer, Bertrand: Applying 'design by contract'. *Computer*, 25(10):40–51, 1992.

Modeling and Safety-Certification of Model-based Development Processes

Oscar Slotosch¹ and Mohammad Abu-Alqumsan²

Abstract: In this paper, we describe a two-step approach to show evidence for compliance with safety standards within certification efforts for model-based development projects that share some commonalities (i.e. using the same metamodel). The approach is based on modeling model-based development processes in combination with the requirements imposed on them by safety standards. Besides the typical benefits of model-based approaches (modularity, rigor, formalization and simulation), we use the combined hierarchic processes-requirements model in order to automatically generate formalized descriptions of processes, standard compliance report and verification check-lists. The process description can be used to introduce new team members to the deployed development processes. As a concrete example of the proposed approach, we present representative parts of the Validas model-based tool qualification process that has been fully modeled and certified based on the automatically generated documents by TÜV SÜD.

Keywords: Model-based Development, Process Model, Safety Standards, Tool Qualification

1 Introduction

Thanks to the easiness in which domain-specific models can be developed for dedicated purposes, model-based development is increasingly used in more areas of applications. Yet, there are remaining areas of applications where models do not enjoy the expected/desired level of acceptance, even when promising examples and pilot cases exist. This may suggest that many developers resist switching to model-based development, possibly because they do not want or are not able to do so. It is common in practice to encounter the following arguments against introducing model-based development: “there is no detailed process description of the model-based approach”, “it is not clear how to do this with my tool X”, “the compliance of the model-based development process to safety standards is unclear”, or “the modeling tools cannot be used in safety critical projects, since they are not qualified (or certified)”. Obviously, these statements/arguments are not against the model-based approach per se, but rather are manifestations of why developers may resist introducing/switching to model-based development.

Furthermore, quality aspects of software like consistency, reproducibility and repeatability and roll-out planning cannot be ensured without the availability of a precise process description. Consider for instance a situation where a company’s management

¹ Validas AG, Arnulfstr. 27, 80335 München, slotosch@validas.de

² Validas AG, Arnulfstr. 27, 80335 München, abu-alqumsan@validas.de

has decided to use UML (and a specific tool) for the specification of software algorithms in a pilot project. Assume as well that the team is willing to do so, but some team members use state transition diagrams, others sequence diagrams and some others use activity diagrams. Furthermore, some even have developed a very sophisticated combination of component diagrams, class diagrams and C++ code that requires new stereotypes and small changes in the tool. Even if the pilot project may become a success, it is obviously impossible to roll it out to the whole company without making a more detailed description of the modeling process available. A qualification of the used tool might additionally be required according to relevant safety standards.

In the present work, we aim at bridging the gap of lacking such descriptions of modeling processes with a novel approach. The same approach is additionally used for the purposes of facilitating certification efforts, by straightforwardly linking relevant process activities to the corresponding requirements imposed by relevant safety standards.

Hereby, the crucial point to meet these two goals (i.e. providing formal description and examining compliance) is to decide upon and to use the right abstraction level when describing processes and tool usage. While it suffices, from a safety point of view, to satisfy a requirement by tracing it to the respective activity/-ies and its/their produced documentation/s, this is yet insufficient, and often not even necessary, for effective and efficient introduction and usage of models/tools. For the latter case, descriptions have to be provided in more details, but not to the point where concrete objects/examples are being described. Otherwise, repeatability and reproducibility in similar projects would be harmed. We recognize that the metamodel is a well suited abstraction level for both the reasoning on compliance to safety standards and the description of model-based development processes. Since processes consist of actions and input/output artifacts we decided to model the metamodel itself as an artifact that is processed (i.e. created, updated/extended) from within process activities. This renders the proposed approach most suitable when the metamodel is used in several concrete development projects of similar nature.

Without harming the generality of the proposed approach, the present paper focuses on the application of the proposed method for model-based tool qualification processes deployed at Validas AG. In particular, we show how the approach is used to show the compliance of our processes to relevant safety standards.

The remainder of this paper is structured as follows. In Section 2 we introduce our general approach for modeling model-based development. Section 3 roughly introduces model-based tool qualification and the metamodel we use to this end. In Section 4, we introduce how the proposed approach is actually done for modeling the MetaModel in tool qualification projects. Section 5 provides more details on the whole structuring of processes and requirements relevant to tool qualification projects. Section 6 provides some details regarding a practical example from the industry: The certification of Validas tool qualification processes by TÜV SÜD. Section 7 concludes with a summary.

2 Modeling Model-Based Development Processes for Compliance and Formal Description

In order to achieve compliance and to provide formal description of model-based development processes, we propose to jointly formalize safety standard requirements and model-based development processes. To this end, we use the process modeling and requirements management framework AutoFOCUS³. This open-source software tool allows modeling of requirements and processes in a hierarchic manner. It additionally supports backward and forward tractability to requirements. Further, the tool provides a strongly-typed environment with support for enumeration and structure data types and additionally provides the possibility to define functions based on user-defined data types.

The tool also supports simulation (module test) of modeled processes with the help of the so called “DTD Evaluator”, which is a functional interpreter able to process user defined functions on user defined data types.

Altogether, these features render the tool ideal for our purposes (some benefits of these features will become clearer in later sections). Standard compliance examination of the model-based development process is developed and achieved with a set of utilities and extensions that allow for automatic generation of documentation and work products. We refer to these extensions as TOPWATER extensions as the name of the project in which they were developed.

In particular, the following documents can be automatically generated through the TOPWATER extensions:

1. *The Formal Process Description*: is integrated eventually as an appendix within a manually written process description
2. *The Compliance Report*: showing the satisfaction of the model to all relevant requirements and their traces.
3. *The verification and validation (V&V) Plan*: V&V have to be performed according to this plan in each project that claims compliance to the standards.

Obviously, evidence for compliance of a concrete project to safety standards is achieved in two steps. Firstly, the general *Compliance Report* shows the general compliance of the methodology to the safety standard. Secondly, the *V&V Report* (produced by following the *V&V Plan*) shows the compliance of the concretization of the general methodology within a concrete project. As such, the *compliance Report* and the *V&V Plan* are the same for all projects that share the same model and modeling process and the *V&V Report* is unique for each project.

The overall safety compliance approach is depicted in Fig. 1.

³ AutoFOCUS3 is an open-source software and can be freely downloaded from <https://af3.fortiss.org/download>

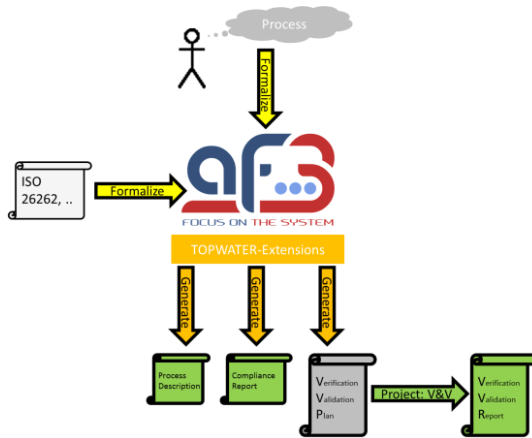


Fig. 1: Validas TOPWATER Compliance Method

3 Model-based Tool Qualification

Tool qualification is imposed by safety standards to ensure that tools can be used with confidence when developing safety-critical items/elements, see [In11], [In10], [RT11]. The main approach adopted by Validas AG for tool qualification is to test the tool in the same exact environment of the tool user (i.e. qualification by validation). This can be achieved using the so-called tool qualification kits (or QKits in short), which additionally can generate the work products (e.g. tool qualification report, tool safety manual) required by safety standards.

The idea of model-based tool qualification is to build a model for the tool qualification (i.e. containing features, known bugs, tests, etc.), that allows performing the available tests and analyzing their results to generate the required documentation and work products.

The tool chain analyzer [Va17] is a tool that supports the modeling of toolchains and qualification kits. It supports a tool qualification model with all required information for classification and qualification of tools, see [Wi12], [SI12].

The metamodel of the TCA (see user manual) consists of the following elements (for the sake of clarity in the present paper, we use a simplified representation and omit the containment hierarchy):

- TOOLCHAIN: root element containing all other elements

- TOOL: represents a tool in the model
- VERSION: represents a version of the tool
- FEATURE: represents a feature of the tool
- ERROR: represents a potential error of a tool feature
- KNOWNBUG: represents a known mal function of the tool
- CHECK: user action to detect potential errors or real bugs
- RESTRICTION: user action to avoid potential errors or real bugs

Every element in the metamodel has attributes like NAME, DESCRIPTION; some also have other attributes (e.g. COMMENT, IMPACT, etc.) that have to be specified in concrete tool classification and qualification projects. Fig. 2 shows a typical representation of the metamodel.

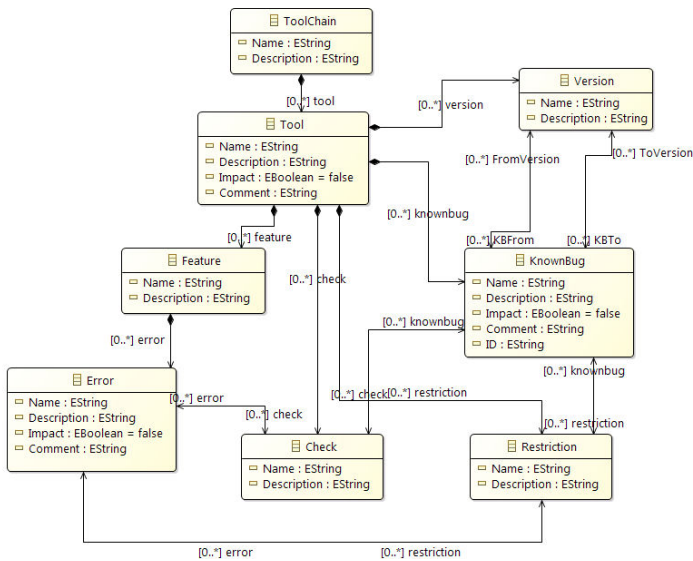


Fig. 2: Example Metamodel of Toolchains/Tools

The tool qualification requirements imposed by the relevant safety standards [In11], [In10] and [RT11] are similar in their nature. They mainly require a three phase approach:

- Classification of the tools
- Qualification of critical tools

- Safe usage of the tools according to safety manuals, which are based on results from the previous two phases (i.e. classification and qualification of tools)

4 Modeling the MetaModel used in Model-based QKit Development Processes

At the core of our approach is the modeling of model-based development processes and at the core of such modeling is the specification of the **MetaModel** being used.

The basis for specifying the **MetaModel** in AutoFOCUS3 is a simple enumeration data type called “**ModelSpecification**” with the enumerator names {**NotSet**, **Specified**, **NotRequired**}, corresponding to the three possible states every model element can have. The states of all model elements are initially set to **NotSet**, and once they are specified, their states change to **Specified**. Sometimes it might be undesirable or even not possible to specify an actual value for a model element. In these cases, the respective model elements are set to be in the **NotRequired** state. To clarify the last point, consider for instance the link `TOOL_VERSION_TO` of a `KNOWN_BUG` element, which defines the version in which the known bug was fixed. This attribute can be therefore **Specified** (in case the known bug is already fixed in a specific version) or **NotRequired** (in case the known-bug remains an open bug). The attribute `TOOL_VERSION_FROM`, which defines the version in which the bug is introduced, has to be specified in either case.

In general, for every class with name `<CName>` with attributes `<A1>`, to `<An>` and links `<L1>` to `<Lm>` in the metamodel, there are three structure types modeled within AutoFOCUS3:

- `<CName>Class: {Attributes: <CName>Attributes, Links: <CName>Links}`
- `<CName>Attributes: {<A1>: ModelSpecification, ...<An>: ModelSpecification}`
- `<CName>Links: {<L1>: ModelSpecification, .. <Lm>: ModelSpecification}`

The type **MetaModel** is a structure data type consisting of components for each class:

- `MetaModel: {<C1>: <CName>Class, ...}`

The types, which are required for modeling the **MetaModel** from Fig. 2 are modeled in AutoFOCUS3 as shown in Fig. 3.

Name	Type
[-] Data Dictionary	
[-] ModelSpecification	
NotRequired	
NotSet	
Specified	
[-] CheckAttributes	
Description	ModelSpecification
Name	ModelSpecification
[-] CheckClass	
Attributes	CheckAttributes
Links	CheckLinks
[-] CheckLinks	
To_Errors	ModelSpecification
To_KnownBugs	ModelSpecification
To_Tool	ModelSpecification
[-] ErrorAttributes	
[-] ErrorClass	
[-] ErrorLinks	
[-] FeatureAttributes	
[-] FeatureClass	
[-] FeatureLinks	
[-] KnownBugAttributes	
[-] KnownBugClass	
[-] KnownBugLinks	
[-] MetaModel	
Check	CheckClass
Error	ErrorClass
Feature	FeatureClass
KnownBug	KnownBugClass
Restriction	RestrictionClass
Tool	ToolClass
Data Dictionary Evaluator	

Fig. 3: Modeling the MetaModel Data Type in AutoFOCUS3

Having modelled the **MetaModel** in AutoFOCUS3, actions that define how it is being created and updated during the modeling processes are defined straightforwardly using function definition.

For example, the modeling of known bugs of a tool requires, as its input, a defined TOOL model with contained FEATURE model. The output would be an updated model, where KNOWN-BUG elements are added together with links to corresponding TOOLS and affected FEATURES. Adding mitigations (CHECKS and RESTRICTIONS) and linking them to the KNOWN-BUGs further extends the model. Such procedures can be described using the attributes of the model elements that need to be described in each step: “TOOL.NAME” and “TOOL.VERSION” or “KNOWN-BUG.ID”. The predicate that checks whether known bugs can be modelled or not can be formulated in AutoFOCUS3 as follows:

```

readyForKBModeling (MetaModel:M) =
    M.Tool.Attributes.Name==Specified() &&
    M.Tool.Attributes.Description==Specified() &&
    M.Tool.Links.To_Features==Specified() &&
    M.Feature.Attributes.Name==Specified() &&
    M.Feature.Attributes.Description==Specified() &&
    M.Feature.Links.To_Tools==Specified() &&
    
```

```

M.Version.Attributes.Name==Specified() &&
M.Version.Attributes.Description==Specified() &&
M.Version.Links.To_Tools==Specified()

```

And the update of the model by specifying known bugs can be done within AutoFOCUS3 in a functional programming style by defining the following function

```

updateKBModeling (MetaModel:M) =
return combineModels (M, addKnownBugClass ( {
  Attributes: combineKnownBugAttributes (
    specifyKnownBugName (),
    specifyKnownBugDescription (),
    specifyKnownBugID (),
  Links: combineKnownBugLinks (
    specifyKnownBugLinkToTool (),
    specifyKnownBugLinkToVersion ())));

```

The helper function can be implemented easily using the definition

```

specifyKnownBugName:KnownBugAttributes = {
  Name:Specified,
  Description:UnSet,
  ID:UnSet}

```

Furthermore, in order to facilitate the generation of the *V&V Plan*, we mark the V&V actions using the keyword *Criterion*.

5 Modeling of Development Processes and Requirements

The process description starts from a high abstraction level describing the process with the customer interaction, see Fig. 4. When constructing a model-based QKit, the core activity is to build the corresponding model. This process is further detailed as depicted in Fig. 5, which also shows the main phases of building the QKit: Structure modeling, analysis modeling and test modeling.

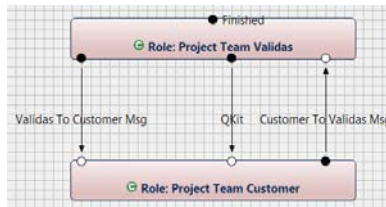


Fig. 4: Validas Interaction Process (High-Level) with Tool Providers

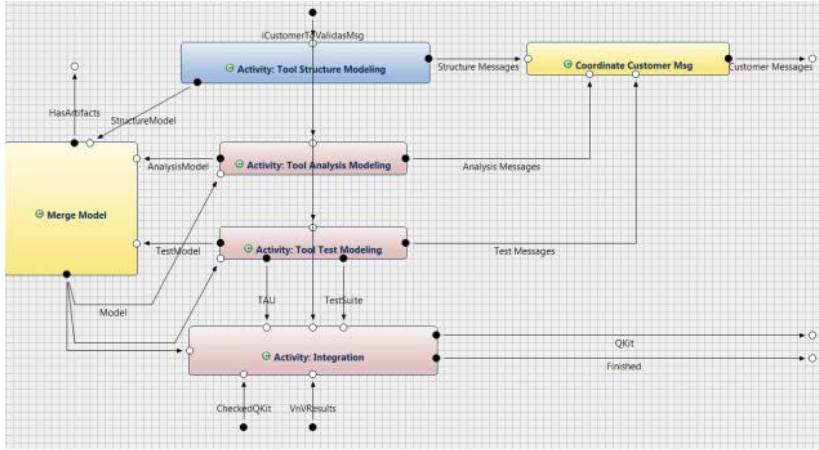


Fig. 5: Model Construction Main Process

The interface of the known bug modeling process (carried out in the analysis modeling phase) is described in Fig. 6. This example emphasizes the adequateness of the strongly-typed environment of AutoFOCUS3 to our modeling purposes. The type of the input model is **MetaModel** as detailed in Fig. 3. The name of the input model is “ToolFeatureModel”, the name of the output model is “KBModel”.

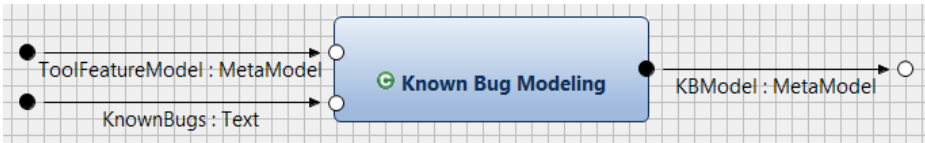


Fig. 6: Known-Bug modeling Process Interface

As has been already shown with the previous figures, AutoFOCUS3 supports a hierarchical description of processes (and requirements as will be seen later). The process of specifying known bugs can then be modelled with more details through a state transition diagram as depicted in Fig. 7 and Fig. 8 for the first part of the modeling process.

