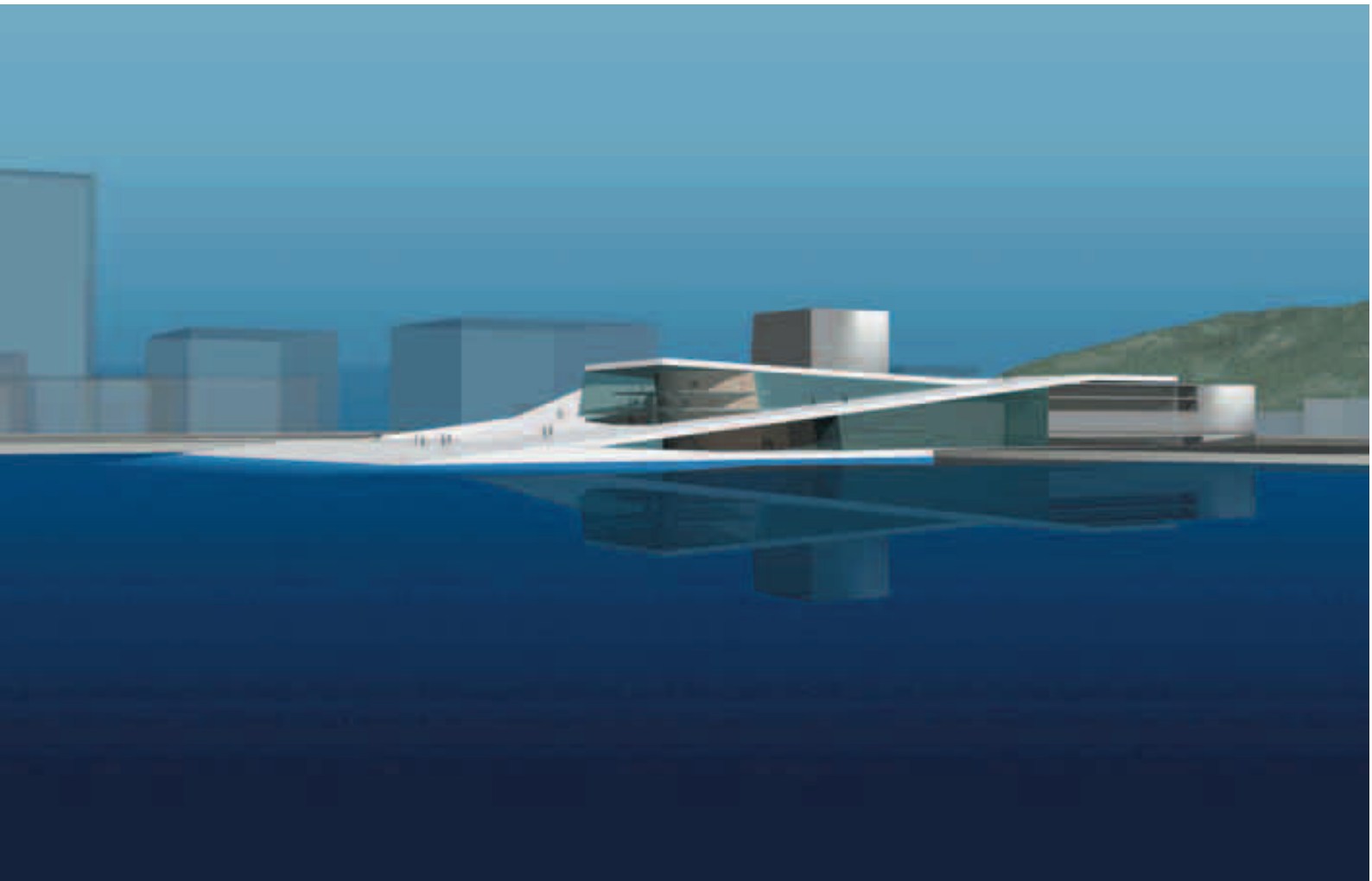
ACM/IEEE 13th International Conference
on Model Driven Engineering
Languages and Systems
Oslo, Norway October 3–8, 2010

Preliminary Proceedings of the

MODELS 2010
Doctoral Symposium



Preface

This volume contains the papers presented at Doctoral Symposium of the 13th ACM IEEE International Conference (MODELS 2010) held on October 4th, 2010 in Oslo.

The goal of the Doctoral Symposium is 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 aims to support students by providing independent and constructive feedback about their already completed and, more importantly, planned research work.

We received 28 submissions, each submission was reviewed by 3 programme committee members. The committee decided to accept 10 papers for presentation in the symposium.

We thank the members of the programme committee for their help in supporting young researchers in shaping their scientific work, the MODELS 2010 Organizing Committee for hosting the Doctoral Symposium, as well as the EasyChair team for providing the system to coordinate the review process.

September 2010

Brian Elvesæter
Bernhard Schätz

Conference Organization

Programme Chairs

Brian Elveæter
Bernhard Schätz

Programme Committee

Ruth Breu
Betty Cheng
Juergen Dingel
Brian Elveser
Gregor Engels
Robert France
Jeff Gray
Gerti Kappel
Gabor Karsai
Joost-Pieter Katoen
Ingolf Krueger
Jochen Küster
Pieter Mosterman
Ivan Porres
Alexander Pretschner
Bernhard Rumpe
Bernhard Schätz
Jonathan Sprinkle
Friedrich Steimann
Ketil Stoelen
Stefan Wagner

External Reviewers

Birgit Penzenstadler
Jan Olaf Blech
Mario Gleirscher
Markus Look
Steven Völkel
Thomas Kurpick
Tim Gülke

Table of Contents

Towards the Verification of State Machine-to-Java Code Generators for Semantic Conformance

Lukman Ab Rahim

School of Computing and Communications, InfoLab21,
Lancaster University, Lancaster LA1 4WA, UK,
abrahim@comp.lancs.ac.uk

Abstract. Verifying code generators is as important as verifying the generated code itself. One of the challenges in verifying code generators is the complexity of the code generator. This project proposes an approach to verify semantic conformance of UML State Machine-to-Java code generator, which is a complex code generator because of the complexity of the UML State Machine notation and the dissimilarity between source and target languages. The approach uses a model checker to verify the generated code that is annotated by assertions. The assertions represent the UML State Machine semantics.

1 Introduction

Verification is a crucial activity in software development. When code generators are used to develop software, the correctness of the code generator is as important as the correctness of the software itself. A faulty code generator means a faulty generated code.

This project proposes an approach, we call Annotation-Driven Model Checking (ADMC), to verify code generators. We restrict this project to UML State Machine-to-Java code generators because of these reasons: 1) UML is a commonly used modelling language, and 2) UML State Machine notation has a well define semantics.

Verifying a code generator is a challenging endeavor because it involves three major tasks: 1) verifying syntactic correctness of the generated code, 2) verifying preservation of source model's semantics, and 3) verifying conformance to the modelling language semantics. The first task can be easily achieved by using a compiler. The second task can be accomplished by comparing the behaviour of the source model with the behaviour of the generated program. The third task checks the behaviour of the generated program with the semantics of the modelling language. Verifying preservation of the semantics of the source model is arguably the hardest but verifying conformance is challenging nontheless. This project focuses on verifying code generators for semantic conformance.

Verifying a code generator may also be difficult because of the complexity of the code generator. A code generator is complex due to the complexity of the

source model's language. For example, UML State Machine notation has Composite and Submachine state that can be entered in various ways. Each entry method needs to be catered and the semantics followed by the code generator. The complexity of a code generator also depends on the similarity between the source model's language and the generated code's language. For instance, generating a Java code from UML class diagram is simpler because both languages support the concept of classes, attributes and methods. However, generating from UML state machine is more difficult because Java does not explicitly support the concept of states, events and transitions.

The main challenges of this project is to come up with an approach that is: 1) applicable to verify semantic conformance of different code generators, 2) usable during development of realistic systems that are modeled using state machine, and 3) advantageous to the software verification activity. The first challenge is important because we do not want to propose an approach that can only be used for specific code generators. The second challenge is a problem of scalability since we are using model checking and using this method tends to suffer from the state-space explosion problem when verifying complex concurrent systems. The final challenge is to produce an approach that is of value to the verification of systems.

The following section discusses existing work related to this project. Section 3 explains our approach to verify semantic conformance, and Section 4 explains our plan to evaluate the approach. The final section desribes the project's current status and contributions.

2 Related Work

There has been a wide body of work on verifying that model transformations are semantics-preserving – in the sense that the application-specific model semantics hold in the generated model (or code). For example, Varro and Pataricza [1] use model checking to check properties in both source and target models. If both models have the intended properties, the transformation is said to be semantically correct. Staats and Heimdahl [2] use a similar approach when verifying semantic correctness of a Simulink-to-C code generator.

To ensure semantic preservation, Barbosa et al. [3] propose an extension to the MDA four layered architecture. They propose the addition of a semantic metamodel and model into the architecture. The verification is carried out for both static (well-formedness contraints) and dynamic (model behaviours) semantics. The approach uses a formal checker to verify the conformance of both static and dynamic semantics based on the semantic metamodel. This approach requires a lot of effort because the semantic models need to be created for each source and target models.

Chaki et al. [4] and Pnueli et al. [5] propose the use of an approach based on proof-carrying code (PCC) [6, 7] but where model checking is combined with the use of a theorem prover, which is normally used in PCC. Autofilter [8], AutoBayes [9], and AutoCert [10] are three tools that apply the principles of PCC

to verify code generators. These tools use theorem provers guided by annotations to verify the preservation of certain semantic properties. Autofilter and AutoBayes are restricted to the domain of geometric state estimation and data analysis problems respectively. AutoCert improves on Autofilter and AutoBayes by being a domain independent tool and provides a model-driven mechanism to generate the annotations.

Ensuring the semantic conformance of Java programs generated from UML state machines are investigated by Blech et al. [11], and Pintér and Majzik [12]. Blech et al. ensure the conformance of automatically generated Java programs by proving the Isabelle/HOL formalizations of the Java programs. Pintér and Majzik ensure the conformance of the Java programs by generating the Java programs from proven state machines. They proof the correctness of the state machine by translating it to Extended Hierarchical Automata (EHA) and check the EHA using the SPIN model checker.

3 Annotation-Driven Model Checking

Annotation-Driven Model Checking (ADMC) is an approach we created to verify semantic conformance of State Machine-to-Java code generators. The basic idea of the approach is to verify the code generator by checking the behaviours of the generated code using a model checker. The generated code's behaviours are checked using assertions, which are added into the generated code. The assertions capture the semantics of state machines as a statement with boolean flags and auxiliary methods[1]. Using assertions made of boolean flags have been proven sufficient to represent the semantics as being shown in [13] where the authors also use boolean flags in formalizing the semantics.

ADMC is an indirect approach to verifying code generators. We chose this indirect method to avoid the complexity of the model transformations. Furthermore, model checking semantic conformance can be practically achieve (using available tools) by checking if the source language semantics are preserved in the generated code. Due to ADMC being an indirect approach, ADMC does not guarantee semantic conformance of all possible state machines. Instead it only guarantees semantic conformance for the state machine use in the verification. Verifying semantic conformance of code generators is more practical this way considering the complexities of UML State Machine notation.

Figure 1 shows the components in ADMC. The Annotation Transformation (AT) is a set of transformation rules that will generate the Verification Component (VC). The AT is unique to the code generator being verified because each code generator translates the UML State Machine notation differently. As a result of these differences, the form and the place to add the assertions are different. To create the AT, how the code generator translates the UML State Machine notation must be understood. The translation can be understood by studying the transformation rules, or if the rules are not available, from the generated code.

[1] Auxiliary methods are used to simplify the assertions.

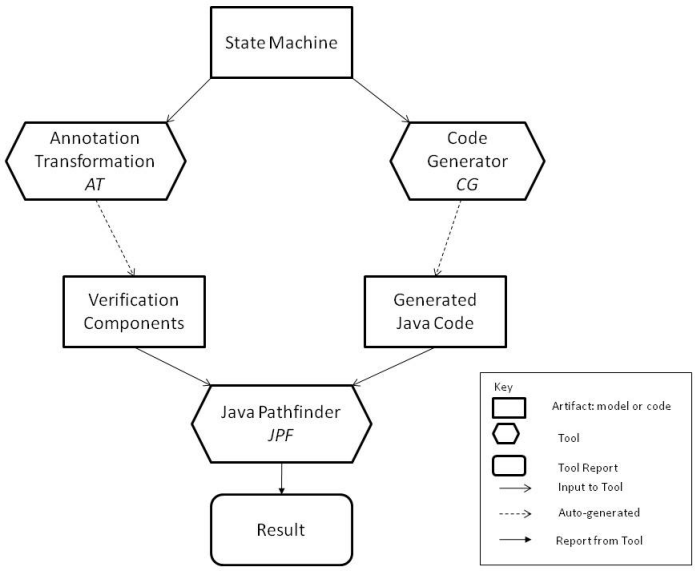


Fig. 1. Annotation-Driven Model Checking

The VC is a Java code that extends the generated code with assertions to verify the generated code. The assertions are in the VC (not in the generated code) because we want ADMC to verify commercial code generators. If the assertions are to be in the generated code, the code generator needs to be modified, and most of the time this is impossible because the transformation rules are not available.

The generated code and the VC will be the input to the Java Path Finder (JPF) [14] model checker. JPF model checks both codes and the VC, and returns the result. The result can be a counter example (if the model checker detects any assertion violation) or a conformation on the success of the model checking. The counter example will show which semantic is violated, where in the code that violates this semantic and the execution trace that leads to the violation. Using these information, we can identify the problematic transformation rules.

The ADMC appproach can be used in three situations: 1) use by tool developers to verify their code generator, 2) use by tool users to evaluate conformance of code generators, and 3) use by testers to verify generated code provided by a different party. When use by tool developers, the AT is developed once for the current release and reuse for future releases. The same advantage is obtained by the testers where they can reuse the AT to verify different generated code created from the same code generator. Tool users can use ADMC to select the best code generators that conform to their properties of importance. These properties may be the tool users' interpretation on the open semantics (known as variation points in UML) of the modelling language. For instance, the UML specification does not specify how the event pool is implemented. Thus, the tool users

may implement the event pool as a queue or a stack, and verify the semantics associated to the two data structures.

4 Evaluation Plan

Three evaluations have been planned for this project. The first evaluation evaluates the applicability of the approach to verify different code generators. This evaluation is conducted by using ADMC to verify several commercial code generators. Two commercial code generators, Rhapsody [15] and Visual Paradigm [16], have been verified.

The second evaluation evaluates the scalability of the approach for the development of realistic systems. One of the way to use ADMC is to verify generated code provided by other parties. Therefore, when confronted with a complex system that might suffer from state space explosion, ADMC must be able to handle this situation. The outcome of this evaluation is a set of state-space reduction methods that can be applied with ADMC, and how to apply them. We plan to use three realistic systems as case studies: 1) Center TRACON Automation System (a system to update weather information for air traffic control), 2) Software Define Radio (a radio communication system), and 3) VoiceYourView (a system that captures public opinions and concerns).

The third evaluation attemps to measure how much ADMC reduces the verification effort during system testing. We plan to investigate what are the activities during verification and how much effort each activity needs. We also need to identify in which activity one verifies semantic conformanace and how much ADMC reduces the effort in this activity. We have not decided on how to measure effort.

5 Contributions and Current Status

This project proposes an approach to verify semantic conformance of code generators. The approach uses model checking to verify the source language semantics, which are specified as assertions. The approach is used to verify UML State Machine semantics for State Machine-to-Java code generators. However, we believe the approach can be used for other modelling and source code languages. Verifying code generators for other languages using ADMC requires the availability of a model checker suitable for the properties and the generated code.

Another contribution is a model-driven implementation to a selected few state-space reduction methods. These methods are necessary since ADMC uses model checking (normally associated with scalability problem due to state space explosion) to verify the generated code.

Currently we have develop the approach and verify two commercial code generators. From these verifications, both code generators do not fully conform to the semantics of UML State Machine.

References

1. Varró, D., Pataricza, A.: Automated formal verification of model transformations. In Jürjens, J., Rumpe, B., France, R., Fernandez, E.B., eds.: CSDUML 2003: Critical Systems Development in UML; Proceedings of the UML'03 Workshop. Number TUM-I0323 in Technical Report, Technische Universität München (September 2003) 63–78
2. Staats, M., Heimdahl, M.: Partial Translation Verification for Untrusted Code-Generators. In: International Conference on Formal Engineering Methods (ICFEM'08), Springer (2008) 226–237
3. Barbosa, P.E.S., Ramalho, F., de Figueiredo, J.C.A., dos S. Junior, A.D.: An extended mda architecture for ensuring semantics-preserving transformations. In: 32nd Annual IEEE Software Engineering Workshop, 2008. (October 2008) 33 –42
4. Chaki, S., Ivers, J., Lee, P., Wallnau, K., Zeillberger, N.: Model-Driven Construction of Certified Binaries. In: 10th International Conference, MODELS 2007, Springer (2007) 666–681
5. Pnueli, A., Shtrichman, O., Siegel, M.: The Code Validation Tool CVT: Automatic Verification of a Compilation Process. Software Tools for Technology Transfer **2** (1998) 192–201
6. Necula, G.C.: Proof-carrying Code. In: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '97), ACM (1997) 106–119
7. Colby, C., Lee, P., Necula, G.C.: A Proof-Carrying Code Architecture for Java. In: Computer Aided Verification, Springer-Verlag (2000) 557–560
8. Denney, E., Fischer, B., Schumann, J., Richardson, J.: Automatic Certification of Kalman Filters for Reliable Code Generation. In: IEEE Aerospace Conference, IEEE (2005) 1–10
9. Schumann, J., Fischer, B., Whalen, M., Whittle, J.: Certification Support for Automatically Generated Programs. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, IEEE (2003) 1–10
10. Denney, E., Fischer, B.: Generating Customized Verifiers for Automatically Generated Code. In: Proceedings of the 7th International Conference on Generative Programming and Component Engineering (GPCE '08), ACM (2008) 77–88
11. Blech, J.O., Glesner, S., Leitner, J.: Formal Verification of Java Code Generation from UML Models. In: 3rd International Fujaba Days 2005-MDD in Practice. (2005) 49–56
12. Pintér, G., Majzik, I.: Automatic Code Generation Based on Formally Analyzed UML Statechart Models. In: Formal Methods in Railway Operation and Control Systems, Budapest, Hungary (May 2003) 45–52
13. Mostafa, A.M., Ismai, M.A., El-Bolok, H., Saad, E.M.: Toward a formalization of uml2.0 metamodel using z specifications. In: Proceedings of the 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, Qindao, China, IEEE (July 2007) 694–701
14. Visser, W., Păsăreanu, C.S., Khurshid, S.: Test input generation with java pathfinder. In: ISSTA '04: Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis, New York, NY, USA, ACM (2004) 97–107
15. IBM: Rhapsody. Website http://www-01.ibm.com/software/awdtools/rhapsody/.
16. Visual Paradigm International: Visual paradigm. Website http://www.visual-paradigm.com/.

