Proceedings of the Doctoral Symposium at MODELS 2009

Technical Report 2009-566
School of Computing, Queen's University

Kingston, Ontario, Canada

Juergen Dingel (Ed)

October 5, 2009

Table of Contents

Juergen Dingel
The Doctoral Symposium at MODELS'09 2

Andreas Svendsen
Application Reconfiguration Based on Variability Transformations 4

Yu Sun
Model Transformation by Demonstration 10

Konrad Wieland
Advanced Conflict Resolution Support for Model Versioning Systems 16

Angelika Kusel
TROPIC: A Framework for Building Reusable Transformation Components 22

Johannes Schoenboeck
Transformation Nets: A Runtime Model for Transformation Languages 28

Tobias Walter
Combining Domain-Specific Languages and Ontology Technologies 34

Zekai Demirezen
A Semantic Framework for DSMLs 40

Stefan Gudenkauf
A Coordination-based Model-Driven Method for Parallel Application Development 46

Janos Mathe
Towards an Adaptable Framework for Modeling, Verifying, and Executing Medical Guidelines 52

The Doctoral Symposium at MODELS'09

Juergen Dingel
Queen's University, Canada
`dingel@cs.queensu.ca`

The goal of the Doctoral Symposium was to provide a forum in which PhD students can present their work in progress and to foster the role of MODELS as a premier venue for research in model-driven engineering. The symposium aimed to support students by providing independent and constructive feedback about their already completed and, more importantly, planned research work. The technical scope of the symposium coincided with that of MODELS'09. Submissions were required to describe research-in-progress that is meant to lead to a PhD dissertation and clearly indentify:

- Problem: The technical problem the research intends to solve and why it is important and needs to be solved.
- Related work: A review of the relevant related work with an explanation of how the proposed approach is different and which advantages it has.
- Proposed solution: A description of the proposed solution and which other work (e.g., in the form of methods, or tools) it depends on.
- Expected contributions: A list of the expected contributions.
- Current status: The current status of the work and how close to completion it is.
- Plan for evaluation: A description of how it will be shown that the work does indeed solve the targeted problem.

Submissions were encouraged from PhD students who had settled on a PhD topic, but were still sufficiently far away from completion to be able to take full advantage of the given feedback. Nineteen papers from nine countries were submitted. Every submission was reviewed by three members of the program committee with respect to

- Overall quality of the submission itself (e.g., clarity, precision (relative to the stage of the research), and adequacy of the problem statement, the solution description, the expected contributions, the plan for evaluation, and the review of related work)
- Potential quality of the (completed and proposed) research (e.g., originality of solution and its impact) and its relevance to the MODELS community

Nine papers were accepted all of which are included in the proceedings. The program committee consisted of

- Ruth Breu, University of Innsbruck, Austria
- Betty Cheng, Michigan State University, USA
- Juergen Dingel, Queen's University, Canada
- Gregor Engels, University of Paderborn, Germany
- Robert France, Colorado State University, USA
- Vahid Garousi, University of Calgary, Canada
- Aniruddha Gokhale, Vanderbilt University, USA

- Jeff Gray, University of Alabama at Birmingham, USA
- Gerti Kappel, Vienna University of Technology, Austria
- Jochen Küster, IBM Research Zürich, Switzerland
- Ingolf Krüger, University of California at San Diego, USA
- Yvan Labiche, Carleton University, Canada
- Pieter Mosterman, The MathWorks, USA
- Iulian Ober, University of Toulouse, France
- Alexander Pretschner, Fraunhofer IESE and Kaiserslautern University of Technology, Germany
- Bernhard Schätz, Munich University of Technology, Germany
- Holger Schlingloff, Humboldt University, Germany
- Michal Smialek, Warsaw University of Technology, Poland
- Stéhane Somé, University of Ottawa, Canada
- Janos Sztipanovits, Vanderbilt University, USA

The organizer would like to thank to the authors of submitted papers for giving us the reason to hold this symposium, to the members of the program committee for their excellent work, to Alexander Pretschner for sharing his experience from last year's symposium, and to MODELS'09 general chairs Robert Pettit and Sudipto Ghosh for their support.

Application Reconfiguration Based on Variability Transformations

Andreas Svendsen[1,2]

[1] SINTEF, Pb. 124 Blindern, 0314 Oslo, Norway,
`Andreas.Svendsen@sintef.no`
[2] Institute for Informatics, University of Oslo, Pb. 1080 Blindern, 0316 Oslo, Norway

Abstract. When variability is applied to a model and the model is transformed, corresponding test-cases may also require a revision. The Common Variability Language (CVL) can describe variability and be executed to generate model variants. We explore the feasibility of an approach using CVL for automatically transforming a test-case of the original model. The transformation of a specific test-case using CVL is walked through demonstrating the approach. Furthermore we discuss the challenges that need to be resolved. The goal of this work is to find a general solution for reconfiguring applications based on varying models.

1 Introduction

The Common Variability Language (CVL) is a domain specific language (DSL) for describing variability [1]. CVL specifies variability models separated from the base models and allows execution to transform these into various resolved models.

When variability is applied to models, test-cases, dependent on these models, may have to change accordingly. Based on how the original model is changed, these test-cases can be transformed accordingly using CVL. Since CVL is a DSL, and not a general-purpose transformation language (e.g. ATL [2]), proper analysis can be performed to retrieve information from the variability models.

As a comparison Uzuncaova et al. [3] present an approach using incremental test generation to test product line models. Each feature of a program is defined as an Alloy formula, which is used to generate incremental tests. Our approach tries to automatically retrieve the information needed from a CVL model to transform the test-cases. In addition we look at the possibility to predict the verdict of a transformed test.

To give a better understanding of the issues, we first give a description of the background, including CVL, and the problem to be analyzed in section 2, before we introduce an example to illustrate the process in section 3. We then present proposed solutions and challenges in section 4 and a conclusion in section 5.

2 Background and Problem

2.1 Common Variability Language

The Common Variability Language is used to specify in separate models the variability that can be applied to a base model. There are three models of interest: The base model is a model in the domain language, the variability model describes the variability applied to the base model and the resolution model describes how the variability model can be resolved to produce a new base model. One or more variability models can be applied to a base model, and one or more resolution models can be applied to a variability model.

In CVL the simplest form for operation is a substitution, describing a transformation of objects (model elements) in the repository:

- Value Substitution: Change a value of an object
- Reference Substitution: Change a reference between two objects
- Fragment Substitution: Substitute a group of objects (placement) for another group of objects (replacement)

The fragment substitution allows replacing a set of objects (representing a fragment of the model) with another set of objects. The variability model stores the references into and out of the fragments to form placement and replacement fragments. These references in the placement and replacement are matched and changed accordingly to form a new model.

Other parts of CVL are basically just abstraction mechanisms on top of the basic substitutions. Notice that CVL is not a general-purpose language, but rather a DSL for expressing variability. This makes CVL more restrictive and applicable for analysis than general-purpose transformation languages.

2.2 Problem Analysis

Assume that we want to apply variability to a model to produce a new resolved model. Several test-cases may be based on the original model. The problem we will investigate is whether it is possible to also transform these test-cases based on the variability model, and if this can be performed automatically. The process is illustrated in Fig.1.

Our starting point is a model (top left) and its test-case (top right). The first step is to define a CVL model describing how the model can be transformed into another model (step 1). Based on the CVL model we retrieve information about which elements that are about to change and how (step 2). This information is used to automatically build a new CVL model to transform the test case (step 3). The transformed test-case should apply to the transformed model.

3 Example

To illustrate the problem further, we introduce an example from the domain of train signaling.

3.1 Train Control Language

The Train Control Language (TCL) is a domain specific language to describe signaling systems on train stations, which are safety-critical systems of the highest classification [4]. Various representations of the stations, e.g. truth tables, functional specifications, source code, can be generated.

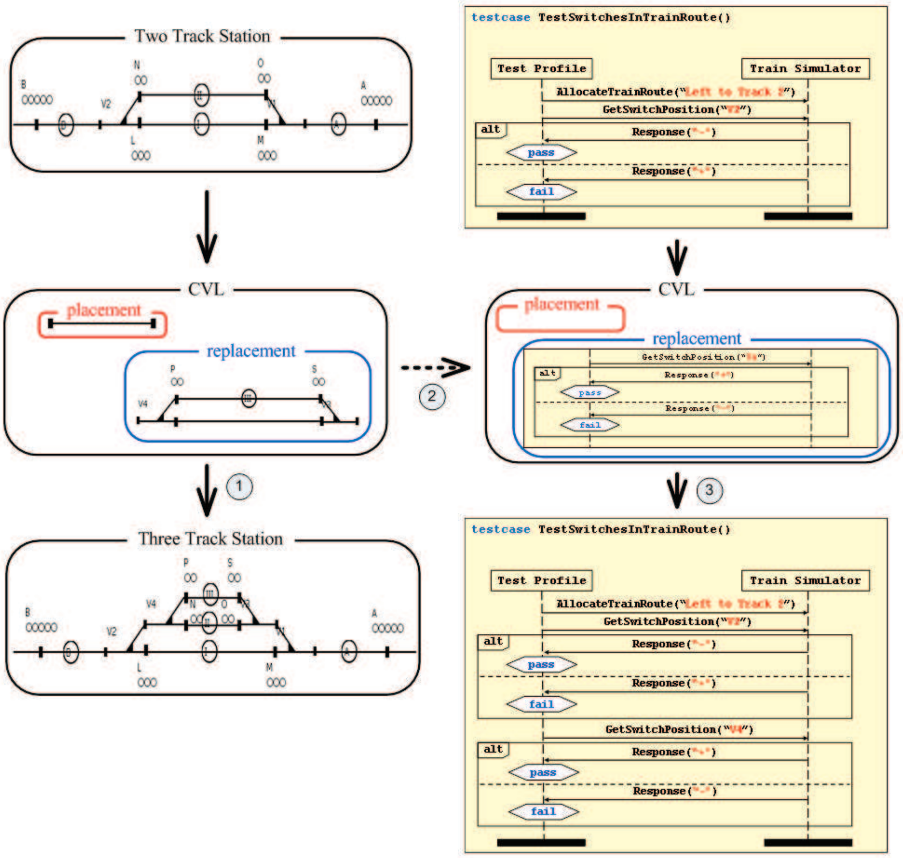

Two Track Station
B
V2
N
O
L
M
A
CVL
placement
replacement
P
S
V4
1
2
3
Three Track Station
testcase TestSwitchesInTrainRoute()
Test Profile
Train Simulator
AllocateTrainRoute("Left to Track 2")
GetSwitchPosition("V2")
alt
Response("-")
pass
Response("+")
fail
GetSwitchPosition("V4")

Fig. 1. CVL transformation of the model (left) and the test-case (right).

TCL defines several elements, such as train routes, track circuits, switches, line segments, signals and endpoints. These elements are illustrated in Fig.2. Train routes are routes between two main signals of the same direction, and these routes need to be allocated before the train can enter or leave the station. The train routes are divided into track circuits, which are further divided into switches and line segments.

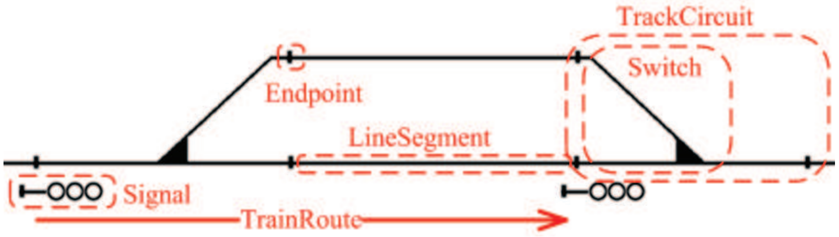


Fig. 2. Concrete syntax of TCL.

Since this is a safety-critical system, the models and the produced representations are tested very thoroughly. The amount of time and resources needed to run all test-cases on a specific station is enormous. Being able to configure a new station based on a well tested one without having to rerun all the test-cases can therefore be a huge saving.

3.2 Challenges to Be Solved

To illustrate the challenges with application reconfiguration based on TCL models, we provide a test-case scenario in TCL.

Assume that we have a two-track station as illustrated in the top left part of Fig.1. This station has eight train routes, from left to track 1 (I), from left to track 2 (II), etc. To be sure to avoid a train from derailing or passing onto a wrong track each train route has to be tested for switch positions. This test is performed on runtime and controls that the switch is actually in the position it is supposed to be. Each switch has two positions: Normal (+) and divert (-) position. For instance the train route “from left to track 2” requires switch V2 to be in divert position (-).

We assume that this two-track station is well tested and approved. Using CVL we transform this two-track station into a three-track station (see the lower left part of Fig.1.) by substituting the second track with a new two-track. Four train routes are added, two in each direction on the third track. Our purpose is to use CVL to transform the test-case for the two-track station to also apply for the three-track station. However, notice that track two and three are divided by an additional switch. The test-case presented for the two-track station is therefore incomplete to describe the three-track station. Regarding the test-case, we have to handle the transformation different for each track:

- **Train routes to/from track 1:** These train routes only include one switch which is equivalent to the train routes in the two-track station. It is therefore no need to update the tests of these train routes.
- **Train routes to/from track 2:** These train routes include another additional switch which has to be included in the tests. Consequently, the original tests have to be extended to also test the position of the newly added switch.

- **Train routes to/from track 3:** These train routes are not present in the two-track station, and tests for them must be created. The created tests can be based on the tests to/from track 2, with a recalculated verdict.

We will in the following examine the transformation of the test of the following train route: “From left to track 2”. The test-case for the original two-track station is illustrated in the top right part of Fig.1, where we get the verdict “pass” if switch V2 is in divert position (-). The notation in this figure can represent any testing profile, e.g. UML testing profile [5] or TTCN-3 [6].

The CVL transformation process is illustrated in Fig.1. A CVL model describes the variability applied to a two-track station to transform it to a three-track station (step 1). The purpose of the test-case transformation is to retrieve enough information from the CVL model (step 2) to automatically transform the test-case to be meaningful for the three-track station (step 3).

The test-case for the three-track station requires an additional test of the position of switch V4. Transforming the test-case for the two-track station therefore requires a substitution where an empty placement is replaced by this test. What kind of replacement and how it is configured is based on the information from step 2. Notice that this is a special case, but we want it to be applied to the general case too.

This test-case also yields other parts of interest. For instance tests of train routes to/from track 1 do not need to be retested. Since we assume that the two-track station is well tested, and since the changes do not affect these train routes, this should be a viable option. However, we need to be certain that other changes cannot have cascading effects.

4 Proposed Solution and Challenges

For this scenario to be possible, we need to retrieve the elements that is examined from the test-case (i.e. train route and switch). The connection between the original elements found in the test-case must be analyzed to gather information about the requirements of the test-case. Then the CVL model describing the variability must be analyzed and put in context of the original test-case to examine the impact the change will have on the test-case. Our proposed solution is to use CVL to describe the transformations of both the model and the test-case.

Since CVL is not a general-purpose transformation language, information about the properties of the variable elements can be identified. CVL uses no search for patterns, but rather describes exactly where we apply a change and how. Detailed information about which elements that vary and how they vary are therefore available in the variability model. Accompanied by an analysis of the domain language and the test-case, properties of these test-cases can be applied. The purpose of CVL and application reconfiguration is to be general enough to support many cases.

However, there are some challenges. Can this approach be fully automated, or will there be a need for manual intervention? Even though we can retrieve information from the CVL model, this may not be enough to perform transformation of the test-case. Another question that is applicable is how we can be certain about the completeness and correctness of the transformed test-case.

Another matter is whether a well tested test-case needs to be retested. If the verdict of the transformed test can be safely derived, a rerun of the test is not necessary. It should therefore be possible to automatically decide whether the verdict is safely derived. However, how can we for certain decide whether we need to rerun a test? Even though the element which is tested is not changed, there may be cascading effects. It may be necessary to restrict and formalize CVL and the domain language. The question is, however, if this will make the approach more general.

If test-cases can be transformed successfully based on CVL models, it is applicable to also transform other kinds of applications. Whether this approach can support general application transformation is also something we would like to investigate.

5 Conclusion

We have seen a case where transforming a model yields a need for modifying a corresponding test-case. Based on the CVL model applied to the base model we investigated how we can automate this process. Proposed solution and challenges concerning our approach were listed. The expected contributions of this work is tool (based on the CVL tool) and methodology support for application reconfiguration. This tool will be evaluated by industrial partners.

References

1. Haugen, Ø., Møller-Pedersen, B., Oldevik, J., Olsen, G.K., and Svendsen, A., "Adding Standardized Variability to Domain Specific Languages," SPLC 2008, Limerick, Ireland, (2008)
2. ATL: ATLAS Transformation Language : http://www.eclipse.org/m2m/atl/.
3. Uzuncaova, E., Daniel, G., Sarfraz, K., and Don, B., "Testing Software Product Lines Using Incremental Test Generation," in Proceedings of the 2008 19th International Symposium on Software Reliability Engineering: IEEE Computer Society, (2008)
4. Svendsen, A., Olsen, G.K., Endresen, J., Moen, T., Carlson, E., Alme, K.-J., and Haugen, Ø., "The Future of Train Signaling," Model Driven Engineering Languages and Systems (MoDELS 2008), Tolouse, France, (2008)
5. OMG, "UML Testing Profile, V 1.0," OMG. formal/05-07-07 (2005)
6. TTCN-3: Test & Test Control Notation : http://www.ttcn-3.org/.