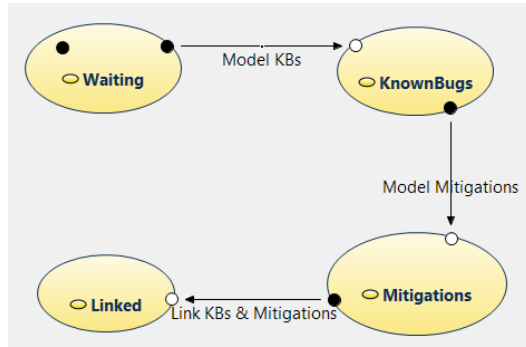


Fig. 7: Specification of Known Bug Modeling Behavior

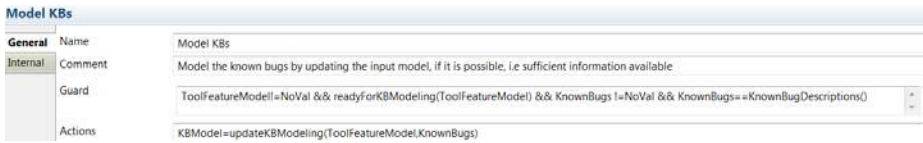


Fig. 8: Specification of the Transition link “Model KBs” from Fig. 7.

As an example of defining verification and validation activities for known bugs, we define the corresponding *Criterion* for known bugs modeling as depicted in Fig. 9. Note that this example shows the relevant V&V activities only partially.

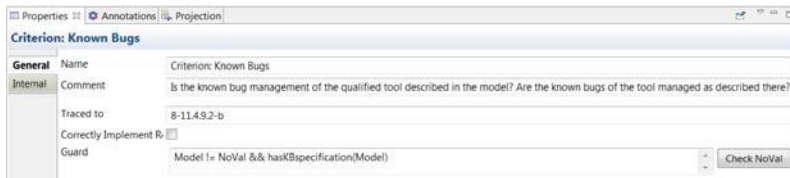


Fig. 9: Criterion Known Bugs

Requirements are typically structured in a hierarchic manner, which can be done in AutoFOCUS3 straightforwardly. It remains however to link/trace these requirements with the process modeling. To do so, we recognize that safety standard requirements are typically imposed on a) Processes and b) Products. For the purposes of certification and showing compliance, evidence should be provided that a) comply with requirements and that these processes have been actually followed/deployed for creating b). In our approach, we model the modeling activities/actions themselves including V&V activities that can be seen as a checklist for the concrete project. As such, every requirement has at least two traces in the processes model:

1. One to a process activity that describes it
2. Another to a V&V activity that checks it on the concrete example.

By splitting the requirements into these two parts, we can demonstrate that our process satisfies the requirements, provided that for each project the V&V activities are performed successfully. The concrete projects do not need to bother on the standard compliance, but have only their concrete checklists to perform.

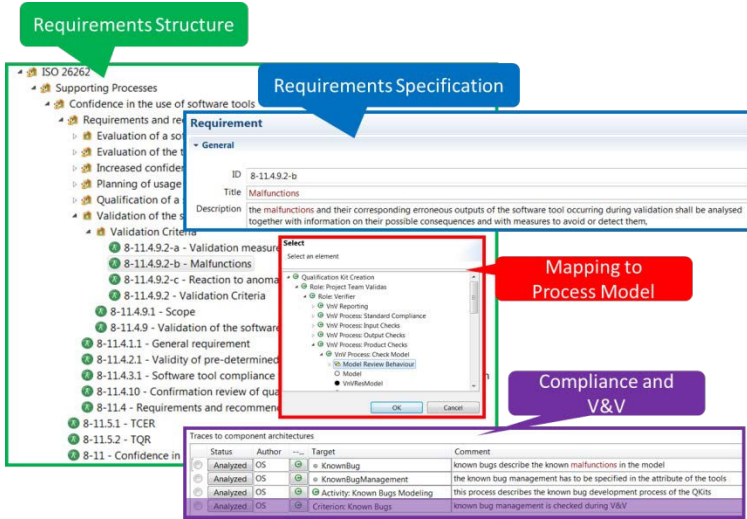


Fig. 10: Requirements Management and V&V Activities

Fig. 10 shows some examples of how the requirements from the safety standards are managed. The highlighted trace to the “Criterion: Known Bugs” is a trace to the V&V activity. The way in which this Criterion is defined is already discussed (and shown in Fig. 9).

6 Example: Certification of Tool Qualification Processes

Validas has applied (and validated) the proposed approach on the example of tool qualification as has been detailed in previous sections. Validas claims to build ISO 26262 and IEC 61508 compliant tool qualification kits using a model-based qualification process. The Validas process for building qualification kits has been modeled within AutoFOCUS3 and it has been shown to satisfy all 120 requirements from ISO 26262 and IEC 61508 for tool qualification. 13 additional requirements from Validas (functional and quality requirements for the model-based QKit) have been satisfied using a process description consisting of over 1150 element describing the process and the interaction with the customer. Every Criterion (V&V Check) consists of several questions, separated by “?”. In total over 250 (simple and concrete) questions in the criteria have to be answered for each QKit to pass the V&V.

According to the Method described in Section 2 the relevant documents have been generated.

Throughout the certification process regarding Validas process, TÜV requested not only the compliance with the tool qualification requirements, but also some general requirements on the management of functional safety. Those requirements could be easily satisfied by filling out the required Excel checklists and providing the evidences from the general Validas processes.

The main requirements have been successfully certified based on the generated documents: Validas Qualification Method (including the generated, formal process description), compliance report (generated as described) and V&V Plan (generated from the model as described).

7 Summary

We have presented a general two-step compliance approach that builds upon formalizing and modeling arbitrary requirements and model processes. In the first step, compliance to safety standards of the general deployed methodology is shown and compliance report is generated, whereas in the latter step, compliance of concrete projects that make use of that general methodology is checked against a V&V plan. Both the compliance report and the V&V plan are automatically generated from the constructed model. The method has been applied successfully within our certification efforts of the Validas model-based qualification kit construction processes with TÜV SÜD, according to ISO 26262 and IEC 61508.

8 Acknowledgments

This work has been supported in part by the German Federal Ministry of Research and Education (BMBF) within the project TOPWATER (ZIM) under research grant ZF41611701BZ5. The authors would like to thank Joachim Schramm for the fruitful and stimulating discussion within the TOPWATER project and Thomas Escherle for proof reading and formatting the paper.

9 References

- [In10] International Electrotechnical Commission: IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related systems, Edition 2.0, 2010.
- [RT11] RTCA: DO-330: Software Tool Qualification Considerations 1st Edition, 2011.

- [In11] International Organization for Standardization: ISO 26262 Road Vehicles –Functional safety–. 1st Edition, 2011.
- [Sl12] Slotosch, Oscar: Model-Based Tool Qualification - The Roadmap of Eclipse towards Tool Qualification – opencert, 2012.
- [Wi12] Wildmoser, Martin; Philipps, Jan; Jeschull, Reinhard; Slotosch, Oscar; Zalman, Rafael: ISO 26262 - Tool Chain Analysis Reduces Tool Qualification Costs. In SAFECOMP 2012, 2012.
- [Va17] Validas AG: Tool Chain Analyzer Tool, can be downloaded from www.validas.de/TCA.html, 2017.

Controlled Complexity for Future Mobility – Methodology, Guidelines and Tooling

Christian Reuter¹

Abstract: The automotive industry is currently facing the most extensive changes since its invention. Connected, autonomous, shared, and electric crystallize as game changers whereupon new key player arise with expertise therein. Internet of things principles bring the vehicles online and so the product life cycle shortens. Although new functionalities should be rolled out quickly. This contradiction needs new methodologies, guidelines and tooling. In this paper, the current usage of combined textual and model-based requirement specification as well as variant management techniques at Daimler is presented in context to research outcomes. Furthermore, upcoming challenges in the field of handling complexity are described to give an example for ongoing investigations.

Keywords: Complexity, Variant Management, Systems Engineering.

1 Introduction

A bunch of innovations has led the automotive industry to a complex system architecture in modern vehicles. A Mercedes-Benz S-Class (W222) has about 150 systems with functions realized on 100 electronic control units with over 80 million lines of code [Sc15]. The degree of dependency is rising, since the number of sensors and actors nearly allows full automation up to autonomy. As known by modern smartphones, innovation drivers come by combining the persisting hardware with new software functionalities. These functionalities have to be implemented in development artifacts like requirements, models, test cases, and code. This principle is keeping the costs per piece down, but demands higher efforts in creating and integrating new concepts as well as validating the larger variability and mitigating malfunctions. Hereby, one of the biggest challenges is the short-term evaluation of changes by recognizing the development artifacts, which were affected.

In the next section, the current approaches for handling complexity are presented. In addition to the applied methodologies, guidelines, and tooling, also scientific analysis on these are provided.

Given that practical application falls short of academic research the pure form of approaches is not always one-to-one implementable into company processes. Therefore, dimensions of stakeholders of the development process and resulting realization tracks are described. Hence, appearing challenges, which need approaches to handle this

¹ Daimler AG, Research & Development, X426, Sindelfingen, 71059, christian.c.reuter@daimler.com

imperfection, are discussed in the third section. Finally, the presented approaches and challenges were summarized.

2 Foundations in Practical Handling of Complexity

Complexity pushes the need to establish a more high-level view on relations. Therefore, at Daimler Research & Development, for example, UML activity diagrams are used to structure functions and encapsulate them according to the IPO model [BVR17]. In addition, further diagrams are used to describe behavior (e.g. state machines, sequence diagrams) or structure.

One issue in splitting the development into different levels is keeping them consistent. A previous case study has shown that the combination of graphical models and a textual representation can match different needs of stakeholders. Although it should be considered, that the implementation follows an in advance with all participating partners agreed guideline [BRV18].

Regarding a software product line, changes can increase variability of development artifacts. Bauer et al. discussed “methods for the determination of change impacts” [Ba15]. The major problem in industrial application of these methods is a high effort to reach the initial conditions for using them. As well as the point of view, where based on a mainly product structure orientated approach the evaluation of efforts is getting more complex, since the derivation of development artifacts from the elements of a product structure needs further information.

Besides that, the concept of FODA (feature-oriented domain analysis) domain modelling [Ka90] is based on an easier implementable concept, of a tree of features. This furthermore allows extending details as of more feature levels over time. Aside from that, no differentiation of links is necessary. A link therein describes the relation of a development artifact to a feature, to such an extent as a development artifact relies on a feature.

At Daimler, this is used for a multi-level concept, which gives assistance to the developers in managing complexity. Here, all mentioned steps are based upon the same principle. The specification is called a 150%-specification, as it comprises different self-contained specifications with a common core and individual extents:

- On level one, complexity can be handled with explicit marks within a requirements specification or a graphical model, so that according to that mark, the document can be filtered and only relevant aspects appear. This concept is applicable for systems with a smaller complexity.
- On level two, a simplified form of feature-based variability management can be used, which is in daily use for systems with a medium complexity. Therefore, e.g. a requirements specification is linked to a feature tree within the same tool, so

every requirement is assigned to a feature. At the feature tree, there are features and their characteristics listed in a flat hierarchy and finally variants can be built up as a set of features.

- In an environment with high demands on managing complexity, on level three, the combination of a specialized variant management tool (e.g. pure::variants [PU17]) with interfaces to – for the specific company or department – most common development tools (e.g. DOORS [IB17], Enterprise Architect [SP17], Simulink [MA17]) can be used. In this case, the feature tree is independent of a single development tool. The mentioned specialized variant management tool supports a capable multi-level feature tree, which is furthermore extendable with in-depth logic-based constraints [Bo11].

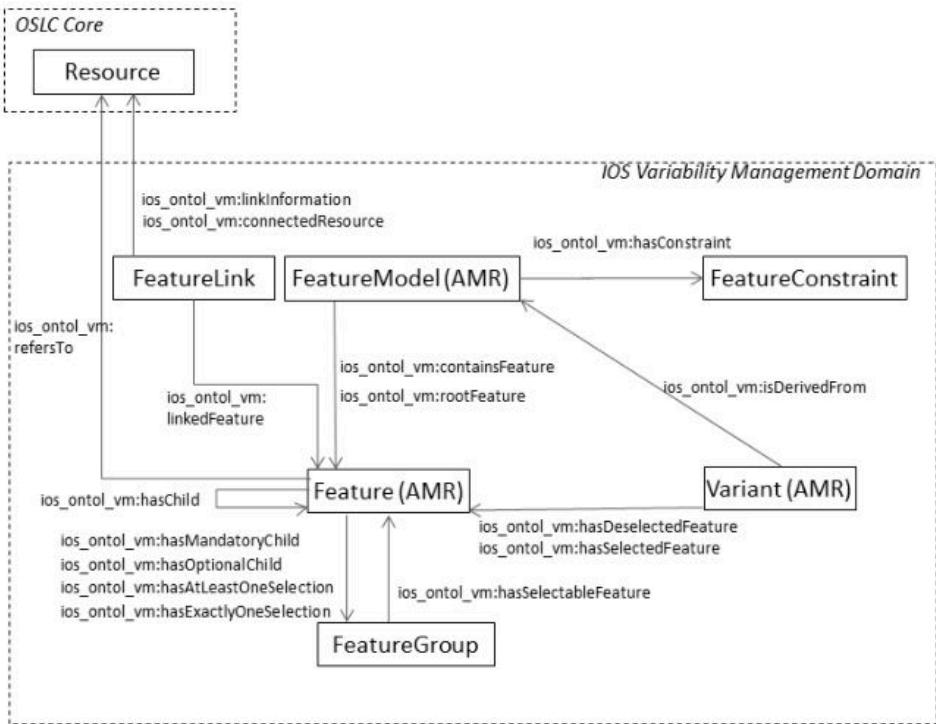


Fig. 1: Draft version of Variability Management Domain for OSLC [Re15]

Besides these in-tool, one-to-one tool, or one-to-many tools solutions, previous work has shown that a continuous systems engineering environment would bring the highest benefit to the users and IT operations. To reach this aim, an extension of the OSLC (Open Services for Lifecycle Collaboration) standard was proposed as part of the ARTEMIS EU project – CRYSTAL (CRITICAL sYSTEM engineering AccELeration). Part of this was to strengthen the concept of an Interoperability Specification and an

interrelated Reference Technology Platform [CR17]. As a contribution to the Interoperability Specification, the diagram in Fig. 1 pictures the objects “FeatureLink” and “Feature”, which can relate to other OSLC resources in order to attach variant information to a variety of development artifacts [OS17]. Several tool implementations (e.g. PTC Integrity [PI17], PTC Modeler [PM17], Enterprise Architect) were realized to demonstrate the capabilities of this concept [Re15].

In the area of alternative drive systems, an interview-based study of Beckmann et al. [BRV18] depicts that overlooking today's challenges the trade-off between the separation of concerns and the interlink of different levels and views like function orientation, component mapping, signal flow, coding of electronic control units and also safety and security topics is difficult to handle.

3 Upcoming Challenges for a Variety of Development Artifacts

The described methodology gives advice to the developer how variant handling can be pursued. However, in daily use, the comprehension for variant management is essential for developing a variant friendly architecture and therein comprised functions. This means that widely generic interfaces between the components as well as a clearly defined hierarchical decomposed structure inside the components is a prerequisite for the flexibility the industry is dependent on nowadays [CN02].

To illustrate the stakeholders of the development process resulting in different dimensions, which have to be handled, the following list gives assistance:

- Development Process (from system design till system validation with stakeholders like strategy, sales, user interaction)
- Additional Processes (change requests after contracting or issues in the supply chain)
- External regulations (legislation, state of the art e.g. documented by ISO or GB/T standards)
- Manufacturer/OEM-specific formalities (e.g. hardware/software architecture, communication matrix, security requirements)

These influences lead to a functional structure with a variety of requirements, models, software with related calibration, and more. For handling their complexity, two major tracks should be considered:

- The first track includes all information, which is well known and can be considered from the beginning of the development process. That applies also to manufacturer/OEM-specific formalities and mostly to external regulations.

- The second track of late changes is more difficult to handle. Depending on the scale of the change, this track starts earliest after contracting with Tier-1 suppliers and latest when the milestone of 100% functionality is reached. This track also applies to a software product line related development, when the newer software version relies on the previous one.

For the first track, the strategies mentioned in Section 2 are applicable, although the efficiency of exchanged variant data between the different development tools could be increased. Also, a comprehensive controlling of variability over the single levels from product over system to components and across the horizontal departments (like development, validation, after sales) would bring a better integration to fasten realization phases.

Quite more challenging are late changes. Here, usually time, cost, quality, and package are the most significant measurements. If market influences or misleading design decisions result in a late change, the appreciation of values needs more information. At first sight, it needs a clearly defined aim. This is the basis for alternating concepts. In turn, this leads to a scope, what has to be adapted (e.g. components, communication, documentation, testing). For getting a fast decision the more helpful it is for the involved departments the closer the scope is that they have to investigate for implementing the concept. Finally, the evaluated concepts need to be weighed, which concept fits better or is even one at least suitable.

4 Conclusion

The handling of complexity questions will be a key skill for developers as well as for companies. The foundations must be well known and day-to-day routine. Therefore, the knowledge base has to be broadened and common guidelines have to be agreed. Areas of high variability can be a starting point for that, but cross-sectional support is required. Because managing complexity cannot be reached by finding a local optimum, also the interaction of tools has to be pushed to prevent isolated applications. Furthermore, some challenges still persists, which need new methodologies. A short-term evaluation of a rating of change impacts and their extent with preferably low entry requirements rises the attractiveness of such an approach.

