Electronic Communications of the EASST
Volume 21 (2009)

Proceedings of the
3rd International Workshop on
Multi-Paradigm Modeling
(MPM 2009)

Toward Automated Verification of Model Transformations: A Case Study of Analysis of Refactoring Business Process Models

Márk Asztalos, László Lengyel and Tihamér Levendovszky

5 pages

Guest Editors: T. Levendovszky, L. Lengyel, G. Karsai, C. Hardebolle
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Toward Automated Verification of Model Transformations: A Case Study of Analysis of Refactoring Business Process Models

Márk Asztalos, László Lengyel and Tihamér Levendovszky

Budapest University of Technology and Economics
Department of Automation and Applied Informatics
Magyar Tudósok körútja 2., I. em., Budapest, H-1111, Hungary
{asztalos, lengyel, tihamer}@aut.bme.hu

Abstract: Verification of the transformations is a fundamental issue for applying them in real world solutions. We have previously proposed a formalization to declaratively describe model transformations and proposed an approach for the verification. Our approach consists of a reasoning system that works on the formal transformation description and deduction rules for the system. The reasoning system can automatically generate the proof of some properties. In this paper, we present a case study, to demonstrate our approach of automated verification of model transformations in a multi-paradigm environment.

Keywords: model transformations, automated verification, offline analysis

1 Introduction

In Multi-Paradigm Modeling , model transformations are used to transform models between different paradigms (for example, synchronization), or for generating new models by composing existing models (for example, generating source code using different models in different paradigms) . Verification of model transformations means proving some properties of the model transformations functional and non-functional properties as well, and some properties of the models under transformation. The analysis of a transformation is called *offline*, when only the definition of the transformation itself and the metamodels of the target and source languages are used during the analysis process and no concrete input models are taken into account. Formal verification methods proposed in related work can usually be applied only to a certain (type of) transformation or for the analysis of only a certain (type of) property (see Section 3 for details). Moreover, many of these methods can only be applied manually, therefore, no automation can be built in the analysis process at all.

2 Background

Graph rewriting-based model transformations offer a strong mathematical background for the formalization and analysis of model transformations [EEPT06] . In model transformation frameworks, each transformation consists of separate rules and an additional control structure (control flow) that defines the execution order of the rules. Models can be considered special graphs, for these applications, attributed typed graphs have been introduced.

Previously [ALL09], we have introduced the terms *transformation description* and *assertion*: an assertion is a formal expression that describes some characteristics of a model under transformation or the model transformation itself. Informally, a *transformation description* is a set of assertions. In other words, an assertion states something about the model or the model transformation. We want our system to prove if some assertions are true or prove if they are false. We have proposed the formal definition language for describing the assertions. We have also defined deduction rules for the reasoning system that works on an initial assertion set that describe a model transformation and may derive some more assertions automatically applying the deduction rules. We want to generate some assertions automatically from the very definition of the model transformations. We also let the developer of the model transformation to provide additional assertions for different parts of the transformation manually, therefore, the knowledge of domain experts can be used. The reasoning system should use these assertions as well.

3 Related Work

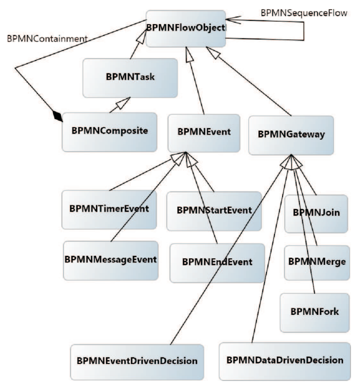
Offline analysis of model transformations have been performed in several cases, but the approaches presented can usually be applied for only certain (type of) transformations, or only for certain (type of) properties. In [Var02], syntactic correctness and semantic correctness are aimed to be verified by the VIATRA transformation system. Converting system models into Kripke structures allows to verify certain properties of a single transformation starting from a single model. [KN08] presents how behavior preservation of a model transformation can be verified via goal-directed certification. The proposed verification realized in GReAT is based on the generation of assurances for code produced by automatic synthesis tools. Some papers propose approaches for verification of model transformations in special domains, such as mechatronic systems [BGL05], or Java code generation [BBG+06]. [ABK07] presents an approach similar to the one presented in this paper: UML metamodels along with embedded well-formedness rules (typically OCL constraints) can be translated to the formalism Alloy. Then, the Alloy Analyzer can conduct fully automated analysis of the transformation. The difference between our approach and the one presented in that paper is that the Alloy Analyzer uses a simulation that generates a random instance model of the input metamodel, then analyzes the behavior of the transformation by transforming this instance model.