Reuse in Modelling Method Development based on Meta-modelling*

Alexander Bergmayr

Faculty of Computer Science, University of Vienna
Bruenner Strasse 72, 1210 Vienna, Austria
ab@dke.univie.ac.at

Abstract. Systematic reuse is one obvious step to cope with the complexity and high effort involved in modelling method development. This principle, however, is still rarely adopted since today's development platforms lack adequate mechanisms promoting reusability. To alleviate this deficiency, an ontology-based description framework, providing operators for the extraction, discovery, match and assembly of *Method Design Fragments* is proposed. In this way, potential reusable artifacts are regarded as explicit abstractions of existing method designs and become operationalized through an extensible reuse library leading to increased productivity during method development and higher quality results with reasonable effort.

Keywords: Modelling Method Development, Design Fragment, Reuse

1 Introduction

The construction of new modelling methods [1], providing the necessary concepts capable to capture the relevant domain knowledge represented through models, is one major challenge in Model-Driven Engineering (MDE) [2]. Modelling method development is a complex task and the effort involved is usually extensive [3]. One obvious step to mitigate this complexity and high effort is to emphasize the *reuse* during method development. Systematic reuse in the development of modelling methods is, however, rarely adopted. This is, among others, due to the fact that in contrast to contemporary 'general-purpose' languages most formalisms of today's platforms for modelling method development lack adequate mechanisms promoting reusable artifacts. As a consequence, method engineers can hardly draw on support by method reuse libraries, thus modelling methods become realized usually from scratch without considering existing ones that might entirely or partly be reused. Support for extracting these reusable method artifacts in a form to allow discovering them, identifying composable elements thereon, and their assembly leverage the reuse in modelling method development. To explore this assumption, the rest of this work discusses the notion of *Method Design Fragments* inspired by Brinkkemper [4] but interpreted in terms of concerns known from aspect-oriented modelling.

* This work has been partly supported by the EU project PlugIT (ICT-231430) and the Austrian Federal Ministry of Science and Research (BMWF).

2 Proposed Solution

When adopting the principle of reuse during development of new modelling methods, higher quality results with reasonable effort can be expected. Consequently, an ontology-based framework for establishing an extensible library of reusable method design fragments is outlined (cf. Fig.1 (A)). Core activities supported involve the *Extraction* of such fragments from existing modelling methods, their *Discovery* in a library relating and maintaining them, the *Match* between (retrieved) fragment elements and the *Assembly* of matched ones to provide a single unit for reuse or their amalgamation resulting in the pursued method.

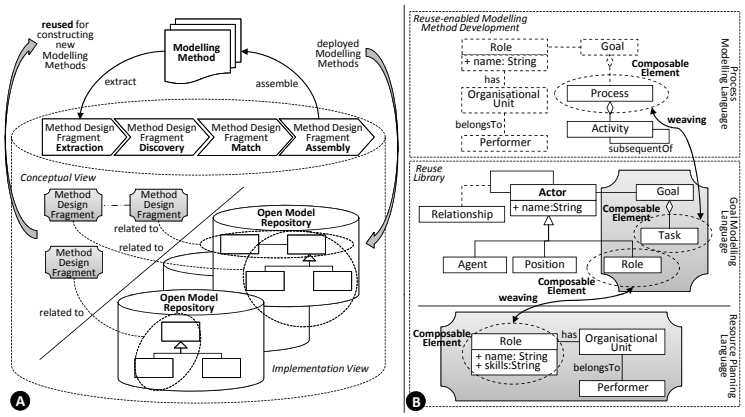


Fig. 1. Framework for reusable Method Design Fragments (A), Example (B)

In more detail, fragment extraction is operationalized through abstraction [5] involving to select or generalize particular method artifacts, or filter properties assigned to them. Considering the example on the right part of Fig. 1, a simple process modelling language (as one major part of a prospective method), comprising `Activity` and `Process` extended with concepts from the area of goal modelling and resource planning is going to be developed. In fact, different concerns become composed into one language. A reuse library contains the language constructs for the aspired extension. Whereas the applied selection on the resource planning language comprise all constructs, the `skill` property of the `Role` element has been filtered. In terms of the goal modelling language, the selection prunes the former while preserving compatibility [6] as (instance) models conforming to the extracted fragment still conform to the original language. This compatibility is, however, not always intended (e.g., deep inheritance hierarchies) hence, flexibility of extraction operators is required during application. To allow discovering reusable fragments, the focus is turned on concepts, relationships and properties underlying a method, i.e., the ontological viewpoint of a methods's conceptualization is going to be addressed. Accordingly, an ontology-based approach for the description of fragments is aspired not only as a vehicle

for abstracting over potential multiple model repositories to allow high-level queries, i.e., a data integration scenario but also to benefit from capabilities for consistency or membership validation. The latter is of particular interest for validating extraction results when, for instance, compatibility is expected as mentioned before. On the other hand, consistently assembled concepts admit their instantiation. Before addressing the assembly, the concepts involved need to be identified. Considering the example, the `Role` constructs, and the `Process` and `Task` constructs are candidates for assembly. Matching techniques, available for ontologies support the finding of concept correspondences even if syntactical matches could not have been derived as it is the case for `Process` and `Task`. Finally, the assembly supports *weaving* matched fragments following a certain strategy (cf. [7]). In the example, constructs highlighted as *composable* became assembled using a merge operation to form a single construct.

3 Expected Contribution

By the proposed approach for reuse in modelling method development, three main contributions are expected.

Reusable Business Process Modelling Design Fragments. To allow utilizing a concrete library comprising different reusable artifacts, the first main contribution is extracting and cataloging design fragments. To initiate this investigation, the area of business process modelling is intended to be considered due to the diversity of available formalisms with different expressiveness but also concepts that are common to all of them. In this way, a business process modelling reuse library becomes established, indeed also useful for experiments to validate the proposed approach.

Operators for Method Design Fragment Extraction and Assembly. The adaption or extension of existing, or the building of new operators is required allowing their applicability for a broad range of method development scenarios when considering current formalisms (cf. ECORE, EMOF or KM3, to mention just a few) for meta-modelling and intractable diversities (e.g., different terminology for concepts identical in meaning) that come along with them. Since most formalisms, however, adhere to class-based concepts (cf. [8]), the provisioning of generic operators is feasible. Considering extraction operators, process models as discussed in [9] may serve as a source for deriving requirements while on the other hand, literature in the area of meta-model composition (as further discussed in Section 4) and approaches applying aspect-oriented techniques for weaving language concerns (cf. [10]) pose grounding for further investigation on assembly operators.

Description Framework for Reusable Method Design Fragments. Extracted method design fragments need to be described and properly arranged in a catalog establishing a reuse library utilized by engineers during method design. To support the construction of such a library, the development of an ontology-based description framework is the third main contribution. In this way, potential reusable artifacts implemented by arbitrary formalisms for method development become explicit through method design fragments although the utilized formalisms may not necessarily provide reuse mechanisms. The advocated

ontology-based approach allows anchoring these fragments to the body of domain knowledge [11] a method is (explicitly or implicitly) committing to. Sharing the analysis results of a particular domain across methods with similar needs, requires carefully considering their intended meaning which may include operational semantics particularly when methods are behavioral in nature. Available literature aiming to couple ontologies and meta-modelling in general (cf. [12] a workshop series dedicated to ontologies in MDE), and proposals for capturing method concepts in terms of ontological descriptions (cf. [13]) and model-based (cf. [14]) as well as ontology-based search (cf. [15]) in particular are of relevance.

4 Related Work

Work that is related to the proposed approach is discussed according to the reuse principle as well as the consideration of ontologies in modelling method development.

Modelling Language Reuse. In the work of [16] a template-based reuse approach to alleviate reoccurring language design problems is advocated. Templates capture these design problems and allow their instantiation. In fact, instantiated templates become embedded in a language reusing them. Another approach for reuse comprise the composition of existing language constructus as suggested in [17] and [18]. Whereas the former approach is inspired by UML's package merge and particularly address the generalization of reused elements of different packages, the latter suggests components in terms of UML to provide reuse. Their actual composition is based on coupling required and provided interfaces related to concrete elements of the reused components. Another compositional approach is proposed by [19] discussing the concept of modularity in terms of textual domain-specific languages defined by a grammar-based approach. Considering proposals relying on aspect-orientation, in [20] extension mechanisms for EMF-based meta-models are discussed. Operators applied in these approaches to foster reuse is in this work subsumed under the term *assembly*. Beside the assembly, the *extraction* of reusable fragments is still underrepresented in current approaches.

Ontologies in Language Development. In [21] a domain-specific language (DSL) development framework relying on class-based meta-concepts enriched by description logics (DLs) using the Web Ontology Language (OWL) is proposed. Another alternative approach to the traditional class-based development of modelling languages is [22] proposing a meta-language based on a philosophical theory although a reference implementation is missing. While both approaches expose insights into ontology-based meta-modelling, their focus is not on reuse.

5 Planned Evaluation

The proposed approach will be evaluated based on a three-level strategy.

Assessment of Reuse Library. To evaluate the utility of a reuse library for business process modelling as described in Section 3 both, the technical as well as the human viewpoint will be considered according to [23], supporting criterion for each viewpoint (e.g., precision and recall or difficulty of use and transparency, resp.). Regarding the human viewpoint, empirical studies based on

questionnaires are planned to be conducted with partners of the plugIT[1] project, proposing a model-based approach for the currently imposed practice of a tight coupling between business and IT [24], and members of an emerging initiative advocating 'open' models[2].

Controlled experiments with Reusable Fragments. Considering this criterion, business informatics students are instructed to evaluate the applicability of operators for extraction and assembly, and fragments extracted for the business process modelling area. The task involves to reassemble fragmented methods and extend assembled ones with new constructs by relying on a predefined baseline. The exploitation of the reuse capability assumed needs to be evaluated based on corresponding metrics [25] (e.g., ratio of reused to total size of lifecycle constructs and properties).

Increase of Productivity and Quality. Considering this criterion, several small development teams composed of students with similar experience are going to realize an excerpt of a modelling method specification. One half of the teams need to implement the specification from scratch, while the other half becomes access to a library consisting of potential reusable method design fragments. The evaluation is conducted based on empirical studies, investigating the productivity and quality of reuse in modelling method development while applying corresponding reuse metrics [25] (e.g., ratio of reused to manually implemented 'code' or error rates, resp.).

6 Current Status and Next Steps

This research effort is still in an initial state currently addressing the elaboration of an environment that allows experimenting with reuse in modelling method development. While the scope is currently on structural language concerns, forthcoming experiments need to be extended on entire method definitions.

References

1. Karagiannis, D., Kühn, H.: Metamodelling Platforms. In: EC-WEB '02: Proceedings of the Third International Conference on E-Commerce and Web Technologies, London, UK, Springer-Verlag (2002) 182
2. Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., Völkel, S.: Design Guidelines for Domain Specific Languages. In: 9th OOPSLA Workshop on Domain-Specific Modeling (DSM' 09). (October 2009)
3. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4) (December 2005) 316–344
4. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information and Software Technology **38**(4) (1996) 275–280
5. Kramer, J.: Is abstraction the key to computing? Commun. ACM **50**(4) (2007) 36–42

[1] http://www.plug-it-project.eu
[2] http://www.openmodel.at

6. Sen, S., Moha, N., Baudry, B., Jézéquel, J.M.: Meta-model Pruning. In: 12th International Conference on Model Driven Engineering Languages and Systems, Berlin, Heidelberg, Springer-Verlag (2009) 32–46
7. Reddy, Y.R., Ghosh, S., France, R.B., Straw, G., Bieman, J.M., McEachen, N., Song, E., Georg, G.: Directives for Composing Aspect-Oriented Design Class Models. Transactions on Aspect-Oriented Software Development I **3880** (2006) 75–105
8. Sprinkle, J., Rumpe, B., Vangheluwe, H., Karsai, G.: Metamodelling: State of the Art, and Research Challenges. In: Model-Based Engineering of Embedded Real-Time Systems. Springer (2010) 59–78
9. Henderson-Sellers, B., Ralyté, J.: Situational Method Engineering: State-of-the-Art Review. Journal of Universal Computer Science **16**(3) (2010) 424–478
10. Schauerhuber, A., Wimmer, M., Schwinger, W., Kapsammer, E., Retschitzegger, W.: Aspect-Oriented Modeling of Ubiquitous Web Applications: The aspectWebML Approach. In: 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems. (2007) 569–576
11. Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: What Are Ontologies, and Why Do We Need Them? IEEE Intelligent Systems **14**(1) (1999) 20–26
12. Ghosh, S., ed.: Second Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2009). Volume 6002. (2010)
13. Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. Model Driven Engineering Languages and Systems **4199** (2006) 528–542
14. Lucrédio, D., M. Fortes, R.P., Whittle, J.: MOOGLE: A Model Search Engine. In: 11th International Conference on Model Driven Engineering Languages and Systems, Berlin, Heidelberg, Springer-Verlag (2008) 296–310
15. Uschold, M., Benjamins, V.R., Ch, B., Gomez-perez, A., Guarino, N., Jasper, R.: A framework for understanding and classifying ontology applications. In: IJCAI99 Workshop on Ontologies. (1999) 16–21
16. Emerson, M., Sztipanovits, J.: Techniques for metamodel composition. In: 6th OOPSLA Workshop on Domain-Specific Modeling. (2006) 123–139
17. Blanc, X., Ramalho, F., Robin, J.: Metamodel Reuse with MOF. Model Driven Engineering Languages and Systems **3713** (2005) 661–675
18. Weisemöller, I., Schürr, A.: Formal Definition of MOF 2.0 Metamodel Components and Composition. In: 11th International Conference on Model Driven Engineering Languages and Systems, Berlin, Heidelberg, Springer-Verlag (2008) 386–400
19. Krahn, H., Rumpe, B., Völkel, S.: MontiCore: Modular Development of Textual Domain Specific Languages. In: TOOLS (46). (2008) 297–315
20. Reina Quintero, A.M., Torres Valderrama, J.: Using Aspect-orientation Techniques to Improve Reuse of Metamodels. Electron. Notes Theor. Comput. Sci. **163**(2) (2007) 29–43
21. Walter, T., Silva Parreiras, F., Staab, S., Ebert, J.: Joint Language and Domain Engineering. In: ECMFA 2010. Volume 6138., Springer (2010) 321–336
22. Laarman, A., Kurtev, I.: Ontological Metamodeling with Explicit Instantiation. In: Software Language Engineering. Volume 5969 of Lecture Notes in Computer Science., Heidelberg, Springer Verlag (January 2010) 174–183
23. Mili, A., Mili, R., Mittermeir, R.T.: A survey of software reuse libraries. Ann. Softw. Eng. **5** (1998) 349–414
24. Woitsch, R., Karagiannis, D., Plexousakis, D., Hinkelmann, K.: Business and IT alignment: the IT-Socket. E & I Elektrotechnik und Informationstechnik **126** (2009) 308–321
25. Frakes, W., Terry, C.: Software reuse: metrics and models. ACM Comput. Surv. **28**(2) (1996) 415–435

Rearrange: Rational Model Transformations for Performance Adaptation

Mauro Luigi Drago

Politecnico di Milano
DeepSE Group - Dipartimento di Elettronica e Informazione
Piazza Leonardo Da Vinci, 32 - 20133 Milano, Italy
drago@elet.polimi.it
WWW home page: http://home.dei.polimi.it/drago

1 Introduction and Problem Setting

Software is now pervasive and plays an important role in everyday life. People rely on hand-held applications, such as GPS navigation systems, to take decisions. Critically safe applications, such as avionics or driving control systems, are wide spread and neither failures nor unresponsiveness are tolerated. In the development of such systems, assessment of non-functional properties, even in the early stages of design, has been recognized as mandatory and useful.

In this direction, many Model Driven Engineering (MDE) approaches for performance prediction have been developed [1][2][3]. Well established techniques can be used to generate performance models from abstract models of systems, to compute performance metrics, and to check whether requirements are met. However, poor support is provided to interpret results, usually given at a low abstraction level, and to identify appropriate solutions when requirements are not met. It is still entirely up to the engineer and to its expertise to understand what numbers mean and what changes to apply.

The need of feedback provision methodologies is even more compelling if we consider that many performance problems can discovered only at run time, when real usage information is gathered or assumptions about the environment and users evolve. If changes occur frequently and fast, software must evolve at the same speed and by preserving service provision. For critical applications, it may be neither feasible nor acceptable to adapt by stopping an application, by re-entering the software development cycle, and by waiting for a new release to be shipped.

Software adaptation, both at design time and at run time, is however everything but easy, even in the restricted area of performance engineering. Many research challenges are still open, especially for run time evolution [4]: changes in the environment must be sensed and propagated across abstraction layers, problems and solutions must be formally specified, fast reflective capabilities are needed to consistently select and apply countermeasures.

Our research project focuses on these research issues and, more formally, on the problem of *how to close the feedback loop* from a performance engineering perspective. As the distinction between design time and run time is becoming

blurred, we believe that similar methodologies may be adopted to cope with feedback provision in both situations. In particular, the research questions we are tackling are three: *i)* how can solutions to performance problems be provided to engineers at design time, *ii)* can design time techniques be useful also at run time, *iii)* how can changes be applied at different abstraction layers and propagated consistently across the modeling stack.

2 Related Work

Several methodologies are available in literature for model driven performance evaluation at design time [1]. However, all of them fail or provide limited support to the feedback problem. Existing approaches can be divided in two categories: meta-heuristic approaches [5][6][7] and rule based approaches [8][9][10]. Meta-heuristic approaches use optimization and genetic algorithms to address architectural evolution. Despite being efficient in finding solutions, all these approaches work only for specific types of evolution (e.g., selection or allocation of components). Rule-based approaches adopt instead Event Condition Action (ECA) rules to identify problems and propose alternative solutions, which are usually identified by considering well known performance anti-patterns. For example, Xu describes in [10] a rule-based approach for the PUMA framework, in which declarative rules are implemented with the Jess language. However, solutions are proposed by rules only for trivial performance problems and, if this is the case, changes apply to the lowest abstraction layer without propagation along the modeling stack.