Bibliography

- [Ba15] Bauer, W.; Bosch, P.; Chucholowski, N.; Elezi, F.; Maisenbacher, S.; Lindemann, U.; Maurer, M.: Complexity Costs Evaluation in Product Families by Incorporating Change Propagation. In: 9th Annual IEEE International Systems Conference, 2015.
- [BVR17] Beckmann, M.; Vogelsang, A.; Reuter, C.: A Case Study on a Specification Approach using Activity Diagrams in Requirements Documents. In: International Requirements Engineering Conference (RE), 2017.

- [BRV18] Beckmann, M.; Reuter, C.; Vogelsang, A.: Coexisting Graphical and Textual Representations of Requirements: Insights and a Guideline. In: International Working Conference on Requirements Engineering — Foundation for Software Quality (REFSQ), 2018.
- [Bo11] Boutkova, E.: Experience with Variability Management in Requirement Specifications. In: 15th International Conference on Software Product Lines (SPLC), 2011.
- [CN02] Clemens, P.; Northrop, L.: Software Product Lines – Practices and Patterns, Addison-Wesley, Boston, 2002.
- [CR17] Crystal, <http://www.crystal-artemis.eu/>, accessed 21/12/2017.
- [IB17] IBM Rational DOORS Family, <https://www.ibm.com/us-en/marketplace/rational-doors>, accessed: 21/12/2017.
- [Ka90] Kang, K.; Cohen, S.; Hess, J.; Novak, W; Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. In: Technical Report CMU/SEI-90-TR-21 ESD-90-TR-222, 1990.
- [MA17] MathWorks, Simulink – Simulation and Model-Based Design, <https://www.mathworks.com/products/simulink.html>, accessed: 21/12/2017.
- [OS17] Specifications – Open Services for Lifecycle Collaboration, <http://open-services.net/specifications/>, accessed: 21/12/2017.
- [PI17] Integrity | PTC, <https://www.ptc.com/en/products/plm/plm-products/integrity>, accessed: 21/12/2017.
- [PM17] Integrity Modeler | PLM | PTC, <https://www.ptc.com/en/products/plm/plm-products/integrity-modeler>, accessed: 21/12/2017.
- [PU17] pure-systems – The leading provider of software for product line and variant management tools | pure::variants, <https://www.pure-systems.com/products/pure-variants-9.html>, accessed: 21/12/2017.
- [Re15] Reuter, C.: Variant Management as a Cross-Sectional Approach for a Continuous Systems Engineering Environment. In: Graz Symposium Virtual Vehicle, 2015.
- [Sc15] Schneider, J.: Software-innovations as key driver for a Green, Connected and Autonomous mobility. In: ARTEMIS-IA/ITEA-Co-Summit, Berlin, 2015.
- [SP17] Sparx Systems, Enterprise Architect – UML Design Tools and UML CASE Tools for software development, <http://www.sparxsystems.com/products/ea/>, accessed: 21/12/2017.

Taming the Software Development Complexity with Domain Specific Languages

Experiences from Deploying MPS-based DSLs for Computed Tomography Scanners at Siemens Healthineers

Daniel Ratiu,¹ Holger Nehls,² Jochen Michel³

Abstract: Modern computed tomography (CT) scanners are highly complex and flexible devices. This versatility is realized with a multitude of interconnected parameters and rules which are defined by domain experts in so-called scanner model specifications distributed over almost one hundred documents. The primarily used tools to write these documents (e.g. MS Word, MS Excel) are domain agnostic and they support only plain natural-language for the specification. Consequently, maintaining a valid scanner specification is a tedious, error-prone and therefore expensive process. To tackle the complexity of scanners parameters specifications, over the last two years we developed and deployed an eco-system of domain specific languages (DSLs) and associated tooling, covering a central portion of the scanner domain. The languages are developed using the JetBrains' MPS language workbench. In this paper, we present our experiences with developing our language eco-system. We briefly describe the language architecture, the design and development process that led us there, and discuss variation points of our approach and present in more detail a set of lessons learnt and best practices.

Keywords: domain specific languages, industrial experience, JetBrains' Meta-Programming System

1 Introduction

Non-invasive imaging is one of the most important improvements in medicine and enables doctors to diagnose and heal diseases that are not visible without having insights into the human body. Modern Computed Tomography (CT) scanners are highly complex machines, enabling radiologists to perform examinations of patients, like trauma scans, evaluation of neurological abnormalities, detection of tumors or diagnostic of heart diseases. While the x-ray beam rotates around the patient, the detector measures the attenuation, which represents the composition of the scanned object. Based on this volume data, it is possible to reconstruct slice images and calculate 3D visualisations of the human body and organs. This data is the basis for applications that support the radiologist in the diagnose process.

CT scanners are perfect examples of software intensive cyber-physical systems – a large amount of software enable the realization of complex use-cases and the interaction with

¹ Siemens Corporate Technology, Munich, daniel.ratiu@siemens.com

² Siemens Healthineers, Forchheim, holger.nehls@siemens-healthineers.com

³ Siemens Healthineers, Forchheim, jochen.michel@siemens-healthineers.com

the real world. The system depends on a wide set of parameters that represent quantities from the physical world, such as dose parameters, geometric properties and special scanner capabilities. Besides the program code per se, the complexity of software is also due to the big variability space defined by these configuration parameters. Valid combinations of parameters reflect physical capabilities of the devices and the desired clinical cases to be performed. A central challenge that the scanner development teams need to address is to keep the parameters consistent for a wide variety of clinical cases, on different hardware and across product lines. Inconsistencies of the parameters configurations can lead to bad imaging or even damages to the CT scanners.

Traditionally (Figure 1-up), experts from the CT domain (e.g. physicists) define possible parameter configurations using tables in MS Excel and MS Word documents. These documents are written in plain natural language and have a very weakly defined structure. Quality assurance of these specification documents is realized exclusively through manual reviews. Once the valid configurations are defined, a semi-automatic process involving manual transformations and different scripts is employed to generate configuration files in XML format which can be loaded by the scanner software. This manual process of creating the specifications is slow, prone to inconsistencies and reaches its limits due to the complexity of modern CT scanners.

The use of domain specific modeling tools drastically increases the development productivity and quality: on the productivity side domain experts benefit from a higher level of abstraction and higher automation; on the quality side they benefit from modeling guidance and advanced consistency checks. To cope with the complexity of the CT domain, we have built a set of

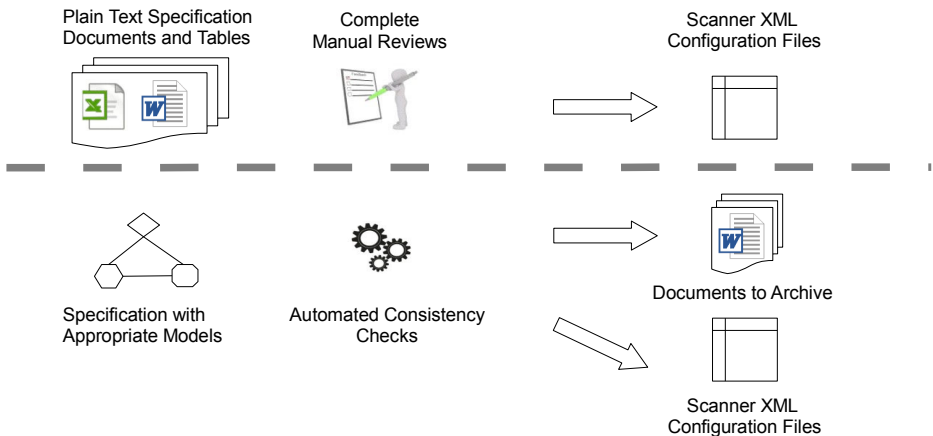


Fig. 1: Specify parameters and their relations using plain text and Excel tables requires big review effort for quality assurance and fragile semi-automatic generation of XML files (top); Model-driven specification of parameters enables deep and automatic consistency checks and automatic generation of XML-based configuration files to be loaded on the server (bottom).

domain specific languages and tooling (in the following called *Scanner-DSL*) which we use to specify the configurations of parameters of CTs. These rich models allow a wide range of consistency checks and automatic generation of configuration files in XMLs format and which are subsequently loaded in CT scanners (Figure 1-bottom). Besides XML, for process compliance reasons, we also generate PDF documents which are subsequently archived.

Contributions: In this paper we present our experiences with developing an eco-system of domain specific modeling languages over the last two years. Model-driven development approaches are widely used within Siemens Healtineers. However, the Scanner-DSL project is the first big project based on language engineering technologies. Thereby, besides presenting the tooling per se, this paper also describes our approach on technology transfer and adoption of domain specific modeling approaches in an industrial setting. Last but not least, based on our experience, we derive a set of lessons learnt and open challenges which need to be addressed for a broader adoption of the technology.

Structure: In Section 2, we give a brief overview of the technologies used and the developed DSLs. In Section 3 we present the development process and the major phases of our project. In Section 4 we discuss variation points of the approach and present our lessons learnt. In Section 6 we conclude the paper and give an outlook on future work.

2 Scanner-DSL

Scanner-DSL is an eco-system of languages and tooling built with JetBrains' MPS⁴ language workbench and the extensions offered by mbeddr-platform⁵. In Figure 2 we illustrate the architecture of our tool.

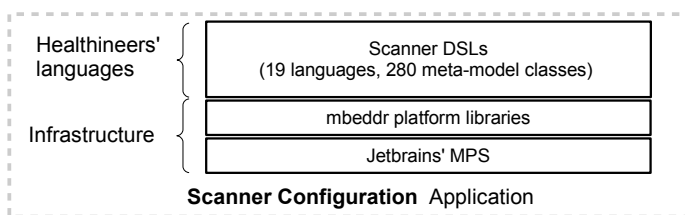


Fig. 2: Scanner-DSL architecture at a glance: the application, built around a set of domain specific languages, is based on Jetbrain's MPS and the extensions provided by the mbeddr-platform.

For reasons of brevity, we will not describe MPS or the mbeddr-platform libraries in detail; we refer the reader to [Ca14], [Vo13] or [Vo16]. However, in order to make this paper self-contained, we will briefly present in the following two subsections the major features

⁴ <https://www.jetbrains.com/mps/>

⁵ <http://mbeddr.com/platform.html>

of MPS and of mbeddr-platform which we used in our project. In Section 2.3 we briefly describe our tool.

2.1 JetBrains's MPS

Jetbrains' Meta-Programming System⁶ is an open-source language workbench which offers comprehensive support for all concerns of the development of DSLs and associated tooling. In MPS, a language implementation consists of several *language aspects* – e.g. structure, concrete syntax, constraints, type system, transformations, interpreters, or debuggers. MPS ships with a set of dedicated DSLs for implementing each language aspect.

Editors. MPS, at its core, features a projectional editor to display models. Projectional editors do not use parsers; instead, they render, or project, a program's abstract syntax tree (AST) in a notation defined by the language developer. Language engineers can choose to use specific notations appropriate for the business domain they address – e.g. plain text, forms, tables, diagrams, mathematical formula or trees.

Context Sensitive Constraints. MPS guides the language users (i.e. domain experts) towards building models in two ways: 1) *constructively* by preventing the definition of invalid models up-front using an advanced set of scopes and constraints; and 2) *analytically* by allowing the definition of advanced checks in the IDE. The constructive way is using the projectional nature of MPS directly – the users are allowed to enter only valid content. Further constraints are essentially implemented as if-statements that check some property of the AST and report errors if invalid code is detected.

Generators. MPS generators usually work as a chain of model-to-model transformations where domain-specific ASTs get enriched by platform-specific information with each transformation step. Eventually, a chain reaches the target language and a model-to-text transformation produces an output text file. Whilst MPS generators are not bound to this design, it is the most common use case since it allows for a very modular approach to combine and interchange transformation steps.

Tooling via IDE Extensions MPS also allows the definition of IDE extensions such as new menus or views; language engineers use these extensions extensively for building domain specific tooling or to integrate external tools. The IDE extensions are implemented via regular Java/Swing programs and a couple of MPS-specific extension points.

⁶ <https://www.jetbrains.com/mps/>

Foundational Support for Model-driven Development MPS offers comprehensive support for the entire life-cycle of model based development with domain specific languages: integrating with different version control systems, merging and diffing at model level, testing of different aspects of the language definition, refactorings of languages and models, migration of models when languages evolve.

2.2 mbeddr Platform

Besides the set of DSLs and language definitions aspects shipped with MPS, we made use of an additional set of libraries for developing new languages. These libraries are offered by the *mbeddr platform* [mbe15] provide additional language definition aspects and DSLs for defining special editors. We have made use of tabular and mathematical notations [VL14], grammar cells for the definition of consistent editors and language documentation aspect.

2.3 Scanner DSL

In Figure 3 we present a screenshot of our tool which contains examples of three models, each built with a different DSL: a model which describes the available parameters with their set of possible values (top-left) and two models describing possible valid combinations of these parameters. Our models make use of textual, mathematical and tabular notations. On the bottom-right we illustrate errors caused by a failed consistency check.

Our eco-system of DSLs contains 19 languages providing 264 concepts (i.e. meta-model classes), with 57 properties and 264 relations between them. We implemented 59 constraints and scoping rules which restrict constructively the building of semantically flawed models and 112 more complex consistency checks. For quality assurance of these languages, we have extensively used the testing infrastructure of MPS. We created ca. 60 test-cases with more than 450 assertions. The generators are tested by comparing the generated artifacts with a manually reviewed baseline.

Usage Our tool is currently used in production to model three existing scanners. The initial modeling was done by one of the members of the language engineering team. We were constantly in touch with our domain experts and future users of the tool. At the end of 2017, the user models have 104 parameters, 1111 composite rules containing 5553 atomic rules distributed across more than 200 tabular rules. These models have been recently taken over and enhanced by three domain experts. Our domain specific tool went into production several months ago and is used today full-time by two users (both non-computer scientists).

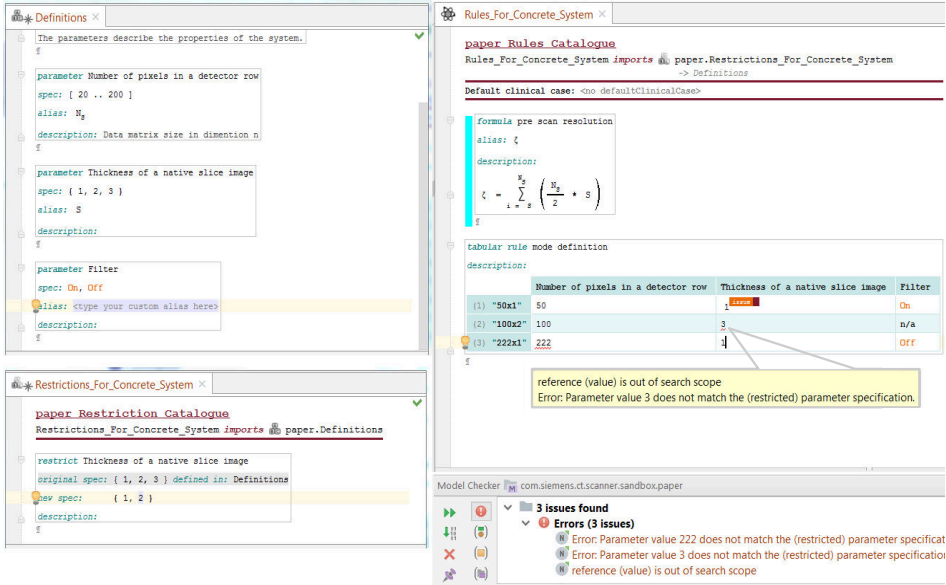


Fig. 3: Examples of models built with three different DSLs. The models are deeply integrated with each other which enables complex consistency checks.

3 Development Process

In the following we present three highlights of our project concerning the development process: the phases of our project, enabling continuous integration of languages and models and involving domain experts.

3.1 Project Phases

Our development process varied substantially within the last two years of the project and can be divided into four phases. In Table 1 we present an overview over these phases, their duration and the number of persons involved.

Phase 1: Ramping-up During the first 6 months of the project, until the team got confident with the technology, we had a ramping-up phase. It was characterized by the exploration of the MPS technology stack and rapid prototyping of relevant use-cases in order to understand the limitations. The main goal was to produce enough functionality to convince the other teams about the meaningfulness of the modeling approach. During the ramping-up phase most of the development took place during several hackathons, each of them being three days long.

Phase 2: Initial development After the ramping-up phase, several team members got confident with the MPS technology and the team allocated half a person for the development. Soon after, the first interns joined the team and the development got a higher dynamics also in-between the hackathons. The hackathons were used to explore advanced features of MPS and to solve more complex problems.

Phase 3: Mature project After one year, the development was accelerated in order to synchronize with the planned deadlines. Our team grew further and this lead to a significant increase of the code-base and functionality. We integrated our project in the existing continuous integration infrastructure and increased the coverage of our tests.

Phase 4: Production After one and half years since project start, our system went into production and the size and the number of user-models describing parameters configurations started to grow rapidly. The first domain experts started to use our DSLs and the development of languages and user models got different dynamics. This lead to the necessity to decouple the life-cycles of language development and the use of languages for developing CT specifications.

Phase	Duration (in months)	Team-size (#developers / #students)
Ramping-up	6	0.5
Initial devel.	6	0.8 / 1
Mature project	9	1.2 / 2
Production	5	2.8 / 2

Tab. 1: The dynamics of our project changed substantially between phases: we started with a very small team and and once the value of the technology had been proven, the team was increased in size and contributes directly to productive software development projects.

3.2 Continuous Integration

Both the language engineering and the domain experts teams are distributed: language development happens in Forchheim, Cologne and Munich. The domain experts work distributed from Germany and China. Starting relatively early in the development of the Scanner-DSLs, we introduced continuous integration for the development of DSLs. After reaching the first functional milestone, we deployed the languages and tooling as a standalone application to be used by domain experts. In Figure 4 we illustrate these two delivery pipelines: the domain experts use releases of the Scanner-DSL tooling for building their models. Artifacts (like configurations as XML) are generated automatically from the models and integrated within the continuous integration pipeline for our scanner software.