Model Transformation by Demonstration[1]

Yu Sun

Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294
yusun@cis.uab.edu

Abstract. A common approach toward model transformation is to write transformation rules in a specialized language. Although such languages provide powerful capabilities to automate model refinements, their usage may present challenges to those who are unfamiliar with a specific model transformation language or a particular metamodel definition. The research described in this paper makes a contribution toward simplifying the creation of model transformations by recording and analyzing the operational behavior exhibited by an end-user. The paper presents the motivation and the current status of this research, followed by the project objectives, methodology and evaluation plan. A prototype is described that provides initial evidence of the benefits of the approach.

Keywords: Model transformation, code generation, by demonstration.

1 Background and Motivation

Model transformation has emerged as a core part of Model-Driven Engineering (MDE). Examples of model transformation include code generation from models, model synchronization and mapping, model evolution, and reverse engineering [1]. Several approaches have been developed to perform model transformations, such as: direct model manipulation, intermediate representation, and transformation language support [2].

Direct model manipulation accesses the internal structure of a model instance using an API provided by a host modeling tool, and encodes the transformation procedures in a general-purpose programming language (GPL), such as Java or C++. This approach may be used by software developers who are familiar with programming languages, but is not feasible for end-users who do not have programming experience. The power of a transformation is often restricted by the supported API within the modeling tool. Furthermore, GPLs lack the high-level abstractions that are needed by end-users to specify transformations, making the transformations more challenging to write, understand, and maintain.

Many modeling tools support importing and exporting model instances in the form of XMI (a standard interchange format for UML models). It is possible to use existing

---

[1] This work is supported in part by an NSF CAREER award (CCF-0643725).

XML tools (e.g., XSLT) to perform model transformations outside of a modeling tool using XMI as an intermediate representation. Although XSLT is specifically used to transform models, it is tightly coupled to XML, requiring experience to define the transformations using concepts at a lower level of abstraction.

A more common and popular approach toward implementing model transformations is to specify the transformation rules by using a model transformation language. Although most of these languages are powerful, they are not perfect, still presenting several challenges to users, particularly those domain experts and non-programmers who are unfamiliar with a specific transformation language. Even though declarative expressions are supported in most model transformation languages, they may not be at the proper level of abstraction for an end-user, and may result in a steep learning curve and high training cost. Furthermore, the transformation rules are usually defined at the metamodel level, which requires a clear and deep understanding about the abstract syntax and semantic interrelationships between the source and target models. In some cases, domain concepts may be hidden in the metamodel and difficult to unveil [3][4] (e.g., some concepts are hidden in attributes, association ends or enumerations, rather than being represented as first-class entities). These implicit concepts make writing transformation rules challenging. Thus, the difficulty of specifying metamodel-level rules and the associated learning curve may prevent domain experts from contributing to model transformation tasks from which they have much domain experience.

The research described in this paper contributes a new approach to simplify the realization of model transformations, enabling general users (e.g., domain experts and non-programmers) to specify model transformations without knowledge of a specific model transformation language or metamodel definition.

2 Related Work

Model Transformation By Example (MTBE) is an innovative approach to address the challenges inherent from using model transformation languages. Instead of writing transformation rules manually, MTBE enables users to define a prototypical set of interrelated mappings between the source and target model instances. From those mappings, the metamodel-level transformation rules can be inferred and generated semi-automatically. In this context, users work directly at the model instance level and configure the mappings without knowing any details about the metamodel definition or the hidden concepts. With the semi-automatically generated rules, the simplicity of specifying model transformations may be improved.

MTBE was first introduced by Varró [5], where the prototypical transformation rules were partially generated from the user-defined mappings by conducting source and target model context analysis. Varró later proposed a more practical and efficient way to realize MTBE by using inductive logic programming [6][7]. The basic idea is to represent the initial mappings in the form of logic clauses and then infer the transformation rules using a logic programming engine.

Similarly, Strommer and Wimmer implemented an Eclipse prototype to enable generation of transformation rules from the semantic mappings between domain models [3][8]. Instead of using a logic programming engine, their inference and reasoning process was based on pattern matching and was applied to business process modeling languages [9].

However, the current state of MTBE research still has several limitations that prevent it from being a widely used model transformation approach. The semi-automatic generation often leads to an iterative manual refinement of the generated rules; therefore, the model transformation designers are not isolated completely from knowing the transformation languages and the metamodel definitions. In addition, the inference of transformation rules depends on the given sets of mapping examples (i.e., the model inference is only as good as the seeded examples). In order to get a complete and precise inference result, one or more representative examples must be available for users to setup the prototypical mappings, but seeding the process with such examples is not always an easy task in practice. Furthermore, current MTBE approaches focus on mapping the corresponding domain concepts between two different metamodels without handling complex attribute transformations. For instance, in practice, it is quite common to transform an attribute in the source model to another in the target model with some arithmetic or string operations, which is expressed by imperative transformation rules in a transformation language. Unfortunately, these imperative expressions can only be added manually to the generated rules using current MTBE approaches. The related work mentioned here primarily has been applied to exogenous model transformation (i.e., transformation of model instances from different metamodels), but they are not as beneficial for inferring the refinements that are typical of endogenous model transformations where the source and target models are from the same metamodel.

3 Goals and Objectives

The Model Transformation By Demonstration (MTBD) research described in this paper further simplifies the model transformation process initiated by MTBE. The contribution of MTBD is a technique that will enable all model users (i.e., not only model experts and programmers, but also domain experts and non-programmers) to specify the desired model transformations, without knowing any model transformation language or metamodel definition. The realization of MTBD has the potential to provide fully automatic generation of transformation rules without manual refinement of a transformation specification. MTBD will also be applicable to both exogenous and endogenous model transformations, enabling complex attribute computations. In addition, MTBD can be applied to any model instance without being restricted by the availability of appropriate source and target models.

4 Proposed Methodology

Figure 1 provides an overview of the scope of the research on MTBD. The core idea is a new technique that records user interactions within a modeling tool and infers a representative model transformation specification. Instead of inferring the rules from a prototypical set of mappings (as done in MTBE), users are asked to demonstrate how the model transformation should be done by directly editing (e.g., add, delete, connect, update) the model instance to simulate the model transformation process step by step. The user transforms a source model to the target model during the demonstration process. A recording and inference engine will capture all user operations and infer a user's intention in a model transformation task. A transformation pattern will be generated from the inference, specifying the precondition of the transformation (i.e., *where* the transformation should be done) and the sequence of actions needed to realize the transformation (i.e., *how* the transformation should be done). This pattern serves as an intermediate transformation representation, which can be used to generate different model transformation rules, code, data and other necessary transformation artifacts. The final generated rules and code can be reused in any model instance at any time.

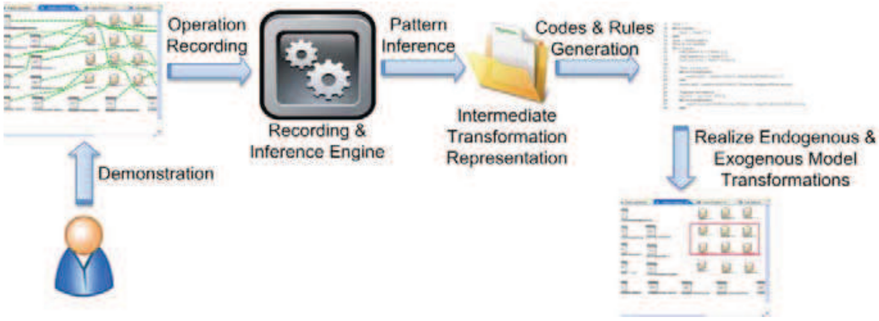

Operation Recording
Pattern Inference
Codes & Rules Generation
Recording & Inference Engine
Intermediate Transformation Representation
Realize Endogenous & Exogenous Model Transformations
Demonstration

Fig. 1. Overview of research scope

5 Experimental Evaluation

The goal of MTBD is to provide transparency of a model transformation language to allow end-users to specify the essence of the operational behavior of a desired transformation. Therefore, this approach must be as powerful as using a general model transformation language; i.e., it must have the capability to correctly realize the same kinds of model transformation tasks that can be done by applying a general model transformation language.

The evaluation of MTBD will be based on three criteria – completeness, correctness and simplicity. Regarding the first two criteria, for each kind of model transformation (i.e., exogenous and endogenous model transformations), some existing transformations written in a specific model transformation language will be selected. For instance, the ATL transformation zoo [10] provides a list of model transformation scenarios that have been implemented by ATL, such as Class to Relational, UML to OWL, among others. MTBD will be used to automatically generate the transformation rules. Given the same set of source models, we can finally compare the target models produced by executing the selected existing transformation rules and those produced by applying the transformation rules generated by our approach. The similarity between the two sets of target models reflects the completeness and correctness of our approach.

The simplicity of MTBD will be evaluated by observing the time and procedure for applying a model transformation by demonstration and the scale of the transformation rules to realize the same task. For example, the size and complexity of an ATL transformation will be compared to the relative effort (in terms of mouse clicks and time) to specify the same transformation by demonstration.

6 Current Results

The current focus of this work is the implementation of endogenous model transformation by demonstration. Since both the source and target models conform to the same metamodel, they can be presented in the same model editor, which facilitates the demonstration process and operation recording. Our work is implemented in the Eclipse-based domain-specific modeling tool – GEMS (Generic Eclipse Modeling System) [11]. An Eclipse plug-in has been developed, which partially realizes the MTBD idea in endogenous model transformations. More specifically, the current status of the MTBD prototype includes: (1) a recording engine to completely capture all user operations and related context; (2) an algorithm to optimize the recorded operations, eliminating meaningless operations (e.g., an add operation followed by a delete operation are both meaningless if they operate on the same object); (3) the inference and generation of a transformation pattern from the recorded operations that describe the weakest precondition and the transformation actions; (4) an algorithm to automatically match a transformation precondition in any model instance, and execute the necessary transformation actions; (5) support to infer transformations with attribute operations; (6) a correctness checking and undo mechanism to guarantee the correctness of the transformation process; (7) fully automatic generation of a transformation pattern, without iterative manual refinement.

We have applied our approach successfully to complete some model refactoring tasks in sample domains, demonstrating transformation correctness and simplicity improvement. More information (e.g., video demonstrations) about the capabilities of the current MTBD prototype is available on the project's web site at: http://www.cis.uab.edu/softcom/mtbd.

7 Overall Contributions

The current work has led to an initial prototype of MTBD that provides initial evidence that the approach can assist end-users in specifying comparatively simple endogenous model transformation tasks that do not require complex preconditions. Such evidence suggests that MTBD is possible without learning any model transformation language and the metamodel definition of the domain. Future contributions are expected to improve the specification of more complex transformation preconditions and more powerful actions so that large-scale and complex model transformations could be supported. In addition, implementing the MTBD idea in exogenous model transformations will be our next focus.

With a complete realization of the MTBD technique, a number of important model engineering applications that are currently defined by model transformation languages may be improved. For example, MTBD can be used to demonstrate the transformation process to generate ATL codes to transform, map and synchronize two domains; MTBD can demonstrate the process of weaving a crosscutting concern to the base model so that Aspect-Oriented Modeling (AOM) is simplified; some commonly used refactoring rules such as Extract Class, Move Method can also be demonstrated, automatically generating the model refactoring rules; some traditionally complex model evolution tasks such as model scalability can be simplified as well through demonstrating the scaling process.

References

1. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal, vol.45 no.3, pp.621-645, 2006.
2. Sendall, S., Kozaczynski, W.: Model transformation - The heart and soul of model-driven software development. IEEE Software, Special Issue on Model Driven Software Development, vol. 20, no. 5, pp. 42-45, Sep./Oct. 2003.
3. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards model transformation generation by-example. In Proceedings of the 40th Hawaii International Conference on Systems Science, Big Island, HI, January 2007, pp. 285.
4. G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer.: Lifting metamodels to ontologies - a step to the semantic integration of modeling languages. In Proceedings of International Conference on Model Driven Engineering Languages and Systems, Genova, Italy, October 2006, pp. 528–542.
5. Varró, D.: Model transformation by example. In Proceedings of Model Driven Engineering Languages and Systems, Genova, Italy, October 2006, pp. 410–424.
6. Balogh, Z., Varró, D.: Model transformation by example using inductive logic programming. Software and Systems Modeling, DOI 10.1007/s10270-008-0092-1, Springer Berlin / Heidelberg.
7. Varró, D., Balogh, Z.: Automating model transformation by example using inductive logic programming. In Proceedings of the 2007 ACM Symposium on Applied Computing, Seoul, Korea, March 2007, pp. 978–984.
8. Strommer, M., Wimmer, M.: A framework for model transformation by-example: Concepts and tool support. In Proceedings of the 46th International Conference on Technology of Object-Oriented Languages and Systems, Zurich, Switzerland, July 2008, pp. 372–391.
9. Strommer, M., Murzek, M., Wimmer, M.: Applying model transformation by-example on business process modeling languages. In Proceedings of Third International Workshop on Foundations and Practices of UML, Auckland, New Zealand, November 2007, pp. 116–125.
10. ATL Transformation Zoo. http://www.eclipse.org/m2m/atl/atlTransformations/
11. Generic Eclipse Modeling System (GEMS). http://www.eclipse.org/gmt/gems/

Advanced Conflict Resolution Support for Model Versioning Systems*

Konrad Wieland

Business Informatics Group
Institute of Software Technology and Interactive Systems
Vienna University of Technology, Austria
`wieland@big.tuwien.ac.at`

Abstract. Collaborative software development is nowadays inconceivable without optimistic version control systems (VCSs). Without such systems the parallel modification of one artifact by multiple users is impracticable. VCSs have proved successfully in the versioning of code, but they are only conditionally appropriate to the management of model versions. Hence, much research effort is currently spent in the development of dedicated model versioning systems. Whereas those approaches mainly focus on an accurate detection of conflicts which may occur when two versions of one model are merged, the actual conflict resolution is hardly considered. Conflict resolution constitutes the phase in the merge process which involves the most human interaction and in which very little automatic support is provided by the current versioning systems. In this proposal, we present research objectives as roadmap towards enhanced user support in conflict resolution for model versioning systems.

Key words: model versioning, conflict resolution, model merging

1 Background

The development of software systems without version control systems (VCSs) is nowadays unimaginable. Especially optimistic VCSs are of particular importance because such systems effectively manage concurrent modifications on one artifact performed by multiple developers at the same time. Besides appropriate infrastructural means like versioning systems for the management of software artifacts, abstraction mechanisms in terms of model-driven engineering (MDE) are required to handle the complexity of modern software systems. Hence software models nowadays are an indispensable source of information for software engineering—either traditionally for documentation purpose or now for MDE (cf. [5]) where code is automatically generated from models. Like other software artifacts, models are developed in teams and evolve over time, consequently they also have to be put under version control.

Standard VCSs for code usually work on file-level and perform conflict detection by line-oriented text comparison. When applied on the textual serialization

---

* This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and FFG under grant FIT-IT-819584.

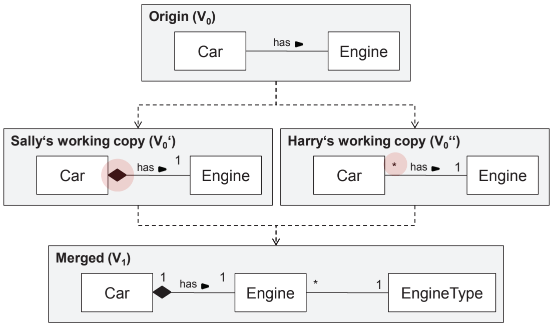


Fig. 1. Motivating Example.

of models, the result is unsatisfactory because the information stemming from the graph-based structure is destroyed and the associated syntactic and semantic information is lost. Consequently, dedicated VCSs for model versioning have been proposed which realize model specific comparison, conflict detection, conflict resolution, and merge components.

Especially when resolving complex conflicts between two versions of a model, user interaction is required. A high degree of automation would enable an effective and time-saving development of software models [7]. However, a pantheon of conflicts exists, where automation is currently at its limit (cf. [15]) and only advanced user support in terms of proper presentation and visualization of the conflicts (e.g., by grouping the conflicts) make the manual resolution practicable.

Further difficulties in merging software models arise from the fact that they express aspects of a software system at a very high level of abstraction and, therefore, reveal a high amount of semantics, domain specific knowledge, and modeling experience. A concrete example in the context of UML Class Diagram is depicted in Figure 1. This example shows that concurrent changes of the same model motivated by different but partly overlapping intentions need manual interventions by the modelers for resolving conflicts. After checking out the actual version of the origin model V0 consisting of the classes *Car* and *Engine* and the association *has*, the modeler Sally replaces the association with a composition in her working copy V0'. Hence, she defines an *Engine* instance as part of one *Car* instance. In parallel, the modeler Harry increases the multiplicities in his working copy in a different way to unbound in order to declare that more than one car may use the same type of an engine (e.g., an engine of the type *Diesel*). Both versions express different understandings of the class *Engine* and therefore an automatic merge is not possible. A naive merge including both variants would lead to a semantically incorrect model as the upper bound for the multiplicity of the composition is restricted to one. A *collaborative interaction* of both modelers is necessary to find a solution combining both intentions. This exchange of information between the modelers leads to a merged model V1 covering both aspects by introducing a third class named *EngineType* (or maybe even a class *CarType*) and consequently result in a model of higher semantics and quality.