4 Business Process Modeling

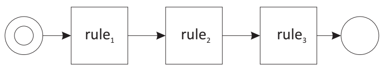
VMTS is our n-level metamodeling and model transformation framework that applies the MPM concept. Business Process Model Notation (BPMN) and Business Process Execution Language (BPEL) are both standards for describing business process models [AML09]. In VMTS, for each business process, a BPMN and a BPEL model is created describing different aspects of the same system, therefore, we apply an MPM concept. Model transformations have been implemented from BPMN to BPEL and in the reverse direction as well. The presentation of the whole concept, and the BPMN, and BPEL metamodels are detailed in [AML09]. A fundamental part of the transformation from BPMN to BPEL is to flatten BPMN models, which is an in-place transformation that works on a single BPMN model. Figure 1 presents the BPMN metamodel, the transformation control flow, and the rules. Each rule is applied exhaustively.

In the following, we outline the main elements of the analysis that is performed by the reasoning system automatically. The properties that should be analyzed are the properties presented above: (i) all nodes of type *BPMNComposite* will be deleted. (ii) All edges of *BPMNContainment* will be deleted. (iii) If a path exists between two nodes of type other than *BPMNComposite* through edges of type *BPMNSequenceFlow*, it will be so after the transformation. (i) and (ii) can be described by assertions presented in Section 2 , but (iii) cannot be expressed by means of assertions. This is an abstract property that is not understandable by the reasoning system, therefore, in this case, we need to manually define what properties should be hold for the transformation that results (iii). If the property (iii) was true before $rule_1$, then it will be so after the execution of this rule, since this rule performs the refactoring operation by its definition. $rule_2$ does not modify the property either, since it only deletes edges of type *BPMNContainment*, which does not take part in the paths. The third rule, $rule_3$, deletes a node of type *BPMNComposite*. We can state that if there is no incoming, or outgoing edge of type *BPMNSequenceFlow*, then this rule does not modify the property either, since, in this case, all paths remain unmodified. But if there are edges of type *BPMNSequenceFlow*, then this rule may modify some paths in the model. Until this point, we have presented the discussion of the property (iii), this must be performed manually by domain experts, since this property cannot be expressed in the formalism used to describe the transformation. The automated analysis should be used to help the proof of the property. In this case, we want to prove with the reasoning system, the there are no incoming or outgoing edges to or from nodes of type *BPMNComposite* before $rule_3$. If this property can be proven, than (iii) can be proven as well.

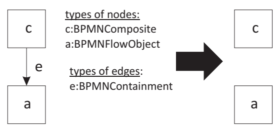
VMTS automatically generates a transformation description from the flattening transforma-



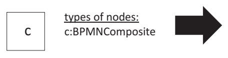
(a) BPMN Metamodel of models under transformation