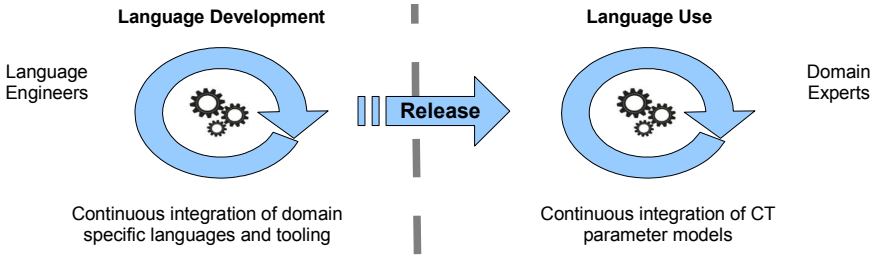


Fig. 4: The continuous integration is performed both for the language development and for the language use. When the eco-system of DSLs reach a new baseline, we make a release of the Scanner-DSL tooling and the domain experts use this release for their development of parameters configurations for the CT scanners.

3.3 Involving Domain Experts

The DSLs we develop capture the semantics of the CT scanner domain in an explicit and precise manner. The language development process went closely together with a knowledge engineering process inside the organization. At any point in time, at least one member of the language development team had several years of experience with the scanner software development and thereby in-depth domain knowledge.

During creation of languages, we had to make the domain knowledge explicit (i.e. choose which concepts and relations among them are captured as first class language constructs, and which constraints need to be implemented). We identified edge-cases for which several domain experts needed to be involved in discussions. Many times these experts had a slightly different view over their domain and they needed to agree upon how the domain looks like.

4 Discussion and Lessons Learnt

In this section we discuss important variation points of our approach and present our lessons learnt and open challenges.

4.1 Discussion

On Projectional Editing One of the most important distinguishing feature of MPS is the projectional editing. Being a projectional editor, MPS does not feel like text editors when users edit their models. In order to increase the fluency of models' creation and modification, MPS allows advanced customization of editor actions – e.g. what happens when the user presses "backspace" in a certain editor cell, or how are linear sequences of lexical items are transformed into models. We have invested some effort to make the editing experience as

intuitive as possible. At the same time this means that the users, who know and sometimes expect fundamental features of Excel or Word, need to understand that the focus of the new tool (from their point of view) is not to mirror known textual editor features, but to model the semantics of the computed tomography scanner data. The initial feedback from our users with respect to the usability of the editor is positive – the users immediately understood that they are not editing “simple” text but rich models and thereby they calibrated their expectations.

On MPS’s Extensibility The seamless extension capabilities of MPS with additional language definition aspects is a key feature which we made use of in our project. Besides the standard aspects shipped with MPS, as presented in Section 2, we have intensively used the “mbeddr-platform” libraries featured as part of mbeddr. Dependencies among different extensions need to be managed appropriately such that the deployed Rich Client Platform (RCP) can be built in a meaningful manner.

On Language Evolution and Models Migration on Multiple Branches MPS provides out-of-the-box advanced support for evolving DSLs and migrate the models to the new versions of the DSLs. We have used these features intensively to perform agile language development. However, if the models are built on different branches then they also need to be migrated to new language versions individually. Comparing (or merging) these branches after migrations have been performed on them proves to be challenging.

4.2 Lessons Learnt

DSL Development and Domain Engineering go Hand-in-hand. In addition to the DSL development itself, substantial effort has been involved in domain engineering. The knowledge of domain experts (i.e. domain concepts, their relations and constraints on valid combinations) at a certain point in time was formalized in the DSL. The DSLs help us better understand, manage and consolidate the knowledge in our organization. Once initial versions of the DSL was built, it was subsequently piloted to model different aspects of the scanner domain and by doing this we identified improvements of DSL.

Along with the development of DSLs the team went through a learning process about the domain – we continuously got feedback from domain experts – and ca. 10-15 persons are aware about different details of our DSLs and continuously validate what we are developing.

Continuously Demonstrate the Added Value from Early Stages Initial experience with creating the user models was very useful to convince the stakeholders about the value of the model-based approach. We were able to detect several inconsistencies in the original

data which passed through multiple-review sessions and this was a strong argument for semantically rich models. The possibility to generate XML configuration files from models quickly and in a fully automated way served as additional argument for our approach.

MPS Enables Highly Efficient Development of DSLs The infrastructure provided by MPS and mbeddr-platform allowed us to develop the languages in a highly efficient manner. After a few hours of development, we could get a baseline for languages and tooling which can be used as input to engage in discussions with domain experts. This baseline is then subject to iterative improvements, each iteration consisting often only of several hours of development.

Need for Support for the Entire Life-cycle of DSLs The support for entire life-cycle of DSL engineering and development offered by MPS proved to be essential - starting from the DSLs used to define different language aspects, with testing, refactoring and support for continuous integration.

Configuration Management The DSL development team is distributed between Forchheim, Cologne and Munich. The team of domain experts using the DSLs is distributed as well between Germany and China. The support offered by MPS for advanced versioning and merging both for language development as well as for the language use is of high importance for the adoption.

Semantic Richness Besides the definition of appropriate language constructs and constraints which prevent up-front building meaningless models, we have implemented a rich set of consistency and plausibility checks on the scanner models. These have proven to be highly useful and are appreciated by domain experts since they get feedback immediately in the IDE (e.g. in case consistency is violated). Corrective actions can be taken immediately before errors are discovered later in process or even introduced into the production.

Testing and quality assurance We have developed a comprehensive test-suite for testing the context-sensitive constraints and the generators. The checking rules are developed test-driven; the generator for XML artifacts is validated by using a baseline test method and reaches a block coverage of about 97% (measured at Java level using the *EMMA*⁷ code coverage tool). Each test is performed latest on our build server, triggered by every commit. These tests proved to be highly useful whenever we evolved the DSLs or migrated them to a new version of MPS. Overall, we estimate that the effort spent on writing unit test was 30% of the total development effort. The integration of testing in our continuous integration framework accounted for further 10% of the effort. We feel however that the testing capabilities of MPS could be enhanced towards support for "end-to-end" testing.

⁷ <http://emma.sourceforge.net/>

5 Related Work

[Vo17] presents lessons learnt from developing mbeddr, an open-source stack of domain specific languages built on top of C using JetBrains' MPS. mbeddr is one of the biggest DSLs based projects involving 10+ person years of development effort. This is the closest work on experience with instantiating DSLs technology with JetBrains' MPS. Compared to [Vo17], this paper presents experiences and particularities with transferring the language engineering technologies into industrial context and the entire life-cycle (i.e. from domain engineering to supporting domain experts in using the tooling) of deploying DSLs.

[MPP14] presents experiences and challenges with domain specific modeling in industrial automation domain. [To16, TK16] describes experiences with introducing MetaEdit+ language workbench in industrial context to create domain specific modeling languages. The experienced presented by Tolvanen and colleagues are very much similar to our experiences: support for the entire life-cycle of languages and models is needed, domain specific modeling and domain specific tooling drastically increase the productivity of the software development. Compared to these works, our experiences in this paper are based on a single, medium sized language engineering project in the healthcare domain. We also describe our approach to transferring language engineering technology in industrial practice.

6 Conclusions

In this paper we presented the first results from a two years endeavor on deploying domain specific languages to describe parameters and their configurations for CT scanners. This is only a first step to a holistic model based approach tailored to the needs of Siemens Healthineers Computed Tomography. The efforts reported here are part of a longer term project aimed at increasing the automation of the development of Computed Tomography scanner software. This is only the initial baseline representing one specification document – we plan to extend the languages and models for up to other 100 specifications successively. In parallel, the development of new innovative systems continues and new areas of applicability can be anticipated.

Acknowledgements. We would like to thank Robert Walter⁸ for the discussions and feedback on this paper.

References

- [Ca14] Campagne, Fabien: The MPS Language Workbench. CreateSpace Publishing, 2014.
[mbe15] mbeddr Platform. <http://mbeddr.com/platform.html>, 2015. Accessed: 2017-12-15.

⁸ Independent Consultant, Gleueler Str. 179, 50931 Köln, info@digital-ember.com

- [MPP14] Moser, Michael; Pfeiffer, Michael; Pichler, Josef: Domain-specific Modeling in Industrial Automation: Challenges and Experiences. In: Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation. 2014.
- [TK16] Tolvanen, Juha-Pekka; Kelly, Steven: Model-Driven Development Challenges and Solutions - Experiences with Domain-Specific Modelling in Industry. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development - Volume 1: Ind Track MODELSWARD. 2016.
- [To16] Tolvanen, Juha-Pekka: MetaEdit+ for Collaborative Language Engineering and Language Use (Tool Demo). In: Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering. 2016.
- [VL14] Voelter, Markus; Lisson, Sascha: Supporting Diverse Notations in MPS Projectional Editor. In: Workshop on The Globalization of Modeling Languages, co-located with MODELS. S. 7–16, 2014.
- [Vo13] Voelter, Markus; Benz, Sebastian; Dietrich, Christian; Engelmann, Birgit; Helander, Mats; Kats, Lennart; Visser, Eelco; Wachsmuth, Guido: DSL Engineering. dslbook.org, 2013.
- [Vo16] Voelter, Markus; Szabó, Tamás; Lisson, Sascha; Kolb, Bernd; Erdweg, Sebastian; Berger, Thorsten: Efficient development of consistent projectional editors using grammar cells. In: Proceedings of the International Conference on Software Language Engineering. 2016.
- [Vo17] Voelter, Markus; Kolb, Bernd; Szabó, Tamás; Ratiu, Daniel; van Deursen, Arie: Lessons learned from developing mbeddr: a case study in language engineering with MPS. *Software & Systems Modeling*, 2017.

Tutorials

Tutorials

Der Tutorial-Track der Modellierung 2018 umfasst Präsentationen zu aktuellen Methoden, Techniken und Werkzeugen für die Modell-basierte Entwicklung und Analyse von Software-intensiven Systemen. Die Tutorien sind in einem didaktisch ausgereiften und Praxis-orientierten Workshop-Format gehalten und kombinieren Vorträge zu konzeptionellen Grundlagen mit interaktiven Beispielen sowie Live-Demos.

Es wurden drei hochwertige Vorschläge für Tutorials für den Track eingereicht, von denen alle drei in das Programm der Modellierung 2018 aufgenommen werden konnten.

Das Tutorial *eMoflon: A Tool for Building* bietet einen Überblick über Prinzipien und Praktiken im modernen Modell-getriebenen Software Engineering. Unter Verwendung des Meta-CASE Tools eMoflon wird in dem Tutorial die Verwendung von Meta-Modellierungsansätzen und Techniken für unidirektionale sowie bidirektionale Modell-Transformationen beschrieben und anhand einer Fallstudie zu Objekt-orientierten Refaktorisierungen demonstriert.

Das Tutorial *Henshin: A Model Transformation Language and its Use for Search-Based Model Optimisation in MDE Optimiser* bietet einen Überblick über die Anwendung des Werkzeugs MDE Optimiser, welches eine Erweiterung des Modell-Transformations-Frameworks Henshins bereitstellt. Es wird demonstriert, wie mithilfe von MDE Optimiser Modell-Optimierungs-Probleme durch die Suche nach optimalen Modellen hinsichtlich beliebiger Fitnessfunktionen gelöst werden können.

Das Tutorial *Feature Modeling and Development with FeatureIDE* bietet schließlich einen Überblick zur Modellierung, Entwicklung und Analyse Feature-orientierter Software-Produktlinien. Es wird demonstriert, wie mit dem Werkzeug FeatureIDE Variabilitätsmodellierung im Problemraum unter Verwendung von Feature-Modellen sowie Techniken zur Abbildung von End-Nutzer-Features auf Domänen-Artefakte im Lösungsraum eingesetzt werden können.

Wir möchten dem Organisations-Team der Modellierung 2018, insbesondere den Organisatoren vor Ort, für die exzellente Unterstützung bei der Durchführung der Tutorials danken. Weiterhin geht unser Dank an die Mitglieder des Programm-Komitees für ihre hervorragende Arbeit bei der Begutachtung der Einreichungen. Schließlich möchten wir den Autoren und Präsentierenden für ihre hochwertigen Tutorials sowie allen Tutorial-Teilnehmern für ihre fruchtbaren und inspirierenden Diskussionsbeiträge danken.

Berlin / Darmstadt, im Februar 2018

Malte Lochau, TU Darmstadt

Timo Kehrer, HU Berlin

Programmkomitee

Anthony Anjorin
Thorsten Berger
Daniel Strüber
Mattias Ulbrich
Manuel Wimmer

Universität Paderborn
Chalmers Universität Göteborg
Universität Koblenz-Landau
KIT Karlsruhe
TU Wien

Feature Modeling and Development with FeatureIDE

Thomas Thüm,¹ Thomas Leich,² Sebastian Krieter³

Abstract: FeatureIDE is an open-source framework to model, develop, and analyze feature-oriented software product lines. It is mainly developed in a cooperation between TU Braunschweig, University of Magdeburg, and Metop GmbH. Nevertheless, many other institutions contributed to it in the past decade. Goal of this tutorial is to illustrate how FeatureIDE can be used to develop software around end-user features. We will show how feature models are connected to and synchronized with other artifacts. The hands-on tutorial will be highly interactive and is devoted to practitioners facing problems with variability, lecturers teaching product line development, and researchers who want to save resources in building product-line tools.

Keywords: software product lines; feature-oriented software development; feature modeling; product configuration; feature traceability; consistency checking; Eclipse; FeatureIDE

1 Motivation and Overview

Software systems often have to be tailored to the needs of different customers. If differences between those systems are made explicit in terms of features, feature-oriented software product lines can be used to automatically generate software variants based on a selection of features [Ap13].

In feature-oriented software development, valid combinations of features are defined in a feature model during domain analysis. In domain design and domain implementation, those features are mapped to development artifacts, such as models, code, documentation, or tests. Preprocessors support a fine-grained mapping, as illustrated in the tutorial. Guided by the feature model, valid configurations are derived and then used as input for the preprocessor.

Since 2004 we are developing tool support for feature-oriented software development for Eclipse in the FeatureIDE project [Me17]. Since 2009, FeatureIDE is open source and received contributions from all over the world. While FeatureIDE started as a tool for teaching and a vehicle for research prototypes, today it is also applied in industrial projects with thousands of features.

The tutorial is planned to be a highly interactive, half-day event. We will demonstrate FeatureIDE's functionality in addition to interleaved hands-on sessions, in which participants

¹ TU Braunschweig, Germany

² Metop GmbH, Germany; Harz University of Applied Sciences, Germany

³ University of Magdeburg, Germany; Harz University of Applied Sciences, Germany

can to tryout FeatureIDE and rely on our assistance. In the interactive parts, the goal is to modify an example product line with FeatureIDE. Participants are asked to *bring a notebook* for the hands-on sessions. The tutorial will cover the following topics:

1. Introduction to feature-oriented software development
2. Setting up Eclipse and FeatureIDE
3. Analysis of feature models and configurations
4. Analysis and testing in feature-oriented software development

We gratefully acknowledge *all* who contributed to the open-source project FeatureIDE. In particular, a special thanks for recent contributions to Timo Günther, Christopher Sontag, Joshua Sprey, Paul Westphal, Chico Sundermann, Holger Fenske, Jens Meinicke, Reimar Schröter, Gunter Saake, Ina Schaefer, Mustafa Al-Hajjaji, and Alexander Knüppel. A prior version of this tutorial has been presented at SPLC'16 [TLK16].

Literaturverzeichnis

- [Ap13] Apel, Sven; Batory, Don; Kästner, Christian; Saake, Gunter: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer, Berlin, Heidelberg, 2013.
- [Me17] Meinicke, Jens; Thüm, Thomas; Schröter, Reimar; Benduhn, Fabian; Leich, Thomas; Saake, Gunter: Mastering Software Variability with FeatureIDE. Springer, Berlin, Heidelberg, 2017.
- [TLK16] Thüm, Thomas; Leich, Thomas; Krieter, Sebastian: Clean Your Variable Code with FeatureIDE. In: Proc. Int'l Software Product Line Conf. (SPLC). ACM, New York, NY, USA, S. 308–308, September 2016.

Henshin: A Model Transformation Language and its Use for Search-Based Model Optimisation in MDEOptimiser

Daniel Strüber,¹ Alexandru Burdusel,² Stefan John,³ Steffen Zschaler⁴

Abstract: This tutorial presents Henshin, a versatile model transformation language increasingly used in academic and industrial applications. Henshin is based on the paradigm of graph transformation and provides a comprehensive tool set that supports largely declarative transformation specifications and various formal analyses. We present the application of Henshin in a *search-based model optimisation* task, where the goal is to find an optimal model regarding a given fitness function. Using Henshin, we specify evolutionary operators for MDEOptimiser, a novel search-based model optimisation tool.

Keywords: model transformation; graph transformation; model optimisation; evolutionary optimisation

1 Summary

Model transformation has been called the heart and soul of model-driven engineering, a paradigm in which models are continuously improved, refined, and translated. While transformations can, in principle, be developed using any general-purpose-language, these languages usually do not offer any support for challenges faced during transformation development, such as the need for verification, traceability, and optimisation. To better support developers, a variety of dedicated model transformation languages has emerged.

Henshin [Ar10, St17b] is a model transformation language based on the paradigm of algebraic graph transformations. Henshin's key benefits are: (i) a visual syntax, supporting a largely declarative specification of transformations, (ii) a mature formal foundation, enabling various formal analyses, and (iii) a comprehensive tool chain, comprising various editors, execution engines, and analysis tools. Supported analyses include model checking as well as conflict and dependency analysis. In academia, Henshin has been used in settings such as model versioning, model refactoring, and software product line transformations. In industry, Henshin has been used to verify the correctness of satellite control procedure translations.