Approaches	Graphical Visualization	Grouping	Automatic Resolution	Dependency Detection	Collaborative Merge	Model Verification
Subversion[1]	×	×	×	×	×	×
EMF Comp.[2]	✓	×	×	×	×	×
Alanen & Porres[1]	×	×	×	×	×	∼
Unicase[12]	✓	×	×	×	×	∼
Oda & Saeki[17]	n.a.	n.a.	×	×	×	×
CoObRA[19]	×	×	×	×	×	×
Cicchetti et al.[8]	×	×	∼	×	×	×
Odyssey-VCS[16]	n.a.	n.a.	×	×	×	×
SMoVer[2]	×	✓	×	×	×	×
Ohst et al.[18]	✓	×	×	×	×	×
Küster et al.[13]	✓	×	×	∼	×	×

Table 1. Evaluation of Existing VCS Approaches. (✓...feature is provided, ×...not provided, ∼...partly provided , n.a....not available)

In this research proposal methods focusing on the resolution of conflicts to fulfill the user's requirements of effectively developing software models in a collaborative manner are elaborated. Therefore we review the conflict resolution support in state-of-the-art versioning systems in Section 2 which allows us to formulate six research objectives in Section 3. These objectives will be addressed in the context of the PhD work.

2 Related Work

In the last decades a lot of research approaches in the domain of software versioning have been published which are profoundly outlined in [9] and [15]. Most of them mainly focus on versioning of source code, e.g., Subversion[1], as they deal with software artifacts in a textual manner. Still, dedicated approaches, depicted in Table 1, exist aiming at the comparison and conflict detection for software models. In our evaluation we focused on the features offered for conflict resolution. Some systems provide a graphical visualization of conflicts to support the user in understanding and resolving them. For instance, EMF Compare[2] presents both versions of the model in a tree-based manner linking the modified and conflicting elements by colored lines. Furthermore, in the approach of Ohst et al. [18] the differences are illustrated in the actual model using the concrete syntax of the modeling language. Common model elements in both versions are painted gray whereas differences of interest are highlighted.

To increase the understandability the differences and conflicts may be grouped by certain categories. The versioning system SMoVer [2] groups conflicts by their type.

---

[1] http://subversion.tigris.org

[2] http://www.eclipse.org/modeling/emft/?project=compare

Besides a dedicated conflict presentation the key challenges for an advanced user support are automatic resolution, detection of conflict dependencies as well as collaborative merging. However, these features are hardly provided by any approach of our evaluation. Only Cicchetti at al.[8] presents an idea by one example on how to resolve conflicts automatically and Küster et al.[13] presents an approach for detecting dependencies and conflicts between changes in process models.

Model verification is only partly considered by the approaches of [12] and [1]. In Unicase, for instance, a verification of the conformance to the metamodel is executed before the versions are merged.

3 Research Objectives

The overall goal of the thesis is the development of methods which improve user support during the conflict resolution phase in model versioning systems. In the following, the concrete research objectives addressed in the thesis are shortly presented including the current status of the work as well as a short discussion of the expected contributions.

Evaluation of Existing Versioning Approaches. In a first step we started to review existing model versioning systems based on a literature study (cf. the overview given in the previous section). In the next steps we will extend this work by considering even more approaches from related fields like systems dedicated to ontologies or certain programming languages. Furthermore, we will also perform a comprehensive practical evaluation for which we currently develop a benchmark set consisting of various problems arising in model versioning. We plan to create a repository consisting of conflicting and non-conflicting versions of models which we will provide to the research community working in this area.

Analysis of Status Quo and Requirements Arising from Practice. In order to better understand the requirements on versioning systems and in particular on model versioning systems arising from practice, we developed a questionnaire (available on our project page[3]) which will give a first impression how versioning systems are applied. In this questionnaire the questions are kept quite general with the intention to capture a wide range of developers and project managers. In expert interviews we will capture specific requirements on model versioning systems.

Classification of Conflicts. Although numerous work (cf. [9] and [15]) discuss conflicts in versioning in general, to the best of our knowledge no systematic investigations and classification on the kind of conflicts has been performed so far in the area of model versioning. In the context of data integration, a lot of research has been conducted to classify heterogeneities many years ago [11] but also in recent years [14]. Based on the findings from literature including works from this field and other related fields, the experiences from our evaluation and hopefully also from the expert interviews we will establish a detailed survey on the identified types of conflicts, which occur when two different versions of one model are merged, and on how different conflicts are related. Based on this classification, we hope to get a better understanding how to resolve conflicts.

---

[3] http://www.modelversioning.org

Reduction of Conflict Resolution Effort. Based on the assumption that conflicts will never be completely resolved automatically, we will develop techniques to support the user during the merge process. This includes on the one hand investigations how to present and graphically visualize conflicts in a well-arranged manner by, e.g., grouping the conflicts. On the other hand, we expect that a concise classification of conflicts will help us to better understand the relations and dependencies between different conflicts. This will allow to assign resolution priorities to the conflicts as the resolution of one conflict may automatically reduce the total number of unresolved conflicts, whereas the resolution of another conflict will introduce several new problems.

Verification of the Merged Version. After performing the merge, the following two aspects concerning the quality of the merged model have to be considered.

- The first aspect is the provision of the possible different user intentions, when merging two different versions of a model. In standard versioning systems, the developer who performs the later commit is sole responsible for the often time-consuming, error-prone task of resolving the conflicts. If he has a different understanding of, e.g., the domain, the danger is very high that he destroys the work of the other modeler resulting in unintended models. Recently, collaborative merge approaches for code versioning systems have been proposed to minimize this risk [10]. In [6], we proposed to apply similar techniques in the context of model versioning where the challenge of merging two versions is even more formidable due to their graph-structure and their rich semantics.
- The other quality aspect of a merged model is consistency, i.e., the conformance to its metamodel. This aspect has to be considered also during the merge process [4] and not after the new version is checked into the repository.

Implementation of a Prototype and Evaluation. The research results will finally be integrated within the model versioning framework AMOR [3] and implemented as a plug-in for Enterprise Architect[4], a well-known UML modeling environment developed by our industry partner Sparx Systems. In this context we will perform an extensive evaluation. Test users will be on the one hand our numerous students as we plan to integrate AMOR in our model engineering course. On the other hand we plan to conduct case studies in cooperation with Sparx Systems.

---

[4] http://www.sparxsystems.com.au

References

1. M. Alanen and I. Porres. Version Control of Software Models. In *Advances in UML and XML-Based Software Evolution*. Idea Group Publishing, 2005.
2. K. Altmanninger. Models in Conflict — Towards a Semantically Enhanced Version Control System for Models. In *Models in Software Engineering*. Springer, 2008.
3. K. Altmanninger, G. Kappel, A. Kusel, W. Retschitzegger, M. Seidl, W. Schwinger, and M. Wimmer. AMOR — Towards Adaptable Model Versioning. In *Proc. of the 1st Int. Workshop on Model Co-Evolution and Consistency Management, MCCM'08*, 2008.
4. S. Barrett, P. Chalin, and G. Butler. Model Merging Falls Short of Software Engineering Needs. In *Proceedings of the Workshop on Model-Driven Software Evolution, MoDSE'08*, 2008.
5. J. Bézivin. On the Unification Power of Models. *Journal on Software and Systems Modeling*, 4(2):171–188, 2005.
6. P. Brosch, P. Langer, M. Seidl, K. Wieland, and M. Wimmer. We Can Work It Out: Collaborative Conflict Resolution in Model Versioning. *Accepted for the Europ. Conf. on Computer Supported Cooperative Work, ECSCW'09*, 2009.
7. P. Brosch, M. Seidl, and M. Wimmer. Mining of Model Repositories for Decision Support in Model Versioning. *Accepted for 2nd ECMDA Workshop on Model-Driven Tool & Process Integration*, 2009.
8. A. Cicchetti, D. Ruscio, and A. Pierantonio. Managing Model Conflicts in Distributed Development. In *Proc. of the 11th Int. Conf. on Model Driven Engineering Languages and Systems, MoDELS'08*. Springer, 2008.
9. R. Conradi and B. Westfechtel. Version Models for Software Configuration Management. *ACM Computing Surveys*, 30(2):232, 1998.
10. P. Dewan and R. Hegde. Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development. In *Proc. of the 10th Europ. Conf. on Computer-Supported Cooperative Work, ECSCW'07*. Springer, 2007.
11. W. Kim and J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer*, 24(12):12–18, 1991.
12. M. Kögel. Towards Software Configuration Management for Unified Models. In *Proc. of the 2nd Int. Workshop on Comparison and Versioning of Software Models, CVSM'08*. ACM, 2008.
13. J. M. Küster, C. Gerth, and G. Engels. Dependent and Conflicting Change Operations of Process Models. In *ECMDA-FA*, pages 158–173, 2009.
14. F. Legler and F. Neumann. A Classification of Schema Mappings and Analysis of Mapping Tools. In *Proc. of the 12. GI-Fachtagung fr Datenbanksysteme in Business, Technologie und Web, BTW'07*, 2007.
15. T. Mens. A State-of-the-Art Survey on Software Merging. *IEEE Transactions on Software Engineering*, 28(5):449–462, 2002.
16. L. Murta, C. Corrêa, J. G. Prudêncio, and C. Werner. Towards Odyssey-VCS 2: Improvements over a UML-based Version Control System. In *Proc. of the 2nd Int. Workshop on Comp. and Versioning of Software Models, CSVM'08*. ACM, 2008.
17. T. Oda and M. Saeki. Meta-Modeling Based Version Control System for Software Diagrams. *IEICE Trans. on Information and Systems*, E89-D(4):1390–1402, 2006.
18. D. Ohst, M. Welle, and U. Kelter. Differences Between Versions of UML Diagrams. In *Proc. of the ESEC / SIGSOFT FSE*. ACM, 2003.
19. C. Schneider, A. Zündorf, and J. Niere. CoObRA - A Small Step for Development Tools to Collaborative Environments. In *Proc. of the Workshop on Directions in Software Engineering Environments, WoDiSEE'04*, 2004.

TROPIC - A Framework for Building Reusable Transformation Components[⋆]

Angelika Kusel

Information Systems Group
Johannes Kepler University Linz
Altenberger Straße 69, 4040 Linz, Austria
kusel@bioinf.jku.at

Abstract. Model transformation languages are crucial for the success of Model-Driven Engineering (MDE), being comparable to the importance of compilers for high-level programming languages. The support of large transformation scenarios, however, is still in its infancy since the development of transformations currently takes place on a low-level of abstraction, lacking appropriate reuse mechanisms. We propose a framework called TROPIC (Transformations on Petri Nets in Color) for developing model transformations which tackles these limitations. Firstly, TROPIC allows to specify model transformations on different abstraction levels by providing an abstract mapping view and a concrete transformation view. Secondly, TROPIC facilitates reusability by providing an extensible library of reusable transformation components leading to increased productivity of model transformation development and to higher quality of the resulting model transformations.

Key words: Generic Model Transformations, Reuse, Abstraction

1 Introduction and Problem Description

Model-Driven Engineering (MDE) places models as first-class artifacts throughout the software lifecycle, leading to a change from the "everything is an object" paradigm to the "everything is a model" paradigm [1]. In this respect, model transformations play a vital role, representing *the* key mechanism for *vertical transformations* like the generation of code or documentations and *horizontal transformations* like translations, augmentations and alignments of models, to mention just a few. Several kinds of dedicated model transformation languages have emerged (see [2] for a comparison), which allow specifying and executing transformations between source and target metamodels and their corresponding models, respectively. None of these languages, however, not even the QVT-standard [3] proposed by the OMG, became generally accepted as a state-of-the-art approach. This rare adoption of model transformation languages in practice seems to be, among others, due to the following reasons. Firstly, existing model transformation languages do not provide *appropriate abstraction mechanisms* to deal with the complexity of overcoming structural heterogeneities between different metamodels, a form of heterogeneity well known in the area of database

---

[⋆] This work has been partly funded by the Austrian Science Fund (FWF) under grant P21374-N13.

systems when creating mappings between different schemata [4]. Secondly, current approaches lack *suitable reuse mechanisms* in order to reduce the high and error-prone effort of specifying recurring transformations.

2 Proposed Solution

To alleviate the above mentioned problems, a framework for building reusable transformation components is proposed (denoted as *mapping operators* in the following) which are used to resolve recurring transformation problems in model translation scenarios (cf. Figure 1 (a)). The framework provides two views on a transformation problem, namely an abstract *mapping view* which declaratively describes the semantic correspondences on a high-level of abstraction and a *transformation view* which reveals all details of the transformation logic.

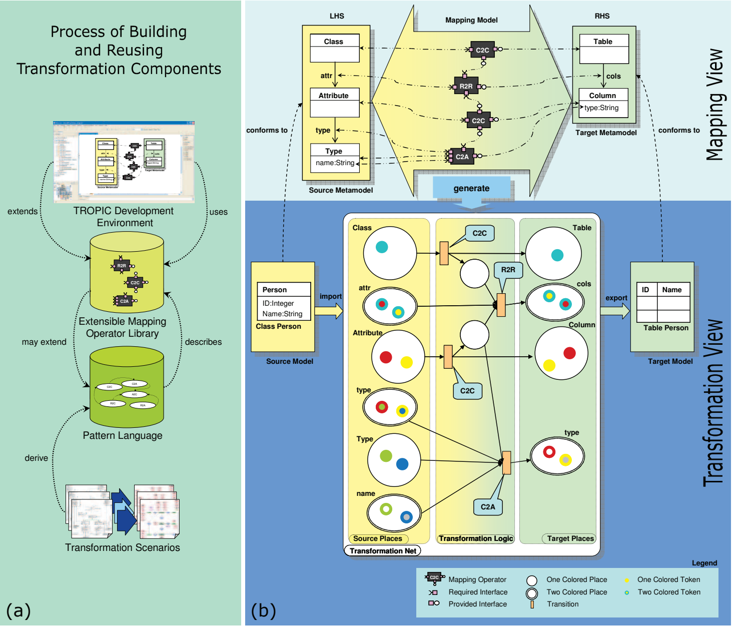

Process of Building
and Reusing
Transformation Components
TROPIC Development Environment
extends
uses
Extensible Mapping Operator Library
may extend
describes
Pattern Language
derive
Transformation Scenarios
(a)
LHS
Mapping Model
RHS
Class
attr
Attribute
type
Type
name:String
Source Metamodel
conforms to
C2C
R2R
C2C
C2A
Table
cols
Column
type:String
Target Metamodel
conforms to
Mapping View
generate
Person
ID:Integer
Name:String
Class Person
Source Model
import
Class
attr
Attribute
type
Type
name
Source Places
C2C
R2R
C2C
C2A
Transformation Logic
Table
cols
Column
type
Target Places
Transformation Net
export
ID
Name
Table Person
Target Model
Transformation View
Legend
Mapping Operator
Required Interface
Provided Interface
One Colored Place
Two Colored Place
Transition
One Colored Token
Two Colored Token
(b)

Fig. 1. (a) Mapping Framework (b) Multiple Views on a Transformation Problem

Mapping View. The mapping view level comprises mapping operators which connect source metamodel elements to target metamodel elements. These mapping operators encapsulate recurring transformation logic and are offered to a transformation designer by means of an extensible library. As a representation formalism, we intend to use a subset of the UML 2 component diagram concepts due to the following reasons. Firstly, this formalism supports a declarative description of mappings. Secondly, a black-box view for transformation logic is provided. And finally, the component's provided and required interfaces enable the composition of mapping operators in order to resolve complex structural heterogeneities. These interfaces are typed by the meta-metamodel datatypes (i.e. in Ecore EClass, EReference and EAttribute), allowing mapping operators to be bound to arbitrary metamodels.

In order to exemplify our approach, Figure 1 (b) illustrates the overall idea from a user's point of view. For this, a simple example is used, transforming some basic object-oriented concepts (`Classes`, `Attributes` and `Types`) into corresponding relational concepts (`Tables` and `Columns`). In order to resolve the occurring structural heterogeneities, three different mapping operators are used, namely a *C2C-component* (transforming class instances, e.g., of the class `Attribute` into instances of the class `Column`), a *R2R-component* (transforming reference instances, e.g., of the reference `attr` into instances of the reference `cols`) and a *C2A-component* (transforming class instances into attribute instances, e.g., of the class `Type` into instances of the attribute `type`).

Transformation View. On basis of this mapping view, an executable transformation view is generated. For this, each mapping operator of the mapping view must have a well-defined operational semantics in the form of some executable piece of transformation logic. For realizing the transformation view, we are planning to use a modified form of Coloured Petri Nets [5], in the following denoted as Transformation Nets [6] due to the following reasons. Firstly, Transformation Nets enable the execution of the transformation without introducing an *impedance mismatch between the mapping view and the transformation view* as each mapping operator can be realized by an independent set of transitions and places without the need for an explicit control flow between the mapping operators. Secondly, this formalism allows for a *homogenous representation of all artifacts involved* in a model transformation, thus being especially suited for gaining an understanding of the intricacies of a specific model transformation. Finally, since Transformation Nets are already executable, an *explicit runtime model* is provided facilitating the debugging of model transformations [7].

3 Expected Contributions

Three main contributions are expected which foster reuse and abstraction allowing for larger transformation scenarios. Firstly, abstract reuse will be supported by the development of a *pattern language* for model transformations. Subsequently, concrete reuse will be supported by offering an extensible mapping operator library. Finally, abstraction will be facilitated through a development environment, that can be used to realize a mapping view on a concrete transformation problem and generate the corresponding exeutable transformation logic.