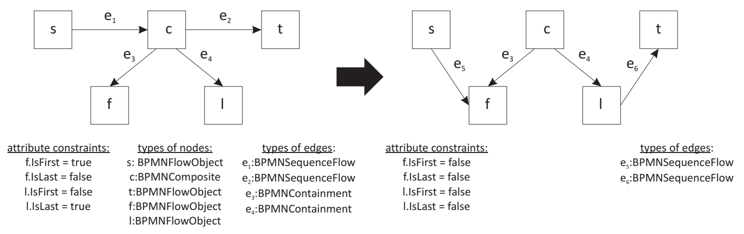
(b) Transformation BPMN Flattening



(c) Rule 2 of Transformation BPMN Flattening

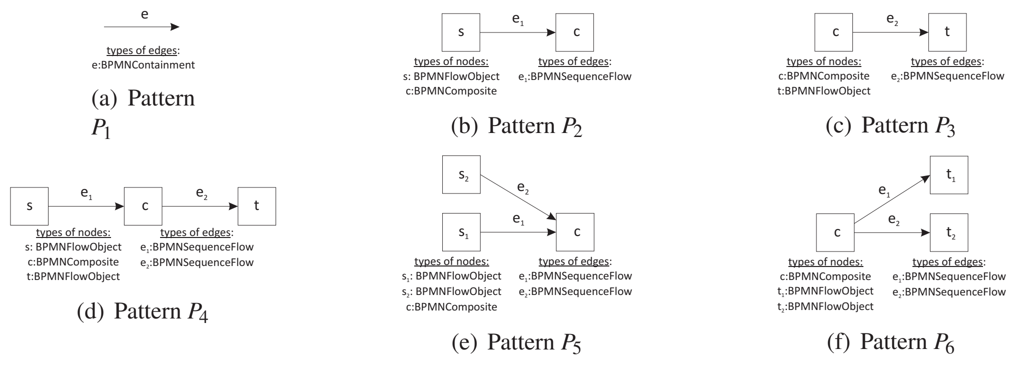


(d) Rule 3 of Transformation BPMN Flattening (RHS is empty)



(e) Rule 1 of Transformation BPMN Flattening

Figure 1: Definition of the transformation BPMN Flattening and the metamodel of the models



(a) Pattern P_1 (b) Pattern P_2 (c) Pattern P_3

(d) Pattern P_4 (e) Pattern P_5 (f) Pattern P_6

Figure 2: Additional patterns used in assertions

tion. This description describes the control flow, which has three steps conforming to the three rules of the transformation. Assertions describing the rules and the preconditions of the successful application of the rules are also generated. Let P_{LHS_i}, and P_{RHS_i} the patterns in the left hand side and right hand side of $rule_i$ $(i = 1,2,3)$. The generated assertions are as follows: $\textbf{pre}(rule_i)$ **Exists** P_{LHS_i} $(i = 1,2,3)$, and $\textbf{at}_{\textbf{succ}}(rule_i)$ **ForEach** $P_{LHS_i} \rightarrow P_{RHS_i}$ $(i = 1,2,3)$.

In the following, we will refer patterns depicted in Figure 2 . We need to add some assertions manually that define some properties that must be true for the input model: (1) Each edge of type *BPMNContainment* must have a left node of type *BPMNComposite* and a right node (since, dangling edges are forbidden): $\textbf{before}(rule_1)$ **Any** $P_1 \rightarrow P_{LHS_2}$ (2) It is forbidden that a node of type *BPMNComposite* has two incoming edges of type *BPMNSequenceFlow*: $\textbf{before}(rule_1)$ **None** P_3. (3) Similarly, it is forbidden that a node of type *BPMNComposite* has two outgoing edges of type *BPMNSequenceFlow*: $\textbf{before}(rule_1)$ **None** P_4 (4) Each node of type *BPMNComposite* must have one incoming edge of type *BPMNSequenceFlow*, one outgoing edge of type *BPMNSequenceFlow*, and a first and last node connected by edges of type *BPMNContainment*: $\textbf{before}(rule_1)$ **Any** $P_{LHS_3} \rightarrow P_{LHS_1}$.