Rule-based approaches are popular also for run time adaptation. An overview of the various approaches and of hot research topics can be found in [4][11][12]. Of the various approaches, it is worth mentioning the foundational work on the Rainbow project [13] by Garlan et al.: one of the first approaches recognizing the power of abstractions for adaptation and providing an holistic solution. They propose to reason about adaptation at the level of architectural models: probes sense the environment and reflect changes on models, while ECA rules specify when to adapt and how to react. More recent work can also be found in [14], where Morin et al. use aspect oriented modeling techniques to manage adaptation and, most notably, introduce the idea of automatically generate feedback rules by comparing application run time and design time models. Interesting are also recent approaches where software product lines techniques are used at run time and viewing evolution as a variability concern [15][16].

3 Proposed Solution

Rule-based approaches represent the most adopted solution in literature but, although ECA rule have a very well defined structure, every approach uses its own formalism to specify them and a babel of languages exist. Furthermore, there is still a strong separation between design time — when rules are used to build software, to foster performance issues in design models, and to find

alternative solutions — and run time — when rules are used to automatically modify software artifacts in response to environmental changes —. Many design decisions taken by engineers at design time may constitute a valid basis for run time evolution. For example, in the context of component based software, to cope with performance issues the allocation of components may be changed or different implementations showing better Quality of Service (QoS) can be selected. The rationale behind these decisions must be preserved, and many performance issues at run time can be solved by following the same decision process used at design time.

This intuition must be indeed supported by adequate approaches and technologies; a common mechanism to specify rules suitable both for design time evolution and for automatic run time usage is required. We believe that model transformations absolve to this task, by being *"definitions for Round-Trip Engineering"* [17]. Several languages exist to specify model transformations [18]: they are currently used to support software design, and they have been already applied for run time adaptation [19]. However, current transformation languages do not natively support feedback and performance. Existing approaches for feedback provision leverage model transformations only to specify solutions and reactions in adaptation rules. The logic to identify problems, viable alternatives, and to apply them resides outside in separate artifacts (e.g., planners, monitors, or actuators). We instead believe that this strong separation should be encompassed: feedback and performance concepts should be promoted to first class citizens in transformation languages in order to fully exploit their potential.

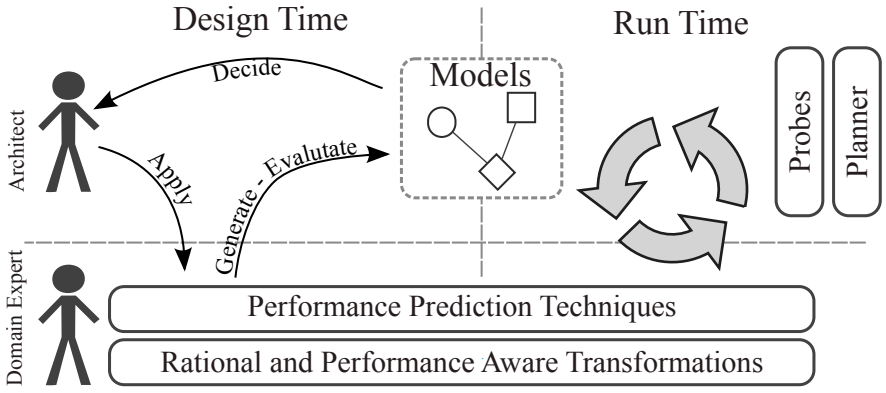


Fig. 1: Overview of the ReARRANge approach.

To close the feedback loop we propose the ReARRANge (Rational trAnsfoRmations for peRformance AdaptatioN) approach depicted in Figure 1. The common background between design time and run time is what we call *"rational and performance aware model transformations"*. They extend traditional transformation languages by providing concepts to clearly identify and state alternative design solutions (captured by 1-to-n transformations, i.e., when multiple output models correspond to the same input model), and to decorate them with

performance engineering concepts (e.g., how to predict non-functional properties of a solution, which non-functional attributes are impacted by a solution). Domain engineers write transformations as usual and, at the same time, use our additional concepts to embed expertise in performance engineering. Knowledge can be then reused by end-user architects to close the feedback loop: the state-space of solutions can be automatically explored by leveraging rational information, non-functional properties of viable alternatives can be computed by leveraging the performance related information, and architects may eventually be guided toward the best design solution.

This general scheme can be adopted to close the feedback loop also at run time: architects are substituted by planners, probes detect changes, transformations implement reactions. Keeping alive models, using probes to update models with changes in the environment, and adopting planners to reason about reactions is common for several approaches [4][13]. We believe this methodology is worth and that existing solutions can be integrated into ReARRANge. In the specific case, we intend to integrate with Kami [20]: a methodology (and a framework) to keep alive models at run time, providing support for model-aware probes, actuators, and for reasoning. Indeed, to integrate with adaptation frameworks — and also with Kami — probes and actuators must be first implemented. Every application domain has its own adaptation needs, and manual implementation can be cumbersome. We believe that rational and performance aware transformations contain all the information necessary to automatically (or at least partially) generate probes and actuators (i.e., what to monitor and what to change); our research will tackle also this aspect.

4 Expected Contributions

In detail, our research will provide the following scientific contributions:

- an extension to existing model transformation languages promoting performance engineering concepts to first class citizens, and tackling knowledge sharing about solutions to performance issues between design time and run time. The kind of information we intend to embed into transformations concerns how to evaluate the performance of candidate alternatives, which performance indexes are of interest, and how to select the appropriate solutions.
- an approach to close the feedback loop at design time leveraging rational and performance aware transformations. We intend to provide a mechanism to explore the state-space of solutions, to evaluate the non-functional properties of the generated alternatives, and to guide architects in decision making.
- an approach to reuse the design time rationale to perform run time evolution, with specific attention to performance issues. We intend to investigate what kind of knowledge can be effective also at run time, how to integrate with existing frameworks for run time evolution, and how to generate, from rational and performance information embedded into transformations, the artifacts (e.g., probes and actuators) necessary for run time evolution.

5 Current Status

Concerning the current status of our research, we have designed and implemented a prototype of the transformation language at the basis of our approach, called QVTR^2 [21]. Since performance issues and adaptation may occur at different abstraction levels, we believe that declarative languages are the most interesting for our goals: traceability and bi-directional capabilities can help in pushing changes across the whole modeling stack. Indeed, QVT-Relations is the base language we have selected for extension. The current implementation concentrates in particular on closing the feedback loop at design time, by providing constructs to: *i)* identify which transformation rules specify alternatives, *ii)* specify which analysis technique should be used to predict the QoS of the candidate model, and *iii)* basic support for constraints between alternatives (i.e., exclusion and inclusion). We have also implemented an execution engine for our language, enabling iterative exploration of the various alternatives and guidance for decisions. As the next step, we are investigating which constructs are required to support run time evolution in QVTR^2, we are adding new constructs to represent performance indexes, and we are implementing more refined constraints for alternatives (e.g., constraints to predicate over indexes) to improve exploration of the solutions state-space at design time and planning at run time.

6 Plan for Evaluation

Our approach tackles both design time and run time evolution; evaluation must cover then both aspects and we concentrate on service and component based systems, given the availability of techniques to predict non-functional properties. For the design time, we plan to compare with the rule-based frameworks described in [10] and in [8]. The common scenario used by both approaches is a multi-tiered web based application designed and developed with a component based approach. Both approaches use performance anti-patterns as the starting point to identify performance issues and to specify rules. We intend to adopt this use case, by extending it with the model transformations necessary to automate development and to provide evolution with ReARRANge. About the results, we expect to show that it is possible to identify at least the anti-patterns covered by other approaches, that we can propose adaptations also for performance issues not fully-covered by existing work (i.e., anti-patterns for which solutions are not proposed), and that engineers can be guided toward alternatives in which anti-patterns do not occur.

For the run time evolution, the use cases adopted to show the approaches in action are usually component based systems similar to the one mentioned for design time evolution. With some extensions, the same use case proposed for design time can be used to assess run time evolution capabilities of ReARRANge. In particular, we plan to show that our approach can handle common evolution patterns (e.g., re-allocation or substitution of components) and more complex adaptation schemes (e.g., changing the structure of the persistence layer to optimize access efficiency).

6

References

1. Balsamo, S., Marco, A.D., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: A survey. IEEE Trans. Software Eng. **30**(5) (2004)
2. Woodside, M., Petriu, D.C., Petriu, D.B., Shen, H., Israr, T., Merseguer, J.: Performance by unified model analysis (puma). In: WOSP, ACM (2005)
3. Becker, S., Koziolek, H., Reussner, R.: Model-based performance prediction with the palladio component model. In: WOSP, ACM (2007)
4. Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J.: Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]. Volume 5525 of LNCS. Springer (2009)
5. Martens, A., Koziolek, H., Becker, S., Reussner, R.: Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: WOSP/SIPEW. (2010)
6. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An approach for qos-aware service composition based on genetic algorithms. In: GECCO, ACM (2005)
7. Grunske, L.: Identifying "good" architectural design alternatives with multi-objective optimization strategies. In: ICSE, ACM (2006)
8. Cortellessa, V., Martens, A., Reussner, R., Trubiani, C.: A process to effectively identify "guilty" performance antipatterns. In: FASE. (2010)
9. Parsons, T.: A framework for detecting performance design and deployment antipatterns in component based enterprise systems. In: DSM, ACM (2005)
10. Xu, J.: Rule-based automatic software performance diagnosis and improvement. In: WOSP, ACM (2008)
11. Oreizy, P., Medvidovic, N., Taylor, R.N.: Runtime software adaptation: framework, approaches, and styles. In: ICSE. (2008)
12. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. IEEE Computer **42** (2009)
13. Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B.R., Steenkiste, P.: Rainbow: Architecture-based self-adaptation with reusable infrastructure. IEEE Computer **37**(10) (2004)
14. Morin, B., Barais, O., Nain, G., Jézéquel, J.M.: Taming dynamically adaptive systems using models and aspects. In: ICSE. (2009)
15. Cetina, C., Giner, P., Fons, J., Pelechano, V.: Autonomic computing through reuse of variability models at runtime: The case of smart homes. IEEE Computer **42** (2009)
16. Bencomo, N., Blair, G.S.: Using architecture models to support the generation and operation of component-based adaptive systems. In: Software Engineering for Self-Adaptive Systems. (2009)
17. Hettel, T., Lawley, M., Raymond, K.: Model synchronisation: Definitions for round-trip engineering. In: ICMT, Springer-Verlag (2008)
18. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal **45**(3) (2006)
19. Vogel, T., Giese, H.: Adaptation and abstract runtime models. In: SEAMS. (2010)
20. Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model evolution by run-time parameter adaptation. In: ICSE. (2009)
21. Drago, M.L., Ghezzi, C., Mirandola, R.: Qvtr^2 : a rational and performance-aware extension to the relations language. In: NFPinDSML 2010, workshop at Models. (2010) Submitted to MoDELS.

A Model Driven Approach to Test Evolving Business Process based Systems

Qurat-ul-ann Farooq

Technische Universität Ilmenau
P.O. Box 10 05 65, 98684 Ilmenau, Germany
qurat-ul-ann.farooq@tu-ilmenau.de

Abstract. Testing business process based systems needs to deal with some unique problems typically caused by dynamic nature of the systems, hierarchy of the processes and service composition. Evolution of these systems is often rapid and can make the task of testing more complex.In this paper, we present the goals of our PhD thesis and discuss a model-driven methodology for test generation and regression test selection using BPMN 2.0 and U2TP. In our methodology, we establish the traceability between the system models and test suites during the test generation. Later in the case of system evolution we utilize this traceability to perform the change impact analysis for the selection of affected test cases to reduce the testing cost.

1 Introduction and Problem Statement

Mode-driven testing (MDT) is becoming widespread due to its promises of raising the level of abstraction, reuse, scalability, effort reduction, better complexity management and easy traceability between system models and test suites [1]. In MDT, test cases are specified using dedicated visual test specification languages such as UML 2 Testing Profile (U2TP) [2]. Although,there is an argument about the overhead of rework involved in visual modeling of test information. However, studies have shown that the benefits of understanding of test suites during maintenance and offshore testing are far more then the initial investment in visual specification of test suites [3]. In the environments where technological change is rapid, specification of test suites in abstract test specification languages is very attractive. It offers the possibility of reuse even after technological changes, saving huge costs in preparation and re-specification of test suites.

Model-driven testing for enterprises that use business process languages to automate their business processes has been introduced to transfer the benefits of MDT in the enterprise system development and testing [4, 5].However, most of the works are initial investigations in the area and lack rigorous details and tool support. Testing the business process based systems is often difficult due to the dynamic nature of these systems,deep hierarchy of subprocesses and service composition [6]. Another important characteristics of enterprise systems is rapid evolution. Enterprise systems evolve due to changes in requirements, technological advancements, organizational changes and many other factors. Besides

effective change management and version control strategies, testing the system after the evolution is necessary to prevent the adverse affects of the changes. Such a testing is known as regression testing. Since, executing the whole test suite on the changed system is very expensive, a cost effective methodology is selective regression testing. It is a process of selecting test cases corresponding to modified parts of the system [1].

In this paper, we discuss the objectives of our PhD thesis. The goals of our thesis work are two fold. (1)To provide a methodology and tool support for platform independent test suite generation for testing business process based systems. This test methodology will serve as a basis for construction of the baseline test suites [1]. (2)To provide a platform independent regression testing methodology to deal with business process based system evolution by utilizing the dependencies between business process models and test models.

2 Related Work

The idea of specifying tests in an abstract test language is advocated by many researches in web services and business process based testing. Several researchers used TTCN for specification of test concepts in web service and business process domain [7, 8]. Since, U2TP is a standard for visual test specification, and it provides all the essential concepts for test modeling like specification of test architecture, test behavior, test data and test time; hence, it is an ideal candidate for test specification for us. There are some approaches reported in the literature focusing on the issue of model-driven testing of service composition. Stefanescu et.al[4] presented a choreography modeling language MCM and a model based test generation environment. The abstract test cases are represented in TPTP and then translated in concrete test scripts. Yuan et.al[5] transformed business process diagram into an abstract test model. This abstract test model covers structural aspects of test specification. This abstract model is then transformed to TTCN3 test suites for test execution. This work is closely related to our test case generation methodology, however this work is only an initial investigation and does not provide transformation rules, and behavioral test case specifications. Gracia-Fanjul et.al[9] presented a test specification methodology for BPEL based service compositions using SPIN model checker. Most of the approaches for regression testing of business process based systems describing service compositions are code based [10–13]. There are a few model based regression testing approaches for business process based applications as well. Khan et.al[14] presented their preliminary work on regression testing of web services. They used protocol state machine to model external behavior and interactions between different actors are modeled using UML sequence diagrams. Terhini et.al[15] presented service composition as as a time labeled transition system. However, lack of use of standard modeling languages, inability to model the system and test cases at a higher level of abstraction, lack of rigorous change definitions and

[1] A stable tested version of the system is known as baseline; whereas, the system after modifications is referred to as delta version in this paper.

lack of tool support, lack of evaluation based on some fault model to see the fault detection capabilities are major problems in these approaches and create a motivation for further work in this area. A discussion on general model based testing approaches can be found in Farooq et.al[1], due to lack of space we can not discuss these approaches in this paper.

3 Proposed Solution

In this section we will discuss our model-driven test generation and regression testing methodology that uses BPMN2.0 models for test generation and U2TP for test specification.

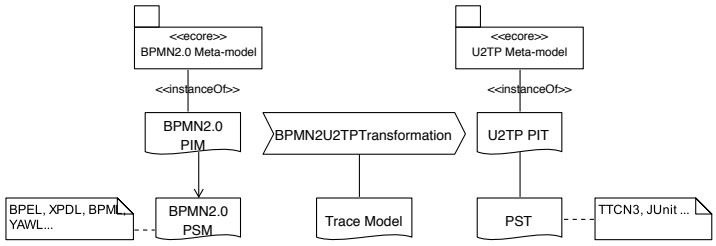


Fig. 1. Model-driven Testing using BPMN2.0 and U2TP

Figure 1 presents our approach for model-driven business process testing. BPMN is a standard visual business process modeling language by OMG[16]. It supports web service choreographies and web service orchestration for web service composition to form an end to end business process model. U2TP is also a standard by OMG for test specification [2]. We adapt the model-driven test process presented by Dai et.al[17] for our test generation approach. For test generation in our approach, a mapping is required between BPMN2.0 and U2TP elements.

We are implementing this mapping using Epsilon Transformation Language (ETL) [2]. Along with U2TP test suites, a major output of this transformations is a *Trace Model* to express links between source and target elements. This model will be used later by our regression test selection approach. ETL seems to be a suitable candidate transformation language due to its support of multiple source and target models and its easy syntax. A trace model can also be constructed during the ETL transformation to link source and target models. During the test generation, we intend to focus not only on test behavior generation, but also on test architecture and test data generation aspects to make our test cases executable on system under test.

[2] http://www.eclipse.org/gmt/epsilon/doc/etl/

3.1 Model Driven Regression Testing of Business Process based Systems

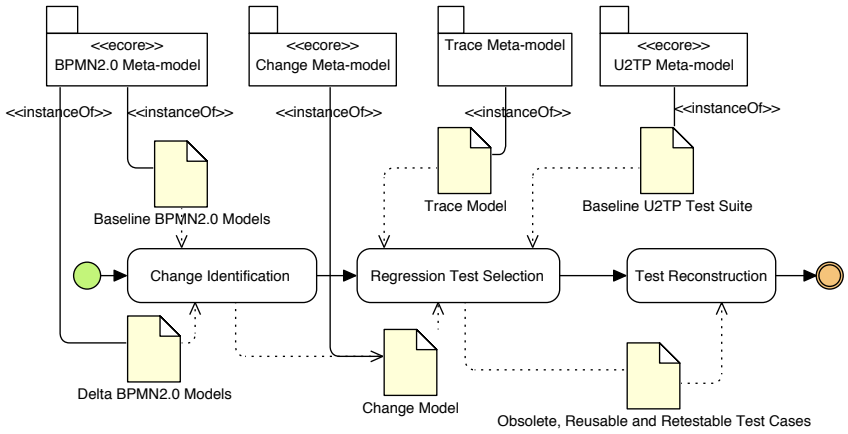


Fig. 2. Model-Based Regression Testing Process

We divide the model driven regression testing process into three major activities as depicted in figure 2. We discuss these activities in detail in the following subsections.