Pattern Language for Model Transformations. A major task will be the investigation of existing model transformations to build up a catalog of transformation patterns for recurring transformation problems in the form of a textual description comprising the standard parts of a design pattern, i.e., name, description as well as concrete implementation. For identifying these patterns, different sources will be investigated like (1) existing lists of patterns for resolving structural heterogeneities [8], [9], (2) existing model transformations in the ATL model transformation zoo (www.eclipse.org/m2m/atl/atlTransformations/), and (3) transformation scenarios between metamodels for structural domains (e.g., ER models and UML class models) as well as for behavioral domains (e.g., BPMN models and BPEL models). Finally, common problems in the area of information integration will be investigated since the mapping of schemas is closely related to the mapping of metamodels whereby, [10] and [11] provide starting points. On top of the resulting list of found patterns, a useful categorization will be established resulting in a pattern language.

Extensible Mapping Operator Library. It goes without saying, that the resulting library of mapping operators being part of the pattern language can not be complete with regard to solving arbitrary transformation problems. Therefore, the transformation designer must be able to define his/her own mapping operators leading to the need of a mapping operator editor which allows to extend the library of existing mapping operators by user-defined ones and thus potentially extending the pattern language. User-defined mapping operators can be defined from scratch or by reusing existing ones. In this respect, different reuse mechanisms should be possible like building a new operator by (1) *black-box reuse* comprising the sequencing and/or nesting of existing ones or by (2) *white-box reuse*, i.e. inheriting from an existing one and further refining it.

Development Environment. Finally, mapping operators must be applicable in concrete model transformation scenarios representing the mapping view of a transformation problem. Therefore, a development environment is needed, which allows first, to build a mapping model consisting of mapping operators between a concrete source metamodel and a concrete target metamodel and second, to generate the corresponding executable transformation view.

4 Related Work

Related Work is discussed along three dimensions: abstraction, abstract reuse and concrete reuse.

Abstraction. The ATLAS Model Weaver (AMW) [12] offers abstraction mechanisms by the definition of simple correspondences (denoted as weaving operators) between two metamodels. The operational semantics of the weaving operators is determined by a higher-order transformation that takes a weaving model as input and generates model transformation code. The weaving models are compiled into low-level transformation code in terms of ATL which is in fact a mixture of declarative and imperative language constructs. Thus, it is difficult to debug a weaving model in terms of weaving operators, because they do not explicitly remain in the model transformation code. Moreover, although it is possible to add new weaving operators, the specification of the operational semantics thereof is cumbersome, since the whole higher-order transformation must be adapted. Finally, a weaving operator always connects source metamodel elements to target metamodel elements, so it is not possible to realize complex transformation logic by the composition of operators.

Abstract Reuse. Abstract reuse in the form of transformation patterns is still in its infancy. A first list of patterns in the context of graph transformations has been proposed by Agrawal et al. [8]. Another initial list of patterns originating from QVT Relations specifications has been collected by Iacob et al. [9]. These two lists can act as an initial input for our pattern language.

Concrete Reuse. Typically, model transformation languages, e.g., ATL [13] and QVT [3], allow to define transformation rules based on types of the corresponding metamodels. Consequently, model transformations are not reusable and must be defined from scratch again and again. One exception is the approach of Varró et al. [14] who define a notion of generic transformations within their VIATRA2 framework, which in fact resembles the concept of templates in C++ or generics in Java. Another approach which is now integrating the idea of genericity are TGGs [15]. Therefore, VIATRA2 as well as TGGs also provide a way to implement reusable model transformations and could be principally used to implement our mapping operators. Nevertheless, they do not foster an easy to debug execution model as is the case with our proposed Transformation Nets.

5 Evaluation

The evaluation of our approach is based on the following four research questions:

Question 1: *Are the found patterns useful/applicable in diverse scenarios?* Concerning this question, the following strategy will be applied. The case studies consisting of numerous transformation examples as described in Section 3 will be divided into a training set and a test set. The training set will be taken for finding recurring transformation patterns. Afterwards the test set will be realized with the found patterns in the training set and evaluated on the basis of corresponding reuse metrics [16].

Question 2: *Does the approach lead to a better understanding of large scenarios?* Regarding this issue, an empirical study will be conducted with students from our model engineering courses (around 200 master students every year). The aim of this empirical study is to evaluate whether the abstract mapping view leads to a better understanding of a large problem. Therefore, the students will be divided into two subgroups, whereby one subgroup gets the transformation definition in our proposed formalism and the other subgroup gets the transformation definition in a low-level transformation language. The understandability will then be evaluated based on questionnaires.

Question 3: *Is productivity of the development process increased by the usage of reusable components?* Concerning this point, again an empirical study will be conducted. Thereby, three distinct transformation approaches will be presented, including our proposed approach. Afterwards the students will have to solve a certain problem with each of these approaches. The productivity will then be evaluated based on corresponding metrics.

Question 4: *Is the quality in the sense of correctness of the resulting model transformations increased by the usage of the reusable components?* Regarding the correctness of the resulting model transformations, also an empirical study will be conducted in conjunction with the study evaluating question 3. Thereby also the quality in terms of freedom from errors will be measured.

6 Current Status

This research effort is still in an initial stage comprising one publication [17] which describes a first set of mapping operators. Furthermore, a complementing research effort realizing the transformation view is currently conducted by [18].

References

1. Bézivin, J.: On the unification power of models. Journal on Software and Systems Modeling **4(2)** (2005) 171–188
2. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal **45**(3) (2006) 621–645
3. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/-Transformation Specification. www.omg.org/docs/ptc/07-07-07.pdf (2007)
4. Kashyap, V., Sheth, A.: Semantic and schematic similarities between database objects: A context-based approach. The VLDB Journal **5**(4) (1996) 276–304
5. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modeling and Validation of Concurrent Systems. Springer (2009)
6. Reiter, T., Wimmer, M., Kargl, H.: Towards a runtime model based on colored Petri-nets for the execution of model transformations. In: 3rd Workshop on Models and Aspects-Handling Crosscutting Concerns in MDSD, Berlin, Germany. (2007)
7. Wimmer, M., Kusel, A., Schoenboeck, J., Kappel, G., Retschitzegger, W., Schwinger, W.: Reviving QVT Relations: Model-based Debugging using Colored Petri Nets. In: Proc. of MoDELS '09, Denver (2009)
8. Agrawal, A., Vizhanyo, A., Kalmar, Z., Shi, F., Narayanan, A., Karsai, G.: Reusable idioms and patterns in graph transformation languages. Electronic Notes in Theoretical Computer Science **127**(1) (March 2004) 181–192
9. Iacob, M.E., Steen, M.W.A., Heerink, L.: Reusable model transformation patterns. Volume 0., Los Alamitos, CA, USA, IEEE Computer Society (2008) 1–10
10. Legler, F., Naumann, F.: A Classification of Schema Mappings and Analysis of Mapping Tools. DB-Systeme in Business, Technologie und Web **12** (2007) 449–463
11. Alexe, B., Tan, W., Velegrakis, Y.: STBenchmark: Towards a Benchmark for Mapping Systems. Proc. of the VLDB Endowment archive **1**(1) (2008) 230–244
12. Fabro, M.D.D., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: AMW: A generic model weaver. In: Proceedings of the 1ères Journées sur l'Ingénierie Dirigée par les Modèles, Paris, France. (2005) 10
13. Jouault, F., Kurtev, I.: Transforming Models with ATL. Model Transformations in Practice Workshop of MODELS'05 (2005)
14. Varró, D., Pataricza, A.: Generic and meta-transformations for model transformation engineering. In Baar, T., Strohmeier, A., Moreira, A., Mellor, S., eds.: Proc. UML 2004: 7th International Conference on the Unified Modeling Language. Volume 3273 of LNCS., Lisbon, Portugal, Springer (October 10–15 2004) 290–304
15. Amelunxen, C., Legros, E., Schurr, A.: Generic and reflective graph transformations for the checking and enforcement of modeling guidelines. In: IEEE Symposium on Visual Languages and Human-Centric Computing. (2008) 211–218
16. Frakes, W., Terry, C.: Software reuse: metrics and models. ACM Comput. Surv. **28**(2) (1996) 415–435
17. Kappel, G., Kargl, H., Reiter, T., Retschitzegger, W., Schwinger, W., Strommer, M., Wimmer, M.: A framework for building mapping operators resolving structural heterogeneities. In: Proc. of Information Systems and e-Business Technologies (UNISCON'2008), Springer, LNBIP 5 (2008) 158–174
18. Schoenboeck, J.: Transformation Nets - A Runtime Model for Transformation Languages. Doct. Symp., Models 2009, Denver, USA (2009)

Transformation Nets - A Runtime Model for Transformation Languages[*]

Johannes Schoenboeck

Institute of Software Technology and Interactive Systems
Vienna University of Technology
Favoritenstraße 9-11/188-3, 1040 Vienna, Austria
`schoenboeck@big.tuwien.ac.at`

Abstract. Model-Driven Engineering places models as first-class artifacts throughout the software lifecycle requiring the availability of proper transformation languages. Although numerous approaches are available, they lack convenient facilities for supporting debugging and understanding of the transformation logic. This is not least because transformation engines operate on a low level of abstraction, hide the operational semantics of a transformation and scatter metamodels, models, transformation logic, and trace information across different artifacts. To tackle these problems, we propose a DSL on top of Colored Petri Nets (CPNs)—called Transformation Nets—for the development, execution and debugging of model transformations on a high level of abstraction. This formalism makes the afore hidden operational semantics explicit by providing a runtime model in terms of places, transitions and tokens, and ensures a homogenous view on transformations by representing them on the basis of the runtime model.

Key words: Model Transformation, Debugging, CPN, Runtime Model

1 Introduction and Problem Description

The availability of proper model transformation languages is *the* crucial factor in MDE, since transformation languages are as important for MDE as compilers are for high-level programming languages. Several kinds of dedicated transformation languages have been proposed (see [1] for an overview), comprising imperative, declarative and hybrid ones. Imperative approaches allow to specify complex transformations more easily but induce more overhead code as many issues have to be accomplished in an explicit way, e.g., specification of the execution order. Although hybrid and declarative model transformation languages relieve transformation designers from these burdens, specification of transformation logic is still a tedious and error prone task due to the following reasons.

First, transformation engines used for executing model transformations operate on a considerably lower level of abstraction than the transformation logic itself. This leads to an impedance mismatch between specification and execution, thus hampering understandability and debuggabilty. Second, declarative and hybrid approaches use black-box transformation engines hiding the actual operational semantics, e.g., the Atlas Transformation Language (ATL) uses a stack

---

[*] This work has been funded by the Austrian Science Fund (FWF) under grant P21374-N13.

machine [2]. As a consequence, debugging of model transformations is limited to the information provided by the transformation engine, most often just consisting of variable values and logging messages, but missing important information e.g., why certain parts of a transformation are actually executed or not. Finally, comprehensibility of transformation logic is hampered as current transformation languages provide a limited view on the execution of model transformations, since metamodels, models, transformation specification, and trace information are scattered across different artifacts.

What is needed is a declarative approach that integrates all artifacts in a common view thereby providing a runtime model that makes the operational semantics of a transformation specification explicit. Based on this runtime model, debugging on the level of transformation specifications should be enabled rather than just forcing transformation designers to interpret low-level error messages.

2 Proposed Solution

The conceptual architecture of our approach tackling the aforementioned limitations is shown in Fig. 1. The Transformation Net formalism [3], a DSL on top of CPNs [4], follows a process-oriented view towards model transformations making the operational semantics of the transformation logic explicit. Transformation Nets form a runtime model that provides the explicit statefulness of imperative approaches through tokens contained within places. The abstraction of control flow known from declarative approaches is achieved as the net's transitions can fire autonomously, thus making use of implicit, data-driven control flow.

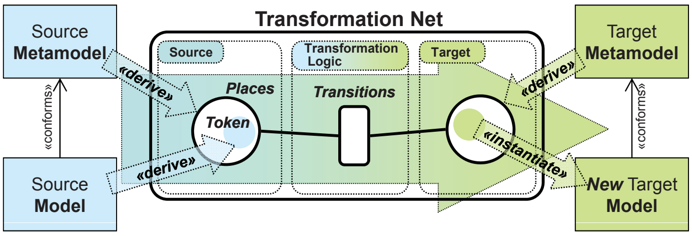


Fig. 1. Conceptual Architecture of Transformation Nets

Furthermore, Transformation Nets provide a uniform formalism not only for representing the transformation logic together with the metamodels and the models themselves, but also for executing the transformations. In particular, places in Transformation Nets are derived from elements of metamodels, whereby a place is created for every class, attribute and reference in a metamodel. Tokens are created from elements of models and then put into the according places. Finally, transitions represent the actual transformation logic. The existence of certain model elements (i.e., tokens) allows transitions to fire and thus stream these tokens from source places to target places finally representing instances of the target metamodel to be created and thereby establishing trace information in terms of tokens within trace places. The abstract syntax of the Transformation Net language is formalized by means of a metamodel (see [3]) conforming to the Ecore meta-metamodel, the Eclipse realization of OMG's MOF standard.

3 Expected Contributions

By the proposed solution we expect three main contributions: (1) a runtime model based on CPNs being the prerequisite for both, (2) debugging of transformation languages and (3) an environment to specify and to debug Transformation Nets.

Runtime Model for Model Transformation Languages. The runtime model based on CPNs allows transformation designers to gain an explicit, integrated representation of the semantics of model transformations which particularly favors debugging and understanding. The runtime might act as a transformation engine for various declarative transformation languages, e.g., QVT Relations, to benefit from our debugging features. As Petri Nets provide formal definitions of concurrent operations, parallel execution of transformation logic is possible to increase efficiency of the execution phase. To ensure valid target models it should be possible to specify different levels of integrity constraints, i.e., an optimistic approach, where conformance will be checked after transformation or a pessimistic approach, where conformance is ensured during transformation.

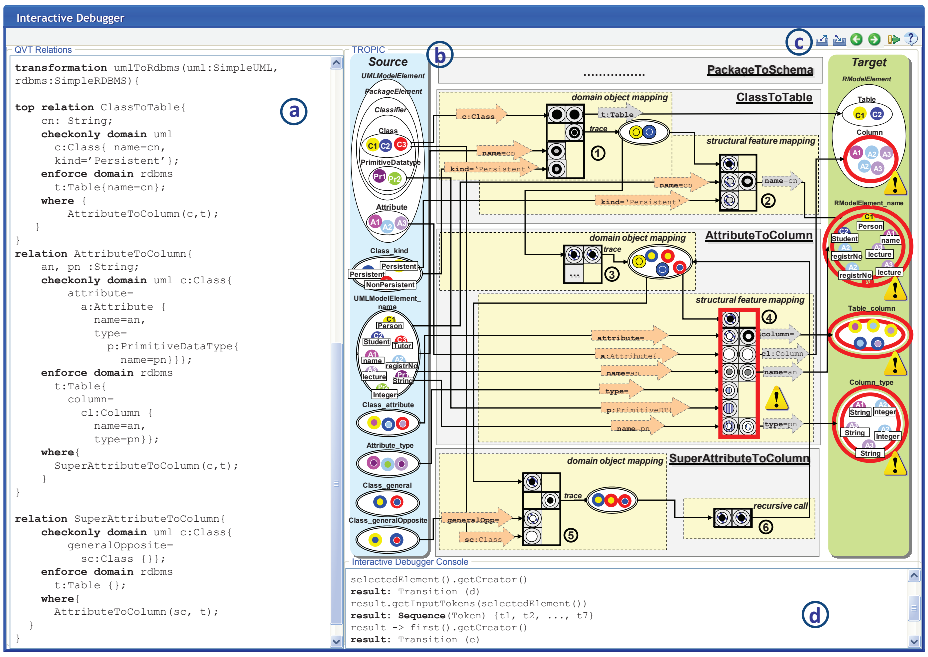
Debugging of Model Transformations. The runtime model supports transformation designers in debugging transformation logic along the three main phases of debugging: (1) observing facts, (2) tracking origins and (3) fixing bugs. Observing facts and tracking origins can be achieved using appropriate mechanisms before (i.e., static debugging), during (i.e., life debugging) or transformation execution (i.e., forensic debugging).

Observing facts. Formal properties of CPNs [5] such as *Reachability*, *Liveness* or *Persistence* can be exploited for *static debugging. Reachability* allows to check if the desired final state (i.e., the expected output model) is reachable from the initial state (i.e., the given input model) with the defined transformation logic. *Liveness* properties can be applied to detect "dead" transformation logic (*L0-liveness*) or for defining test cases, e.g., to check if a transition fires as many times as expected (*L2-liveness*). Finally, the *persistence* property can be used to detect non-determinism or erroneous race conditions. Besides *live debugging* (simulation) also *forensic debugging* is supported in that the resulting target model can be compared to an expected target model to identify wrong target tokens similar to unit-based testing of software.

Tracking origins. The transformation process can be executed stepwise revealing which tokens enable a certain transition and which tokens get produced by firing this transition, enabling *live debugging.* This is possible because Transformation Nets provide a white-box view on model transformation execution, i.e., the specification does not need to be translated into some low-level executable artifact but can be executed right away. Additionally, the runtime metamodel also allows to employ MDE standards for debugging such as OCL to define conditional breakpoints or to explore the execution state by using queries on the runtime model to reason backwards in time. Additionally, *forensic debugging* is enabled by tokens in the corresponding trace places indicating which source elements were used to create specific target elements.