Search-based model optimisation [ZM16] is a recent trend that combines the benefits of model-driven and search-based software engineering. Search-based software engineering

¹ Universität Koblenz-Landau, Universitätsstr. 1, 56070 Koblenz, Germany, strueber@uni-koblenz.de

² King's College London, London WC2R 2LS, United Kingdom, alexandru.burdusel@kcl.ac.uk

³ Philipps-Universität Marburg, Hans-Meerwein-Str., 35032 Marburg, Germany, stefan.john@uni-marburg.de

⁴ King's College London, London WC2R 2LS, United Kingdom, steffen.zschaler@kcl.ac.uk

provides software developers with technologies to solve optimisation tasks such as test case selection, component deployment, or release bundling. Yet, the encoding of these tasks as search problems is a complex and error-prone task that requires substantial expertise in meta-heuristic technologies, such as evolutionary optimisation algorithms.

To alleviate this issue, search-based model optimisation makes this expertise available by incorporating it into an optimisation framework. The user provides a solution-space specification, comprising a meta-model and some in-place model transformations for modifying the candidate solutions, i.e., the meta-model's instances. The model transformations can be specified manually, or even generated automatically [St17a]. In summary, search-based model optimisation supports the black-box use of search technologies, by enabling a problem specification based on domain concepts, rather a technology-specific encoding.

MDEOptimiser [BZ17] is a search-based model optimisation framework based on Henshin. Like other recent frameworks, in particular MOMoT [FTW16], MDEOptimiser involves Henshin to bridge model transformation with optimisation based on genetic algorithms. As its distinguishing feature, MDEOptimiser uses Henshin rules to specify the genetic operators, in particular the mutation operator. This set-up is particularly efficient in situations where the model itself, rather than the orchestration of rules, is to be optimised.

Goals. Participants will take three things from the tutorial: (i) How to specify and apply model transformation rules using Henshin, (ii) How to specify an optimisation problem using MDEOptimiser and Henshin, and (iii) How to design a high-quality mutation operator.

Prerequisites. Participants should be familiar with the Eclipse Modeling Framework (EMF), which provides the underlying modeling platform for Henshin. In particular, they should know how meta-models and model instances are specified using EMF.

References

- [Ar10] Arendt, Thorsten; Biermann, Enrico; Jurack, Stefan; Krause, Christian; Taentzer, Gabriele: Henshin: advanced concepts and tools for in-place EMF model transformations. In: MODELS. Springer, pp. 121–135, 2010. <https://www.eclipse.org/henshin/>.
- [BZ17] Burdusel, Alexandru; Zschaler, Steffen: , MDE Optimiser. <https://mde-optimiser.github.io/>, 2017.
- [FTW16] Fleck, Martin; Troya, Javier; Wimmer, Manuel: Search-based model transformations with MOMoT. In: ICMT. Springer, pp. 79–87, 2016.
- [St17a] Strüber, Daniel: Generating Efficient Mutation Operators for Search-Based Model-Driven Engineering. In: ICMT. pp. 121–137, 2017.
- [St17b] Strüber, Daniel; Born, Kristopher; Gill, Kanwal Daud; Groner, Raffaella; Kehrer, Timo; Ohrndorf, Manuel; Tichy, Matthias: Henshin: A Usability-Focused Framework for EMF Model Transformation Development. In: ICGT. pp. 196–208, 2017.
- [ZM16] Zschaler, Steffen; Mandow, Lawrence: Towards model-based optimisation: Using domain knowledge explicitly. In: MELO. pp. 317–329, 2016.

eMoflon: A Tool for Tools and Transformations

Lars Fritsche,¹ Géza Kulcsár²

Abstract: eMoflon is a model-based meta-CASE framework, which allows users to build their own solutions for modern MDE scenarios. Particularly, eMoflon supports meta-modeling and unidirectional as well as bidirectional model transformation. In this tutorial, those major functionalities of eMoflon are presented using a case study of object-oriented refactorings.

Keywords: Model-driven engineering; Model transformation; Bidirectional model synchronization

Model-Driven Engineering (MDE) is a software engineering principle, which tackles the challenges of complex and long-living software systems by treating architectural and behavioral system models as an inherent part of the software product. Nowadays MDE scenarios pose challenges of increasing complexity to software engineers, incorporating multiple models as well as diverse model engineering tasks:

- (T1) model-based specification of application domains,
- (T2) integration and evolution of models and
- (T3) verification and preservation of consistency between models which represent different system aspects but share some semantic features.

Particularly, while model evolution is often described by *unidirectional model transformation*, model integration and consistency requires the use of *bidirectional synchronization* techniques.

With the increasing popularity of MDE, a number of industrial as well as academical tools have appeared to facilitate the specification and transformation of models, relying on well-founded techniques such as *meta-modeling* and *model transformation*. Regarding the technical foundations, the Eclipse Modeling Framework (EMF) has become a de-facto standard for developing MDE tooling. However, the large majority of prevalent tools focuses on a single aspect of MDE-related activities like model creation or transformation, while not providing a holistic approach addressing each facet of the MDE process.

In this tutorial, we present eMoflon³, a model-based meta-CASE framework. Using eMoflon, we demonstrate how to specify EMF-conform meta-models as well as unidirectional and

¹ TU Darmstadt, Germany, lars.fritsche@es.tu-darmstadt.de

² TU Darmstadt, Germany, geza.kulcsar@es.tu-darmstadt.de

³ <http://emoflon.org/>

bidirectional transformations of EMF models. All features are shown and discussed on the example of a real-world case study of model-based *object-oriented refactorings*.

In particular, the refactoring case study is used in this tutorial to highlight the aforementioned major functionalities of the eMoflon suite:

- **(F1)** Representation of a custom-tailored abstraction over an object-oriented program using eMoflon and EMF-based meta-modeling techniques.
- **(F2)** Specification of correct refactoring rules over our custom-tailored program model by using the SDM model transformation language in eMoflon.
- **(T3)** Consistency preservation between refactored models and the original code base; propagation of changes in the source code or the program model to the other side using bidirectional model transformation in eMoflon.

The tutorial is aimed at model engineers of any experience level who are interested in a multi-purpose meta-CASE tool for tackling diverse challenges posed by the modern MDE landscape, within a single, flexible framework.

Overview:

- Metamodeling with eMoflon
- Unidirectional model transformation using *Story-Driven Modeling*
- Bidirectional model transformation with *Triple Graph Grammars*
- Case study of model-based *object-oriented refactorings*

Lars Fritsche is a doctoral student at the Real-Time Systems Lab at the Technical University of Darmstadt. His research interests are concurrent engineering and bidirectional model transformation.

Géza Kulcsár is a doctoral student at the Real-Time Systems Lab at the Technical University of Darmstadt. His research interests are the semantics of controlled graph transformation and its application for model transformation.

Werkzeugpräsentation

Werkzeugpräsentationen

Ziel des Tracks „Werkzeugpräsentation“ war es, Modellierungswerkzeuge zu präsentieren, die im wissenschaftlichen Umfeld entwickelt wurden und werden. Die Leitung für diesen Track haben Prof. Dr. Hans-Georg Fill und PD Dr. Agnes Koschmider übernommen. Vorgestellt wurden fünf Modellierungswerkzeuge, die einen interessanten Einblick in die aktuellen Entwicklungen im Bereich Modellierungswerkzeuge geben. Die ausgewählten Beiträge wurden auf Basis von jeweils drei Gutachten eines internationalen Programmkomitees ausgewählt. Die ausgewählten Werkzeuge bieten verschiedene Modellierungsunterstützungen. Es wurde ein Werkzeug für eine prozessbasierte Modellierung und Generierung mobiler Cloud-Anwendungen vorgestellt sowie ein Werkzeug zur interaktiven Simulation der musterbasierten Kontrollflusssemantik von Geschäftsprozessmodellen. Weitere Demonstratoren wurden für eine Augmented Reality-basierte Prozessmodellierung, für die Erfassung der konzeptuellen Modellierung und für ein interaktives Zoomen in Fehlerbäumen (Component Fault Trees) gezeigt.

Die Realisierung der Werkzeugpräsentation wäre nicht ohne die Unterstützung von zahlreicher Seite möglich gewesen. Die Organisatoren möchten sich daher besonders bei den Autoren der Beiträge für die Einreichung der Arbeiten und die Demonstrationen der Tools sowie bei den Mitgliedern des Programmkomitees für die fristgerechte Anfertigung der Gutachten bedanken. Ebenso ergeht der Dank an die Organisatoren der Modellierung 2018 für die Ermöglichung des Tracks im Hauptprogramm der Tagung. Die Werkzeugpräsentation fand am Donnerstag, den 22. Februar 2018 in Braunschweig statt.

Braunschweig, im Februar 2018

Hans-Georg Fill, Universität Wien

Agnes Koschmider, Karlsruher Institut für Technologie

Programmkomitee

Dominik Bork
Robert Buchmann
Michael Fellmann
Florian Johannsen
Birger Lantow
Judith Michael
Kristina Rosenthal
Robert Woitsch

Universität Wien
Universität Babes-Bolyai, Cluj-Napoca
Universität Rostock
Universität Regensburg
University of Rostock
Alpen-Adria-Universität Klagenfurt
Fernuniversität Hagen
BOC AG

Graphical App Designer

Konzept für die prozessbasierte Modellierung mobiler Cloud-Anwendungen

Prof. Dr. Gabriele Roth-Dietrich¹, Prof. Dr. Rainer Gerten² und André Schäfer³

Abstract: Der Graphical App Designer (GAD) unterstützt Unternehmen, die Cloud-Plattformen betreiben und versetzt Anwender⁴ und Berater, die über fachliche Prozessexpertise sowie über Prozessmodellierungskennntnisse, aber nicht über Programmiererfahrung verfügen, in die Lage, Cloud-Anwendungen für heterogene mobile Frontends zu modellieren und zu implementieren. Der GAD stellt die Modellierungsumgebung bereit, die sich an Prozessdarstellungen in BPMN orientiert, und stellt durch Prüfungen bereits während der Modellierung die Generierbarkeit und die Funktionsfähigkeit der Entwürfe sicher. Aus den Modellen kann ein plattformspezifischer GAD-Compiler später die ausführbaren Anwendungen auf Cloud-Seite generieren.

Keywords: Graphical App Designer, Prozessmodellierung, BPMN, Cloud-Plattform, mobile Endgeräte.

1 Modellierung und Generierung mobiler Anwendungen durch Business-User

Die Nutzung mobiler Cloud-Plattformen und die Umsetzung mobiler Geschäftsprozesse in einer App stellt Unternehmen vor große Herausforderungen, weil die Prozessexperten zwar die Abläufe bis ins Detail kennen, aber zu wenig technische Vorkenntnisse haben, um die Prozesse in der Programmierumgebung der Plattform zu implementieren. Der Graphical App Designer (GAD) soll Anwender aus Fachabteilungen und IT-Berater mit Business-Schwerpunkt in die Lage versetzen, ihre Prozesskenntnisse direkt in Form eines Ablaufmodells für die mobile Anwendung zu übertragen. Er übernimmt dazu den Ansatz grafischer Modellierungswerkzeuge, mobile Anwendungen visuell zu designen, sowie die Idee von App-Baukästen, Apps aus Bausteinen zusammensetzen. Der GAD stellt eine visuelle Programmierumgebung für die Modellierung und spätere Generierung

¹ Hochschule Mannheim, Fakultät für Informatik, Mannheimer Wirtschaftsinformatik-Institut, Paul-Wittsack-Straße 10, 68163 Mannheim, g.roth-dietrich@hs-mannheim.de

² Hochschule Mannheim, Fakultät für Informatik, Mannheimer Wirtschaftsinformatik-Institut, Paul-Wittsack-Straße 10, 68163 Mannheim, r.gerten@hs-mannheim.de

³ Movilizer GmbH, Konrad-Zuse-Ring 30, 68136 Mannheim, andre.schaefer@honeywell.com

⁴ Soweit im Folgenden bei der Bezeichnung von Personen die männliche Form verwendet wird, schließt diese Frauen in der gleichen Funktion ausdrücklich mit ein.

verteilter Cloud-Lösungen und ihrer Nutzung auf heterogenen mobilen Endgeräten bereit.

2 Konzept

Um die Zielgruppe optimal zu unterstützen, muss sich die Modellierung der mobilen Anwendung im GAD-Editor am bekannten BPMN-Standard für Geschäftsprozesse orientieren, von dem jedoch kontextabhängig nur Teilmengen der Notationselemente in Frage kommen. Die Modellierung soll die Nutzung von Vorlageprozessen ermöglichen, die versierte Berater oder Entwickler für typische Prozessszenarien vorfertigen und vielfach wiederverwenden. Diese Templates können Anwender als anpassbare Copy-and-Paste-Kopiervorlagen oder als referenzierte Modelle für die Ablaufmodellierung in den GAD-Editor importieren, um die Modellerstellung oder -aktualisierung zu beschleunigen, wiederverwendbare Modellbibliotheken vorzuhalten und auf Best Practice Know-how dezidierter Branchen und Anwendungsdomänen zurückzugreifen.

Die Modellierung darf sich nicht auf die für die Nutzer auf dem Mobile Device sichtbaren Prozessschritte beschränken, sondern muss Backend-Prozesse der Cloud-Plattform einschließen, etwa für die Synchronisation mit Cloud-Funktionalitäten oder für Datenmanagementaufgaben. Die Modellierungsphase sollen umfassende Prüfungen begleiten, so dass die Entwürfe zum Modellierungsende weitgehend fehlerfrei sind. Diese Verprobungen gewährleisten, dass nur lexikalisch erlaubte Modellierungselemente zum Einsatz kommen und dass Anwender diese auf syntaktisch korrekte Weise zusammenfügen. Gewünschte Semantikprüfungen kann der GAD über Regeln integrieren, etwa zum grundsätzlichen Prozessablauf, zur Reihenfolge der Templates (z.B. vorbereitende Aktivitäten stets vor durchführenden), zur Template-Schachtelung (beispielsweise Prozessabschluss immer nur mit Dokumentation der Tätigkeit) oder zu Mussbestandteilen für die Prozesse (etwa an Synchronisationspunkten mit der Cloud). Der durch den GAD erzeugten Zwischencode soll sich wiederum an Standards anlehnen und fertige Modelle in einem plattformunabhängigen XML-Format ablegen, aus denen ein auf die Cloud-Plattform zugeschnittener GAD-Compiler die fertigen Anwendungen generiert und über einen Konnektor an die Cloud-Plattform überträgt.

3 Umsetzung

Das Konzept des GAD wurde zusammen mit Studierenden des Bachelor-Studiengangs Unternehmens- und Wirtschaftsinformatik der Hochschule Mannheim im Rahmen eines Projektsemesters im Sommer 2017 beispielhaft für das Unternehmen Movilizer GmbH und ihre Cloud-Plattform Honeywell Movilizer Cloud for Field Operations umgesetzt. Die Movilizer Cloud verbindet Akteure im Bereich Field Operations miteinander und orchestriert Prozesse über Unternehmens- und Anwendungssystemgrenzen hinweg durch Movilizer Mobile Apps, die sich aus Movilizer Movelets, d.h. mobilen Prozessen

zusammensetzen. Um einen Überblick über die Breite der in der Praxis genutzten mobilen Prozesse zu bekommen, analysierten und modellierten die Studierenden vier Prozessszenarien in BPMN: Pickup and Delivery, Direct Store Delivery, Installed Base Management sowie Field Service Sales. Nach Zusammenführung der Szenarien in einen vereinheitlichten Gesamtprozess und Modularisierung mit Hilfe von wiederverwendbaren Subprozessen konnten etwa 20 GAD-Templates extrahiert werden.

Die Marktrecherche konzentrierte sich auf Metamodellierungstools, die sowohl Prozesse als auch die Besonderheiten von verteilten mobilen Cloud-Anwendungen modellieren können, sich dabei an BPMN anlehnen, aber Anpassungen in der Modellierungsnotation ermöglichen. Weitere K.O.-Kriterien waren importierbare Shapes und Notationselemente, XML-Unterstützung und Copyleft. Zu den sonstigen Kriterien gehörten mit absteigender Gewichtung User Experience, Art und Ort der Modelldatenspeicherung, Attributierungsmöglichkeiten für Screens und Felder, Dokumentation bzw. Support, bereits vorhandene Syntax-Checks sowie Lizenzkosten. Es wurden folgende Tools untersucht: Adonis, ADOxx, BIC Design Free WebEdition, BPMN.io, Camunda, ConceptBase, draw.io, [em], Fujaba, JastAdd, MetaCase, metaDepth, OpenPonk, RMT Framework und Signavio. Gemäß dem aufgestellten Kriterienkatalog und der festgelegten Kriteriengewichtung fiel die Entscheidung auf die Verwendung des Frameworks bpmn.js, das die Grundlage der beiden als gut bewerteten Werkzeuge bpmn.io und Camunda bildet.

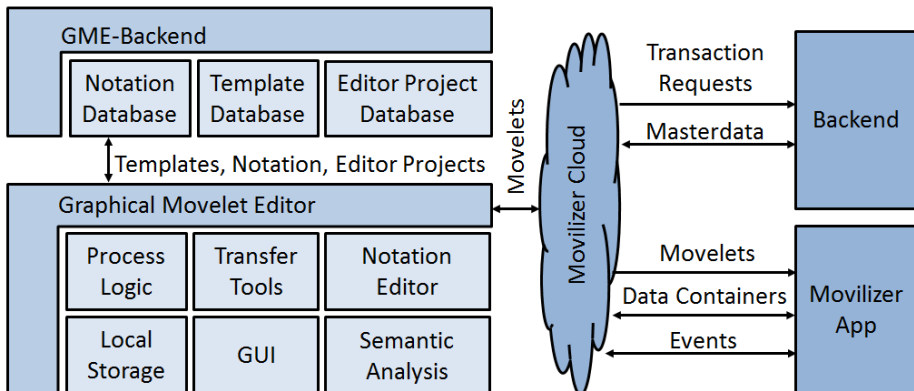


Abb. 1: Architektur des Graphical Movelet Editor

Um die Modellierungsoberfläche so einfach wie möglich zu halten, nutzt der Proof-of-Concept des GAD nur wenige BPMN-Notationselemente, darunter ein Start- und ein Ende-Ereignis sowie den Sequenzfluss. Das Hauptelement für die Ablaufreihenfolge ist der Screen, dargestellt als BPMN-Aktivität, der entweder eine für Nutzer sichtbare Oberfläche oder einen verdeckt ablaufenden Prozessschritt abbildet. Entsprechend der Task-Typen einer BPMN-Aktivität kann auch der Screen verschieden ausgestaltet sein, z.B. als Text Item Screen, der ein Eingabefeld bereitstellt, mit Kalenderdarstellung für die Datumsselektion oder als Unterschriftenfeld.