Change Identification: The change identification activity involves many sub activities such as model comparison and change impact analysis. Another very interesting concept that can be useful for this activity is considering the refinement patterns for evolving the system and then using these patterns for change identification. This activity takes baseline and delta versions of BPMN2.0 models and generates a set of changes between both versions of the system depicted as *Change Model* in figure 2.

Regression Test Selection: It involves selection of test cases corresponding to modifications and then classification of these test cases. We adopt the well known classification of Leung and White also discussed by Farooq et.al[1] for this purpose and will classify our test suite into obsolete, reusable and re-testable test cases. In this activity the baseline test suite is required for test selection. As mentioned earlier, our baseline test suite consists of U2TP test models which will be generated by adopting the test generation process depicted in figure 1. The *Trace Model* generated by the transformation at the time of test generation will be used to identify the affected test cases.

Test Reconstruction: After the changes, some test cases might need to be modified in order to conform the changed system. In this phase the test cases which belong to the modified parts of the system will be repaired so that they could be executed on the new version of the system.

4 Expected Contributions

Our expected contributions from this PhD thesis are as follows:

1. Test Generation:
   (a) Provision of mapping and transformations between BPMN2.0 and U2TP constructs with traceability support.
   (b) Creation of a U2TP editor to support U2TP standalone meta-model.
   (c) To analyze and integrate an effective test data generation strategy and integrating some existing test environment e.g. TTWorkbench for the test execution, [3].
2. Traceability and Regression Testing:
   (a) To discover the relationship between different PIM models and express them as traces for effective impact analysis.
   (b) Provision of a concrete model for expressing change information in the models.
   (c) Provision of a method for test suite classification considering changes in the baseline PIM models.
3. To evaluate our methodology on an effective case study to see its fault detection and test reduction capabilities.

5 Current Status and Evaluation Plans

At present, the work on this project is in its initial stages. We are working on the mapping of BPMN2.0 models to U2PT test suites. In parallel, the development of a U2TP editor is also under consideration. At present, none of the tools available in the market, support U2TP modeling explicitly. We are using GMF modeling framework [4] to build an eclipse plug in that support the standalone U2TP meta-model and also provide the modeling capabilities.

We intend to evaluate our approach on two medium size case studies that are under development by the MOPS (Adaptive Planning and Secure Execution of Mobile Processes in Dynamic Scenarios) project team [5]. The application of our approach on MOPS case studies can provide us significant results regarding reduction capabilities of our approach. For evaluating our baseline test generation strategy, we plan to perform mutation analysis to see the fault detection effectiveness. We plan to perform empirical studies between our approach and other relevant model based regression testing approaches discussed in related work to see the effectiveness of our approach in comparison with other studies in the area.

[3] http://www.testingtech.com/products/ttworkbench.php
[4] http://www.eclipse.org/modeling/gmp/
[5] http://fusion.cs.uni-jena.de/professur/research/research-projects/mops

6

References

1. Farooq, Q.u.a., Iqbal, M.Z., Malik, Z., Riebisch, M.: A model-based regression testing approach for evolving software systems with flexible tool support. In IEEE, ed.: 17th IEEE International Conference on Engineering of Computer-Based Systems (ECBS), IEEE Computer Society (2010) 41–49
2. Uml, O.M.G.: 2.0 testing profile specification. Technical report, Object Management Group, 2002
3. Baker, P., Loh, S., Weil, F.: Model-Driven engineering in a large industrial context motorola case study. In: Model Driven Engineering Languages and Systems. (2005) 476–491
4. Stefanescu, A., Wieczorek, S., Kirshin, A.: MBT4Chor: a Model-Based testing approach for service choreographies. In: Model Driven Architecture - Foundations and Applications. (2009) 313–324
5. Yuan, Q.: A model driven approach toward business process test case generation. 2008 10th International Symposium on Web Site Evolution **2008** (October 2008) 41–44
6. Bruning, S., Weissleder, S., Malek, M.: A fault taxonomy for Service-Oriented architecture. In: High Assurance Systems Engineering Symposium, 2007. HASE '07. 10th IEEE. (2007) 367–368
7. Werner, E., Grabowski, J., Troschatz, S., Zeiss, B.: A TTCN-3-based web service test framework. (2009)
8. Schieferdecker, I., Stepien, B.: Automated testing of XML/SOAP based web services. In: Kommunikation in Verteilten Systemen. (2003) 4354
9. Garcia-Fanjul, J., Tuya, J., de la Riva, C.: Generating test cases specifications for BPEL compositions of web services using SPIN. In: Proceedings of the International Workshop on Web Services: Modeling and Testing (WS-MaTe 2006). (2006) 8394
10. Ginige, J.A., Sirinivasan, U., Ginige, A.: A mechanism for efficient management of changes in BPEL based business processes: An algebraic methodology. In: Proceedings of the IEEE International Conference on e-Business Engineering, IEEE Computer Society (2006) 171–178
11. Liu, H., Li, Z., Zhu, J., Tan, H.: Business process regression testing. Service-Oriented Computing ICSOC (2007) 157–168
12. Ruth, M.E.: Concurrency in a decentralized automatic regression test selection framework for web services. In: Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids. (2008)
13. Wang, D., Li, B., Cai, J.: Regression testing of composite service: An XBFG-Based approach. In: IEEE Congress on Services Part II, 2008. SERVICES-2. (2008) 112119
14. Khan, T.A., Heckel, R.: A methodology for Model-Based regression testing of web services. In: Practice And Research Techniques, Testing: Academic & Industrial Conference on. Volume 0., Los Alamitos, CA, USA, IEEE Computer Society (2009) 123–124
15. Tarhini, A., Fouchal, H., Mansour, N.: Regression testing web services-based applications. In: Computer Systems and Applications, 2006. IEEE International Conference on. (2006) 163170
16. OMG: BPMN 2.0 Specification. (August 2009)
17. Uml, O.M.G.: 2.0 testing profile specification. Technical report, Object Management Group, 2002

A Transformational Approach for Component-Based Distributed Architectures

Fabian Gilson

University of Namur - Faculty of Computer Science
PRECISE Research Center in Information Systems Engineering
Rue Grandgagnage 21 - B-5000 Namur - Belgium
fgi@info.fundp.ac.be

Abstract. Nowadays information systems are becoming larger, more complex and they are frequently integrating *Commercial Off-The-Shelf* components. Furthermore, stakeholders are having architecturally significant non-functional requirements that may influence the system architecture at early architecture design stages. As systems are intended to cope with evolving requirements, traceability mechanisms are needed to keep track of design decisions and resulting system architectures. In addition, distributed systems are often deployed on heterogeneous physical infrastructures with many deployment constraints. The present work aims at defining a transformation-oriented design method mixing architecture and requirement models in order to define, build, abstractly deploy and transform component-based information systems.
Keywords: *component-based application - architecture description language - model transformation - abstract deployment*

1 Problem overview

With the emergence of component-based and distributed software applications, many design languages appeared focusing on a high-level representation and on the communication facilities between those systems. Amongst such languages, architecture description languages (ADLs) are probably the most known. As proposed by Bass *et al*, "The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them." [1] Furthermore, as such systems complexity, size and requirements grow, some authors [2,3,4] stressed the need for using model transformations to refine a high-level solution into more detailed ones in order to ease the understanding of the system.

Nowadays systems are intended to evolve in order to cope with frequent business evolutions and these changes often impact the system architecture. Currently, these evolutions are somehow documented, when actually documented, but there is still a lack in their formalisation resulting in losses about the design or evolution rationale traceability.

Moreover, distributed systems are often deployed on heterogeneous existing infrastructures with many deployment constraints. Application modellers check such constraints mostly at the end of the development process. This results sometimes in incompatibilities between the software solution and its target environment. This kind of incompatibilities could be avoided by integrating deployment and infrastructure needs at the architecture design and checking frequently the compatibility between the application and its infrastructure.

By reconciling both model transformations and design decisions traceability, this thesis aims at proposing a transformation-oriented design method for component-based information systems. Using our proposal, designers define a system architecture in three linked layers separating the conceptual definition of components and connectors, their implementations and the target or existing physical infrastructure. Coupled with the architecture, modellers use a previously defined list of architecturally significant non-functional requirements in order to define transformations that implement one-by-one these requirements.

The present document first presents existing approaches in architecture description languages, abstract deployment languages, design traceability techniques and model transformation languages in Section 2. Then, we give an overview of our proposed solution in Section 3. In Section 4, we present the expected outcomes of this thesis. Next, in Section 5, we expose the current achievements of our work. Last, in Section 6, we explain the evaluation method we plan to use.

2 Related work

As information systems become more and more decentralized, modelling languages appeared in order to cope with the growing in size, complexity and spreading. Amongst such approaches, we note the architecture description languages (ADL). In [5], the authors compared some of them and highlighted their strengths and weaknesses. All focused languages have at least one notable lack. For example, Darwin [6] and Rapide [7] do not offer facilities for non-functional requirements definition. Only a few languages were developed after that comparison report, such as AADL [8] and SafArchie [9]. AADL goes a step further than other ADLs in semantics definition of components by providing facilities to specify hardware elements. Designers can also add properties on components such as *Quality of Service* (QoS) properties. However, AADL constructs make models really close to the implementation, missing abstraction possibilities. In SafArchie, components are completed with behavioural specification. Model transformations intended to cope with internal and external evolution of models can be expressed in a dedicated transformation language. Unfortunately, expressed connectors are only synchronous method calls and no QoS-related properties can be expressed on model constructs.

Because component-based architectures involve many parties and often integrate already existing hardware, it is crucial to model the deployed artefacts on an abstract representation of the infrastructure. Very few methods integrate

abstract deployment facilities or when existing, they have limited semantics. Unified Modeling Language (UML) deployment diagrams [10] from the Object Management Group (OMG) offers a basic way to deploy artefacts on physical nodes connected by communication links. Because it has been thought generic, the construct semantics is very poor, especially for the communication paths. The OMG itself created another deployment standard [11] linked to the CORBA [12] middleware. It is richer than the UML deployment diagram as it allows to define deployment plans and target hardware descriptions, but sticks too much to CORBA concepts, making it hard to use with other component languages.

Software systems have to cope with changes in requirements as nowadays environment is evolving rapidly [13]. Recent approaches have been specified to formalise the design rationale and design decisions knowledge. One of the most crucial requirements of such approaches is the binding between the software architecture and its design choices [14,15]. An ontology of design decisions for software intensive systems has been proposed in [16] and a tracing mechanism has been presented in [17].

As model driven engineering (MDE) approaches became more popular and in response to the OMG request for proposal on Query/View/Transformations (QVT) [18], transformation languages appeared to manipulate models. The work in [19] highlighted the main features of transformation languages and provided a feature-based framework classifying a couple of existing approaches. Atlas Transformation Language (ATL) [20] is a declarative and imperative textual language working on *Meta Object Facility* (MOF)-compliant models [21]. It offers traceability facilities, but transformation are only unidirectional. In [22], the authors compared three transformation language approaches: a textual-based transformation language (ATL), an abstract syntax-based language (attribute graph grammar, AGG) and their proposal – a concrete syntax-based graph transformation (CGT) –. With concrete syntax-based languages, developers define transformations directly on models instead of working on the abstract syntax. This eases the rules writing and increases their readability. For both graph-based languages, transformation rules are also often more concise and readable than for textual-based approaches.

3 Proposed solution

The thesis proposes a transformation-oriented design method based on an ADL reconciling three main challenges in distributed applications design. First, we want to represent component-based architectures with architecturally significant requirements and infrastructure constraints. Designers use coarse-grained components and connectors specifications that have been highlighted by previous tasks and draw a first architecture model. The constructs semantics are refined by quality attributes expressing their related non-functional requirements. Connectors allow communication protocol abstractions in order to cover a wider range of connection facilities between components. Connectors are then the support for the connection (saying, for example, if we have a *point-to-point* or a

broadcast-like connection) and the communication protocols are defined separately, enabling more flexibility in components composition. Parallelly to the architecture model, the architecturally significant non-functional requirements (ASNFR) are listed and linked to the model element in charge of implementing that requirement.

Second, we use these ASNFRs to define, trigger and trace model transformations that fully or partially implement them. Starting from a first coarse-grained model and the list of unimplemented ASNFRs, we want to formally transform the model step by step. This way, each refined or extended model is linked to its source model by a transformation, enabling tracing capabilities: where we come from, where we go and how we do it. This traceability mechanism permits to structure and capture the design decisions taken by software engineers during the application design. Designers can retrieve later the reasons why the final architecture model is as it is and can also reuse architectural knowledge for other similar systems. Furthermore, in some cases, system evolution will be easier to manage as system engineers can select a particular model in the transformations graph (before the injection of a particular technology, for example) and build the new solution from that point.

Third, the deployment environment hardware is specified by dedicated model constructs after the first coarse-grained model. After each transformation, the resulting architecture is validated with the infrastructure constraints in order to identify inconsistencies or possible threats at deployment time. Using this method, deployment related problems will be identified as soon as a model does not conform to the target deployment environment. Corrective decisions will be easier to take as we know exactly which design decision caused the problem. By constantly confronting the model to its deployment infrastructure, no bad surprise should arise when installing the resulting application onto the target hardware.

This thesis does not address the preliminary task concerning the coarse-grained components identification as well as the ASNFRs elicitation. This can be done by any other method as long as we obtain a first high-level architecture and a list of ASNFRs related to single architectural elements. Furthermore, our approach does not focus on design alternatives exploration, but propose design decisions traceability features.

4 Expected results

First, we will create an ADL conforming to the specification expressed in the previous section and answering to the lacks of existing languages. We will define a metamodel of the language and precise its constructs semantics. Then, we will create the textual and graphical concrete syntaxes. Coupled with the ADL, we will create a simple graphical notation to summarise the ASNFRs. Second, we will reuse, extend or create a transformation language in order to correctly justify architecture model refinements and element substitutions or extensions. Used with the ASNFR model, this will help to keep track of the application

design decisions path and will be used for further evolutions of the system. For both tasks, we will use academic examples in order to validate and revise our method with the development of a realistic, even though simple, distributed application. Third, we will investigate the possibility to add behavioural semantics to architectural elements semantics. We will make a survey of existing methods and languages, and if possible, integrate one of them that answers our need. Finally, after the definition and validation of our languages, we will create at least a textual editor and if we did not run out of time, a graphical editor at last.

5 Current status and achievements

At present time, we already achieved a couple of tasks. We went through existing ADLs and stressed the important concepts and features of such languages. We also found a few lacking properties that we find necessary when designing distributed software. Based on our observations, we built an ADL and we defined his textual syntax with an LL-grammar. We implemented our textual syntax using an *Eclipse* plug-in called *Xtext* which is an editor and a parser-generator for LL-grammars. This plug-in dynamically generates a configurable textual editor as an Eclipse plug-in with, among others, syntax coloring and auto-completion.

We defined a simple graphical syntax for ASNFR models as a UML profile. We created also a textual syntax for these models that we added to our Eclipse-based editor. For both languages, we used academic examples to illustrate and validate the approach.

6 Evaluation plan

Coupled with illustrative examples, we plan to make a controlled experiment in order to validate our proposal. We will test our method on a last graduation year student group. We will compare our method to an iterative one supported by UML diagrams for static models and free text for tracing the design decisions. First, the students, separated into two groups, one for each method, will be asked to build a distributed application from scratch. Second, we will ask them to make functional and non-functional changes in the application. Regarding ASNFR, we will focus on *QoS*-related properties. We will evaluate their experiences with both design methods by questionnaires to capture their feelings about the methods. We will also compare the correctness of the built systems (before and after the changes) regarding to functional test cases and non-functional requirements. Last, we will compare the time spent by students for designing and developing their applications.

References

1. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, Second Edition. Addison-Wesley Professional (2003)

2. Moriconi, M., Qian, X., Riemenschneider, R.A.: Correct architecture refinement. IEEE Transactions on Software Engineering **21** (1995) 356–372
3. Bosch, J., Molin, P.: Software architecture design: Evaluation and transformation. IEEE International Conference on the Engineering of Computer-Based Systems (1999) 4
4. Khriss, I., Keller, R.K., Hamid, I.A.: Supporting design by pattern-based transformations. In: 2nd International Workshop on Strategic Knowledge and Concept Formation. (1999) 157–167
5. Medvidovic, N., , Taylor, R.N.: A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering **26** (2000) 70–93
6. Magee, J., Dulay, N., Eisenbach, S., Kramer, J.: Specifying distributed software architectures. In: 5th European Software Engineering Conference. (ESEC 95), Sitges, Spain, Springer (1995) 137–153
7. Luckham, D.C., Kenney, J.J., Augustin, L.M., Vera, J., Bryan, D., Mann, W.: Specification and Analysis of System Architecture Using Rapide. IEEE Transaction on Software Engineering **21** (1995) 336–355
8. Feller, P.H., Gluch, D.P., Hudack, J.J.: The Architecture Analysis & Design Language (AADL) : An Introduction. (2006)
9. Barais, O.: Construire et Maîtriser l'évolution d'une architecture logicielle à base de composants. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Lille, France (2005)
10. Object Management Group: Chap.10. In: OMG Unified Modeling Language (OMG UML), Superstructure, version 2.2. Object Management Group (2009) 193–214
11. Object Management Group: Deployment and Configuration of Component-based Distributed Applications, version 4.0. (2006)
12. Object Management Group: Common Object Request Broker Architecture (CORBA) Specification, version 3.1. (2008)
13. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Transactions on Software Engineering **27** (2001) 58 –93
14. Bosch, J.: Software architecture: The next step. In: 1st European Workshop on Software Architecture, Springer (2004) 194–199
15. Tyree, J., Akerman, A.: Architecture decisions: Demystifying architecture. IEEE Software **22** (2005) 19–27
16. Krutchen, P.: An ontology of architectural design decisions in software intensive systems. In: 2nd Groningen Workshop Software Variability. (2004) 54–61
17. Feng, Y., Huang, G., Yang, J., Mei, H.M.: Traceability between software architecture models. Computer Software and Applications Conference, Annual International **2** (2006) 41–44
18. Object Management Group: MOF 2.0 Query/View/Transformations RFP. OMG Document ad/2002-04-10 (revised on April 24,2002)
19. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal **45** (2006) 621–645
20. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica (2005)
21. Object Management Group: Meta Object Facility (MOF) Core Specification, version 2.0. (2006)
22. Grønmo, R., Møller-Pedersen, B., Olsen, G.K.: Comparison of three model transformation languages. In: 5th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 09), Springer (2009) 2–17

Modeling Complex Situations in Enterprise Architecture

Hyeonsook Kim

School of Computing, Thames Valley University, UK, W5 5RF
hyeonsook.kim@tvu.ac.uk

Abstract. Over the decades, Enterprise Architecture (EA) has been researched to supply all the necessary components for enterprise system modeling including taxonomies, meta-models, architecture development methods, and modeling tools. Main benefits of EA are the knowledge infrastructure for analysis and reporting by all stakeholders and the possibility of designing new conditions in an organized manner. However, EA now faces a big challenge with the growing dynamic of market demands and the rapid changes in business environments, which requires agile system response and self evolutionary behavior to support quick decision making. In technology side, there are already matured, promising paradigms to tackle this challenges, which are Complex Event Processing (CEP) and Context-Awareness (CA), however they have not been integrated to EA level yet. This research proposes integration of CEP and CA concepts into EA by replacing or extending related EA components. Sophisticated event modeling to detect complex situations of organization will provide the recognition and adoption of environment changes with increased enterprise agility in more strategic and business level. Adding semantic descriptions and annotation to event models will improve context awareness for dynamic decision support and autonomic behavior. Case studies will be conducted to define specific boundaries of functional and design requirements, and application scope. Meta-model of complex situation will be developed and evaluated before being integrated into existing EA model. Modeling tools such as model editor and model transformer will also be developed to facilitate the integrated EA modeling.