Fixing Bugs. Since model transformations can be "simulated" on the basis of CPNs, a bug can be fixed right away without interruption of the simulation.

Development Environment for Transformation Nets. The runtime model as well as the debugging techniques will be integrated in a development environment supporting the creation, execution and debugging of Transformation Nets. We will additionally provide mappings from declarative transformation languages to Transformation Nets for debugging purposes, e.g., for QVT Relations [6] as shown in Fig. 2. The editor toolbar provides common debugging functionalities such as enabling stepwise debugging to figure out the operational semantics by firing transitions including an undo/redo mechanism. Besides these standard debugging functionalities, there are additional debugging features which result as a benefit of using a dedicated runtime model, e.g., an *Interactive OCL Console* to explore and to understand the history of a transformation by determining and tracking paths of produced tokens [7].


Interactive Debugger
QVT Relations
a
TROPIC
b
c
Source
UMLModelElement
PackageElement
Classifier
Class
PrimitiveDatatype
Attribute
Class_kind
Persistent
NonPersistent
UMLModelElement_name
Class_attribute
Attribute_type
Class_general
Class_generalOpposite
PackageToSchema
ClassToTable
domain object mapping
structural feature mapping
trace
AttributeToColumn
SuperAttributeToColumn
recursive call
Target
RModelElement
Table
Column
RModelElement_name
Table_column
Column_type
Interactive Debugger Console
d

```
transformation umlToRdbms(uml:SimpleUML,
rdbms:SimpleRDBMS){

top relation ClassToTable{
    cn: String;
    checkonly domain uml
      c:Class{ name=cn,
      kind='Persistent'};
    enforce domain rdbms
      t:Table{name=cn};
    where {
        AttributeToColumn(c,t);
   }
}
relation AttributeToColumn{
    an, pn :String;
    checkonly domain uml c:Class{
        attribute=
          a:Attribute {
            name=an,
            type=
              p:PrimitiveDataType{
                name=pn}}};
    enforce domain rdbms
      t:Table{
        column=
          cl:Column {
            name=an,
            type=pn}};
    where{
      SuperAttributeToColumn(c,t);
    }
}

relation SuperAttributeToColumn{
    checkonly domain uml c:Class{
        generalOpposite=
          sc:Class {}};
    enforce domain rdbms
      t:Table {};
    where{
      AttributeToColumn(sc, t);
  }
}
```

```
selectedElement().getCreator()
result: Transition (d)
result.getInputTokens(selectedElement())
result: Sequence(Token) {t1, t2, ..., t7}
result -> first().getCreator()
result: Transition (e)
```

Fig. 2. Development Environment for Transformation Nets.

4 Related Work

Related work regarding the use of Petri Nets for model transformations and debugging support of transformation languages is presented in the following.

Petri Nets and Model Transformations. In the area of graph transformations, some work has been conducted that uses Petri Nets to check formal properties of graph production rules. Thereby, the approach proposed in [8] translates individual graph rules into a place/transition net and checks for its termination. Another approach is described in [9], which applies a transition system for modeling the dynamic behavior of a metamodel.

Compared to these two approaches, our intention to use Petri nets is entirely different, i.e., not just using them as a back-end for automatically analyzing properties of transformations, but additionally use them as a front-end for fostering debuggability and understandability.

Debugging Support for Model Transformations. In the Fujaba environment, a plugin called MoTE [10] compiles TGG rules [11] into Fujaba story diagrams that are implemented in Java, which obstructs a direct debugging on the level of TGG rules. Furthermore, approaches like VIATRA [12] produce debug reports that trace an execution, only, but do not allow to debug certain transformation rules. Debugging of ATL [2] is based on the step-wise execution of a stack-machine that interprets ATL byte-code. In contrast to the above language-specific debugging facilities, Hibberd et al. [13] present forensic debugging techniques by utilizing trace information of model transformations for localizing bugs. In addition, they present a technique based on program slicing for further narrowing the area where a bug might be located.

While Hibberd focuses only on forensic debugging, Transformation Nets additionally enable live debugging. What sets our approach apart from these approaches is that all debugging activities are carried out on a higher level of abstraction and on a single formalism. Current approaches do not provide an integrated view on the whole transformation process in terms of the past state, i.e., which rules fired already, the current state, and the prospective future state, i.e., which rules are now enabled to fire. Therefore, these approaches only provide snapshots of the current transformation state. Furthermore, our approach is unique in allowing interactive execution.i.e., fixing bugs during execution.

5 Plan for Evaluation

The plan for evaluating our approach builds on empirical studies and on applying case studies.

Empirical studies. To evaluate usability and applicability of Transformation Nets we intend to conduct empirical studies with students from our model engineering courses (around 200 master students every year) based on questionnaires. Additionally, we will use the debugging questions of Hibberd et al. [13] and let students answer those questions with our approach to verify the debugging support.

Case Studies. Case studies for transforming models will be set up and implemented with distinct existing model transformation languages, including Transformation Nets. The results will be evaluated on the basis of a suitable subset of the ISO 9126 software quality model [14]. We intend to use a representative selection of metamodels defining structural and behavioral languages. For this, we aim to use the well-known Class2Relational example [15] and the CSP2ActivityDiagrams example [16]. These case studies will also be used to evaluate to what extent concurrent execution of transformation logic improves the performance of model transformations.

6 Current Status

Currently a first prototype to specify and execute Transformation Nets is available which is applied in several case studies to verify the basic approach, as the research is in it's initial state. Additionally, a complementing research focuses on how Transformation Nets can be employed in reusable mapping operators [17].

References

1. Czarnecki, K., Helsen, S.: Feature-based Survey of Model Transformation Approaches. IBM Systems Journal **45**(3) (2006)
2. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: Atl: A model transformation tool. Science of Computer Programming **72**(1-2) (June 2008) 31–39
3. Wimmer, M., Kusel, A., Reiter, T., Retschitzegger, W., Schwinger, W., Kappel, G.: Lost in Translation? Transformation Nets to the Rescue! In: Proc. of 3rd Int. United Information Systems Conf. (UNISCON'09), Sydney, Australia, Springer (April 2009) 315–327
4. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modeling and Validation of Concurrent Systems. Springer (2009)
5. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (1989) 541–580
6. Kusel, A., Schwinger, W., Wimmer, M., Retschitzegger, W.: Common pitfalls of using qvt relations - graphical debugging as remedy. In: 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2009), Potsdam, Germany, IEEE Computer Society (June 2009) 329–334
7. Wimmer, M., Kusel, A., Schoenboeck, J., Kappel, G., Retschitzegger, W., Schwinger, W.: Reviving QVT Relations: Model-based Debugging using Colored Petri Nets. In: Proc. of MoDELS '09, Denver (2009)
8. Varró, D., Varró-Gyapay, S., Ehrig, H., Prange, U., Taentzer, G.: Termination Analysis of Model Transformation by Petri Nets. In: Proc. of Int. Conf. on Graph Transformation. Volume 4178., Natal, Brazil, LNCS (September 2006) 260–274
9. de Lara, J., Vangheluwe, H.: Translating Model Simulators to Analysis Models. In: Proc. of 11th Int. Conf. on Fundamental Approaches to Software Engineering, Budapest, Hungary (April 2008) 77–92
10. Wagner, R.: Developing Model Transformations with Fujaba. Technical report, University of Paderborn (September 2006)
11. Koenigs, A.: Model Transformation with Triple Graph Grammars. Model Transformations in Practice Workshop of MODELS'05, Montego Bay, Jamaica (2005)
12. Balogh, A., Varró, D.: Advanced model transformation language constructs in the VIATRA2 framework. In: Proc. of SAC '06, NY, USA, ACM (2006) 1280–1287
13. Hibberd, M.T., Lawley, M.J., Raymond, K.: Forensic Debugging of Model Transformations. In: Proc. of MoDELS'07, Nashville, USA (October 2007)
14. International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC), Geneva, Switzerland: ISO/IEC Standard No. 9126: Software engineering Product quality; Parts 14 (2004)
15. Bézivin, J., Rumpe, B., Schürr, A., Tratt, L.: Model Transformations in Practice Workshop of MODELS'05, Montego Bay, Jamaica. http://sosym.dcs.kcl.ac.uk/events/mtip05/long_cfp.pdf (2005)
16. Taentzer, Gabriele and Rensink, Arend: AGTIVE 2007 Tool Contest. http://www.informatik.uni-marburg.de/∼swt/agtive-contest/ (2007)
17. Kusel, A.: TROPIC - A Framework for Building Reusable Transformation Components. Doctoral Symposium, Models 2009, Denver, USA (2009)

Combining Domain-Specific Languages and Ontology Technologies*

Tobias Walter**

[1] Institute for Software Technology, University of Koblenz-Landau
Universitätsstrasse 1, Koblenz 56070, Germany
[2] ISWeb — Information Systems and Semantic Web,
Institute for Computer Science, University of Koblenz-Landau
Universitätsstrasse 1, Koblenz 56070, Germany
`walter@uni-koblenz.de`

Abstract. This paper presents an approach of using ontology technologies together with domain-specific languages (DSLs). After identification of some issues and challenges in the development and use of DSLs, we propose a solution that allows the specification of DSLs with seamless integrated ontologies. The specified DSL can be used to build several domain-specific models. Simultaneously reasoning services can be invoked to support the modeling by suggestions and debugging. Furthermore we comment on future work of this approach and on related work.

1 Introduction

Domain-Specific Languages (DSL) are used to model and develop systems of different application domains. DSLs are high-level languages and provide abstractions and notations for better understanding and easier modeling. To develop large software systems different domain-specific languages, respectively fragments of them, may be used. Each DSL focuses on a different problem domain and as far as possible on automatic code generation [1].

DSL designers are dealing with developing domain-specific languages. They specify such languages by defining abstract and concrete syntax and semantics. The new designed DSL is provided to the *DSL user*. He uses the language to create domain models.

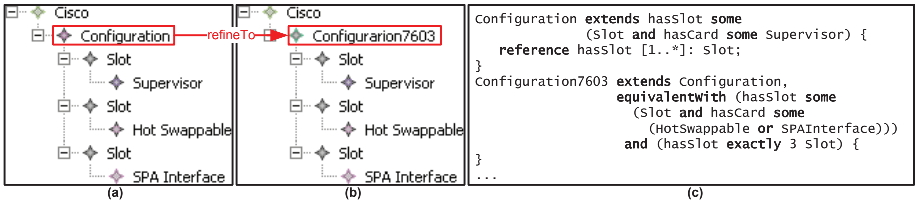
Figure 1 *(a)* and *(b)* depict two domain models which conform to the DSL metamodel (partially depicted in figure 1 *(c)*, defined by using a KM3 syntax [2] extended by description logics). In figure 1 *(a)* a DSL user describes a Cisco device by defining a configuration which in the example has three slots with three inserted cards, namely Supervisor, HotSwappable and SPAInterface. At some point in the domain modeling phase he requires without accomplishing any extra effort suggestions of concepts to be used or wants to validate or refine his domain model. In the example he classifies the Configuration element to find its most specific type, for instance Configuration7603. Thus he refines his domain model.

---

* This work is supported by EU STReP-216691 MOST.

** Under supervision of Prof. Dr. Jürgen Ebert[1] and Prof. Dr. Steffen Staab[2]

To define precisely DSLs and to provide services to the DSL user the DSL designer has to enrich the DSL by additional formal constraints. In figure 1 *(c)* he defines very simple constraints which are integrated and embedded within the DSL metamodel. For instance he defines that a Configuration7603 is equivalent with an anonymous concept which has exactly 3 Slots whereas in one of them either a HotSwappable card or an SPAInterface card is plugged in.



```
Configuration extends hasSlot some
              (Slot and hasCard some Supervisor) {
   reference hasSlot [1..*]: Slot;
}
Configuration7603 extends Configuration,
                  equivalentWith (hasSlot some
                    (Slot and hasCard some
                      (HotSwappable or SPAInterface)))
                   and (hasSlot exactly 3 Slot) {
}
...
```

Fig. 1. *(a)*, *(b)*: Domain Models in concrete Syntax, *(c)*: Excerpt of a Metamodel specifying a DSL

1.1 Problems and Challenges

Although the usage of DSLs provides many advantages there are some problems and challenges to be considered. We differ between problems and challenges for DSL designers and DSL users. The following ones are issues of DSL designers:

1. *Constraint Definition*: To check the consistency of domain models it is essential for the DSL designer to define constraints during the definition of DSL metamodels, which have to be fulfilled by elements in the domain models which are created by the DSL user.

Considering the example above the DSL designer for instance wants to restrict the use of concept Configuration7603 by defining an equivalent, anonymous concept. The additional constraint allows only to use the concept if and only if a valid combination of slots and plugged in cards is given.

The following ones are problems and challenges of DSL users:

2. *Incomplete Knowledge*: DSL users sometimes do not have complete knowledge about all domain concepts. They require suggestions of domain concepts to be used.
3. *Debugging*: For many DSL tools debuggers are missing. For DSL designers and users it might be useful to see the consequences of applying different constructs.

Considering again the example above the DSL user for instance wants to specialize his domain model by requiring suggestions of more specific concepts like the Configuration7603. Or he wants to check the consistency of his domain model with regard to the DSL metamodel and gets some debugging information, if the model is inconsistent.

The remaining sections at first present the proposed solution (cf. section 2) based on the above mentioned problems and challenges. Here we give the idea of a development environment for DSLs, where an instantiable DSL, in the following KM3 [2], is combined with an ontology language to define linguistic instantiable DSLs. In our future work beside DSLs with linguistic instantiation, also the design and use of DSLs with ontological instantiation and non-instantiable DSLs [3] will be considered. In section 3 we discuss the solution by comparing it with related work. Furthermore we present a workplan which also contains the current status and a plan for evaluation.

2 Proposed Solution

We suggest defining DSLs in a way that makes use of ontology technologies, in particular description logics [4]. Description logic is a family of logics for concept definitions that allows for joint as well as for separate sound and complete reasoning at the model and at the instance level given the definition of domain concepts.

OWL2, the Web ontology language, is a W3C recommendation with a very comprehensive set of constructs for concept definitions [5]. Nevertheless, OWL2 has not been designed to act as a metamodel for defining DSL models. Hence, we propose to build an integrated language by combining a pure DSL metamodel and an OWL metamodel in order to benefit from both and thereby tackle the above mentioned problems.

In the following we explain some relevant parts of an environment where KM3 [2], an instantiable DSL, is combined with OWL2. Result of the combination is an integrated metamodel which is used to design further DSLs. In our example we are designing instantiable DSLs with embedded constraints and axioms. Instead of KM3 also other metamodels of languages could be used to be combined with OWL.

In the environment which is depicted in figure 2, we consider different roles. A DSL designer creates DSLs by defining an abstract syntax model which consists of abstract instances of the integrated metamodel, a concrete syntax model (which again could be instantiable and thus could act as a metamodel) and the semantics of the new language. The concrete syntax model is provided to the DSL user. He builds domain models and wants for example only by pressing one button within the environment to check the consistency and debug his domain model or wants to be guided through the modeling process.

In the following we consider metamodel hierarchies to describe the specification and the use of DSLs. At the *M2 layer* the language is specified by defining a metamodel. At the *M1 layer* the specified language can be used by creating an abstract syntax model, which is an instance of the DSL metamodel.

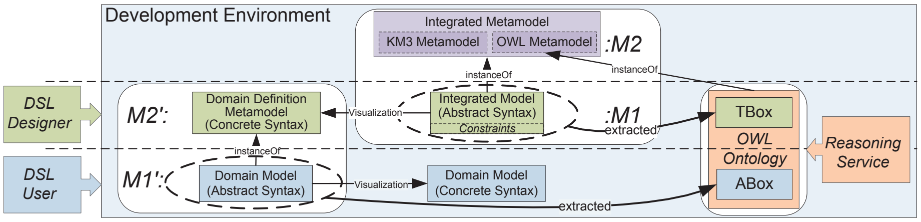

Development Environment
Integrated Metamodel
KM3 Metamodel
OWL Metamodel
:M2
instanceOf
instanceOf
DSL Designer
M2':
Domain Definition Metamodel (Concrete Syntax)
Visualization
Integrated Model (Abstract Syntax)
Constraints
:M1
extracted
TBox
OWL Ontology
ABox
Reasoning Service
DSL User
M1':
instanceOf
Domain Model (Abstract Syntax)
Visualization
Domain Model (Concrete Syntax)
extracted

Fig. 2. Overview of Integration Approach

2.1 Combining DSLs and Ontology Languages

As we see in figure 2 the metamodel of the DSL KM3 and the one of OWL are integrated at the *M2 layer* in the middle column. To accomplish the integrated metamodel, we want to provide an integration approach, which exactly defines how concepts of DSL metamodels and the ones of an OWL metamodel can be combined [6].

Result of the integration is an integrated metamodel. One part of the integrated metamodel consists of the KM3 metamodel, another part consists of OWL constructs. This new metamodel is used by the DSL designer to define integrated abstract syntax models with embedded constraints at the *M1-layer* simultaneously in a seamless manner (*problem 1*).

It is important for the integration that the combination of different metamodels is loss-free. Especially it must be possible to automatically extract the integrated ontology language OWL from the integrated model at the *M1 layer*.

2.2 Designing Domain-Specific Languages

The DSL designer is able to define an integrated model that consists of abstract instances conforming to the KM3 part in the integrated metamodel and - to the extent possible - consists of semantics and constraints for the *M1'-layer* defined by the OWL part of the integrated metamodel. The additional semantics are useful to discuss the meaning of the domain model (*M1' layer*) as well as to indicate constraints that apply at the level of the language itself (*M2' layer*). Figure 1 *(c)* depicts an example of an integrated language.