Die Templates für die vier Prozessszenarien sind baumartig strukturiert und enthalten Regeln für Reihenfolge, Schachtelung und Mussbestandteile. Der Movelet-Container auf oberster Ebene lädt zu Beginn benötigte Daten aus der Movilizer Cloud und schreibt veränderte Daten am Ende zurück. Da alle Prozessszenarien Touren beinhalten, ordnen sich darunter sich Sub-Templates für das Vorbereiten, Durchführen und Abschließen einer Tour an. Je Sub-Template stehen verschiedene Ausgestaltungsvarianten zur Verfügung, etwa die Tour-Durchführung mit Auslieferungen (Delivery), Abholvorgängen (Pick-Up) oder Vertriebsbesuchen (Sales Visit).

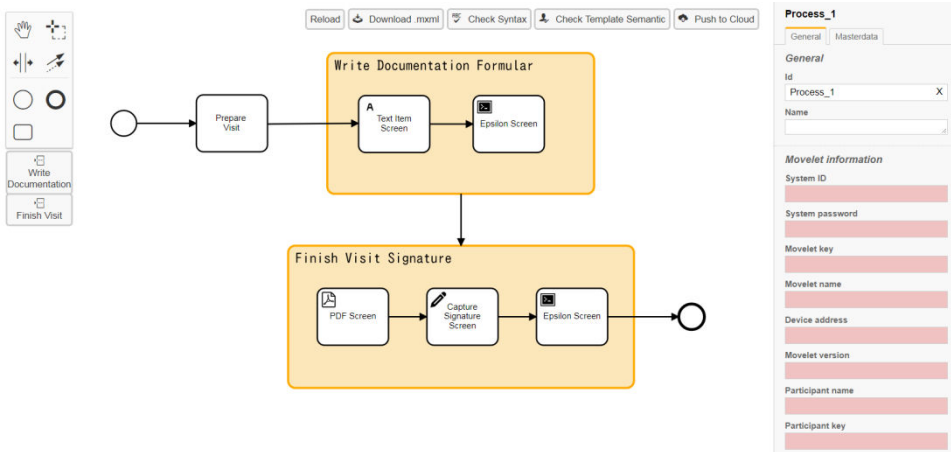


Abb. 2: Modellierung im Graphical Movelet Editor

Für die Implementierung wurde der Tool-Name auf das Unternehmen Movilizer GmbH angepasst. Im Backend des Graphical Movelet Editor (GME) liegen die Notationselemente, der Template-Vorrat sowie die Modellierungsprojekte (siehe Abb. 1). Das GME-Frontend besteht aus GUI-Komponenten, Werkzeugen für die Prozesslogik und Möglichkeit zur lokalen Speicherung der Modelle. Im GME erstellen Anwender neue mobile Anwendungen oder importieren existierende Movelets aus der Movilizer Cloud, nutzen Templates, versorgen die Screens mit Daten und prüfen Syntax und Semantik der Modelle (siehe Abb. 2). Nach Modellierungsende generiert der GME-Compiler lokal gespeicherten Movilizer-spezifischen MXML-Zwischencode und übermittelt bei eingerichteter Verbindung zur Cloud-Plattform die fertigen Modelle an die Movilizer Cloud.

Der Editor des Graphical App Designers ist in der Stand-Alone-Version nutzbar, um mobile Prozesse BPMN-nah zu modellieren und mit Templates plattformunabhängig zu orchestrieren. Er kann auf Anfrage zur Verfügung gestellt werden, um ihn zu erweitern oder um den Compiler für die Integration in weitere Cloud-Plattformen auszubauen.

Interactive information zoom on Component Fault Trees

Santiago Velasco¹, Jan Reich² and Maxime Tchangou³

Abstract: The visualization approach for Component Fault Trees (CFTs) realized in this work was implemented as an extension of the safeTbox modeling tool (safeTbox.iese.fraunhofer.de). Its goal is to enhance the understandability of compositional and hierarchical models while facilitating reviewing purposes. Safety Analysts makes use of CFTs to perform fault analyses at system level. However, such analyses are hindered by the traditional approach of hiding the realization information of components behind the specification views. The approach presented here overcomes this problem thanks to an information-based zoom. Through it, it is possible to gradually present at the specification level information extracted out of the internal realization views.

Keywords: Component Fault Trees, Information Zoom, Visual Zoom, Black-box, specification view, realization view, hierarchical abstraction.

Fault Tree Analysis (FTA) is a deductive technique for the identification of faults in a system. It is very well-known in the safety-critical domain of embedded systems, since its use is demanded or recommended by several standards (e.g. IEC 61508 or ISO 26262) depending on the criticality of the system to be designed. As its name indicates, this technique leads to the construction of a tree of undesired events (see Fig1 left): on its top, a top event (event under analysis) will be found, on the leaf, the so called “Basic Events” (being the root causes), and in between intermediate events (depict the logical combinations of basic or other intermediate events).

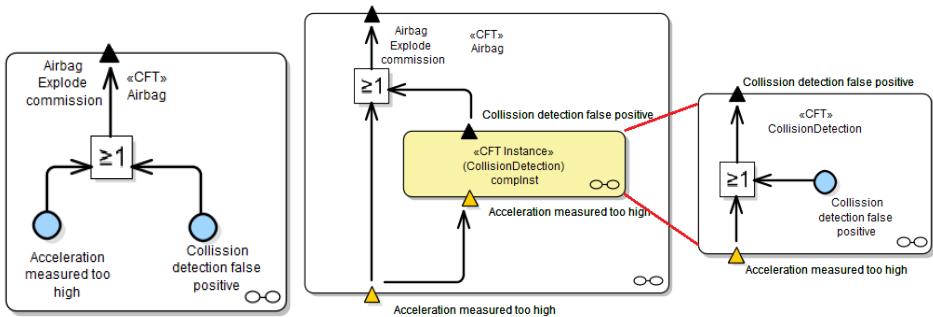


Fig. 1: Fault Tree (Left), Component Fault Tree (Right)

¹ Fraunhofer IESE, Embedded Systems Quality, 67663 Kaiserslautern, santiago.velasco@iese.fraunhofer.de

² Fraunhofer IESE, Embedded Systems Quality, 67663 Kaiserslautern, jan.reich@iese.fraunhofer.de

³ TU-Kaiserslautern, 67663 Kaiserslautern, maximetchangou2011@yahoo.fr

Kaiser and Liggesmeyer [BPO00] extended the FTA approach with modularization concepts to encapsulate parts of a typically monolithic representation into blocks (see Fig1 right). The resulting Component Fault Tree (CFT) concept introduces new modeling elements to support a compositional approach, namely CFT instances and input events. The former allow to reuse existing modularized fault trees within other fault trees, and the later (Yellow triangles in the figure) represent failure modes propagating into the modularized fault trees.

Domis [D05] enhanced the CFT approach to get a formal traceability to system and architectural design models by introducing traces between CFT artifacts and architecture artifacts. For this reason this approach has the advantage that fault models are easier to maintain with respect to changes in system design models. In Adler [ASH17], the integrated approach has been further adapted to support cause analyses in models that describe control schemes. However, CFTs modeled following these approaches have the disadvantage that they have to adopt the hierarchical abstraction of the architecture models they have been integrated to.

In the embedded systems domain, hierarchical abstraction is typically used as means to handle complexity. This is frequently used in the modeling of large systems when building component networks in which it is easy to distinguish between the black-box and white-box views. Meaning the specification and the realization by means of composition of other components. A black-box representation can be very useful to get the big picture in terms of composition, but it can be difficult to work with, when model details are required for reasoning. This is the case in the area of fault analysis for instance, where it is quite important to have a better understanding of the behavior of the system parts. This will be required to identify faults that might not only occur as a pure fault of a component but due to their intricate interaction. Traditionally, compositional models have been supported by most modeling tools by enabling the navigation from the black-box (i.e. specification) to the white-box (i.e. realization) representation of a component and by supporting different visual zoom levels. The navigation from the white to the black representation is generally not well-supported, since the navigation is not unique (e.g. in the case of reuse). For this reason an analyst trying to review the system model has a hard time, since he has to switch continuously between the specification and realization views. Only in this way he will be able to get a big picture of the system, while being able to understand the details of each component at the same time. Such context switching is quite demanding, since it assumes that the analyst is able to retain the realization view of one or more components in mind while analyzing others.

In order to facilitate the review process of complex models, particularly in the context of Component Fault Tree analysis, we have implemented a prototype of a visualization feature as part of the safeTbox™ modeling tool (safetbox.iese.fraunhofer.de). It allows performing an information zoom, in which details of the realization view of a component are brought up to the level of the specification representation. The goal of this feature is to eliminate the need of switching context while obtaining more details within a hierarchical and compositional model. With the start of the visualization feature, the

model will be displayed in the visualization feature as displayed in the modeling tool (see Fig 2 - Entry level). From this point on an analyst has the possibility to control the amount of information being displayed by means of a visual and information zoom. The deeper the selected information level the more details of the realization views of the instances will be displayed. This will occur recursively until no refinement is possible, i.e. the user sees the full realization of the deep most component in the hierarchy.

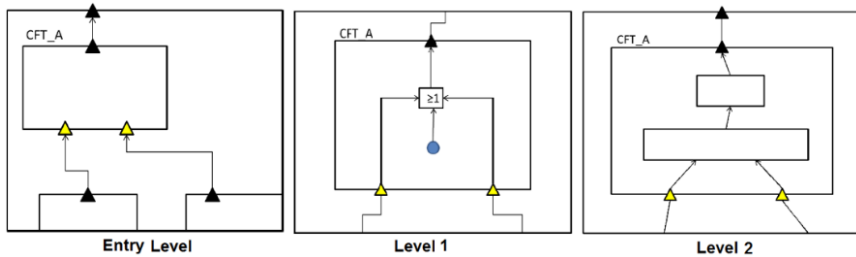


Fig. 2: Solution concept

One of the major differences between the regular visualization approach and ours is that in the black/white box alternative the modeler has a big jump with respect to the amount of information being displayed whenever he switches from the specification (only interfaces being displayed) to the realization (i.e. all information). In our approach, we have introduced several intermediate levels of details which are computed out of the model information and which smoothly increments the amount of information being displayed (see level 1 and 2 in figure 2). From the entry point, i.e. the typical black-box view, the user can zoom into Level 1, which does not present the realization of CFT_A as it would be expected, but a reduced fault tree which has been computed out of the results of the Minimal Cut Sets (MCS) analysis. The MCS Analysis is used to compute the minimal set of events required to trigger the occurrence of a top event. This view abstracts from the underlying fault model with a small set of modeling elements, but is still showing the most important information, namely the faults of all internal sub components of CFT_A. If the user needs to know where these faults originate he can zoom in further into zoom level 2, where he will start seeing an abstract version of the realization of the component. In this level, the focus rests on depicting the fault relationship among sub components. The information with respect to the instance interfaces is removed by abstracting the fault relationships between each pair of sub components. In summary, the feature will display the same information as in the entry level, but for the sub components of CFT_A.

As shown in the figures 3, while zooming in purely through the information zoom, the feature will add more and more information to the model. For this reason, a visual pan and zoom functionality is still required to allow the user focus temporary in certain aspects.

In summary, our visualization approach provides an alternative way in displaying information for component-oriented fault trees. It shall enhance the understandability of hierarchical models while facilitating reviewing purposes by avoiding context switching. In the near future this feature will be extended with other visualization functionalities to support other analysis activities, e.g. Common Cause Failures (CCF) and Boolean cycle visualization in CFTs.

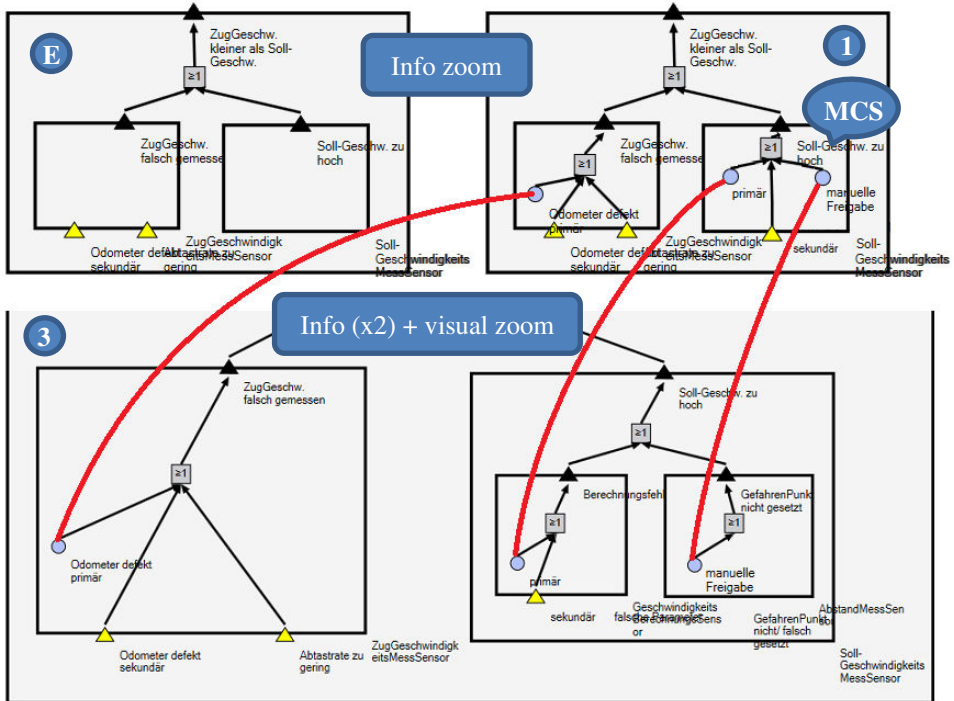


Fig. 3: Visualization feature levels example

- [ASH17] Adler,R., Schneider D., Höfig K., "Evolution of fault trees from hardware safety analysis to integrated analysis of software-intensive control systems", In proceedings of ESREL 2017 (Portoroz, Slovenia, 18-22 June, 2017)
- [D05] Domis, D.: "Integrating fault Tree Analysis and Component-Oriented Model-Based Design of Embedded Systems", PHD Thesis, Technische Universitaet Kaiserslautern, Fachbereich Informatik, 2005
- [BPO03] Bernhard Kaiser , Peter Liggesmeyer , Oliver Mäckel, A new component concept for fault trees, Proceedings of the 8th Australian workshop on Safety critical systems and software, pp. 37-46, October 01, Canberra, Australia, 2003

Eine musterbasierte Kontrollflussemanik zur interaktiven Simulation von Geschäftsprozessmodellen

Andreas Drescher¹

Abstract: Für die Modellierung von Geschäftsprozessen existieren eine Vielzahl von Modellierungssprachen, die allerdings häufig eine unpräzise Kontrollflussemanik besitzen. Dadurch werden die Analysefähigkeit sowie auch die Austauschbarkeit der Modelle eingeschränkt. In diesem Beitrag wird eine leichtgewichtige Methode am Beispiel der Business Process Model and Notation (BPMN) zur präzisen Beschreibung der Kontrollflussemanik einer beliebigen graphischen Modellierungssprache als Microsoft Visio (MS Visio) Add-In vorgestellt. Die Kontrollflussemanik wird durch eine fallbasierte Zuweisung zu den Kontrollflussmustern der Workflow Pattern Initiative definiert. Die entstehende Analysefähigkeit wird durch eine interaktive Simulation demonstriert.

Keywords: Modellierungssprachen, Kontrollflussmuster, Kontrollflussemanik, Simulation

1 Einleitung

Die Wertschöpfung eines Unternehmens ist von einer effizienten und effektiven Ausführung der Geschäftsprozesse abhängig. Das Geschäftsprozessmanagement kann für die strukturierte Herangehensweise von der Erfassung bis hin zur Überwachung der Geschäftsprozesse angewandt werden. In diesem Zusammenhang werden die Geschäftsprozesse mit (1) domänenspezifischen Sprachen (z. B. PICTURE), (2) Standardmodellierungssprachen (z. B. BPMN) oder (3) Standardmodellierungssprachen mit domänenspezifischen Erweiterungen, z. B. um Nachhaltigkeits- oder auch Gesundheitsaspekte (z. B. BPMN4CP), modelliert [Br15]. Insbesondere die Modellierungssprachen aus der Kategorie (1) und (3) besitzen typischerweise eine unpräzise Kontrollflussemanik sowie eine fehlende Werkzeugunterstützung, wodurch deren Verbreitungsgrad reduziert ist. Dies führt häufig dazu, dass MS Visio zur Dokumentation eingesetzt wird [Ko15]. Jedoch unterstützen insbesondere MS Visio und vorhandene Erweiterungen kaum die Analysefähigkeit und Austauschbarkeit der Modelle für (1) und (3). Daher wird in diesem Beitrag eine leichtgewichtige Methode zur präzisen Beschreibung der Kontrollflussemanik beliebiger graphischer Modellierungssprachen als MS Visio Add-In vorgestellt. Auf dieser Grundlage wird eine IT-gestützte Analyse und XML-basierte Speicherung der Modelle sowie deren Kontrollflussemanik möglich. Im Vergleich zu anderen Software-Werkzeugen wird die Sprache nicht durch das Werkzeug vorgegeben, sondern kann durch den Modellierer selbstständig definiert und

¹ Karlsruher Institut für Technologie (KIT), Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB), Andreas.Drescher@kit.edu

entsprechend die Kontrollflussesemantik durch die Muster präzisiert werden, um z. B. kontrollflussabhängige Analysen zu ermöglichen.