Keywords: Modeling complex situation, Enterprise Modeling, Enterprise Architecture, Complex Event Processing.

1 Background and Motivation

Over the years, model-based development has gained rapidly increasing popularity across various engineering disciplines [1]. Numerous efforts resulted in the invention of concepts, languages, and tools for the definition, analysis, transformation, and extension of domain-specific modeling languages as well as general-purpose modeling language standards. For enterprise systems, Enterprise Architecture (EA)

has been researched to supply all the necessary components for enterprise system modeling including taxonomies, meta-models, architecture development methods, and modeling tools. Main benefits of EA are the knowledge infrastructure for reporting and analysis by all stakeholders and the possibility of designing new conditions in an organized manner [3]. EA is not only an instrument for strategic planning of IS/IT but also other business functions, such as compliance control, continuity planning and risk management.

EA now faces a big challenge with the growing dynamic of market demands and the rapid changes in business environments, which requires agile system response and self evolutionary behaviour by quick decision making [4], [5]. Complex Event Processing (CEP) and Context-Awareness (CA) are promising matured paradigms which can support enterprise agility and intelligence in technology level [6]. However, these concepts have not been considered or applied in EA.

In this research, I propose to integrate CEP concepts into EA by replacing or extending EA components. Sophisticated event modeling to detect complex situations of enterprise will provide the recognition and adoption of environment changes with increased enterprise agility in more strategic and business level. Adding semantic descriptions and annotation to event models will improve context awareness for dynamic decision support and autonomic behaviour.

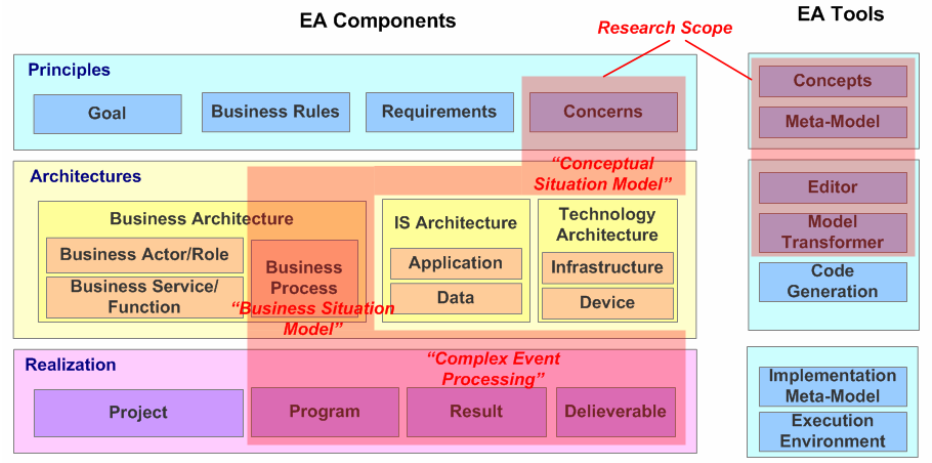


Fig. 1. The scope of this research

Based on these assumptions, my research focuses on modeling business environment changes; it is also known as event. The modeling includes developing meta-models in conceptual level and architectural level. The models and model mappings will be enhanced by semantic description. The created model will be linked to system realization level such as event processing infrastructure. The model editor and model transformation will also be implemented to support situation modeling. Figure 1 briefly shows key components related to the research field and identifies the scope of this research.

2 Related Works

An event is a change of state that has significance for the management of system in the context of system architecture [7]. Inauguration of new vice president, launch of new project, increase of sales, and a system failure are examples of enterprise events. Most enterprise models represent the business states as conditions or effects rather than adopting the concept of event directly. Without consideration of state changes, it is not simple to design complicated business concerns and processes. Besides, not only a simple event but also complex patterns of events including time, location, user, and cost need to be considered in reality. Mapping events from distributed sources, sharing event context with external enterprises, and mapping unknown event from external with internal one are also important parts for enterprise event handling.

In the distributed computing area, the event processing has been actively researched under the topic of Complex Event Processing (CEP). The system vendors such as IBM, TIBCO and Oracle have released their own event processing platform competitively to support complex event pattern recognition. However, they have not provided proper modeling and method to express this process in the manner of model driven engineering. My research will more focus on modeling of complex situation and applying it to EA, while avoiding re-wheeling the existing event processing platform.

2.1 Enterprise Architecture

Enterprise Architecture (EA) is comprehensive description of all of the key elements and relationships that make up an organization [2]. In this manner, the EA which outlines business goals and activities of an enterprise must be able to answer the questions of changes that the enterprise is confronting with. The main objectives of EA include not only better strategic planning but also integration between business and IT.

With increasing interest in business-IT alignment, great efforts have been made for EA development in industry. For instance, TOGAF [8] is one of the most popular and outstanding EA frameworks defined by industrial vendors. The framework addresses all the aspects of enterprise system design not only in development-led view but also in business-led view. Business goals, business data, business activities, business constraints, and resources are designed abstractly in the framework to produce comprehensive guideline for enterprise system development. A few case studies [9] are introduced to demonstrate its feasibility. However, the proposed case studies do not provide enough use cases for complex business scenario and processes. The gap between abstract designs and concrete components is also an open issue [4].

A number of researches have been conducted in academia as well. For example, Deitz [10] proposes enterprise ontology in his thesis and Zhang [11] builds enterprise taxonomies. In particular, Land [12] focuses on describing collaboration and social

interaction among enterprise actors in his paper. The author uses an ontology language, OWL-DL, to describe business activities, and the focus on transactions between actors gives a better insight into the construction of an organization. With this approach all transactions can be interpreted and inferred by an ontology reasoning system. Unexpected events, exceptions and errors also can be handled with increased intelligence. However, There is no enough consideration about IT realization model and no enough evaluation for ontology reasoning.

2.2 Complex Event Processing

Complex Event Processing (CEP) is the analysis of event data in real-time to generate immediate insight and to enable instant response to changing conditions – what we call continuous intelligence [6]. Numerous efforts have been made to develop strong event models and methods for event processing.

Paschke [15] and Vincent [16] analyze the patterns of complex events in order to show how the existing CEP model can be fitted into business process modeling. On the other hand, Kellner [17] demonstrates several patterns of complex event processing discovered from case studies but the patterns are linked to design patterns of business process for system development. In the context of system development methodology, business process modeling is a part of enterprise system modeling in the physical level. Therefore both of the approaches are too tightly bound by system implementation concern rather than business concern.

Another paper written by Zang [19] insists that event processing can fit well in enterprise information systems in terms of facilitation of event aggregation into high level actionable information, and improved responsiveness by nature of event response. The author implements the event processing mechanism based on RFID and evaluated the performance with a sample application. From the proposed event meta-model, the author tries to implement different abstraction hierarchy of space context. However, due to the limited examples and applications, more consideration regarding the scalability and capability of the model is required.

From another point of view, Barros [18] developed a graphical notation for modeling complex events in business processes. Although this approach resides in concrete level without enough abstraction, various types of event composition patterns are identified and implemented as graphical expression. The suitability of the proposed language is assessed by expressing common event scenarios in business process. The most novelty of this paper is that the semantics is defined to provide a mapping from core event models to logical expressions, and a transformation from non-core event models into a set of core event models.

3 Research Objectives

The objective of this research is to develop a framework that extends existing EA with complex situation modeling method in order to support better enterprise agility and intelligence. Business concerns will be designed and expressed using situation

model in abstraction level. The abstracted model will be gradually developed to business architecture level and system realization level in consistent view within the framework. The following sub goals have been formulated:

1. Study and evaluate methods for modeling complex situations of enterprise environment. Several approaches such as ontology model and rule model will be investigated to find the most suitable method. Case studies will be conducted to set up specific boundaries of functional and design requirements, and application scope.
2. Design meta-model of complex situation in conceptual level by mapping business concerns to situation models. Meta-model of complex situation in business architecture level will be designed as well as model transformation between models in different levels.
3. Extend existing EA modeling language such as Archimate [14]. with the proposed complex situation model. Several existing EA meta-models and frameworks will be reviewed and compared to identify the most proper base architecture.
4. Add semantic description or annotation on complex situation model to improve enterprise intelligence.
5. Implement model editor to provide graphic based modeling facilitate and model mappings between models in different levels.
6. Validate proposals with real business cases and its application. The experiments will enable testing the feasibility of the proposed situation model. The application such as simulation will illustrate the benefits of the extended EA.

4 Current Status and Future Works

This research is still in early stage as it was began in 2010. As an initial step, case study is being progressed in order to gather sample business scenarios and data which can identify specific design issues and functional requirements for enterprise complex situation modeling. Sample business scenarios will be implemented using existing EA modeling language such as Archimate. This work will identify specific shortages in expressing complex situation and semantic information. The recognized problems will be good sources to design a new model to extend EA. Further detailed literature review will be also essential.

References

1. Schmidt, D. C., Model-driven engineering. Computer, 39 (2), p.25-31 (2006)
2. Fox, M. & Gruninger, M., Enterprise Modeling. Artificial Intelligence Magazine, 19 (3). (1998)
3. Lankhorst, M., Enterprise architecture at work - Modelling, communication and analysis. Springer-Verlag, Heilderberg (2005)

4. Sinderen, V. & Marten, Challenges and solutions in enterprise computing, Enterp. Inf. Syst. Proceedings of the 11th International IEEE EDOC Conference (EDOC) p.341 – 346 (2008)
5. Kim, T.-Y., Lee, S., Kim, K. & Kim, C.-H., A modeling framework for agile and interoperable virtual enterprises. Comput. Ind., 57 (3), p.204-217 (2006)
6. Luckham, D., The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. In (eds.). Rule Representation, Interchange and Reasoning on the Web (2008)
7. Michelson, B., Event-Driven Architecture Overview. Patricia Seybold Group (2006)
8. The Open Group Architecture Framework (TOGAF) version 9. Van Haren Publishing (2009)
9. Buckl, S., Ernst, A. M., Matthes, F., Ren, Ramacher & Schweda, C. M., Using Enterprise Architecture Management Patterns to Complement TOGAF. In (eds.). Proceedings of the 2009 IEEE International Enterprise Distributed Object Computing Conference (2009)
10. Dietz, J. L. G., Enterprise Ontology: Theory and Methodology. Springer-Verlag New York, Inc. (2006)
11. Zhang, L., Li, T., Liu, S. & Pan, Y., An integrated system for building enterprise taxonomies. Inf. Retr., 10 (4-5), p.365-391 (2007)
12. Land, M. O. T. & Dietz, J. L. G., Enterprise ontology based splitting and contracting of organizations. In (eds.). Proceedings of the ACM symposium on Applied computing. Fortaleza, Ceara, Brazil, ACM. (2008)
13. IDS (Intelligent Decision Support) Special Interesting Group, http://samsa.tvu.ac.uk/ids/blog/ , accessed Jun-2010.
14. Lankhorst, M.M., Proper, H.A., Jonkers, H., The Anatomy of the ArchiMate Language, International Journal of Information Systems Modeling and Design (IJISMD), Vol. 1, Nr. 1 IGI Publishing (2010)
15. Paschke, A. & Ammon, R., Domain-specific Reference Models, Architectures and Patterns for CEP. In (eds.). EuroPLoP - CEP Focus Group. Irsee, Germany (2008)
16. Vincent, P., How does CEP fit into BPM and SOA environments? TIBCO, Complex Event Processing. (2010)
17. Kellner, I. & Fiege, L., Viewpoints in complex event processing: industrial experience report. In (eds.). Proceedings of the Third ACM International Conference on Distributed Event-Based Systems. Nashville, Tennessee, ACM. (2009)
18. Barros, A., Decker, G. & Grosskopf, A., Complex Events in Business Processes. In (eds.). Business Information Systems. Lecture Notes in Computer Science (2007)
19. Zang, C. & Fan, Y., Complex event processing in enterprise information systems based on RFID. Enterp. Inf. Syst., 1 (1), p.3-23 (2007)

Reference Modeling for Inter-organizational Systems

Dieter Mayrhofer

Institute of Software Technology and Interactive Systems
Vienna University of Technology, Austria
mayrhofer@big.tuwien.ac.at

Abstract. Inter-organizational business process models are created and used by different partner networks. Often, those models are similar but differ in certain business needs. Due to the lack of methods and tools, companies are required to start from scratch to create their specific models instead of taking advantage of and re-using already existing models. Therefore, we first analyze reference modeling design techniques and their applicability on inter-organizational models and second create a prototype implementation of inter-organizational reference modeling. By creating models out of reference models we expect to gain efficiency in the modeling process and a higher quality of inter-organizational solutions.

1 Introduction and Motivation

Nowadays, the use of business models and business process models often concentrates on intra-organizational systems. However, companies also take part in inter-organizational systems, but they are mostly limited to their implementation such as Web Services by following a bottom-up approach. To overcome these limitations, we developed a model-driven approach towards inter-organizational systems named *Business Semantics on top of Process Technology (BSopt)*.

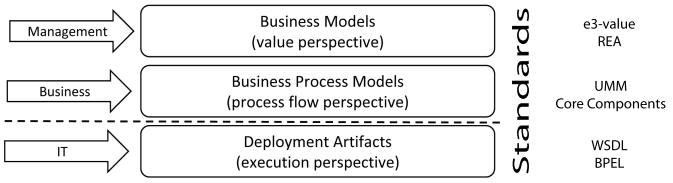


Fig. 1. Layers in inter-organizational systems

The *BSopt* approach [1] builds upon three layers, as shown in Figure 1. The first layer, the *value perspective*, captures the rational as well as the economic resources being exchanged with business partners. To model these aspects, we use two well established business modeling languages, *e3-value* [2] and the *Resource-Event-Agent ontology (REA)* [3], which will be explained later.

The second layer, the *process flow perspective*, is partly derived from the first layer and uses business process models. It describes the flow of business activities and their dependencies in accordance with the business goals to be reached. For this purpose we use *UN/CEFACT's Modeling Methodology (UMM)* [4]. Another important part of this layer are the messages being exchanged between the business partners. To describe these messages we use *Core Components*, which define messages on a conceptual level. In order to get a deployable system, the models defined on the second layer are translated into deployable artifacts covered by the IT layer (the execution perspective). The IT layer implements the business processes by means of tools, frameworks, API's, Web Services, etc. In *BSopt*, the deployable artifacts covered and generated out of the business process models of the second layer are the *Business Process Execution Language (BPEL)* and the *Web Services Definition Language (WSDL)*. We developed a tool for *BSopt* that provides a sophisticated solution to create inter-organizational systems based on models. It allows the user with the help of wizards to model the top two layers mentioned before and generate the artifacts for deployment.

Although the *BSopt* tool proved to seamlessly create inter-organizational systems, we encountered the following problem concerning the re-use of models during evaluation: *If a company wants to use existing business and business process models and wants to change them slightly, it generally has to reinvent the wheel and start again from scratch.*

Therefore, we propose the following solution: *Using various design techniques from reference modeling on the inter-organizational models will foster the re-use of models. Due to less recreation of already existing models which only need to be slightly changed, we expect faster model creation time as well as better quality of models by incorporating best practice into reference models. We anticipate that the design techniques can generally be applied on a variety of model languages.*

The remainder of this paper is structured as follows: Section 2 briefly discusses the models considered for reference modeling and the design techniques in question. Additionally, we introduce existing work in the field. In Section 3 we define the contributions concerning re-using business models and business process models. Section 4 describes the research methods being used in order to create reference models and a prototype implementation. In Section 5 we conclude the paper by showing the advantages we expect from reference modeling used on the inter-organizational models.

2 Background and Related Work

In this section we elaborate on the models used in *BSopt* and on the design techniques of reference modeling. These two areas will serve as the base of the thesis. At the end of the section we discuss additional related work.

Inter-organizational models. The Inter-organizational models are implemented using Domain Specific Languages (DSL). Thereby, *e3-value* describes a business model as a value web where multiple actors exchange objects of economic values such as money, goods, or services. Next, *REA* was originally in-

troduced by McCarthy [3] as an accounting model and extended to an ontology by Geerts and McCarthy [5]. In contrast to *e3-value*, *REA* captures the bilateral agreements and commitments between exactly two agents. *REA* stands for the three main concepts *Resource*, *Event*, and *Agent*. After the business models are defined, business process models based on *UMM* can be partly derived to realize the value exchange. *UMM* allows to create a business process model from an observer's perspective in order to depict inter-organizational processes. It builds upon three views: the business requirements view capturing the requirements already gathered from the *REA* and *e3-value* models, the business choreography view defining the business transactions between partners, and the business information view responsible for the messages being exchanged. *UMM* does not require a certain standard for messages. The *BSopt* project makes use of *UN/CEFACT's Core Components Technical Specification (CCTS)* [6] for defining business documents on a conceptual level using a DSL implementation [7]. Once all the models are created, deployment artifacts can be created on the IT layer to be used by the process or workflow engines.