The newly created abstract syntax is visualized by a concrete syntax. If it is instantiable it can act as a domain definition metamodel of the new specified DSL. This DSL is provided to the DSL user and lies relatively on an *M2' layer*.

2.3 Using Domain-Specific Languages

Having an instantiable DSL specified by the DSL designer the DSL user is able to build different domain models (cf. figure 1 *(a)* and *(b)*). During the domain modeling the DSL user gets different benefits for free.

In the development environment these benefits are represented by reasoning services, which a DSL user can invoke. Furthermore he needs no background information how the reasoning services work and how they are bound with ontologies. The OWL reasoning engine returns suggestions and explanations to the DSL user. All services are provided to the DSL user without any extra effort.

To allow such services the ABox and TBox of an ontology are extracted from the domain model (*M1' layer*) and the integrated abstract syntax model (*M1 layer*), respectively.

Thus, for example some of the following services can be automatically provided to the DSL user, covering the above mentioned problems of DSL users.

- *Dynamic Classification:* A DSL user might not have the complete knowledge of all concepts a DSL provides. Hence he often creates instances of general concepts in his domain model. Invoking the dynamic classification service a DSL user is able to find the most specific concept of elements in the domain model with regard to all other elements (*problem (2)*).

– *Consistency Checking:* DSL users emphasize to have consistent domain models and instances and thus want to check them (*problem (3)*).

3 Discussion

In this section we want to discuss the above presented solution based on related work and to provide an overview of the ongoing work.

3.1 Related Work

Today there are many model-based development environments for DSLs available in the market like for example MetaEdit+ [7], XMF (eXecutable modeling framework) [8] or ATLAS Model Management Architecture (AMMA) [9]. These environments are aligned with the OMG four-layer metamodel architecture. Some of them provide support for specifying queries and constraints, e.g. with OCL-like languages. Here checking constraints and executing queries takes place on one single layer. Our logic-based approach instead allows defining constraints that cover model and instance layer (e.g. *M2'* and *M1' layer*) and provides querying and reasoning simultaneously on both of them. We provide constructs based on Description Logics like equivalence, class descriptions to DSL designers. Thus our environment allows us to support DSL users by guidance and suggestions.

Several approaches describe transformations of MOF-based models to knowledge representation languages where reasoning and querying is adopted. For example [10] presents transformations from MOF-based models to Alloy, [11] presents an approach to describe semantics of MOF-based models with F-Logic. Instead of these approaches, where the expressiveness available for DSL designers is limited to MOF (plus OCL), we provide integrated modeling. Thus the designer benefits from the expressiveness of OWL additionally to the one of MOF.

3.2 Workplan

At first we present the current work status and the constitutive future work. Finally we give a plan for evaluation of the work.

Current Status. Up to now we have considered different usage scenarios presented by industrial partners in the MOST project[3]. The studies of their internal approaches have shown that a large part of domain-related knowledge is specified in natural language, by annotating the models, or even in external documents. Thus we developed an integration approach presented in [6] which combines DSLs and ontology languages at the *M2 layer*. Beside the research of different metamodel integration approaches we started developing an ontology-based DSL framework called *OntoDSL*[12].

Future Work and Expected Contribution. The work in the future should cover at least the following points with corresponding contributions:

1. Formalization of metamodel integration approaches (at the *M2 layer*).
2. Consideration of different language and instantiation paradigms, e.g. the design and use of DSLs with *ontological* or *linguistic instantiations* and the design and use of *non-instantiable* DSLs at the *M2'*- and *M1' layer* [3].

---

[3] `http://www.most-project.eu`

3. Identification of more reasoning services for DSL designers and DSL users.
4. Implementation of a developement environment for designing and using ontology-enriched DSLs.
5. Evaluation of all approaches.

Plan for Evaluation. The evaluation of the approach concerning the metamodel integration of DSLs and ontologies together with its applications is based on usage scenarios and case studies. These scenarios come from industrial partners of the MOST project, namely *Comarch*[4] and *SAP*[5]. The evaluations consist of the deployment of the implemented ontology-enriched modeling approaches to the industrial partners. Thus detailed tests and validations are possible which result information of adaptability and benefits of the integrated use of ontologies.

References

1. Kelly, S., Tolvanen, J.: Domain-Specific Modeling. John Wiley & Sons (2007)
2. Jouault, F., Bezivin, J.: KM3: a DSL for Metamodel Specification. In: Proc. of 8th FMOODS. Volume 4037 of LNCS., Springer (2006) 171–185
3. Atkinson, C., Kuhne, T.: Model-driven development: a metamodeling foundation. IEEE Software **20**(5) (2003) 36–41
4. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The description logic handbook. Cambridge University Press New York (2003)
5. Motik, B., Patel-Schneider, P.F., Horrocks, I.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. `http://www.w3.org/TR/2009/CR-owl2-syntax-20090611/` (June 2009)
6. Walter, T., Ebert, J.: Combining DSLs and Ontologies using Metamodel Integration. In: Proc. of Working Conference of Domain-Specific Languages, Oxford. Volume 5658 of LNCS., Springer (2009) 148–169
7. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In: Proc. of 8th CAISE. Volume 1080 of LNCS., Springer (1996) 1–21
8. Clark, T., Sammut, P., Willans, J.: Applied Metamodelling: a Foundation for Language Driven Development (2nd Edition). Ceteva (2008)
9. Bézivin, J., Jouault, F., Kurtev, I., Valduriez, P.: Model-Based DSL Frameworks. In: OOPSLA, ACM (2006) 22–26
10. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: UML2Alloy: A challenging model transformation. In: Proc. of 10th MODELS. Volume 4735 of LNCS., Springer (2007) 436–450
11. Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A.: Transformation: The Missing Link of MDA. In: Proc. of First International Conference Graph Transformation. Volume 2505 of LNCS., Springer (2002) 90–105
12. Walter, T., Silva Parreiras, F., Staab, S.: OntoDSL: An Ontology-Based Framework for Domain-Specific Languages. In: Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009. Volume 5795 of LNCS., Springer (2009) 408–422

---

[4] `http://www.comarch.com`

[5] `http://www.sap.com`

A Semantic Framework for DSMLs

Zekai Demirezen

Department of Computer and Information Sciences
University of Alabama at Birmingham, Birmingham, AL 35294-1170
zekzek@uab.edu

Abstract. Domain-Specific Modeling Languages (DSML) enable domain experts to participate in software development tasks and to specify their own programs using domain abstractions. To define programs using domain concepts, rather than programming language concepts, tools should provide model-based syntax and semantic specification techniques. However, many Model-Driven Engineering (MDE) platforms primarily concentrate on structural aspects of DSMLs and only provide techniques to define abstract and concrete syntax. A few platforms provide built-in support for specification of dynamic semantics. The purpose of the research described in this paper is to provide a semantic framework that can be used visually by DSML designers, yet has formal underpinnings (transparent to the end-user) such that interoperation with verification tools is possible to realize model checking tasks. This research is focused on a visual technique based on activity diagrams and graph transformation rules to define the semantics of DSMLs.

Keywords: domain-specific languages, operational semantics, graph transformation systems, activity diagram, model checking.

1 Introduction

Model-Driven Engineering (MDE) has been shown to increase productivity and reduce development costs [1]. The concepts advocated by MDE focus on abstractions tied to a specific domain that provide tailored modeling languages for domain experts. Domain-Specific Modeling Languages (DSML), used within the MDE context, enable end-users who are domain experts to participate in software development tasks and to specify their own programs using domain concepts in the problem space, rather than programming language concepts in the technical solution space. However, there remain several challenges that drive new research in DSMLs. For example, simulation, code generation, model checking and different kinds of analysis require a precise definition of the semantics of a DSML that is not provided sufficiently in many modeling toolsets.

DSMLs, like any other language, consist of definitions that specify the abstract syntax, concrete syntax, static semantics and dynamic semantics of the language. Specification of abstract syntax includes the concepts that are represented in the language, and the relationships between those concepts. In the MDE context, domain metamodels are often used to define the structural rules for the abstract syntax. Concrete syntax definition provides a mapping between meta-elements and their textual or graphical representations. Well-formedness rules, which represent the static semantics of a language, can be defined to check model consistency. Such rules are often specified in constraint languages (e.g., OCL) that enforce rules among metamodel elements. The runtime behavior of each syntactical meta-element defined in the DSML represents the dynamic semantics of the language, which is often more challenging to specify. Each part of a DSML specification may be formulated at various degrees of preciseness and formality. MDE platforms mainly concentrate on the structural aspects of a DSML specification and provide techniques to define the abstract and concrete syntax of a DSML. Very few platforms provide a systematic means toward specifying the dynamic semantics of a modeling language [2].

The purpose of the research described in this paper is an investigation into the design of a semantic framework that enables DSML designers to define semantic specifications using visual models. The proposed framework also addresses issues of model verification and model analysis by defining the verification tasks that are specific to a particular domain.

2 Related Work

Current platforms and toolsets that have provided a means for specifying the behavioral semantics of a modeling language often rely on some formalism based on operational semantics. A common approach is to map the metamodel concepts of a DSML to a mature and well-known existing target semantic domain (e.g., Abstract State Machines (ASM) [3], and Petri Nets [4]). In this context, Agrawal et al. [5] and Chen et al. [6] utilize what they call a semantic anchoring technique to map abstract syntax models to existing ASM semantic domains in the GME platform. Ruscio et al. [7] propose a similar technique, except the ASM mapping is integrated within the AMMA platform. In these approaches, the dynamic behavior of a specific DSML element is modeled as a sequence of ASM state transitions. Although these kinds of definitions enable the adoption of model checking and simulation activities using the target semantic domain, it is challenging for DSML designers to use such approaches (because of unfamiliar formalisms in the target model concepts).

Rivera et al. [8] specified the dynamic semantics of DSML in Maude in terms of rewrite rules. Although these rules are similar to graph transformation rules, specifications are represented as Maude objects which requires mapping from the MDE domain to the Maude domain. Scheidgen [9] combine MOF metamodels with an action language based on UML activities to provide a human comprehensible language. However, fine-grained actions, such as create, add, remove, and call, in the language prevent the designer from specifying semantics in a more abstract way.

Muller et al. [10] extended an abstract syntax metalayer with an action language to weave a semantic definition within a metamodel. Kermeta [10] contains constructs for specifying operations of metamodel elements. This built-in support for specification of operational semantics enables the simulation and testing of metamodels. However, the necessity of defining the behavior of each concept in an imperative way results in code that is written in the style of a general-purpose programming language. Engels [11] provides operational semantics of diagrams by means of collaboration and graph transformations. Knapp [12] uses temporal logic; Overgaard [13] advocates the π-calculus to define semantics. Although the formal structures of these related works are suitable for usage with model verification and simulation tools, the specific approaches require expertise in notations and formalisms that are not generally within the skillsets of most designers.

3 Research Goals

The research described in this paper proposes a semantic framework that can be used by DSML designers in a visual manner, yet has a formal foundation that will permit interoperation with model verification tools. Because designers are familiar with domain concepts, the proposed technique enables them to define semantic specifications using visual models. Non-ambiguous and well-defined DSMLs amenable to various verification and analysis tasks may also permit automatic generation of compilers/interpreters and other language based tools. Each domain also has its own set of verification tasks that must be specified in some manner. A key research question addresses the feasibility of designing a general visual language that can be used to define the dynamic semantics of a modeling language, which can interoperate with analysis tools to allow designers to verify the correctness of their models with respect to domain-specific verification tasks.

4 Approach and Methodology

Existing approaches for defining the formal semantics of programming languages can be used to specify the semantics of DSMLs. However, a critical point of this proposed work is an investigation of the benefits that visual models offer to DSML designers in terms of specifying semantics of a new language. To fulfill this objective and accomplish the project goal of transparency of low-level formalisms, three steps will be followed in this project. The first step focuses on the methodology to specify state transitions to show dynamic behavior of meta-elements. The second step concerns the visual language to control the sequence of the defined state transitions and runtime configurations. The third step includes specification of verification tasks and execution of model analysis tools. The combination of all outcomes of these steps will form the semantic framework.

Figure 1 shows an outline of the approach. The first part of the figure demonstrates abstract syntax and static semantic definitions; current platforms (e.g., Amma [7]) provide a means for specifying them. The second part depicts the dynamic semantics specification technique based on activity diagrams and graph grammars. These tools are used to define a sequence of state transitions. The last part shows specification of verification properties within domain boundaries. Finally, all these specifications can be transformed into existing verification tools (e.g., Alloy [18]) to accomplish model analysis.

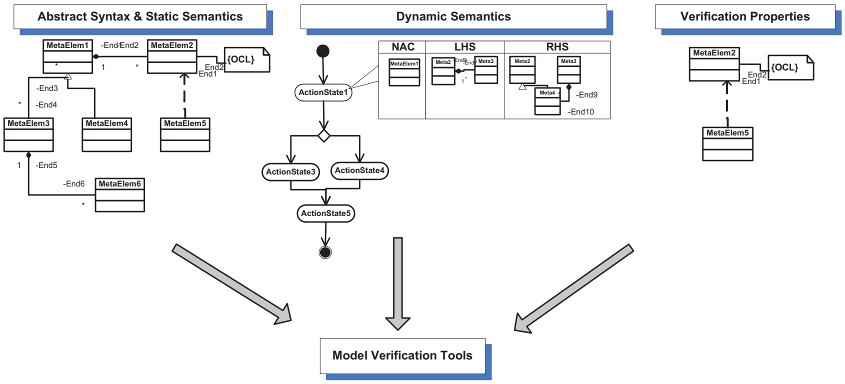

Abstract Syntax & Static Semantics
MetaElem1
-End1
-End2
MetaElem2
{OCL}
End2
End1
-End3
-End4
MetaElem3
MetaElem4
MetaElem5
-End5
-End6
MetaElem6
Dynamic Semantics
NAC
LHS
RHS
MetaElem1
Meta2
Meta3
Meta4
-End8
-End7
-End9
-End10
ActionState1
ActionState3
ActionState4
ActionState5
Verification Properties
MetaElem2
{OCL}
End2
End1
MetaElem5
Model Verification Tools

Fig. 1. An outline of the approach

A first step of this project is to investigate a technique for representing state transitions. Behavior semantics of DSMLs can be represented by a sequence of state transition rules. This approach divides all semantic concerns into discrete states and transition relations. In particular, in-place model transformations [14] represent an approach for designing state transitions. This technique is similar to the Structural Operational Semantics (SOS) defined by Plotkin [15], who proposed SOS to give computational state transitions by means of the abstract syntax of a language. Therefore, SOS defines an abstract behavior for an abstract syntax that allows model checking, correction of proofs and other verification activities.

One of the main characteristics of the in-place model transformation is that target and source models are always instances of the same metamodel. An in-place model transformation rule is defined as L: [NAC]*LHS->RHS, where L is the rule label, LHS denotes the left-hand side rule stating the precondition pattern to trigger the rule; the RHS represents the right-hand side rule that specifies the final model part after execution of a rule. NAC is the optional negative condition that disables the rule if it is satisfied. Graph grammars [4] provide visual rules to specify in-place transformations based on precondition actions and postcondition steps. The notation proposed by AGG [16] to model graph transformations can be used to define these rules visually. AGG is a rule-based visual language supporting an algebraic approach to graph transformation. Available tools associated with AGG (e.g., Graph Transformation Engine, Efficient Graph Pattern Matching, and AGG's analysis techniques for consistency checking) make AGG an attractive candidate for the definition of state translation rules.

Although each AGG transformation shows one of the state transitions of the runtime behavior, to give complete semantics of DSMLs, the sequences of state changes must be defined. These sequence definitions control what state transition is to be fired, in what order, and what condition. Therefore, the second step of this project is to develop a technique for specification of state transition sequences. An activity diagram is an appropriate state machine to define these transition sequences. It enables the design of simple and compound states, branches, forks, and joins. When the activity of a state completes, a transition enables the flow to pass to the next activity. Although flow may continue as sequential transitions, branches can alternate paths. In the proposed framework, each state transition will be mapped with an activity in the activity diagram. Therefore, each activity diagram will depict state transition configurations.

The final step of this project is to facilitate model checking functions by interoperating existing model checking tools with the syntax and semantics of a new modeling language. To enable this capability, an instance model specified in a metamodel must be converted into the formalism expected by an underlying model checking tool. Next, the properties that the model must satisfy need to be stated by a logical formalism expressed in the format expected by the verification tool. For example, Baresi et al. [17] demonstrate how the Alloy tools can be used in graph transformation systems. Alloy [18] is a structural language based on first-order logic. The Alloy analysis tool allows users to prove important properties of a system. Moreover, Alloy can also provide a capability to check the conflict between rules. Graph transformations (described using AGG) can be transformed into an Alloy Model [17]. This transformation can be utilized to map higher level abstractions down to Alloy. Therefore, DSML programs and verification tasks, which are defined by AGG at the domain level, can be transformed into the lower level model needed by Alloy. As a result, the proposed semantic framework can interoperate with the verification tool using transformation between abstraction levels of the different representations.

The current status of this work is represented by a set of formalisms that define the initial approach for specifying the semantics of a modeling language. As a preliminary work, we presented a case study for the verification of simple models using Alloy [19]. Although we introduced the transformation steps from DSML specifications to Alloy models by an example, our objective is to formalize and generalize the mapping rules for integrating analysis tools with a DSML environment. Tool support does not yet exist, but is a focus of near-term future work in order to extend the investigation. At first, an integrated visual editor which provides activity diagram and graph grammar facilities will be completed. After this, integration of analysis tools with the environment will be developed in a PhD thesis.