We have provided all information needed for the automated reasoning, except the deduction rules of the reasoning system, but it would exceed the limits of this paper. In the following, we describe the properties to be verified by formal assertions and informally outline the main steps of the reasoning that is performed completely by the reasoning system automatically:

- Property (i): $\textbf{after}(rule_3)$ **None** P_{LHS_3}. This assertion can be derived, since P_{LHS_3} is the left hand side of $rule_3$ that is applied exhaustively, until it cannot be applied anymore. A rule cannot be applied, when no instances of the left hand side can be found in the input model.

- Property (ii): $\textbf{after}(rule_3)$ **None** P_{LHS_2} (By the metamodel). We can prove that the formula **Any** $P_1 \rightarrow P_2$ can be propagated through the whole control flow, therefore, this formula is true at any point of the transformation. $\textbf{after}(rule_2)$ **None** P_{LHS_2} is true because P_{LHS_2}, and $rule_2$ is applied exhaustively. If there was an instance of P_1 in the model under transformation, there should be an instance of P_{LHS_2} as well, which is a contradiction, therefore, there are no instances of P_1 after the application of $rule_2$. Formula **None** P_1 can be easily propagated through $rule_3$, which means that if $\textbf{before}(rule_3)$ **None** P_1 is true, then $\textbf{after}(rule_3)$ **None** P_1 is true as well.

- Property (iii): **before**$(rule_2)$ **None** P_2, and **before**$(rule_2)$ **None** P_3. From the initial assertion set, it can be derived that **before**$(rule_1)$ **Any** $P_2 \rightarrow P_4$, **before**$(rule_1)$ **Any** $P_3 \rightarrow P_4$, and **before**$(rule_1)$ **Any** $P_4 \rightarrow P_{LHS_1}$. The key of the proof is the propagation of the formulas in these assertions through the whole control flow. Since $rule_1$ is applied exhaustively, P_{LHS_1} cannot be present in the model under transformation after $rule_1$, and formula **None** P_{LHS_1} can be propagated through the whole control flow as well. It results the none of the patterns P_2, P_3, P_4 can be present at any point after $rule_1$.

5 Conclusions

In this paper, we have demonstrated the applicability of our previously presented approach of automated verification of model transformations on a case study. We have shown that an initial assertion set of the system is generated automatically from the definition of model transformations. We have demonstrated how complex analysis is performed automatically, largely decreasing the complexity of the verification of model transformations.

Acknowledgements: This paper was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

Bibliography

[ABK07] K. Anastasakis, B. Bordbar, J. M. Kster. Analysis of Model Transformations via Alloy. In *Workshop MoDeVVA'07*. Pp. 47–56. October 2007.

[ALL09] M. Asztalos, L. Lengyel, T. Levendovszky. A Formalism for Describing Modeling Transformations for Verification. In *MoDeVVA'09*. Denver, Colorado, USA, 2009. http://www.aut.bme.hu/Portal/asztalosmark.

[AML09] M. Asztalos, T. Mészáros, L. Lengyel. Generating Exeutable BPEL Code from BPMN Models. In *GraBaTs'09 Tool Contest*. Zurich, Switzerland, 2009.

[BBG+06] B. Becker, D. Beyer, H. Giese, F. Klein, D. Schilling. Symbolic invariant verification for systems with dynamic structural adaptation. In *ICSE '06*. Pp. 72–81. ACM, New York, NY, USA, 2006.

[BGL05] J. O. Blech, S. Glesner, J. Leitner. Formal Verification of Java Code Generation from UML Models. In *Fujaba Days*. september 2005.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series XIV. Springer, 2006.

[KN08] G. Karsai, A. Narayanan. Towards Verification of Model Transformations Via Goal-Directed Certification. *ASWSD 2006, San Diego, CA, USA, March 15-17, 2006, Revised Selected Papers*, pp. 67–83, 2008.

[Var02] D. Varró. Towards Formal Verification of Model Transformations. In *PhD Student Workshop of FMOODS 2002. Enschede, Hollandia*. 2002.