Design Techniques. Reference modeling is mainly driven by the German Information Systems community and has primarily focused on internal business processes [8–10] . The main idea enables faster creation of models by re-using existing ones as well as delivering higher quality models. Vom Brocke [10] introduces five design techniques for reference modeling (cf. Figure 2). *Configuration* allows to configure a model out of a superset of elements of a core model. *Instantiation* provides a framework of a model with placeholders for parts yet to be defined. These placeholders can then be further specified by plugging in models with specific characteristics. *Specialization* relaxes re-use by allowing to take over a complete general model and changing it by adding or changing arbitrary elements. *Aggregation* allows to combine multiple existing reference models to compose a new complete model. Finally, *Analogy* serves as an orientation for the creation of a new model.

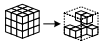
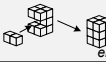
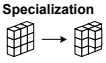
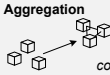
Configuration  *by selection*	*The technique of configuration is characterised by deriving a configured model c out of a configurative model C by means of making choices from a greater variety of alternatives offered in C.*	*The application domain can be described fully in design time including all relevant adaptations that have to be considered in various applications.*
Instantiation  *by embedding*	*The creation of a resulting model "I" by integrating one or multiple original models "e" into generic place holders of the original model "G". The model "I" incorporates the integrated construction results of "e" in "G".*	*The application domain can be covered by a general framework; this framework however, has to be adapted in regard to selected aspects that can not fully be described while building the reference model.*
Specialization  *by revising*	*Derivation of a resulting model "S" from a general model "G". That way, all statements in "G" are taken over in "S" and can either be changed or extended (but generally not deleted).*	*The application domain can be covered by a core solution; but this solution has to be extended and modified (without deleting) in an indefinite manner for various applications.*
Aggregation  *by combination*	*The combination of one or more original models "p" that build "a" resulting model "T", with the models "p" forming complete parts of "T".*	*The application domain can be described partly; each part can fully be specified whereas their combination for replenishing the entire coverage of an application cannot be foreseen when building the reference model.*
Analogy  *by creativity*	*An original model "A" serves as a means of orientation for the construction of a resulting model "a". The relation between the models is based on a perceived similarity of both models regarding a certain aspect.*	*The application domain can be described by certain patterns recurring in each application; the entire solution, however, has to be replenished in an indefinite manner.*

Fig. 2. Design Techniques of Reference Modeling according to [10] and [11]

These design techniques are specified on a general methodological level, independent of special modeling languages. So far, vom Brocke [10] has applied the design techniques to *Entity-Relationship-Diagrams* and *Event-driven Process Chains (EPCs)*. By incorporating the design techniques into inter-organizational models we foster re-use of these models. Hofreiter et al. [11] have already detected an appropriate example for each of the design techniques to be used with *UMM*, but thorough analysis of further re-use possibilities have to be conducted. We have already analyzed first steps towards inter-organizational reference models in [12]. The contributions of this thesis are discussed in the next section.

3 Research Contributions

In our *BSopt* project various modeling languages (i.e., *e3-value, REA, UMM*, and *Core Components*) were used to create an inter-organizational system. These languages include structural and behavioral models, respectively. During the evaluation of these modeling languages we identified a lack of re-use of existing models - except for copying & pasting parts of existing models. Thus, if two business partners want to create a model which is similar to an existing one, they basically have to start from scratch to create a new model and are not able to take advantages of existing ones. To enable and foster re-use of inter-organizational models, we define following research contributions of this thesis.

Inter-organizational Reference Models. The first research contribution addresses the definition of reference models for inter-organizational systems. We will consider the well elaborated modeling languages used in *BSopt*: *e3-value*, *REA*, *UMM*, and *Core Components*. In order to create reference models we have to identify the potential for re-use of the various original models. Therefore, sets of existing similar models have to be analyzed. Model instances are compared to each other and variating elements are identified. Additionally, we have to find the proper design techniques (i.e., *Analogy, Aggregation, Specialization, Instantiation*, or *Configuration*) for these variations in order to create the corresponding reference model which can be used as a basis to create new models.

Metamodel for Reference Models. The second research contribution concerns the creation of an appropriate metamodel for the reference models by extending the BSopt modeling languages. The metamodel has to cover all elements of the models as well as of the reference models (e.g. an additional element for a placeholder in a reference model, which will be further defined in the derived model). First, we have to study the existing metamodels and second, we have to identify locations in the metamodel to allow for placeholders/extension points for the variating elements on the model level. Furthermore, we have to include those extensions in the metamodel allowing for creation of models as well as of reference models for the various modeling languages.

Change Patterns. The third research question deals with identifying the change patterns to derive a model from a reference model. According to the design technique of reference modeling, special points of extensions for the reference model will be enabled for the modeler. The change patterns have to be

specified on a meaningful level of abstraction covering a set of changes on single nodes and edges. To help the modeler to create the model instances, wizards conforming to the change patterns will be created to guide the modeler and keep him from the modeling details. Additionally, traces between the newly created model and the reference model have to be kept. This will allow future changes of the reference model to be propagated to the derived models and is a prerequisite for proper model evolution.

These research contributions allow us to create reference models as well as provide them to modelers in order to create model instances. We expect a more efficient modeling process since the modeler does not have to start from scratch and a higher quality of the models by building upon best practice.

4 Research Methods

In this thesis we follow a Design Science approach [13], which is commonly used in the discipline of information systems (IS). As the area of IS research is at the "intersection" of people, organizations, and technology, Design Science is not only about designing, but also observing and validating "implemented" changes. Design Science identifies design processes (i.e., build and evaluate) and design artifacts. The latter are constructs (e.g., specific vocabularies and symbols), models, methods, and instantiations (e.g., prototype systems). These artifacts enable researchers to understand and to address the problems in developing, implementing, and validating information systems. The following phases are conducted.

Literature Study Right now we study related work in the area of reference modeling and design techniques to identify existing work. Additionally, we will evaluate existing tools in this area with a focus on the five design techniques. Furthermore, we also consider re-use concepts from software engineering, which reference modeling is related to.

Identifying Re-use of Models Once we have studied the different design techniques we will analyze the models used in *BSopt* (i.e., *e3-value, REA, UMM*, and *Core Components*) for their suitability for reference modeling. Thus, similarities in a set of likewise models have to be detected.

Defining Reference Models and Change Patterns After we detected the variating model elements, proper extension points/placeholders in models have to be defined. The existing metamodels have to be extended to allow for these extension points/placeholders. Additionally, change patterns describing the transformation from a reference model to a model instance have to be defined.

Prototype Implementation This phase will use the reference models from the previous phase and implement a prototype using domain-specific languages. The prototype will be built upon our existing *BSopt* tool, which already proved to be consistent.

Prototype Evaluation To validate our assumptions we will evaluate the reference modeling tool. Therefore, the evaluation builds upon three major pillars: *Experiments* with students building models from scratch and students building models using our reference modeling approach; *Case Study* with an appropriate

company including questionnaire as well as face-to-face interviews; *Collaborative studies* in workshops with experts of the used models.

5 Conclusion

The goal of this thesis is to provide reference modeling for inter-organizational systems. Thus, the five design techniques from reference modeling *Configuration*, *Instantiation*, *Specification*, *Aggregation*, and *Analogy* are incorporated on the models used in the *BSopt* methodology. This requires the extension of the existing metamodels to provide further to be specified placeholders in the models. Additionally, change patterns have to be defined to transform a reference model to a model utilizing wizards. The prototype implementation for *BSopt* reference modeling is expected to lower the complexity and fasten the process of modeling inter-organizational systems as well as to provide higher quality models.

References

1. Huemer, C., Liegl, P., Schuster, R., Werthner, H., Zapletal, M.: Inter-Organizational Systems: From Business Values over Business Processes to Deployment. In: Proceedings of the 2nd International IEEE Conference on Digital Ecosystems and Technologies, IEEE Computer Society (2008) 294–299
2. Gordijn, J., Akkermans, H.: Designing and Evaluating e-Business Models. IEEE Intelligent Systems - Intelligent e-Business **16**(4) (2001) 11–17
3. McCarthy, W.E.: The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Env. The Accounting Review **57**(3) (1982)
4. UN/CEFACT: UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - FoundationModule. (October 2009) Implementation Verification Draft 2.0.
5. Geerts, G.L., McCarthy, W.E.: An Accounting Object Infrastructure for Knowledge-Based Enterprise Models. IEEE Intelligent Systems **14**(4) (1999) 89–94
6. UN/CEFACT : Core Components Technical Specification 3.0. (November 2009)
7. Liegl, P., Mayrhofer, D.: A Domain Specific Language for UN/CEFACT's Core Components. In: Proceedings of the 5th World Congress on Services, IEEE Computer Society (2009) 123–131 SC4B2B, Bangalore.
8. Fettke, P., Loos, P.: Perspectives on Reference Modeling. In Fettke, P., Loos, P., eds.: Reference Modeling for Business Systems Analysis. Idea (2007) 1–20
9. Thomas, O.: Understanding the Term Reference Model in Information System Research. In: Business Process Management (2005) Workshops, Revised Selected Papers, Springer LNCS (2005) 16–29
10. vom Brocke, J.: Design Principles for Reference Modeling - Reusing Information Models by Means of Aggregation, Specialisation, Instantiation, and Analogy. In: Reference Modeling for Business Systems Analysis. Idea (2007) 47–76
11. Hofreiter, B., vom Brocke, J.: On the Contribution of Reference Modeling to e-Business Standardization How to Apply Design Techniques of Reference Modeling to UMM. In: BPM Workshops, Springer LNBIP (2009) 671–682
12. Hofreiter, B., Huemer, C., Kappel, G., Mayrhofer, D., vom Brocke, J.: Inter-organizational Reference Modeling - A Position Statement. In: International Workshop on Business System Management and Engineering, Tools conference. (2010)
13. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Quarterly **28**(1) (2004) 75–105

Applying Architecture Modeling Methodology to the Naval Gunship Software Safety Domain

Joey Rivera

US Naval Postgraduate School
1 University Circle
Monterey, California, USA
jrivera@nps.edu

Abstract. In this paper we describe the results of a research project that involved solving the problem of evaluating potential changes to a naval gunship system architecture. The prototype software developed for this domain solution (called "Eagle6") extends the Monterey Phoenix (MP) software architecture formal specification framework by creating an abstract layer of macro commands that are designed for assertion checking. The Eagle6 solution uses assertion checking to test for software states that are considered unsafe.

Keywords: Software System Architecture, Modeling, Environmental Modeling, Assertion Checking

1 Problem

Naval Gunship software system architecture requires consideration for loss of life [6]. The US Navy's Software System Safety Technical Review Panel (SSSTRP) evaluates potential software systems for US Navy Gunships during the initial acquisition process. The SSSTRP organization has identified an unacceptable rate of software acquisition evaluation failures due to issues related to software safety [7]. The high failure rate results in the vendor being required to submit additional documentation and an acquisition process (and timeline) that is unacceptable to the US Navy Acquisition community.

2 Related Works

Leveson's work in Software Safety is the foundation for understanding the risks of poor software engineering [10]. Due to the seriousness of the potential risks, the SSSTRP requires exhaustive testing before modifications are approved. Jackson's "Small Scope Hypothesis" [2] [5] argues that a high proportion of bugs can be found by testing the system within some small scope. Conventional notations, such as UML, have long been the preferred tool of software architects, but the software architect needs a number of different views of the software architecture for the various uses and users [3] [4]. Architecture modeling provides a high-level abstraction for representing structure, behavior, and key properties of a software sys-

tem [8] [9]. These abstractions are useful in describing complex problems to various stakeholders in an understandable manner. The use of the Monterey Phoenix (MP) methodology [1] enables software architects to model system's environments in such a way that enables the non-technical stakeholder to understand the potential effects when making changes to their environment.

3 Proposed Solution: System Architecture Modeling Methodology for Naval Gunship Software

Eagle6 is a software product that uses MP's formal architectural modeling methodology to test software engineering architecture [1]. Because MP is based on Formal Methods, Eagle6 is capable of generating all possible scenarios within scope, and then evaluates the scenarios for assertion violations. Eagle6 models provide high-level abstractions for representing the structure, behavior, and key properties of software system relative to how the software system interacts with its environment. MP models use events to describe how system components interact with their environment. Events contain Attributes that enable the modeler to further refine model representation of system behavior. Attributes are particularly valuable as they represent a more detailed (and measurable) application state, thereby enabling the evaluation of formal assertions. This paper demonstrates how Eagle6 is able to test the high-level relationships between gunship systems, while continuing to support model refinement to reflect a more detailed (and evolving) system view.

3.1 Assertion Checking

Eagle6 processes the MP model and generates all possible scenarios within the model scope. Eagle6 uses an exhaustive search for assertion violations, and then returns a list of assertion counter examples.

The below MP model represents the process the Gun Console Computer (GCC) executes to track target data. The GCC_activity sets target info and starts sending the tracking data to the Command and Decision (CD) system. The GCC also requests more info from the 3D Radar (R3D). If any of these events fail, the event trace may end in the GCC being unable to track the target:

```
SCHEMA Gun_Console_Computer_Activity
ROOT GCC_activity: {(*<0-1/0.2,0.8>  GCC_setTarget *)};
GCC_setTarget: (CD_request_GCC_setTarget
  (GCC_targetNotSet | <0.9>(GCC_request_R3D_setTarget
  GCC_wait_R3D_setTarget (
  (R3D_targetNotSet GCC_targetNotSet)|
  <0.97> (R3D_targetSet GCC_targetSet)))));
```

Notation: {} = Unordered Events; () = Ordered Events; <s/p> = scope/probability; A full explanation of MP notation can be found at www.Eagle6.com.

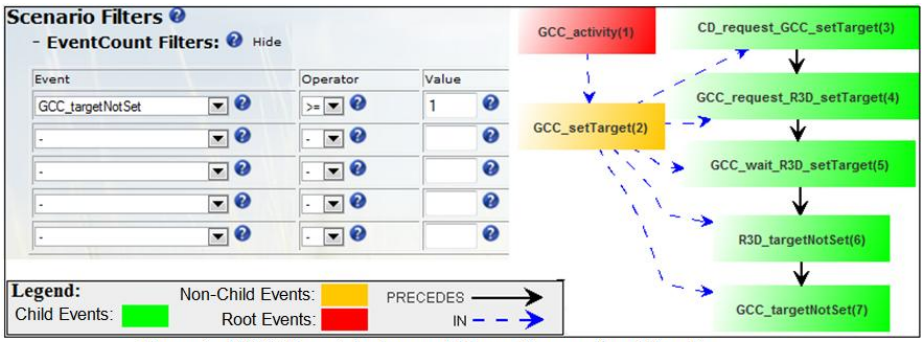


Figure 1: GCC_Target Assignment Macro Command and Results

Figure 1 represents two steps of an Eagle6 process. The Scenario Filter allows for macro commands to be run against the model in order to return all scenarios that meet query conditions. The second is a scenario that was returned from the preceding macro command.

Macro commands can be executed in three predefined formats as individual or combined operations:

- EventCount(EventType, Operator, Value) - used to return only scenarios that have a min/max number of specific events.
- MaxSliceSum(AttributeName, Operator, Value) - used to return only scenarios that have a min/max number of parallel events.
- MaxChainSum(AttributeName, Operator, Value) - used to return only scenarios that have a min/max number of events that happen in sequence within the scenario.

Event grammar rules specify a set of possible event traces representing different scenarios for the GCC_activity. Note the "|" "Or" event grammar allows the modeler to test for scenarios where the GCC_Activity event ended in the target not being set. This capability was particularly helpful when testing the data transfer requirements within the gunship system.

3.2 Formal Verification of Model Properties

The SSSTRP evaluation process requires a formal verification process that evaluates all possible scenarios within the model scope. Eagle6 is able to generate all possible scenarios within scope while simultaneously evaluating the model for assertion violations. In the below example we considered an unsafe state to be any scenario where the Command and Decision (CD) system sends an OpenFire command, and the number of manual approvals required to execute the command is >= 3:

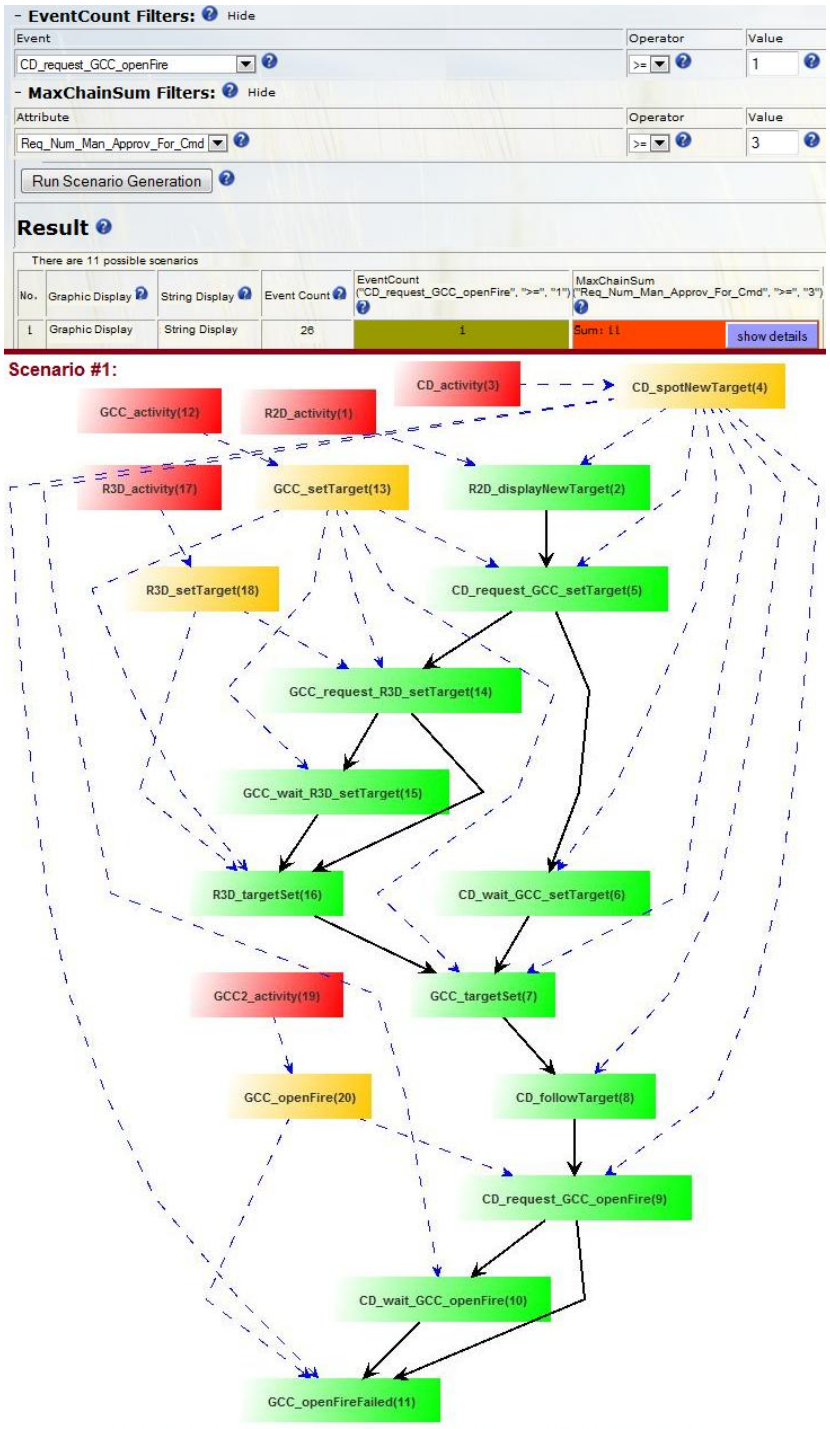


Figure 2: Counter Example of Formal Assertion via Macro Command

4 Expected Contributions

Environment Behavior – Evaluating a gunship system requires the ability to model the system and its environment. Formally testing a system's functional profile while introducing external behavioral events is key to understanding how the system will react in a combat situation. The SSSTRP has the capability to introduce hypothetical environmental events to test for unsafe system states.