5 Evaluation

A research question that will be addressed concerns the key issue of whether semantic specification can be defined visually by DSML designers with model concepts, and whether model verification tools can be instantly executed within domain boundaries. This question will be considered in the research evaluation by using several unique domains that each have a representative DSML. Specification complexities of state transition, sequence of transitions, and verification task definitions, will be checked in detail for each domain. An additional step will be needed to compare the proposed semantic framework with other operational semantics techniques. Visual modeling, comprehensibility, ease of use, amount of time to design a new DSML, and compatibility with verification tools will be used as comparison criteria.

Acknowledgments. Supported by NSF CAREER award (CCF-0643725).

References

1. Vallecillo, A.: A Journey Through the Secret Life of Models. In: Position paper at the Dagstuhl seminar on Model Engineering of Complex Systems (MECS), (2008)
2. Sun, Y., Demirezen, Z., Lukman, T., Mernik, M., Gray, J.: Model Transformations Require Formal Semantics. In: Workshop on Domain-Specific Program Development (DSPD), held at GPCE, Nashville, TN, (2008).
3. Borger, E.: High Level System Design and Analysis using Abstract State Machines. In :FM-Trends 98, Vol. 1641 of LNCS, Springer, pp. 1-43, (1999)
4. de Lara, J., Vangheluwe, H.: Translating Model Simulators to Analysis Models. In: Proc. of FASE 2008, Vol. 4961 of LNCS, Springer, pp. 77-92, (2008)
5. Agrawal, A., Karsai, G., Ledeczi, A.: An End-to-end Domain-driven Software Development Framework. In: Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Anaheim, CA, pp. 8–15, (2003)
6. Chen, K., Sztipanovits, J., Abdelwalhed, S., Jackson, E.: Semantic Anchoring with Model Transformations. In: Model Driven Architecture Foundations and Applications: First European Conference. Springer, pp. 115–129, (2005)
7. di Ruscio, D., Jouault, F., Kurtev, I., Bezivin, J., Pierantonio, A.: Extending AMMA for Supporting Dynamic Semantics Specifications of DSLs. Technical Report 06.02, Laboratoire d'Informatique de Nantes-Atlantique (LINA), Nantes, France, April, (2006)
8. Rivera, J.E., Guerra, E., de Lara, J., Vallecillo, A.: Analyzing Rule-based Behavioral Semantics of Visual Modeling Languages with Maude. In: Software Language Engineering SLE, LNCS, pp. 54-73, (2008)
9. Scheidgen, M., Fischer, J.: Human comprehensible and machine processable specifications of operational semantics. In: Akehurst, D.H., Vogel, R., Paige, R.F. (eds.) ECMDA-FA. LNCS, Vol. 4530, Heidelberg, pp. 157–171, (2007)
10. Muller, P. A., Fleurey, F., Jézéquel, J. M.: Weaving Executability into Object-Oriented Meta-Languages. In: Proc. of MODELS/UML'2005, LNCS, Springer, pp. 264-278, (2005)
11. Engels, G., Hausmann, J. H., Heckel, R., Sauer, S.: Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML. In: The Unified Modeling Language: Advancing the Standard. York, UK, Vol. 1939 of LNCS. Springer, pp. 323–337, (2000)
12. Knapp, A.: A Formal Semantics of UML Interactions. In: Proceedings of the UML – Beyond the Standard, Vol. 1723 of LNCS. Springer, pp. 116–130, (1999)
13. Overgaard, G.: Formal Specification of Object-Oriented Meta-Modelling. In: Fundamental Approaches to Software Engineering (FASE'00), Berlin, Germany, Vol. 1783 in LNCS, Springer, pp. 193–207, (2000)
14. Czarnecki, K., Helsen, S.: Feature-based Survey of Model Transformation Approaches. In: IBM Systems Journal, Vol. 45 , Issue 3, July, pp. 621-645, (2006)
15. Plotkin, G. A Structural Approach to Operational Semantics, Technical Report DAIMI FN-19, Department of Computer Science, Aarhus University, Denmark, (1981)
16. Beyer, M.: AGG1.0. Tutorial, Tech. Univ. of Berlin, Dept. of Computer Science, (1992)
17. Baresi, L., Spoletini, P.: On the Use of Alloy to Analyze Graph Transformation Systems. In: Proceedings of the Fifth International Conference on Graph Transformation (ICGT 2006), Vol. 4178 of LNCS, Springer, pp. 306–320, (2006)
18. Jackson, D., Shlyakhter, I., Sridharan, M.: A Micromodularity Mechanism. In: Proceedings of the 8th European Software Engineering Conference, Vienna, Austria, pp. 62–73, (2001)
19. Demirezen, Z., Gray, J., Mernik, M., Bryant, B.: Verification of DSMLs Using Graph Transformation: A Case Study with Alloy. In: Workshop on Model-Driven Engineering, Verification and Validation - Integrating Verification and Validation in MDE, held at MODELS, Denver, CO, (2009)

A Coordination-Based Model-Driven Method for Parallel Application Development

Stefan Gudenkauf[1,2]

[1] OFFIS Institute for Information Technology, R&D-Division Energy, Escherweg 2,
26121 Oldenburg, Germany
[2] Software Engineering Group, Department of Computer Science,
Christian-Albrechts-Universität zu Kiel, Christian-Albrechts-Platz 4, 24118 Kiel,
Germany
email: `stefan.gudenkauf@offis.de`

Abstract. A continuous trend in computing is the demand for increasing computing performance. With the advent of multicore processors in the consumer market, parallel systems moved out of the scientific niche and became commodity. This raises the need to exploit concurrency in software of all kinds and domains. Unfortunately, the majority of software developers today are short on parallel programming experience, and at least in the near future tools and techniques will not be able to fully exploit concurrency in application development automatically.

In this position paper we propose to regard the coordination model of parallel systems as the first development artifact, focusing on top-down application development. To address the need for higher abstractions and to facilitate reuse, we propose a model-driven software development approach based on a visual domain-specific language that hierarchically separates coordination from computation.

1 Introduction

To continue to improve processor performance, companies such as Intel and AMD turned to hyperthreading and multicore architectures since physical limitations impede further performance gains that base on increasing clock speed and optimizing execution flow [1]. These new performance drivers require to explicitly consider concurrency. Unfortunately, after years of sequential programming practice the majority of software developers today are short on parallel programming experience, and at least in the near future there will be no tools and techniques to fully exploit concurrency automatically.

Concurrency has now to be exploited in applications of all kinds and domains. The challenge is not solely software performance and speedup, but to provide a convenient way to participate in the new performance drivers in general. Reuse and portability may turn out to be of major importance because of the high development costs of (re-)developing failure-safe parallel software. In the following, we present related work in Sec. 2. In Sec. 3 we present our model-driven solution and the expected contributions. Finally, we describe our plans for evaluation in Sec. 4 and conclude the paper in Sec. 5.

2 Related work

Lee argues that most of the difficulties in parallel programming are a consequence of our concurrency abstractions [2]. He shows that the threading model, although being a minor syntactical extension to existing languages, implies severe consequences to programming since it is enormously nondeterministic and requires to cut away unwanted nondeterminism by means of synchronization. This is because the immense number of possible interleavings of thread instructions makes it extremely difficult to reason about the actual behavior of an application. Coordination languages can provide a solution, since they are orthogonal to established programming languages and focus on deterministic communication and cooperation between the computational parts of a program.

Pankratius et al. present a case study on parallelizing the open source compression program BZip2 for multicore systems [3]. At least in the context of this study, it is shown that considerable speedup can be gained by exploiting concurrency on higher abstraction levels, and that parallel patterns turned out to be more significant to speedup than fine-grained loop parallelization. It is also noted that industry often propagates the feasibility of inserting parallelization constructs in existing sequential code, thus limiting the amount of exploitable concurrency.

Lee speculates that most existing multithreaded programs have concurrency bugs that do not show up only because the underlying computer architectures and operating systems currently provide modest parallelism, so that only a small percentage of possible interleavings of thread instructions occurs [2]. Also, even if a programmer's code never creates a thread, frameworks may create threads on behalf of the programmer, and thus require the code that is called from these threads to be thread-safe [4]. Expert systems can support and train programmers both in selecting parallel programming paradigms on the architecture level and in selecting thread-safe design patterns.

There are few approaches in model-driven parallel program development. IBM alphaWorks provides a tool that generates parallel code from UML models and supports concurrent patterns for multicore environments [5]. Using the tool involves different activities such as the creation of concurrency patterns by pattern developers and serial computing kernels by C++ developers. There is few information available and the current status of the project is to the best of our knowledge unclear. Pllana et al. propose an intelligent programming environment that targets multi-core systems and proactively supports a user in program composition, design space exploration, and resource usage optimization [6]. This environment is envisioned to combine model-driven development with software agents and high-level parallel building blocks to automatize time-consuming tasks such as performance tuning. UML extension are proposed for graphical program composition. Both works seem promising and will be watched closely, especially regarding the actual method and DSL, and how software engineers are supported in parallel pattern application and in ensuring thread-safety.

3 Solution approach

We propose a model-driven method that regards the overall coordination model of parallel programs as the first development artifact, based upon the following aspects: (1) Top-down problem decomposition is facilitated. (2) Nondeterminism is introduced when needed instead of being cut away when not needed, preserving an overall deterministic program behavior. (3) Mapping high-level units of execution to low-level processing entities is left for model-driven development or run-time scheduling. (4) The development of large-scale parallel programs of all kinds and domains is facilitated. (5) Ordinary software developers are supported in developing parallel applications with considerable effort.

Hypotheses. (1) An explicit coordination metamodel can be developed that combines data sharing for intra-process coordination and message-passing for inter-process communication. (2) Using such a model for the model-driven generation of the coordinational framework of a software product ultimately results in a product that (a) is more performant than a functionally equivalent sequential version of the software ($speedup > 0$) and (b) is not less performant than a functionally equivalent parallel version of the software, that is developed by using the target platform technologies directly ($speedup \geq 0$). (3) Using the coordination metamodel for the model driven generation ultimately results in software that is scalable and maintainable.

Domain analysis. We define our target domain as *parallel systems software engineering*. Basic stakeholder roles are system architects, application domain experts, software developers, and customers. Regarding the software development stages construction, debugging, and performance tuning (cf. [7]), we focus on program construction. We further focus on the behavioral view, regarding the behavior of a software system as concurrent processes within a coordination model, where each process possesses its own address space. Stakeholders are thus encouraged to decompose their problem in terms of processes that consists of tasks (nodes of computation) and sub-processes (a partition of a process in the same address space) to be performed sequentially or in parallel, and control flows between them. Although emphasizing control flows, data flows and objects may also be parts of the model.

DSL. Developing parallel applications using traditional programming languages can be tedious and error-prone due to the linearity of textual source code. Visual DSLs are multi-dimensional, thus able to present multiple concurrent control flows naturally, while fine-grained concurrency control may be encapsulated in appropriate language feature semantics [8, 9]. We regard the threading model as the primary technology underlying our (therefore horizontal) DSL. We may also consider the message passing model, which can be regarded as the most widely-used model for inter-process communication in distributed computing. Orthogonality to existing programming languages is required since it provides an understandable large-scale overview of program structure, cf. [8]. We require the DSL to be visual (RQ-1) and graph-based (RQ-2). Language constructs must conform to domain concepts, ideally providing distinct constructs for each distinct concept (RQ-3). The vocabulary of the language shall be as small as possible (RQ-4) and the constructs shall facilitate model quality (RQ-5). The language must be scalable (RQ-6) and hierarchically composable (RQ-7, cf. Fig. 1). Concurrency must be expressed explicitly as the coordination of tasks and (sub-) processes (RQ-8). Thereby, tasks represent computation and sub-processes represent further compositions of coordinated tasks and sub-processes. Both should be instantiable to represent concurrent execution of the same computation (RQ-9). Also, the language should be control-driven (RQ-10) [10], implying that computational nodes are almost completely separated from coordination since they are regarded as black boxes with defined in- and outputs.

Method. System architects use a DSL to construct an explicit coordination model for the overall behavior of a parallel system. From this model, there may be subsequent model-to-model transformations before code is generated that represents the coordinational framework of the parallel system. Transformations are created by transformation developers (special domain architects), while functional implementation is left for complementing modeling stages or manual implementation. This method scenario is presented in Fig. 2. The benefits of the method are: (1) *Knowledge capture*. Models provide a basis for communication between domain experts, system architects, and software developers. (2) *Reuse and portability*. Reference models and transformations can be reused, providing a basis for software product lining; different target platform transformation sets can be applied to the same coordination model. (3) *Quality*. Model bugs, as well as the respective responsibilities, are separated from implementation bugs – the former having to be corrected only once in the transformation descriptions instead of multiple times in the source code. (4) *Information Hiding*. Transformations encapsulate platform-specific implementation. (5) *Development time reduction*. Reusing models and transformations saves development time.

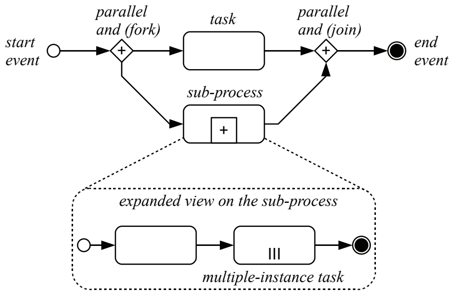

start event
parallel and (fork)
task
parallel and (join)
end event
sub-process
expanded view on the sub-process
multiple-instance task

Fig. 1. DSL example.

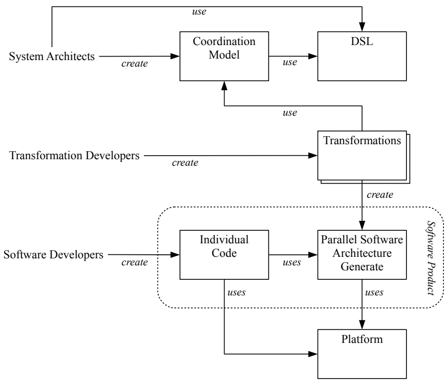

use
System Architects
create
Coordination Model
use
DSL
use
Transformation Developers
create
Transformations
create
Software Developers
create
Individual Code
uses
Parallel Software Architecture Generate
Software Product
uses
uses
Platform

Fig. 2. Method scenario (use of third-party frameworks omitted).

Contributions. (1) *DSL*. A coordination metamodel and a domain-specific concrete syntax that abstracts from the underlying low-level technologies, in particular, from the threading model. Existing DSLs may be reused if matching the DSL requirements. (2) *Method*. An appropriate method to employ the DSL, resulting in a formally specified model as a basis for model-driven development. (3) *Tooling*. Tool support that integrates with existing technologies and complementary tooling. This may comprise a modeling environment, transformation sets to transform coordination models to source code, and support tools for pattern selection and application.

4 Plans for Evaluation and Future Work

Domain model. To refine the domain model, relevant concepts, their (shared and differentiating) features, and additional requirements have to be further identified or revised. This can be done by analyzing reference applications for repetitive patterns [9]. ***Case studies***. We intend to perform case studies on developing an exemplary parallel application, focusing on speedup and scalability. Possible scenarios are: (a) develop the example application based upon a sequential version of the application, (b) if no sequential version is available, develop the example application by using the target platform technologies directly, then re-develop using the proposed method with subsequent domain-specific functional implementation, then compare. Thereby, the case studies have to be carefully designed regarding, for example, knowledge level, learning effects, and favouritism. Also, the application has to be selected carefully considering, for example, source code and documentation availability, implementation language, application size, algorithm complexity, and estimated concurrency, cf. [3]. An application candidate currently regarded is the Desmo-J[3] discrete-event simulation framework. The regarded target platform is Java since it is widely used in industry, supposed to be the first exposure to parallel programming for many programmers, and provides JVM-supported low-level thread management.

Our future work includes: (1) ***Language research***. There are a number of coordination languages that target to reduce complexity by representing parallel program behavior visually [8, 10, 11]. We will examine them for applicability as a DSL. We also suppose that (stereotyped) UML activity diagrams and the Business Process Modeling Notation (BPMN) meet many of the DSL requirements. Both are well-known and may provide a basis for the visual DSL with prospect of broad dissemination, provided that the underlying semantics remain fundamentally intact. (2) ***Model Extension***. We will examine the possibility to extend the domain model to also abstract from the message passing model for distributed computation. (3) ***Tool development***. Tooling may comprise a modeling environment based on the Eclipse Modeling Tools[4] and the openArchitectureWare MDA/MDD generator framework[5], and appropriate model transformations. Also, expert systems may provide assistance for the selection of parallel programming patterns and patterns for thread-safety [12, 13].

---

[3] http://desmoj.sourceforge.net/

[4] http://www.eclipse.org

[5] http://www.openarchitectureware.org/

5 Conclusion

In this position paper we discussed the need for higher abstractions in parallel software development. This need is motivated by the inappropriateness of the threading model since it requires to tame nondeterminism, the lack of parallel programming experience, and the supposed impact of higher-level abstractions on application performance. To satisfy this need, we proposed a model-driven method that regards the coordination model of parallel programs as the first development artifact, and an adequate visual DSL.