2 Werkzeugpräsentation

Das Software-Werkzeug wurde als MS Visio Add-In mit Visual Studio 2017 entwickelt und ist unter <http://wiwi.link/Visio4Analyse> zum Download verfügbar. Es wurde unterstützend das .NET Framework 4.6.1, die Office Developer Tools für Visual Studio 2017 und Visual C# 2017 verwendet. Im Folgenden wird zunächst der Prozess zur Erstellung einer musterbasierten Kontrollflussesemantik beschrieben. Zur Beschreibung der Methode wurde BPMN aus (2) ausgewählt, da typischerweise die Sprachen aus (1) und die Erweiterungen aus (3) keine komplexere Kontrollflussesemantik besitzen, wie z. B. eine BPMN-Aufgabe oder ein komplexes Gateway. Nachfolgend können die Modelle analysiert oder auch mit anderen Software-Werkzeugen austauscht werden. Die entstehende Analysefähigkeit wird durch eine interaktive Simulation in BPMN demonstriert.

2.1 Erstellung einer musterbasierten Kontrollflussesemantik

Für die präzise Beschreibung der Kontrollflussesemantik einer Sprache werden die Kontrollflussmuster der Workflow Pattern Initiative (WPI) eingesetzt [Ru06]. Die Kontrollflussmuster beschreiben das wiederkehrende Verhalten der logisch-kausalen Abhängigkeiten des Kontrollflusses. Daher können sie zur leichtgewichtigen Präzisierung der Kontrollflussesemantik verwendet werden. Derzeit unterstützt das entwickelte Add-In alle Basiskontrollflussmuster und ausgewählte erweiterte Verzweigungs- und Zusammenführungsmuster, d. h. die Kontrollflussmuster: Sequenz, Parallele Aufspaltung, Synchronisation, Exklusive Auswahl, Mehrfachauswahl, Strukturierte synchronisierte Zusammenführung, Mehrfachzusammenführung, Strukturierter, Blockierender und Abbrechender Diskriminator sowie Strukturierte, Partielle und Blockierende Zusammenführung.

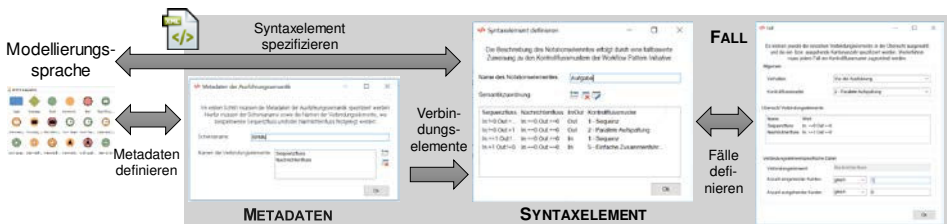


Abb. 1: Musterbasierte Kontrollflussesemantik

Der Prozess zur Erstellung einer präzisen Kontrollflussesemantik wird in Abb. 1 skizziert und die erforderlichen Daten werden mit Dialogfenstern spezifiziert. Im ersten Schritt werden die METADATEN definiert, d. h. der Name der Kontrollflussesemantik sowie die

kontrollflussabhängigen Verbindungselemente, die den Kontrollfluss in MS Visio graphisch abbilden. Beispielsweise ist die Kontrollflusssemantik in BPMN vom Sequenz- und Nachrichtenfluss sowie von der Datenassoziation abhängig. Nachfolgend kann die Kontrollflusssemantik der Syntaxelemente durch die Auswahl des Notationselementes in MS Visio mit verschiedenen Fällen spezifiziert werden. Ein FALL definiert das Verhalten «vor der Ausführung» oder «nach der Ausführung» und wird einem Kontrollflussmuster abhängig von der Anzahl der ein- bzw. ausgehenden Verbindungselemente zugeordnet. Ein Fall für eine BPMN-Aufgabe kann z. B. wie folgt definiert werden: Bei mehr als einem ausgehenden Sequenzfluss (>1), beliebig vielen eingehenden Sequenzflüssen (>0), keinem / keiner ein- bzw. ausgehenden Nachrichtenfluss ($==0$) / Datenassoziation ($==0$) und dem Verhalten «nach der Ausführung» kann die Aufgabe dem Muster Parallele Aufspaltung zugeordnet werden. Für die Spezifikation der erforderlichen Anzahl der Verbindungselemente werden die Operatoren größer ($>$), größer gleich ($>=$), gleich ($==$), ungleich ($!=$), kleiner gleich ($<=$) und kleiner ($<$) angeboten. Die Kontrollflusssemantik der Modellierungssprache ist präzise beschrieben, wenn alle möglichen Fälle definiert wurden. Die Rahmenbedingungen der leichtgewichtigen Methode werden durch die Notation sowie der Kontrollflussmuster bestimmt. Die Spezifikation der Syntaxelemente wird mit Hilfe der Notation realisiert, so dass ein Notationselement mit den entsprechenden graphischen Verbindungselementen für die präzise Beschreibung existieren muss. Daher ist die Beschreibung der Kontrollflusssemantik eines Link-Ereignisses oder Ad-Hoc Teilprozesses problematisch. Darüber hinaus ist eine Verschachtelung mehrere Kontrollflussmuster bei komplexeren Fällen (z. B. eine BPMN-Aufgabe mit angehefteten Ereignissen) erforderlich und die Kontrollflusssemantik muss durch die Kontrollflussmuster abbildbar sein.

2.2 Interaktive Simulation von Geschäftsprozessen

Die präzise Kontrollflusssemantik ermöglicht den Einsatz von kontrollflussabhängigen Analyseverfahren, wie z. B. die interaktive Simulation. In der Praxis ist es jedoch häufig schwierig die genauen Eingabedaten für ein Simulationsmodell zu erhalten. Daher steht insbesondere bei der interaktiven Simulation die spielerische Auseinandersetzung mit den möglichen Verhaltensweisen und der damit einhergehenden Visualisierung zur Förderung der Verständlichkeit von Szenarien im Vordergrund. Alternative Analyseverfahren sind z. B. die Überprüfung der inhaltlichen Korrektheit oder die Testfallgenerierung.

Für die interaktive Simulation des Prozessmodells erhält zunächst jedes Element auf der Zeichenfläche (engl. Shape) in MS Visio einen Zustand. Mögliche Zustände sind (1) «unmarkiert», (2) «markiert», (3) «markiert, nicht ausführbar» oder (4) «hervorgehoben». Ein Shape besitzt den Zustand 1, wenn es aufgrund des aktuellen Zustandes der Geschäftsprozessinstanz nicht ausgeführt werden kann. Es besitzt keine spezielle visuelle Kennzeichnung. Den Zustand 2 hat ein Shape, wenn es aufgrund des zutreffenden Falles «nach der Ausführung» des ausgeführten Shapes, ausgeführt werden kann. Zusätzlich muss für das auszuführende Shape ein «vor der Ausführung» Fall zutreffen, sodass das Shape visuell mit Grün hervorgehoben wird. Sofern der zutreffende Fall «vor der

Ausführung» noch nicht erfüllt ist, der durch das Kontrollflussmuster bestimmt wird, erhält das Shape den Zustand 3. Es wird visuell mit Rot hervorgehoben. Wenn beispielsweise im Dialog in Abb. 2 die Shapes Aufgabe B und D ausgewählt wurden, so muss der Kontrollfluss bei Gateway 2 warten, bis die beiden Aufgaben C und E ausgeführt wurden. Sofern erst eine der beiden Aufgaben ausgeführt wurde, wird das Gateway 2 mit dem Zustand 3 gekennzeichnet. Sobald beide Aufgaben ausgeführt wurden, erhält das Gateway 2 den Zustand 2. Den Zustand 4 besitzt ein Shape, wenn es den Zustand 1 erhalten würde, zugleich aber optionale Ausführungsmöglichkeiten existieren. Dementsprechend kann der Zustand 4 nur in Verbindung mit den implementierten Kontrollflussmustern Exklusive Auswahl und Mehrfachauswahl (vgl. Abb. 2) auftreten. Nach der Auswahl eines oder mehrerer Shapes erhalten die ausgewählten Shapes den Zustand 2 und die anderen Shapes den Zustand 1. Der Zustand 4 wird mit Gelb hervorgehoben.

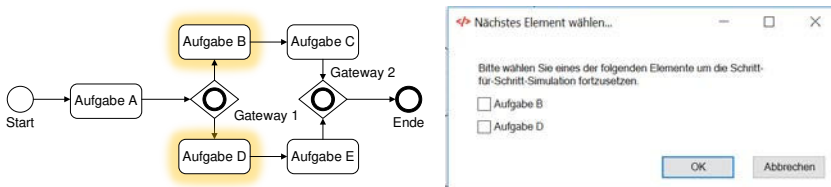


Abb. 2: BPMN Schritt-für-Schritt-Simulation – Kontrollflussmuster Mehrfachauswahl

Für den Start der Simulation müssen die Startelemente ermittelt werden. Startelemente sind dadurch gekennzeichnet, dass sie keine eingehenden Verbindungselemente besitzen, in keinem anderen Shape (z. B. Teilprozess) enthalten und mit keinem anderen Shape verknüpft (z. B. angeheftete Ereignisse in BPMN) sind. Sofern mehrere Startelemente existieren, werden diese mit dem Zustand 4 markiert. Der Folgezustand einer Geschäftsprozessinstanz wird durch das Klicken auf ein Shape mit dem Zustand 2 oder 4 ermittelt.

3 Ausblick

Als weiterführende Schritte sollen Anwender-Studien zur Bewertung der Gebrauchstauglichkeit der musterbasierten Kontrollflussesemantik und der interaktiven Simulation durchgeführt werden. Die Methode und das Add-In sollen dann schrittweise mit dem Feedback verbessert werden. Die nächste Entwicklungsstufe umfasst unter anderem die Simulation mehrerer Fälle inklusive quantitativer Kennzahlen, sodass Aussagen über die Mindstdauer und die Ressourcen- oder Kapazitätsauslastung möglich sind. Ebenfalls sollen die weiteren 31 Kontrollflussmuster der WPI in das Add-In integriert werden.

4 Literaturverzeichnis

- [Br15] Braun, R. et al.: Extending a Business Process Modeling Language for Domain-Specific Adaptation in Healthcare. In (Thomas, O.; Teuteberg, F. Hrsg.): Smart Enterprise Engineering, 2015; S. 468–481.
- [Ko15] Kocbek, M. et al.: Business Process Model and Notation. The Current State of Affairs. In Computer Science and Information Systems, 2015, 12; S. 509–539.
- [Ru06] Russell, N.; Hofstede, A.H.M. t.; Aalst, W. M. P. v. d.; Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. BPMcenter.org, 2006.

Development of a prototype for Smart Glasses-based process modelling

Sven Jannaber¹, Benedikt Zobel¹, Lisa Berkemeier¹ and Oliver Thomas¹

Abstract: The integration of mobile technology is considered a major challenge for the BPM domain. Wearable devices such as smart glasses have already been successfully applied in high-mobility fields such as technical services. However, the utilization of smart glasses to document and model processes still remains on a conceptual level and has not yet been instantiated. This paper demonstrates a prototype for process modelling on smart glasses. It is shown how glasses-specific functionality, e.g. voice recognition, can be incorporated into a modelling environment that facilitates the run-time modelling of processes, even for modelling novices.

Keywords: BPM, process modelling, smart glasses, run-time modelling

1 Towards mobile process modelling

Business process management (BPM) is considered one of the top five management topics for today's enterprises [Lu12]. An integral part of every BPM endeavour is the modelling of business processes [BNT10]. Studies show that the mere documentation of operational activities within an enterprise is able to increase the organizational performance [Me05]. However, despite technological advances, process modelling still relies on traditional modelling methods using desktop computers and complex software suites. This is especially detrimental for high mobility domains such as technical customer service (TCS), which puts heavy emphasis on highly mobile service technicians that perform on-site services such as maintenance and repair [Ma17]. To cope with these requirements, smart glasses have emerged as a suitable tool to support service technicians during service execution [Ni16]. However, while smart glasses have proven to be beneficial in terms of additional information provision [Ni17], challenges arise when implementing process aware information systems on such devices. Major problems particularly address the integration and visualization of processes, since smart glasses come with specific hardware-related restrictions regarding visualization that are diametric to current forms of process models. Consequently, the research question is stated as follows: "*How can business processes be captured and documented in a way that allows for utilization within a process aware information system on Smart Glasses?*" Research on this matter has already conceptualized the use of smart glasses themselves as a means to capture processes [Me17]. Hence, the paper at hand introduces a prototype that addresses this issue by demonstrating how smart glasses functionality can be utilized for process modelling, even in run-time to process execution and on-site.

¹ Osnabrück University, IMWI, Katharinenstraße 3, 49074 Osnabrück, [firstname.lastname]@uos.de

2 A prototype for Smart Glasses-based process modelling

The prototype for process modelling with smart glasses is represented in an exemplary case from the TCS domain in Figure 1. The figure shows part of a process for the replacement of a coil. The technician as end-user of the prototype is recording the process using an intuitive user interface, which does not require modelling experiences. Each modelling action done via the prototype interface results in the ongoing construction of a process model in the background. In the presented case, the EPC language has been chosen for demonstration purposes. The primary form of interaction with the prototype is voice command, as the user has free hands to fulfill his task. The voice control is activated if the icon displaying a mouth (lower right) is green. If there is no connection and the symbol is red, other interaction interfaces like control buttons can be used. When documenting a process, the technician needs to insert and label a function via voice command, as done with the function “Remove Cover” in the example shown in Figure 1. Subsequently, the system requires a feedback to confirm the description. Afterwards, the system awaits the next user interaction, which may be a new function or e.g. a resource. Functions are followed by events that are automatically generated on the basis of the function label. The resulting model can for example be used for the process guidance with smart glasses: The functions will appear as instructions in the glasses’ display, the events can be utilized as a confirmation step. Especially untrained technicians benefit from this documentation, as they receive instructions from their viewing perspective, which are easy to follow and provided hands-free.

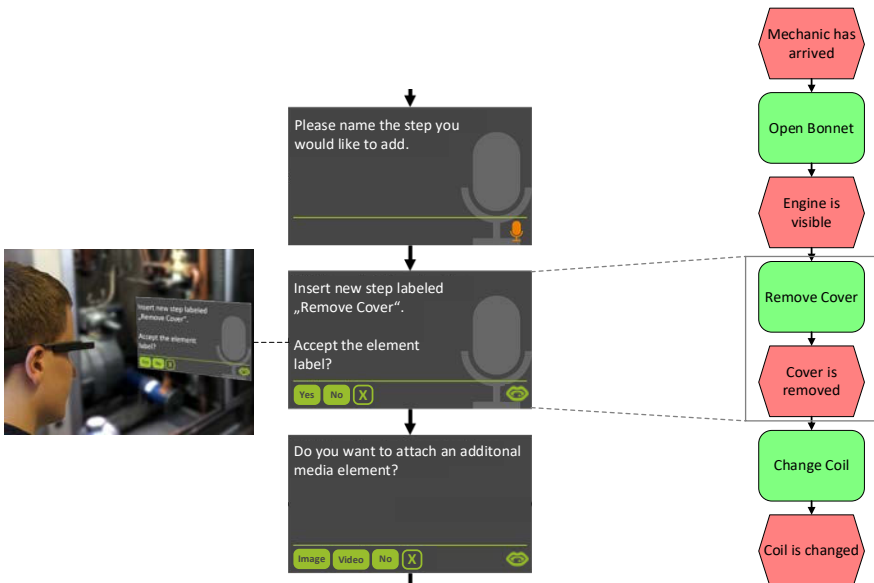


Figure 1: Adding and naming of a new process step

Besides the text-based instructions, smart glasses offer functionalities such as voice or video respectively picture recording. Thereby, the documented process is enhanced by further procedural knowledge as displayed in Figure 2. The technician takes a picture of a service object such as a coil in our example. Therefore, the voice command “record” initiates the camera. After 5 seconds, the picture is taken and displayed to the user afterwards. The user can decide whether to approve the picture or not. If the picture is disapproved, the system returns to the last screen to take another shot. If the picture is confirmed, the technician can proceed with the next step. The recordings are saved as information objects and are assigned to functions as resources. The documentation process can be stopped at any time if the user triggers the button marked with an “x” on every screen.

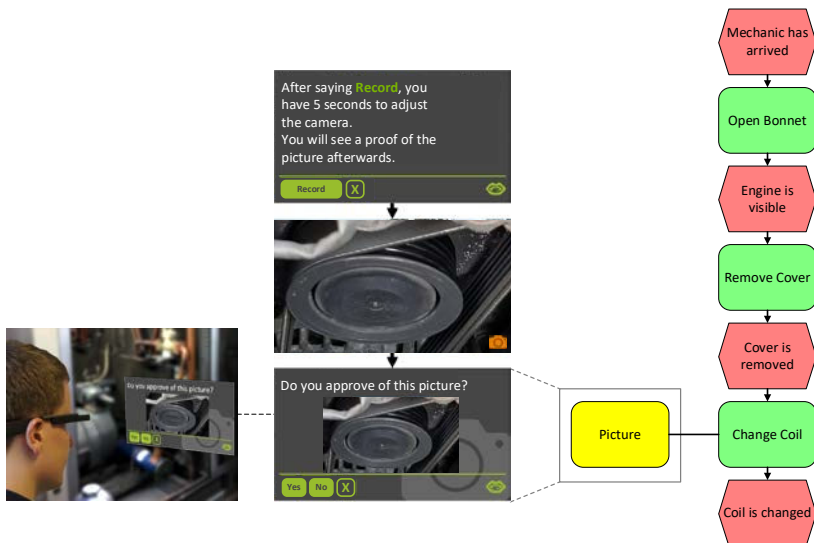


Figure 2: Adding of a media element

As service processes in TCS typically yield a high degree of variability, the documented processes can be adapted and extended at any time. Via voice command, the technician can add or delete new steps or create decision trees by inserting a split. The system transforms these splits into logical operators from the EPC language, such as an XOR to indicate alternative process paths. Additionally, the overall system architecture provides for a manual quality check and enhancement of the resulting models as shown in Figure 3. The architecture encompasses both modelling via smart glasses (focus of this contribution) as well as a back office system that offers traditional modelling software functionality. As demonstrated in Figure 1 and 2, the modelling concept provides that high mobility workers, such as service technicians within the TCS domain, use the prototype to document operational activities in form of process models. However, the expressiveness of such models is limited, since hardware-specific restrictions (e.g. small display size) have to be considered.