Performance Estimates – Early in the design process, non-functional requirements need to be tested in order to see how the integration of a system will impact the overall System of Systems (SoS). Creating a single SoS MP model prior to the acquisition process enables the acquisition community to standardize the vendor response questionnaire, thereby resulting in a formal method of evaluating potential software solutions. Furthermore, incorporating a technical questionnaire that elicits MP model input responses from potential vendors is expected to streamline the vendor's response efforts as well as the evaluation methodology.

Model Views – The SSSTRP evaluation team can extract representations of the model in visual and textual views in order to communicate the results to both technical and non-technical SSSTRP members.

5 Current Status

The initial MP framework was applied to the Navy Gunship Software Safety domain and tested using an MP compiler developed by this research team (www.Eagle6.com). The tool has created an executable code that bridged the gap between concept and design and proved to be critical during the testing process. The tool is made publically available to all internet users, and we are receiving feedback on a daily basis. The initial implementation of the framework satisfied our objective of creating a framework that enables an ability to evaluate the integration risks associated with potential gunship software solutions. We have modeled a gunship software system using Eagle6 and are able to test for unsafe states using the Eagle6 software.

6 Plans for Evaluation

Evaluating the effectiveness of the Eagle6 software as applied to the Gunship Software Safety domain resulted in the evaluation of the following domain-specific solution properties:

Dynamic Reuse and Extensibility - Reuse is supported through the event grammar capability to add/edit/delete MP abstract components (Schemas). Since the interaction among systems and events is decoupled via connectors, a high degree of dynamism is provided with minimal disturbance to the rest of the model. The Eagle6

framework supports a simple, faithful mapping between the architectural elements and their implementation. Eagle6 has the capability to map Use Cases to functional design requirements. This fidelity is necessary to ensure that the conceptual integrity and key properties of an application's architecture are preserved by the architecture's implementation.

Efficiency - We used a custom event trace generator created specifically to test MP models. The Eagle6 event trace generator enabled us to test MP models in relatively short test cycles, thereby resulting in fewer errors in MP programming, standardized MP modeling methodology, and less time needed to change the MP test model to better reflect the Gun System and its environment.

Assertion Checking - MP does an exhaustive search through all possible scenarios (up to the model's scope limit). MP provides a means to write assertions about the system behavior and tools to verify those assertions. This function is non-trivial since having the ability to write event grammar that supports formal assertion checking allows the architect to test for unsafe states by evaluating system attributes.

Acknowledgements: The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

References

1. Auguston, M.:, Software Architecture Built from Behavior Models. ACM SIGSOFT Software Engineering Notes, Volume 34 Number 5 (2009)
2. Jackson, D., Damon, C.A.: Elements of style: Analyzing a software design feature with a counterexample detector. IEEE Transactions on Software Engineering, 22(7) (1996)
3. Kruchten, P.: Architectural Blueprints - the 4+1 View Model of Software Architecture. pp. 42-5 IEEE Software 12 (6) (1995)
4. Perry, D.E., Wolf, A.: Foundations for the Study of Software Architecture. pp. 40-52. ACM SIGSOFT Software Engineering Notes, 17:4 (1992)
5. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. Cambridge, Massachusetts. The MIT Press (2006)
6. Auguston, M., Michael, B., Shing, M.: Environment Behavior Models for Automation of Testing and Assessment of System Safety, Information and Software Technology, pp. 971-980. Vol. 48, Issue 10 (2006)
7. Rivera, J., Luqi, Berzins,V.: Effective programmatic software safety strategy for US Navy Gun System acquisition programs. Proceedings of NPS 6th Annual Acquisition Research Symposium. 159-164, http://edocs.nps.edu/npspubs/scholarly/TR/2010/NPS-GSBPP-10-003.pdf (2009)
8. Perry, D.E., Wolf, A.L.: Foundations for the Study of Software Architectures. ACM SIGSOFT Software Engineering Notes, vol. 17, no. 4, pp. 40-52 (1992)
9. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall (1996)
10. Leveson, N.: Safeware: System Safety and Computers. Addison-Wesley (1995)

A Model-based Framework for Software Performance Feedback

Catia Trubiani

Dipartimento di Informatica, University of L'Aquila
Via Vetoio n.1, 67010 Coppito, L'Aquila, Italy
catia.trubiani@univaq.it

Abstract. The problem of interpreting the results of performance analysis is quite critical in the software performance domain: mean values, variances, and probability distributions are hard to interpret for providing feedback to software architects. Support to the interpretation of such results that helps to fill the gap between numbers and architectural alternatives is still lacking.
The goal of my PhD thesis is to develop a model-based framework addressing the results interpretation and the feedback generation problems by means of performance antipatterns, that are recurring solutions to common mistakes (i.e. bad practices) in the software development. Such antipatterns can play a key role in the software performance domain, since they can be used in the search of performance problems as well as in the formulation of their solutions in terms of architectural alternatives.

Key words: Software Architecture, Performance Evaluation, Antipatterns, Feedback Generation, Architectural Alternatives.

1 Problem

The trend in modeling and analyzing the performance of software systems is to build a cycle where models are derived form design artifacts, and results from models are evaluated in terms of design artifacts, so that the performance issues are brought to the forefront early in the design process [15].

Figure 1 schematically represents the typical steps that must be executed in the software life-cycle to run a complete performance process. Rounded boxes in the figure represent operational steps whereas square boxes represent input/output data. Arrows numbered from 1 through 4 represent the typical forward path from an (annotated) software architectural model all the way through the production of performance indices of interest. While in this path quite well-founded approaches have been introduced for inducing automation in all steps (e.g. [14]), there is a clear lack of automation in the backward path that shall bring the analysis results back to the software architecture.

The *core* step of the backward path (i.e. the shaded box of Figure 1) is the results interpretation and feedback generation: the performance analysis results have to be interpreted in order to detect, if any, performance problems, and once

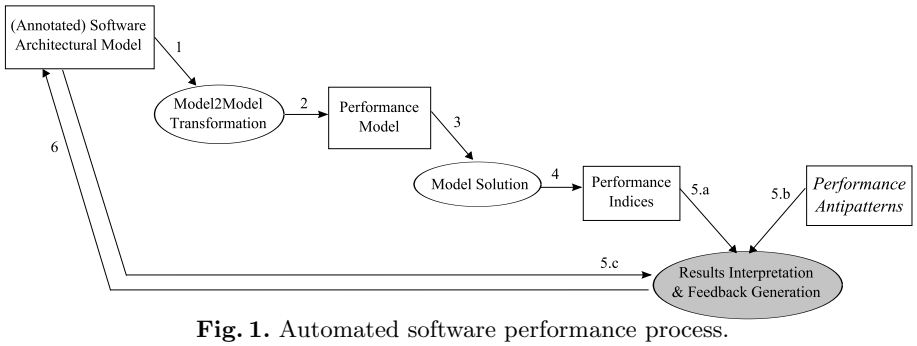


Fig. 1. Automated software performance process.

performance problems have been detected (with a certain accuracy) somewhere in the model, solutions have to be applied to remove those problems[1].

The search of performance problems in the architectural model may be quite complex and needs to be smartly driven towards the problematic areas of the model. The complexity of this step stems from several factors: (i) performance indices are basically numbers and often they have to be jointly examined: a single performance index (e.g. the utilization of a service center) is not enough to localize the critical parts of a software architecture, since a performance problem can be detected only if other indices (e.g. the throughput of a neighbor service center) are analyzed; (ii) performance indices can be estimated at different levels of granularity (e.g. the response time index can be evaluated at the level of cpu device, or at the level of services that span on different devices) and it is unrealistic to keep under control all indices at all levels of abstraction; (iii) architectural models can be quite complex, and the origin of performance problems emerges only looking at the architectural elements described in different views of a system (such as static structure, dynamic behaviour, etc.).

The research activity of my PhD thesis is focused on the core step, and in Figure 1 the most promising elements that can drive this search have been explicitly represented, i.e. *performance antipatterns* (i.e. input labeled 5.b to the core step). The aim is to introduce automation in the backward path providing a feedback to the architectural model (label 6) in the form of an architectural alternative that removes the original performance problems. The benefit of using antipatterns when closing the software performance cycle is that they base on a comprehensive view of the system, thus to capture complex phenomena. An antipatterns-based approach differs from (i) design space exploration techniques that blindly examine all architectural alternatives, (ii) genetic algorithms that search for local changes in the architectural model under analysis.

The main source of performance antipatterns is [13], where modeling notation-independent antipatterns are defined. Some other works present instances of antipatterns, but they are not as general as the ones in [13] (see Section 2).

[1] Note that this task very closely corresponds to the work of a physician: observing a sick patient (the model), studying the symptoms (some bad values of performance indices), making a diagnosis (performance problem), prescribing a treatment (performance solution).

2 Related work

One of the first proposals of automated generation of feedback due to the software performance analysis can be found in [9], where the detection of performance flaws is demanded to the analysis of a specific notation, i.e. Layered Queued Network (LQN) models, and uses informal interpretation matrices as support.

The issue of solving performance issues through antipatterns has been addressed in [12], where a Performance Antipattern Detection (PAD) tool is presented. However, PAD only deals with Component-Based Enterprise Systems and targets Enterprise Java Bean (EJB) applications. It is based on monitoring data from running systems from which it extracts the run-time system design and detects only EJB antipatterns. Its scope is restricted to running EJB systems, therefore it is not applicable in the early development stages.

Another interesting work on the software performance diagnosis and improvements has been proposed in [16]: performance flaws are identified before the implementation, even if they are related only to bottlenecks and long paths. Performance antipatterns, compared to simple bottleneck and long paths identification, help to find more complex situations that embed hardware and/or software problems. Additionally in [16] performance issues are identified at the level of the LQN performance model, and the translation of these model properties into design changes could hide some possible refactoring solutions, whereas the performance antipatterns give a wider degree of freedom for architectural alternatives, since they embed the solutions in their definition.

3 Proposed solution

In this Section a vision of the approach is discussed: the problem of interpreting the performance results and generating architectural alternatives is addressed with a model-based framework that supports the management of antipatterns.

The main activities performed within such framework are schematically shown in Figure 2: *specifying antipatterns*, to define in a well-formed way the properties that lead the software system to reveal a bad practice as well as the changes that provide a solution; *detecting antipatterns*, to locate antipatterns in software models; *solving antipatterns*, to remove the detected performance problems with a set of refactoring actions that can be applied on the system model.

The activity of specifying antipatterns is performed by introducing a metamodel (i.e., a neutral and a coherent set of interrelated concepts) to collect the system elements that occur in the definition of antipatterns (e.g. software resource, network resource utilization, etc.), which is meant to be the basis for a machine-processable definition of antipatterns. An antipattern definition includes: (i) the specification of the problem, i.e. a set of *rules* that interrogate the system elements to look for occurrences of the corresponding antipattern; (ii) the specification of the solution, i.e. a set of *actions* that are applied on the system elements to remove the original performance issues.

The activities of detecting and solving antipatterns are performed by respectively translating the antipatterns rules and actions into concrete modeling notations. In fact, the modeling language used for the target system, i.e. the

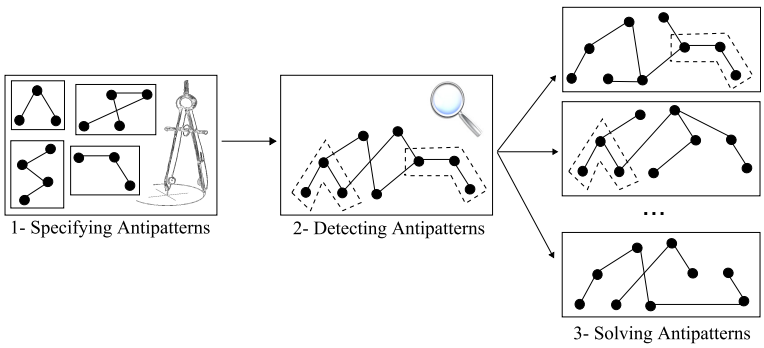


Fig. 2. The main activities of the model-based framework.

box *(annotated) software architectural model* of Figure 1, is of crucial relevance, since the antipatterns neutral concepts must be translated into the actual concrete modeling languages, if possible[2]. The framework is currently considering three notations: a system modeling language such as UML [2] and Marte profile [3]; a domain specific modeling language such as Palladio Component Model (PCM) [4]; an architectural language such as Æmilia [5]. In general, the subset of target modeling languages can be enlarged as far as the concepts for representing antipatterns are available; for example, architectural description languages such as AADL [1] can be also suited to validate the approach.

For example, the "Blob" antipattern [13] can be detected if a software resource requires a lot of information from the other ones, it generates excessive message traffic that lead to over utilize the available network resources. Figure 3 shows an example of the UML and Marte architectural model where the shaded boxes highlight the excerpts of the architecture evidencing the "Blob" (i.e. the *libraryController* UML component)[3]. Such antipattern can be solved by balancing in a better way the business logic among the available software resources and/or by re-deploy the blob software resource to avoid remote communications.

4 Expected contributions

The activity of specifying antipatterns provides several contributions: (i) the identification of the system elements of the antipatterns specification (e.g. software resource, network resource utilization, processing resource, etc.); (ii) the formalization of the antipatterns specification as logical predicates by introducing functions and threshold values[4]; (iii) the definition of a metamodel able to capture the antipatterns properties, i.e. the system model elements.

Performance antipatterns can be translated across the three different notations (i.e. UML and Marte, PCM, and Æmilia) that the framework considers.

[2] It depends on the expressiveness of the target modeling language.

[3] For example, a software resource can be represented as a *UML component*, the network resource utilization can be represented as the tagged value *utilization* of the MARTE stereotype *GaCommHost* applied to a *UML node*, etc.

[4] Threshold numerical values can be assigned by software architects basing on heuristic evaluations or they can be obtained by monitoring the system.

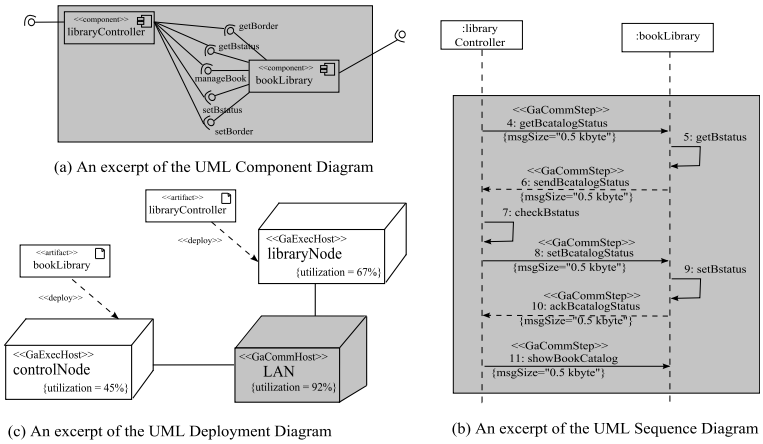


Fig. 3. An example of the "Blob" antipattern [13] in the UML and Marte profile modeling language.

More in general, in a concrete modeling language there are antipatterns that can be automatically detected and/or automatically solved and some others that are neither detectable and solvable. There is an intermediate level of antipatterns that are semi-automatically detectable by relaxing some rules, and/or semi-automatically solvable by devising some actions to be manually performed.

5 Current Status

The activity of *specifying antipatterns* is addressed in [8]: a structured description of the system elements that occur in the definition of antipatterns is provided, and performance antipatterns are modeled as logical predicates. Such predicates could be further refined by looking at probabilistic model checking techniques, as Grunske experimented in [11]. An AntiPattern Modeling Language, i.e. a metamodel specifically tailored to describe antipatterns, is introduced in [6].

The activities of *detecting* and *solving antipatterns* are currently implemented only for the UML and Marte profile notation in [7]: the antipattern rules and actions are translated into that system modeling language.

Additionally, instead of blindly moving among the antipattern solutions without eventually achieving the desired results, a technique to rank the antipatterns on the basis of their guiltiness for violated requirements is defined in [10], thus to decide how many antipatterns to solve, which ones and in what order.

The current work is on investigating the PCM modeling language and the Æmilia architectural language by experimenting the antipatterns that can be specified in these concrete notations.