References

1. Sutter, H.: The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. Dr. Dobb's Journal **30** (2005)
2. Lee, E.A.: The Problem with Threads. IEEE Computer **39** (2006) 33–42
3. Pankratius, V., Jannesari, A., Tichy, W.F.: Parallelizing BZip2. A Case Study in Multicore Software Engineering. Technical report, University of Karlsruhe (2008)
4. Goetz, B.: Java Concurrency in Practice. 7th Printing edn. Addison-Wesley, Pearson Education, Inc. (2009)
5. IBM alphaWorks: Model-Driven Development Tool for Parallel Applications. (http://www.alphaworks.ibm.com/tech/ngspattern) [Date posted: November 1, 2007; last visited: January 16, 2009].
6. Pllana, S., Benkner, S., Mehofer, E., Natvig, L., Xhafa, F.: Towards an Intelligent Environment for Programming Multi-core Computing Systems. In Eduardo, C., Alexander, M., Streit, A., Träff, J.L., Cérin, C., Knüpfer, A., Kranzlmüller, D., Shantenu, J., eds.: Euro-Par 2008 Workshops - Parallel Processing. Volume 5415., Berlin / Heidelberg, Springer (2009) 141–151
7. Zhang, K., Ma, W.: Graphical Assistance in Parallel Program Development. In: Proc. of the 10th IEEE Intl Symp. on VisualVisual Languages, 1994. Proceedings., IEEE Symposium on. (1994) 168–170
8. Browne, J.C., Dongarra, J., Hyder, S.I., Moore, K., Newton, P.: Visual Programming and Parallel Computing. Technical report, University of Tennessee, Knoxville, TN, USA (1994)
9. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley Professional (2008)
10. Papadopoulos, G.A., Arbab, F.: Coordination Models and Languages. In: Advances in Computers, Academic Press (1998) 329–400
11. Lee, P.A., Webber, J.: Taxonomy for Visual Parallel Programming Languages. Technical report, School of Computing Science, University of Newcastle upon Tyne (2003) Technical Report CS-TR-793.
12. Garbe, H., Janssen, C., Möbus, C., Seebold, H., de Vries, H.: KARaCAs: Knowledge Acquisition with Repertory Grids and Formal Concept Analysis for Dialog System Construction. In: EKAW. (2006) 3–18
13. Gudenkauf, S.: Development of a Prototypical Assistance System for Selecting Software Architectures. Master's thesis, Carl von Ossietzky Universität Oldenburg, Germany (2006)

Towards an Adaptable Framework for Modeling, Verifying, and Executing Medical Guidelines

Janos L. Mathe

Vanderbilt University, Institute for Software Integrated Systems
2015 Terrace Pl., Nashville, TN, 37203, USA
janos.l.mathe@vanderbilt.edu

Abstract. Using evidence-based guidelines to standardize the care of patients with complex medical problems is a difficult challenge in healthcare. Computerized support for implementing such guidelines as formalized patient management protocols has tremendous potential, but two hurdles seem to impede the success of most efforts in doing so. 1) Healthcare professionals usually lack the time to learn a mathematical language thus the formalization becomes a tedious process and 2) most of the guideline modeling languages are simply not flexible enough to handle exceptions that real-life situations introduce in execution time. The use of model-integrated techniques for specifying and implementing guidelines as coordinated asynchronous processes is a promising new methodology for tackling the above mentioned two problems and providing advanced clinical decision support.

Keywords: Executable medical guidelines, Model-based development, Design languages, Domain-specific architectures, Medical information systems, Modeling

1 Introduction

Clinical Information Systems (CIS) are increasingly used by HealthCare Organizations (HCO) to improve the quality and decrease the cost of health care delivery. One of the most desired functionalities of a CIS is to support guideline-based health care delivery, a direction that provides treatment for common illnesses by progressing along standardized, but customizable protocols.

Although using these evidence-based guidelines (GLs) to standardize the care of patients with complex medical problems seems to be a promising approach, their implementation poses great challenges in healthcare. One of the main problems is that treatment protocols are hard to capture. There are three reasons why this is a complex problem. *Firstly*, it is because the operation protocols, policies and GLs of healthcare organizations are hardly ever phrased in a mathematically sound manner, which makes them difficult to translate to well-formed computer languages. *Secondly*, it is because they refer to many interdependent aspects of patient care, which makes it difficult to find the proper abstractions to represent them. These abstractions extend to organizational, human behavioral, security and privacy policy, coordination, deployment and document structure concepts. *Thirdly*, using the abstractions of well-

This work was supported in part by TRUST (Team for Research in Ubiquitous Secure Technology, NSF award number CCF-0424422), the Vanderbilt HealthTech Laboratory and the Vanderbilt Informatics Center.

known formal languages, like the coordination abstractions of the Service-Oriented Architecture (SOA) workflow languages, is not an option for healthcare professionals. They are not used to express treatment logic in the form of business processes.

As shown in Section 2, there are several approaches that have attempted to tackle the challenges mentioned above and recommended formalizations for patient management protocols. It seems though that most of the approaches have not been able to address at least one of the following two hurdles, which, at the end, impede their wide acceptance: 1) *Domain specificity*: healthcare professionals usually lack the time to learn a mathematical language thus the formalization becomes a tedious process that involves the healthcare professionals (the domain experts) and the computer scientists who actually perform the implementation. 2) *Expressiveness*: most of the GL modeling languages are simply not flexible enough to handle exceptions that real-life situations introduce in execution time. As a consequence they need a long test period and can usually only be used in a restricted environment.

Our research focuses on the model-based development of Clinical Information Systems (CIS). The specific problems we investigate are modeling, validation, verification and deployment of treatment protocols using the examples of sepsis and congestive heart failure management.

2 Related work

Formalization of medical knowledge has been an active area of research since the 1960s. Early efforts were focused on creating systems that mapped signs, symptoms and laboratory results to probabilistic estimates of different diagnoses [1]. These systems did not prove to be practical for the everyday practice of medicine. Only with the development of the Electronic Medical Record (EMR) have knowledge-based systems proven to be practical and been adopted by practitioners [2].

Medical knowledge-based systems today focus on Computerized Physician Order Entry (CPOE) and clinical decision support advisory systems [3][4]. CPOE systems depend on comprehensive EMRs to provide means to physicians and nurses to create and execute orders for tests, procedures and medications. A system such as WizOrder, contains multiple advisors that help physicians with issues such as identifying potential adverse drug interactions or determining which combination of medicines might be best for a particular patient [5].

Another area being actively explored is the use of computer-generated alerts. By utilizing rule engines through publish/subscribe models to actively monitor the patient's real time status specified problems trigger alerts [6].

The next area of application of knowledge-based systems is process management. One of the approaches in this category is called Digital Electronic Guideline Library (DeGeL) [7]. DeGeL is a Web-based framework and a set of distributed tools that facilitate gradual conversion of clinical GLs from free text, through semi-structured text, to a fully structured, executable representation. The final representations in DeGeL build on two GL ontologies (Asbru [8] and GEM [9]), both of which are formalized textual languages. Though formal and unambiguous, neither [8] nor [9] are suitable for reading.

AsbruView [10] uses two views, including a Gantt chart-like representation, to overcome the readability issue. Proposed views provide a better overview of the therapy steps than one could have from looking at decision tables. Also the precise temporal constraints of plans become visible, which they do not with flow-charts. Unfortunately their paper does not go into a discussion on how nontrivial examples (ones with not only one single thread) would look like using their formalisms.

Quaglini et. al. in [11] provide a classification of exceptions that might occur during the execution of medical processes, but their approach of defining GLs with the help of Petri Nets does not seem to allow easy GL modeling and fails to avoid state explosion in case of complex protocols.

Serban et. al. [12] and Bäumler et. al. [13] both perform protocol verification by translating Asbru-based GLs to SMV.

3 Proposed Solution

Our goal is to satisfy the heterogeneous and conflicting requirements stated in Section 1. For this, we are proposing to show that the management of complex medical processes, operational policies and GLs of HCOs can be translated to a set of semantically well founded and explicitly defined protocols, and by doing this the formal analysis (validation and verification) and execution of these protocols by a computer become feasible. The use of a formal, well defined representation also promotes maintainability and reusability of the software, as the temporal structure and coordination of the tasks will be captured explicitly. This is in sharp contrast to traditional approaches where this information is hidden in the code. It is also required that the domain experts, the health care professionals (and potentially privacy analysts and system integrators) can understand and adopt these domain specific models.

We believe that by using the Model-Integrated Computing (MIC) [14] approach we will be able to capture modeling abstractions in a form of a domain-specific modeling language (DSML), which will help in satisfying the aforementioned requirements. Facilitating the open-source MIC tool suite – built around the Generic Modeling Environment (GME) [15] – enables layered, graphical, multiple-view system modeling, model transformation, model analysis, execution, and design evolution.

The application of MIC principles and tools casts the creation of clinical decision support and process management systems in the following framework: 1) *Design of modeling language for treatment protocols.* In MIC, modeling languages are formally defined by metamodels [15]. The MIC metaprogrammable tools – designed for modeling, model management and model transformation – are automatically customized by the metamodels. 2) *Modeling treatment protocols*. Using the modeling language defined in Step 1, models of specific treatment protocols are created. These models are a formal representation of GLs that drive the management of clinical processes. The precise semantic foundation of the MIC modeling infrastructure and related tools enable the iterative development (the evolution) of modeling language and represented models as well as the validation and verification of the models against a range of safety, privacy and security related criteria defined as constraints or policies. 3) *Generation of process management systems*. Using the MIC model transformation infrastructure, the verified models are translated into configuration files that customize the generic run-time components of the process management system.

As a plan for setting up a pragmatic evaluation of our approach and in an effort to maximize the impact of our system, we sought a clinical paradigm that was common, clinically important, expensive, and had accepted evidence-based treatment GLs. We found sepsis to be an ideal candidate for our intervention. Sepsis treatment is a complex and extremely information-intensive process performed in intensive care units (ICUs) and emergency departments. Application of GLs that can evolve with accumulated experience and can be customized to the needs of individual patients has important implications in the quality and cost of sepsis care, making sepsis management an attractive initial application target.

We expect our contributions to include i) a formal DSML – called Clinical Process Modeling Language (CPML) – that follows the cognitive path physicians take when dealing with clinical problems, ii) an intuitive, graphical modeling environment for healthcare professionals to capture patient management protocols, iii) an implementation of the sepsis treatment GLs using CPML, iv) translators to the our DSML to simulation and verification tools, v) a generated graphical user interface (GUI) that allows healthcare professionals to follow and control the protocol execution while treating the patient, and vi) an execution engine that implements the operational semantics of CPML and the interfaces to other components (e.g.: live patient data feed). For executing the protocol in a clinical environment our team is also developing a tool configured to manage the treatment of sepsis, called Sepsis Treatment Enhanced through Electronic Protocolization (STEEP) [16].

4 Conclusion – Evaluation and Current Status

The use of evidence-based GLs for managing complex clinical problems has become the standard of practice, but GLs are protocols and not patient care plans. To be truly effective, protocols must be deployed as customized, individualized clinical care plans (protocol instances). Our approach inherently supports this idea by allowing tailoring of the protocol models on a per patient basis if necessary as well as customizing the treatment via the graphical interface of STEEP at the bedside.

We believe it was necessary to develop a DSML since there are no widely accepted visual languages for capturing treatment protocols, and generic software modeling languages, such as UML, were not designed for representing medical knowledge. The use of model-integrated techniques provides several tangible benefits. The protocol models capture medical knowledge explicitly and avoid any ambiguity. The models are easily comprehensible by medical professionals and there is no need for IT personnel to act as intermediaries between the medical and the computer fields.

Furthermore, the protocol models enable knowledge transfer since they are based on the best practice available at the time. Medical students and residents using the tool thereby learn expert knowledge in actual practice. Moreover, the models are expected to be updated on a regular basis as new findings emerge in the medical literature.

Finally, the system facilitates the tracking of protocol execution helping to increase compliance, and improve the protocols themselves by enabling the analysis of the outcomes.

While the medical benefits of our approach are clear, it also presents several advantages from a software development perspective. The software architecture is generic and it is expected to work just as well for other illnesses as it does for sepsis. In fact, we have already begun modeling a new condition, congestive heart failure (CHF), a completely different problem. CHF is a chronic condition with patients typically living at home, as opposed to the acute sepsis where treatment is administered in the ICU. We do not expect any software changes to the main components of the system as we attack different illnesses, just as there are no software changes when the protocols are updated based on new medical knowledge.

Treatment protocols, even if they serve only as GLs in patient management, are safety critical and their validation and verification is an essential part of the protocol specification process. We are currently evaluating the expressiveness of the CPML language by experimenting with the semantics. We believe that finding the proper behavioral semantics is crucial, as it will not only allow a comparison with other languages, but define what properties can be verified, how well combined protocols behave when executed parallel, and also how well can the bounded tracking (exception handling) of protocols be implemented.

The project started in 2007 as a collaborative effort between Vanderbilt School of Engineering and Vanderbilt Medical Center to apply advanced model-integrated computing techniques to the management of complex clinical processes. The team has completed the beta version of the generic software infrastructure and the sepsis treatment protocol models resulting in the STEEP toolset. We are in the process of performing a carefully coordinated, multi-phase experiment to evaluate the presented approach in terms of usability and effectiveness, which will involve evaluation of the protocol logic, the patient management interface, and the effects of the altered clinical workflow. Phase one of the clinical tests has already started in two ICUs at Vanderbilt to establish the baseline for the comparative study. The entire STEEP toolset will be introduced later this year. We anticipate showing that the application will 1) decrease time to detection of patients with developing sepsis, 2) improve physician compliance with evidence-based standards, and 3) result in improved clinical outcomes for patients, including ICU and total inpatient length of stay, number of organ system failures, and mortality rate. Once the approach is validated for sepsis, the technology and corresponding tools will be applied to the treatment of other serious illnesses.

5 References

1. Miller RA. Medical diagnostic decision support systems—past, present, and future: a threaded bibliography and brief commentary. J Am Med Inform Assoc. 1994; 1:8–27.
2. Stead WW, Hammond WE. Computer-based medical records: The centerpiece of TMR. MD Comput. 1988; 8:48-62.

3. Hunt DL, Haynes RB, Hanna SE, Smith K. Effects of computer-based clinical decision support systems on physician performance and patient outcomes: a systematic review. JAMA. 1998; 280:1339–46.
4. Bates DW, Leape LL, Cullen DJ, et al. Effect of computerized physician order entry and a team intervention on prevention of serious medication errors. JAMA. 1998; 280:1311–6.
5. Oliven A, Michalake I, Zalman D, Dorman E, Yeshurun D, Odeh M. Prevention of prescription errors by computerized, on-line surveillance of drug order entry. Int J Med Inform 2005; 74(5):377-86.
6. Heather E. Finlay-Morreale, Clifton Louie, and Pearl Toy. Computer-generated Automatic Alerts of Respiratory Distress after Blood Transfusion. J. Am. Med. Inform. Assoc. 2008; 15(3):383-385.
7. Yuval Shahar, Erez Shalom, Alon Mayaffit, Ohad Young, Maya Galperin, Susana Martins, and Mary Goldstein. A Distributed, Collaborative, Structuring Model for a Clinical-Guideline Digital-Library. AMIA Annu Symp Proc. 2003; 2003: 589–593.
8. Seyfang, A.; Miksch, S.; Marcos, M.: Combining Diagnosis and Treatment using Asbru, International Journal of Medical Informatics, pp. 49-57, 68 (1-3), December 2002.
9. Shiffman RN, Karras BT, Agrawal A, Chen R, Marenco L, Nath S. GEM: A proposal for a more comprehensive guideline document model using XML. J Am Med Informatics Assoc 2000;7:488-98
10. Robert Kosara and Silvia Miksch. Metaphors of Movement: A Visualization and User Interface for Time-Oriented, Skeletal Plans. In: Artificial Intelligence in Medicine, Special Issue on Information Visualization in Medicine, pp. 111-131, Volume 22, Number 2, 2001.
11. Quaglini S, Stefanelli M, Lanzola G, Caporusso V, Panzarasa S. Flexible guideline-based patient careflow systems. Artif Intell Med. 2001 Apr;22(1):65-80.
12. Radu Serban, Annette ten Teije, Frank van Harmelen, Arjen Hommersom, Peter Lucas, Perry Groot, Albert Jovell, Sergi Blancafort, Jolanda Wittenberg, Kitty Rosenbrand, Joyce van Croonenborg. Protocure II, Workpackage 5, Deliverable 5.2: How to use model-checking for critiquing using medical guidelines. July 22, 2006
13. S. Bäumler, M. Balser, A. Dunets, W. Reif, J. Schmitt. Verification of Medical Guidelines by Model Checking -- A Case Study. Proceedings of the 13th SPIN Workshop on Model Checking Software, Vienna, Austria
14. Sztipanovits, J., Karsai, G.: “Model-Integrated Computing”, IEEE Computer,V.30. pp. 110-112, April, 1997.
15. Ledeczi, A.; Bakay, A.; Maroti, M.; Volgyesi, P.; Nordstrom, G.; Sprinkle, J.; Karsai, G.: Composing domain-specific design environments, IEEE Computer, Nov. 2001, Page(s): 44 –51
16. Mathe, J., J. B. Martin, P. Miller, A. Ledeczi, L. Weavind, A. Nadas, A. Miller, D. J. Maron, and J. Sztipanovits, "A Model-Integrated Guideline-Driven Clinical Decision Support System", IEEE Software, Special issue on Domain-Specific Languages & Modeling, 2009