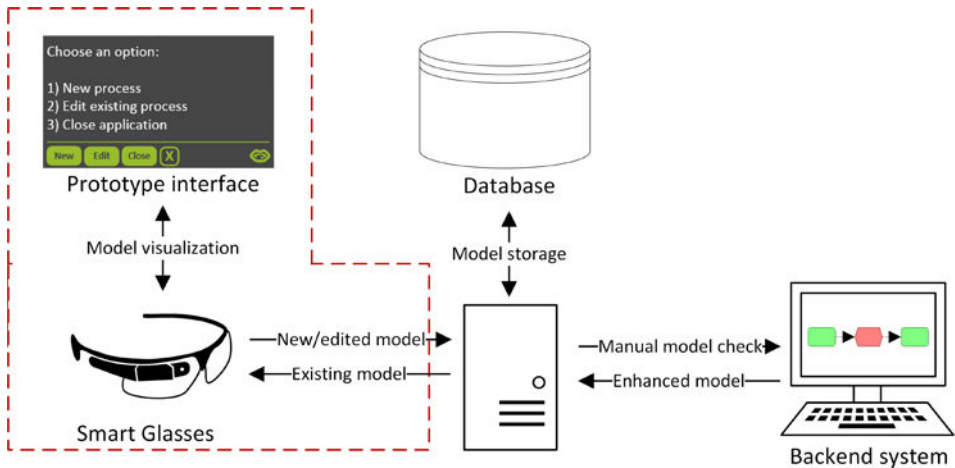


Figure 3: System architecture of the Smart Glasses modelling prototype

Additionally, technicians as end-users are often modelling novices, which may lead to modelling errors. Hence, the backend modelling system ensures that each modelled process using the prototype is manually checked from domain experts. Subsequent to model checking, the models are distributed to a central server, which stores them in a database. Afterwards the models can be used for guiding inexperienced technicians through service processes by visualizing them as instructions on smart glasses.

Bibliography

- [BNT10] Becker, J., Niehaves, B., Thome, I.: How Many Methods Do We Need ? – A Multiple Case Study Exploration into the Use of Business Process Modeling Methods in Industry. In: AMCIS 2010 Proceedings, 2010.
- [Lu12] Luftman, J., Zadeh, H.S., Derksen, B., Santana, M., Rigoni, E.H., Huang, Z. (David): Key information technology and management issues 2011-2012: an international study. *J. Inf. Technol.* 27(3), pp. 198–212 (2012).
- [Ni16] Niemöller, C.; Metzger, D.; Fellmann, M.; Thomas, O.: Shaping the Future of Mobile Service Support Systems – Ex-Ante Evaluation of Smart Glasses in Technical Customer Service Processes. In: *Informatik 2016*. Klagenfurt, 2016
- [Ni17] Niemöller C., Metzger D., Thomas, O. "Design and Evaluation of a Smart-Glasses-based Service Support System", In: Leimeister JM, Brenner W. (eds): *Proceedings der 13. Internationalen Tagung Wirtschaftsinformatik (WI 2017)*, St. Gallen, pp. 106-120 (2017)
- [Ma17] Matijacic, M.; Fellmann, M.; Kammler, F.; Özcan, D.; Nüttgens, M.; Thomas, O.: Elicitation and Consolidation of Requirements for Mobile Technical Customer Services Support Systems – A Multi-Method Approach. In: *Proceedings of the 34th International Conference on Information Systems (ICIS 2013)*, pp. 1-16, 2013.
- [Me05] Melenovsky, M.J.: *Business process management's success hinges on business-led initiatives*. *Gart. Res. Stamford, CT.* 1–6 (2005).
- [Me17] Metzger, D.; Niemöller, C.; Berkemeier, L.; Brenning, L.; Thomas, O.: Vom Techniker zum Modellierer - Konzeption und Entwicklung eines Smart Glasses Systems zur Laufzeitmodellierung von Dienstleistungsprozessen. In: *Smart Service Engineering*, pp. 193-213, 2017.

A web-based modeling tool for studying the learning of conceptual modeling

Benjamin Ternes¹, Stefan Strecker¹

Abstract: How do we learn conceptual modeling? What are common learning difficulties? Which tool support assists learners in what respect? We report on the design and development of a web-based modeling tool aimed at studying the learning of conceptual modeling by observing learner interactions with graphical model editors. Learner interactions with graphical model editors are tracked, recorded and analyzed at the individual and aggregate learner levels with support for graphically reproducing the learner-editor interactions over time. In this short paper, we report on the current state of the tool development.

Keywords: Learning of conceptual modeling; Web-based modeling tool; Prototyping.

1 Introduction

Viewed as an activity, conceptual modeling involves an intricate array of cognitive processes and performed actions including abstracting, conceptualizing, associating, interpreting, visualizing, and, in group settings, communicating, discussing and agreeing. The learning of conceptual modeling, hence, constitutes a complex and challenging task for learners not only at the introductory level. Designing modeling tool support for learners presupposes a differentiated understanding of learning processes, common learning difficulties, and learning barriers. However, surprisingly little is currently known about the learning of conceptual modeling [SSD14, pp. 488]. Research on learning conceptual modeling has only recently seen increasing interest with contributions, e.g., focusing on business process modeling (e.g. [Pi12]), on cognitive aspects (e.g. [TVC17]), and on learning outcomes (e.g. [SDS16]).

In an attempt to contribute to filling this gap, we embarked on a long-term research program with which we aim to better understand how modelers learn a modeling language resp. modeling method and how tool support assists learners in what respect. As part of that research program, we develop a web-based modeling tool aimed at identifying patterns by recording learner interactions with graphical model editors, e.g., patterns of learning difficulties. The current running prototype explores design and implementation strategies for tracking learner-editor interactions, handling and persistency of tracking data and tracking data analytics.

¹University of Hagen, Enterprise Modelling Research Group, Universitätsstr. 41, 58084 Hagen, Germany

2 Tool presentation

Two essential requirements drive the software development, ease-of-use (and installation, configuration) as well as platform independence to the greatest possible extent (based on our primary application scenario of a distance learning context with cohorts of up to 1,500 learners). Hence, in an early design decision, we opted for a web application with a JavaScript-driven browser frontend and an Java EE (Enterprise Edition)-based backend (see Fig. 1). Thus, the tool can be used with popular web browsers and operating systems.

The principle tool operation is as follows: The core component of the web frontend implements the generic handling of nodes and edges on the drawing canvas including higher level features such as creating, reading, and updating entire diagrams. Appropriate resources are dynamically loaded and added to the page as needed, usually in response to user interactions. The created conceptual models are internally represented and stored in the JavaScript Object Notation (JSON) format. Stencil sets are processed by the frontend and provide explicit typing, connection rules, visual appearance, and other features that differentiate a model editor from generic vector-oriented drawing tools.

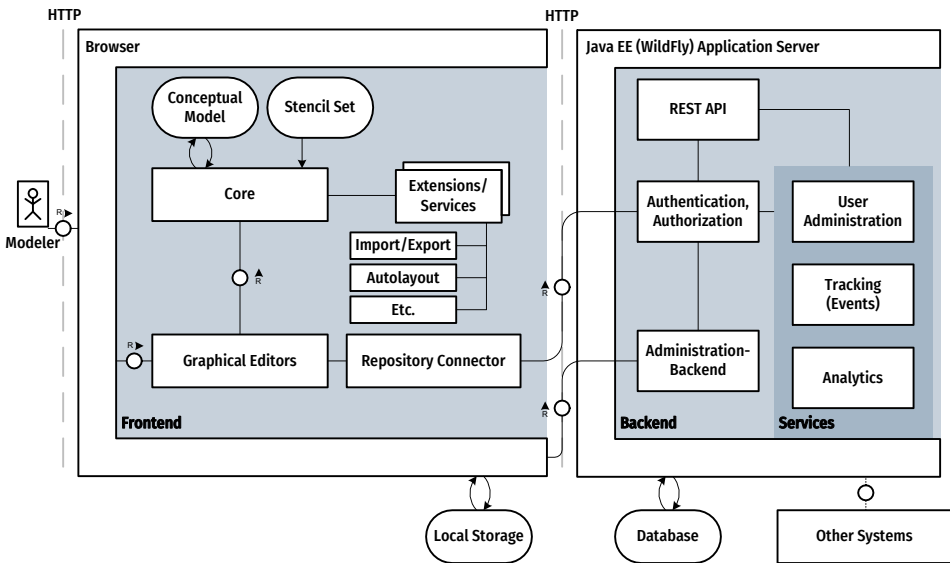


Fig. 1: The software architecture on conceptual design level.

The frontend prototype currently implements two graphical editors we use in an introductory course on conceptual modeling: A variant of the Entity-Relationship Model [Ch76] for data modeling and a subset of the MEMO Organisation Modeling Language [Fr11] for business process modeling (see Fig. 2). With respect to the user interface paradigm, we opted for the widely used stencil set (left) and modeling canvas (right) approach but consider the user

interface subject to future research on better supporting the learning process after having identified patterns of learning and learning difficulties.

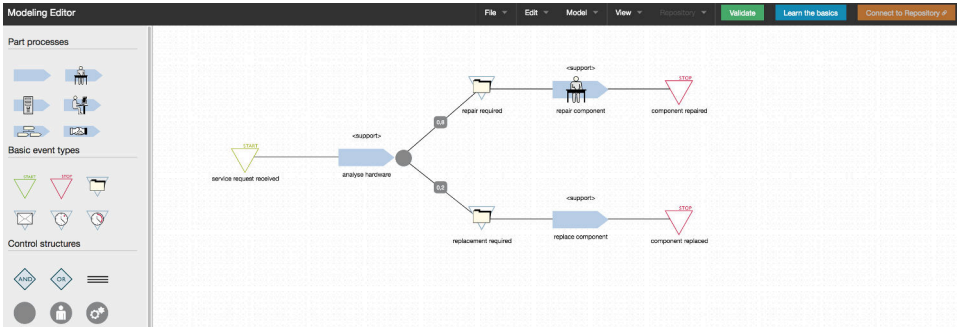


Fig. 2: Overview of the web-based modeling tool.

The backend prototype implements a tracking and an analytic component including algorithms for tracking and functionalities for analyzing learner-editor interactions. In more detail, the prototype implements an algorithm which records the learner-editor interactions while working on, e.g., modeling tasks. An additional analysis interface extends the current prototype, and preliminary comprises two analysis functionalities for reconstructing the learner interactions (see Fig. 3): A step-by-step replay and an automatic replay. Corresponding analytics and visualizations of tracked data will be added in future work.

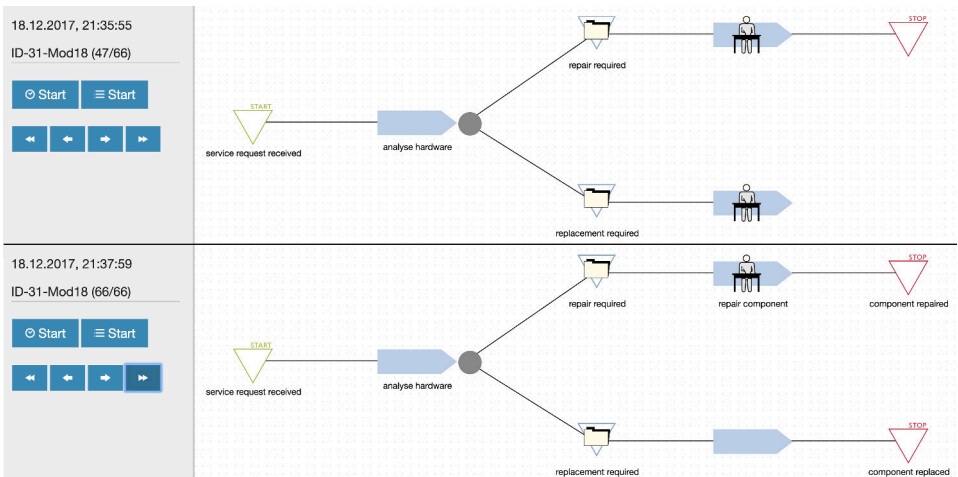


Fig. 3: Overview of the implemented replay analysis.

Please note that due to privacy and security issues, the tool can only be accessed via a VPN connection to the university network at the following link: <http://tool.fernuni-hagen.de>. Further information about the prototype, such as the JSON structure that is used for model serialization, are provided upon request.

3 Limitations and outlook

Observing learning conceptual modeling by learner-editor interaction is a principle limitation of our approach, and neglects other, presumably equally important aspects of the learning process, e.g., learner motivation and willingness-to-learn, use of additional tools outside of our modeling tool, e.g., online tutorial videos etc. In another respect, observing learner-tool interaction is a second- or third-best approach: Asking learners to think out loud (e.g. [Ha16]) while modeling promises further and more detailed insights into their reasoning and is on our agenda as an additional mean of studying the learning of conceptual modeling. We plan to add support for thinking out loud to the prototype in a future version.

Tool development is also confronted with technical challenges, e.g., with performance issues under heavy load which need further investigation and systematic testing (potentially having more than 1,000 students using the tool at the same time). Likewise, the prototype needs further testing of run-time stability under high load which we consider especially important with regard to the implementation of the tracking algorithm. These limitations and challenges remain on our research agenda.

References

- [Ch76] Chen, P. P.-S.: The Entity-Relationship Model—Toward a Unified View of Data. *ACM Transaction Database Systems*, 1(1):9–36, 1976.
- [Fr11] Frank, U.: MEMO Organisation Modelling Language (2): Focus on Business Processes. Technical report, ICB-Research Report No. 49, Institute for Computer Science and Business Information Systems (ICB), University Duisburg-Essen, 2011.
- [Ha16] Haisjackl, C.; Barba, I.; Zugal, S.; Soffer, P.; Hadar, I.; Reichert, M.; Pinggera, J.; Weber, B.: Understanding Declare Models: Strategies, Pitfalls, Empirical Results. *Software and Systems Modeling*, 15(2):325–352, 2016.
- [Pi12] Pinggera, J.; Soffer, P.; Zugal, S.; Weber, B.; Weidlich, M.; Fahland, D.; Reijers, H. A.; Mendling, J.: Modeling Styles in Business Process Modeling. In: *Enterprise, Business-Process and Information Systems Modeling*. LNBIP, vol 113. Springer, Berlin, Heidelberg, pp. 151–166, 2012.
- [SDS16] Sedrakyán, G.; De Weerd, J.; Snoeck, M.: Process-mining enabled feedback: "tell me what i did wrong" vs. "tell me how to do it right". *Computers in Human Behavior*, 57:352–376, 2016.
- [SSD14] Sedrakyán, G.; Snoeck, M.; De Weerd, J.: Process mining analysis of conceptual modeling behavior of novices—empirical study using JMermaid modeling and experimental logging environment. *Computers in Human Behavior*, 41:486–503, 2014.
- [TVC17] Turetken, O.; Vanderfeesten, I.; Claes, J.: Cognitive Style and Business Process Model Understanding. In: *Advanced Information Systems Engineering Workshops*. CAiSE 2017. LNBIP, vol 286. Springer, Cham, pp. 72–84, 2017.

Autorenverzeichnis

A

Abu-Alqumsan, Mohammad, 261

B

Bai, Yu, 245

Berkemeier, Lisa, 321

Burdusel, Alexandru, 299

D

Doan, Khanh-Hoang, 135

Dörndorfer, Julian, 23

Drescher, Andreas, 315

F

Fill, Hans-Georg, 55

Fritsche, Lars, 301

G

Gerten, Prof. Dr. Rainer, 307

Gogolla, Marti, 135

Golubski, Wolfgang, 151

Grabowski, Markus, 245

Greenyer, Joel, 167

Gritzner, Daniel, 167

Grosche, Andreas, 103

I

Igel, Burkhard, 103

J

Jannaber, Sven, 321

John, Stefan, 299

K

Kaiser, Bernhard, 245

Klamma, Ralf, 199

Kolagari, Ramin Tavakoli, 119

Krieter, Sebastian, 297

Kulcsár, Géza, 301

Kuryazov, Dilshod, 183

L

Lange, Peter de, 199

Laue, Ralf, 87

Leblebici, Erhan, 39

Leich, Thomas, 297

M

Mann, Zoltán Ádám, 71

Metzger, Andreas, 71

Michel, Jochen, 281

N

Nehls, Holger, 281

Nicolaescu, Petru, 199

P

Pittl, Benedikt, 55

Q

Queins, Stefan, 151

R

Ratiu, Daniel, 281

Rauh, Alexander, 151

Regnat, Nikolaus, 17

Reich, Jan, 311

Reussner, Ralf, 183

Reuter, Christian, 275

Riebisch, Matthias, 215

Roth-Dietrich, Prof. Dr. Gabriele, 307

S

Schäfer, André, 307
Schmid, Klaus, 119
Schoenen, Stefan, 71
Schürr, Andy, 15, 39
Seel, Christian, 23
Slotosch, Oscar, 261
Spinczyk, Olaf, 103
Stehle, Tilmann, 215
Störrle, Harald, 233
Strecker, Stefan, 325
Strüber, Daniel, 299

T

Tchangou, Maxime, 311
Ternes, Benjamin, 325

Thomas, Oliver, 321
Thüm, Thomas, 297
Tomaszek, Stefan, 39

V

Velasco, Santiago, 311

W

Wägemann, Tobias, 119
Walter, Robert, 281
Wang, Lin, 39
Winkler, Thomas, 199
Winter, Andreas, 183

Z

Zobel, Benedikt, 321
Zschaler, Steffen, 299