6 Plan for evaluation

The aim of my PhD thesis is to provide automation in the backward path from the performance analysis to software modeling by means of antipatterns. The experimentation on the UML and Marte profile modeling language validates the applicability of the whole approach, i.e. the support to results interpretation and feedback generation by specifying, detecting, and solving antipatterns.

As a short term future goal the approach has to be validated to complex case studies, allowing to study its actual usability and scalability. Such experimentation is of worth interest because the final purpose is to integrate the framework in the daily practices of software architects.

In a longer term, some critical pending issues have to be faced in order to automate the whole process. The solution of antipatterns generates three main categories of problems: (i) the convergence problem, i.e. the solution of one or more antipatterns might introduce new antipatterns; (ii) the requirement problem, i.e. when one or more antipatterns cannot be solved due to pre-existing (functional or non-functional) requirements; (iii) the coherency problem, i.e. when the solution of a certain number of antipatterns cannot be unambiguously applied due to incoherencies among their solutions.

References

1. SAE, Architecture Analysis and Design Language (AADL), June 2006, as5506/1, http://www.sae.org.
2. UML 2.0 Superstructure Specification, Object Management Group, Inc. (2005), *http://www.omg.org/cgi-bin/doc?formal/05-07-04*.
3. UML Profile for MARTE beta 2, Object Management Group, Inc. (2008), *http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf*.
4. S. Becker, H. Koziolek, and R. Reussner. The Palladio Component Model for model-driven performance prediction. *JUSS*, 82:3–22, 2009.
5. M. Bernardo, L. Donatiello, and P. Ciancarini. Stochastic Process Algebra: From an Algebraic Formalism to an Architectural Description Language. In *Performance*, pages 236–260, 2002.
6. V. Cortellessa, A. Di Marco, R. Eramo, A. Pierantonio, and C. Trubiani. Approaching the Model-Driven Generation of Feedback to Remove Software Performance Flaws. In *EUROMICRO-SEAA*, pages 162–169, 2009.
7. V. Cortellessa, A. Di Marco, R. Eramo, A. Pierantonio, and C. Trubiani. Digging into UML models to remove performance antipatterns. In *ICSE Workshop Quovadis*, pages 9–16, 2010.
8. V. Cortellessa, A. Di Marco, and C. Trubiani. Performance Antipatterns as Logical Predicates. In *IEEE ICECCS*, pages 146–156, 2010.
9. V. Cortellessa and L. Frittella. A framework for automated generation of architectural feedback from software performance analysis. In *Proc. of Formal Methods and Stoc. Models for Perf. Eval.*, volume 4748 of *LNCS*, pages 171–185, 2007.
10. V. Cortellessa, A. Martens, R. Reussner, and C. Trubiani. A Process to Effectively Identify "Guilty" Performance Antipatterns. In *FASE*, pages 368–382, 2010.
11. L. Grunske. Specification patterns for probabilistic quality properties. In *ICSE*, pages 31–40, 2008.
12. T. Parsons and J. Murphy. Detecting Performance Antipatterns in Component Based Enterprise Systems. *Journal of Object Technology*, 7(3):55–90, 2008.
13. C. U. Smith and L. G. Williams. More new software performance antipatterns: Even more ways to shoot yourself in the foot. In *Comp. Meas. Group Conf.*, 2003.
14. C. M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr, and J. Merseguer. Performance by Unified Model Analysis (PUMA). In *WOSP*, pages 1–12, 2005.
15. C. Murray Woodside, Greg Franks, and Dorina C. Petriu. The Future of Software Performance Engineering. In *FOSE*, pages 171–187, 2007.
16. J. Xu. Rule-based Automatic Software Performance Diagnosis and Improvement. In *WOSP*, pages 1–12. ACM, 2008.

Scenario-based Analysis of UML Design Class Models

Lijun Yu

Department of Computer Science, Colorado State University
Fort Collins, CO 80523, USA
lijun@cs.colostate.edu

Abstract. Identifying and resolving design problems in the early software development phases can help ensure software quality and save costs. In this research we will develop a lightweight approach to uncovering errors in UML design models consisting of class diagrams with invariants and operation specifications expressed in the OCL. In this analysis method a set of scenarios created by a verifier is used to check the functionality specified by operations in the design class diagrams. The scenario generation technique utilizes use cases, misuse cases and activity diagrams describing allowed and disallowed functionality. The method is lightweight in that it analyzes behavior of UML design class model within the scope of a set of scenarios. It is static in that it does not animate or execute the UML design. Empirical studies will be done to evaluate the effectiveness of the method.

Keywords: Scenario, Snapshot, Static Analysis, Scenario Generation, Use Case, Misuse Case, System Operation

1 Problem

Tool-supported rigorous analysis of design models can enhance the ability of developers to identify potential design problems earlier. Detecting and resolving design problems in the design phase prevents them from getting into later development and production phases where they are more costly to fix.

In this doctoral research a scenario-based UML design analysis method will be developed. UML has been accepted by the industry as a de facto standard for object-oriented modeling. The research aims to answer the following questions:

- How can existing class diagram static analysis tools be used to support analysis of functionality?
- What types of errors can be uncovered in scenario-based analysis of class diagrams?
- How can scenarios be used to systematically analyze UML design class diagrams?

In this work scenario is a sequence of system operation calls. System operation is an operation that is directly called by external actors. Scenarios can be described using UML sequence diagrams.

Existing UML and OCL analysis tools such as OCLE [OCLE] and USE [USE] can be used to check whether snapshots describing system state satisfy the properties

expressed in class diagrams. OCLE tool does not analyze operation contracts. The USE tool can be used to analyze operation constraints by simulating behavior of operations in interactive command mode. The proposed research will investigate how these tools can be used to support analysis of functionality against a set of scenarios describing desirable and undesirable behavior.

2 Related work

2.1 UML design analysis and testing techniques

Formal analysis tools such as the Alloy Analyzer [Alloy] and model checking tools [Clarke99] do exhaustive search in the modeled state space to find solutions that satisfy constraints or verify given properties. Formal theorem proving tool such as Isabelle can be used to reasons about properties in the model in interactive mode [Brucker08]. All these techniques need to transform UML designs to their own languages to analyze the UML designs. However, it is a challenging problem to prove the correctness of transformation. For example, [UML2Alloy] has not proved that the transformed Alloy model is syntactically correct.

There is related work that dynamically analyzes executes UML models. In UML Animation and Testing approach (UMLAnT) [Trung05], the UML design under test (DUT) contains UML design class diagrams and UML sequence diagrams. Operations in the class diagrams are specified with OCL pre and post conditions and actions using a Java-like Action Language (JAL). The sequence diagram describes realization of one system operation and test inputs are derived from the sequence diagram using a constraint solver. The UML design is executed to find the inconsistencies between the operation actions and OCL operation contracts. Compared with UMLAnT, the scenario-based UML design analysis method described in this work does not execute the UML design. It is a static technique in which the UML design class diagram is checked with the behavior of scenarios to find design errors. Also the method proposed in this work analyzes sequence of system operations instead of realization of one system operation.

Similar to UML testing technique, UML animation techniques try to execute UML designs. Oliver and Kent propose a UML animation technique to validate a UML design [Oliver99]. In their work UML design class diagrams are animated by mapping OCL constraints to operations on snapshots. In another piece of work Krieger and Knapp use a SAT solver to find new system state that satisfies operation post-conditions [Krieger08]. Compared with UML animation techniques, scenario generation in this work does not depend on operation contracts.

2.2 Test scenario generation from UML requirements model

There are a few approaches that generate test scenarios from use cases in the UML model. All of them generate test cases for testing the code that implements the UML model. Briand and Labiche proposed an approach to generate test data and test oracles from UML analysis model for system testing [Briand02]. In their work system test

requirements are automatically derived from UML analysis artifacts. Valid use case sequences are generated based on use case sequential constraints described using activity diagrams. Use case sequences can be interleaved and each use case may have use case variances which are described using a decision table. The method depends on the verifier's knowledge to select test cases from a large number of interleaved use case sequences and use case variances, also not all the use case and use case variance sequences are feasible. In this work constraint solving technique is used to find initial system state and system operation parameters for all feasible paths in the activity model. Nebut et. al. proposed a use-case driven approach to generate system test inputs [Nebut06]. In their work use cases are fully specified with pre and post conditions. Use cases are built into a Use Case simulation and Transition System (UCTS). Valid instantiated use case sequences are generated by exhaustively simulating the system. The limitation of the approach is that the space of UCTS may be huge when many use cases can be applied at each step of simulation. Kundu and Samanta use UML activity diagram that describes activity sequences inside one use case to generate system test cases [Kundu09]. In their work the activity diagram is converted to an activity graph and test sequences are generated from the graph based on different coverage criteria. Compared with the scenario generation approach in this work, their work focuses on generating action sequences from the UML activity diagram and the effects of actions are not formal specified and taken into consideration in test input generation.

3 Proposed solution

The goal of this work is to develop a tool-supported lightweight static method to analyze functionality specified in UML design class models. The method consists of two independent techniques: a scenario-based UML design analysis technique (Fig.1) and a scenario generation technique.

The scenario-based UML design analysis technique checks a UML design class model against a set of scenarios created by the verifier [Yu08] [Yu07]. The scenario-based analysis method extends the applicability of static analysis tools such as USE and OCLE to behavior analysis. The UML design class model is transformed to a snapshot model which captures valid snapshot transitions as determined by the operation specifications defined in the class model. A scenario captures a sequence of system operation calls and parameters of all the operations are instantiated. The behavior of a scenario is defined as a sequence of snapshot transitions. Snapshot transition describes the behavior of an operation in terms of how system state changes after the operation is invoked. A snapshot transition consists of (1) the name and parameter values of the operation that triggers the transition, (2) a before-snapshot describing the state of the system before the operation is executed, and (3) an after-snapshot describing the state of the system state after the operation has executed. The snapshot transitions which capture the behavior of scenarios are checked against the snapshot model to find inconsistencies between them. This can be done by leveraging existing UML design tool such as USE [USE] and OCLE [OCLE] which checks the consistency between a UML class model and a snapshot. The output of the scenario-

based UML design analysis technique is a set of inconsistencies between the functionality specified in the design class model and the behavior described by a scenario.

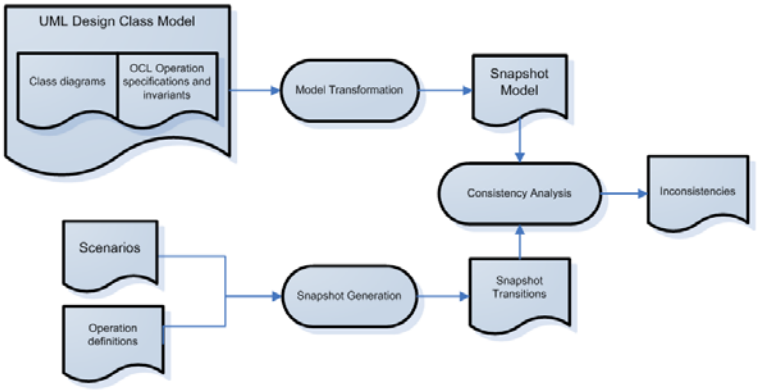


Fig. 1. Scenario-based UML design analysis technique

The scenario generation technique generates scenarios from the requirements or the verifier's domain knowledge. A basic scenario generation technique that generates scenarios from the verifier's domain knowledge was proposed in [Yu09]. A systematic scenario generation technique will be developed to generate a set of scenarios from the requirements model for verification of functionality at the system level. We prefer to generate scenarios from the requirements, because the essence of the scenario-based design analysis technique is to check the consistency between structural UML design and the behavior of scenarios: scenarios generated from the requirements better reflect the verifier's view of system functionality than scenarios generated from the design which capture the designer's view.

The requirements model contains UML use case diagram, requirement class diagram and activity diagram. The use case diagram specifies use cases which describe expected behavior and misuse cases which describe unexpected behavior. Each use case and misuse case is formally described as a system operation with OCL pre and post conditions. The requirement class diagram describes entities and their relationships in the requirements model. It differs from the design class diagram in that the design class diagram refines the requirement class diagram with detailed operation specifications. The activity diagram describes flow of control among use cases and misuse cases. In the Fusion method valid system operation sequences are described in a life-cycle model using life-cycle expressions [Fusion93]. UML activity diagram gives a graphical description of the use case sequences, it also describe parameters and parameter dependencies of system operations by modeling parameters as object nodes.

To generate scenarios from the requirements model, a set of coverage criteria will be defined. For example, action coverage criteria requires that each action (system operation) in the activity diagram must be covered at least once, and activity edge coverage criteria requires that the each edge in the activity diagram be covered at least once. Based on certain coverage criteria, a set of system operation paths are selected from the activity diagram. For each system operation path in the activity diagram, the pre and post conditions of the operations in the path and all branch and guard

conditions are automatically generated and fed to a constraint solver such as Alloy, the constraint solver finds an initial system state and system operation parameters in the system operation path. This will generate a set of requirement-level scenarios and these scenarios are mapped to the design-level system operation sequences for design analysis purpose.

4 Expected contributions

The major contribution of this dissertation is a tool-supported scenario-based UML design analysis technique. In the technique a verifier creates a set of scenarios to analyze a UML design class model. The technique transforms the UML design model to a snapshot model and checks consistencies between the snapshot model and snapshot transitions which capture the behavior of scenarios. Another contribution is a scenario generation technique. The technique generates a set of scenarios from the allowed system operation sequences in the requirements model. The generated system operation sequences can be used as a subset of system test cases for system testing.

5 Current status

The scenario-based UML design analysis technique has been proposed and published in [Yu07] and [Yu08]. A basic scenario generation technique that generates scenarios from the verifier's domain knowledge was proposed in [Yu09]. This technique depends on the verifier's domain knowledge. A tool that supports the model transformation and scenario generation is being implemented in Eclipse EMF and the Kermeta meta-programming environment. After the tool is implemented, empirical studies will be done to evaluate the design analysis technique. The scenario generation technique proposed in this work will be further studied and evaluated.

6 Plan for evaluation

The scenario-based UML design analysis technique will be evaluated by an empirical study. The goal of this study is to investigate: 1) can the technique effectively find inconsistencies between UML class model and scenarios? 2) what types of inconsistencies can the technique uncover? In the study, students with background in object-oriented design, UML and OCL will be assigned to two groups. A UML design model including UML design class diagram and OCL constraints, scenarios described by UML sequence model will be prepared. A number of inconsistencies will be seeded in the design class model and scenarios. The first group of students manually inspects the design model to find inconsistencies. The second group of students uses the scenario-based analysis tool to check the consistencies between the scenarios and the UML design. Number and type of inconsistencies found by each student will be recorded and results from the two groups will be compared.

Another study will be conducted to evaluate the scenario generation technique. The goal of this case study is to investigate whether the scenario generation tool generates scenarios that effectively uncover design errors. The scenario generation technique will be used to generate scenarios. At the same time a random scenario generator is used to generate scenarios as comparison. Design errors uncovered by the scenario generation technique and the random scenario generator will be compared.

References

1. [Alloy] D. Jackson, "Alloy: a lightweight object modeling notation", ACM Transactions on Software Engineering and Methodology, Volume 11, Issue 2, April 2002, pp 256-290.
2. [Briand02] Lionel Briand, Yvan Labiche, “A UML-Based Approach to System Testing”, Software and Systems Modeling, vol. 1 (1), pp. 10-42, 2002.
3. [Brucker08] Achim D. Brucker and Burkhart Wolff. HOL-OCL - A Formal Proof Environment for UML/OCL. In Fundamental Approaches to Software Engineering. Lecture Notes in Computer Science (4961), pages 97-100.
4. [Clark99] E. Clark, O. Grumberg, and D. Peled. Model Checking. The MIT Press, 1999.
5. [Fusion93] Derek Coleman, Object-Oriented Development: The Fusion Method, 1993.
6. [Krieger08] Krieger, M. P. & Knapp, A. Executing Underspecified OCL Operation Contracts with a SAT Solver. ECEASST, 2008, 15
7. [Kundu09] Debasish Kundu and Debasis Samanta, “A Novel Approach to Generate Test Cases from UML Activity Diagrams”, Journal of Object Technology, Volume 8, no. 3 (May 2009), pp. 65-83.
8. [Nebut06] Cle'mentine Nebut, Franck Fleurey, Yves Le Traon, Jean-Marc Je'ze' quel, "Automatic Test Generation: A Use Case Driven Approach," IEEE Transactions on Software Engineering, vol. 32, no. 3, pp. 140-155, Mar. 2006, doi:10.1109/TSE.2006.22
9. [OCLE] D. Chiorean, M. Pasca, A. Carcu, C. Botiza, S. Moldovan, "Ensuring UML Models Consistency Using the OCL Environment", Electronic Notes in Theoretical Computer Science, Volume 102, Nov. 2004, pages 99-110.
10. [Oliver99] Iam Oliver, Stuart Kent, "Validation of Object Oriented Models using Animation," euromicro, vol. 2, pp.2237, 25th Euromicro Conference (EUROMICRO '99)-Volume 2, 1999
11. [Trung05] T. Dinh-Trong, N. Kawane, S. Ghosh, R. B. France, and A. A. Andrews. "A Tool-Supported Approach to Testing UML Design Models", ICECCS '05, IEEE Computer Society Press, pp.519-528, Shanghai, China, June 16-20, 2005.
12. [UML2Alloy] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, Indrakshi Ray: On challenges of model transformation from UML to Alloy. Software and System Modeling 9(1): 69-86 (2010).
13. [USE] Gogolla, M., Büttner, F., and Richters, M. 2007. USE: A UML-based specification environment for validating UML and OCL. Sci. Comput. Program. 69, 1-3 (Dec. 2007)
14. [Yu07] Lijun Yu, Robert B. France, Indrakshi Ray, and Kevin Lano, "A Light-Weight Static Approach to Analyzing UML Behavioral Properties", Proceedings of the 12th IEEE International Conference on Engineering of Complex Computer Systems, Auckland, New Zealand, July 2007.
15. [Yu08] Lijun Yu, Robert France, Indrakshi Ray, "Scenario-based Static Analysis of UML Class Models", Proceedings of ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems, Toulouse, France, Sep. 28-Oct.3, 2008.
16. [Yu09] Lijun Yu, Robert France, Indrakshi Ray, Sudipto Ghosh, "A Rigorous Approach to Uncovering Security Policy Violations in UML Designs", Proceedings of the International Conference on Engineering Complex Computer Systems, Potsdam, Germany, June 2009.

Table of Contents