Electronic Communications of the EASST
Volume 32 (2010)

Proceedings of the Fourth International Workshop on Graph-Based Tools (GraBaTs 2010)

Verification of Model Transformations to Refactoring Mobile Social Networks

Márk Asztalos, Péter Ekler, László Lengyel, Tihamér Levendovszky

12 pages

Guest Editors: Juan de Lara, Daniel Varro
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Verification of Model Transformations to Refactoring Mobile Social Networks

Márk Asztalos, Péter Ekler, László Lengyel, Tihamér Levendovszky

Budapest University of Technology and Economics
Department of Automation and Applied Informatics
{asztalos, ekler.peter, lengyel, tihamer}@aut.bme.hu

Abstract: Verification of model processing programs, where only the definitions of the program and the languages of the models to be transformed are analyzed, has become a fundamental issue in model-based software engineering. This analysis may become very complex, but it is performed only once and the results are independent from concrete input models. The formal background of verification methods for graph rewriting-based model transformations has become a subject of research recently. In previous work, we have provided fundamental formal and algorithmic background of a (semi-)automated verification approach for graph transformations. This work concludes these components and put them together to introduce the implementation of a verification system fully integrated into a model transformation framework, VMTS. The strong points of our approach is its usability, its implementation in an existing tool, and its extendibility, which are demonstrated on a case study in the application domain of mobile centric social networks. Our results show that the verification of graph rewriting-based model transformations can be largely automated.

Keywords: model transformation, automated verification

1 Introduction

In model-based software engineering, developers use programs to process models in a repeatable and automated way. With the increasing need of reliable systems, the verification of such model processing programs has become a fundamental issue. Verification means determining the correctness of the program in the sense that the output satisfies certain functional and non-functional conditions.

Graph rewriting-based model transformation is a frequently used model processing technique, which is well suited to describe several model processing scenarios. We analyze graph rewriting-based model transformations that are based on the formal background graph transformation systems as defined in [EEPT06]. A graph transformation system is defined by a set of rewriting rules (productions), the applications of rules are the elementary operations on graphs. In our terminology, a *model transformation* is the definition of a model processing program specified by a a set of rewriting rules (based on the double-pushout approach [EEPT06]) and an additional directed control flow graph that explicitly defines the execution order of the rules.

In this work, we concentrate on the automated, formal verification of *model transformations* where only the definition of the transformation and the specification of the languages that de-

scribe the models to be transformed are used during the analysis process. We use the term 'verification' as the opposite of 'validation', where, the program is analyzed at runtime. Most online techniques extend the definition of the model processors with additional constraint validation code that guarantees that the program cannot finish successfully if a constraint is not satisfied [Len06]. The main advantage of the verification is that the results of analysis are general in the sense that they are independent from the concrete input models. Moreover, the analysis needs to be performed only once. Comparing to the validation, the disadvantage of the verification is the increased complexity of the analysis, since, for example, the termination of a graph transformation is undecidable in general.

Many published examples can be found for the verification of individual model transformations, however, these methods usually lack generalization possibilities, since the analysis is performed manually or the methods can be applied only to a certain transformation class or to the analysis of only a certain type of property. Therefore, there is an increasing need for automated verification methods and tools. There are few results [Pen09, Ore08, Sch09] related to verification, moreover, several tools [LT04, KR06] that provide methods that can be reused in verification methods as well. For a more detailed discussion on related work, see Section 5.

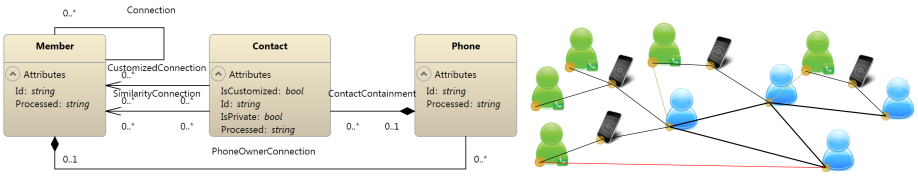
In this paper, we present the concept of an automated verification framework implemented in Visual Modeling and Transformation System (VMTS) [Vis]. We have defined a formal language (Model Condition Description Language [AELL10], detailed in Section 3), which is capable of expressing certain properties of models. Our system can analyze model transformations and prove certain properties for the output models. In previous work [ALL10, AEL$^+$10a, AELL10, AEL$^+$10b] we have described individual components of our verification approach. In this paper, we introduce each of these components shortly (Section 3) to set the grounds for our discussion, then, we present the architecture of our implemented verification system (Section 4). We introduce a case study of refactoring social network models in Section 2 to demonstrate the operation of our system. Section 5 contains discussion on related work and we conclude in Section 6.

2 Case Study

The phone book in our mobile phone is a small part of a social network because every contact has some kind of relationship to us. Given an implementation that allows us to upload as well as download our contacts to and from the social networking application, we can completely keep our contacts synchronized. *Phonebookmark* [EL10] is a phone book-centric social network implementation by Nokia Siemens Networks. We took part in the implementation and before the public introduction it was available for a group of general users from April to December of 2008. It had 420 registered members with more than 72000 private contacts.

2.1 Metamodeling Central Social Network

Visual Modeling and Transformation System (VMTS) [Vis] is a domain-specific modeling, metamodeling, and model transformation system. In the following, we present the model-based representation of *Phonebookmark* networks in VMTS. The metamodel is presented in Figure 1a, its entities are as follows: a *member* is a user of the social network, a *phone* is a mobile device



(a) *Phonebookmark* Metamodel (b) Sample Instance Model

Figure 1: *Phonebookmark* Domain in VMTS

of a member, which can contain phone book entries, a *contact* corresponds to a phone book entry of a phone. Relations between the entities have been defined as follows: each member can own several phones (*PhoneOwnerConnection*), each phone can contain several contacts (*ContactContainment*). A contact can be connected to a member with a *CustomizedConnection* or a *SimilarityConnection* edge. A *CustomizedConnection*, or shortly *customization* edge, means that the current entry corresponds to the member of the social network. Whenever the owner member of the entry connects to the social network, the data can be synchronized. A *SimilarityConnection*, or shortly *similarity* edge, between a member and a contact, denotes that a *similarity detecting algorithm* has found similarities between their data so the user has to decide whether to accept this relation and convert it into a customization edge or reject it. For this purpose, *ApprovalState* attribute has been defined for similarity edges, whose value can be *approved*, *rejected*, or, the default value, *ignored*, which means that the user has not decided yet. In VMTS, a concrete syntax extension has been defined for the instance models. A sample instance model is presented in Figure 1b. The entities can be easily differentiated by their icons. Similarity edges are denoted by red, customization edges by goldenrod colors.

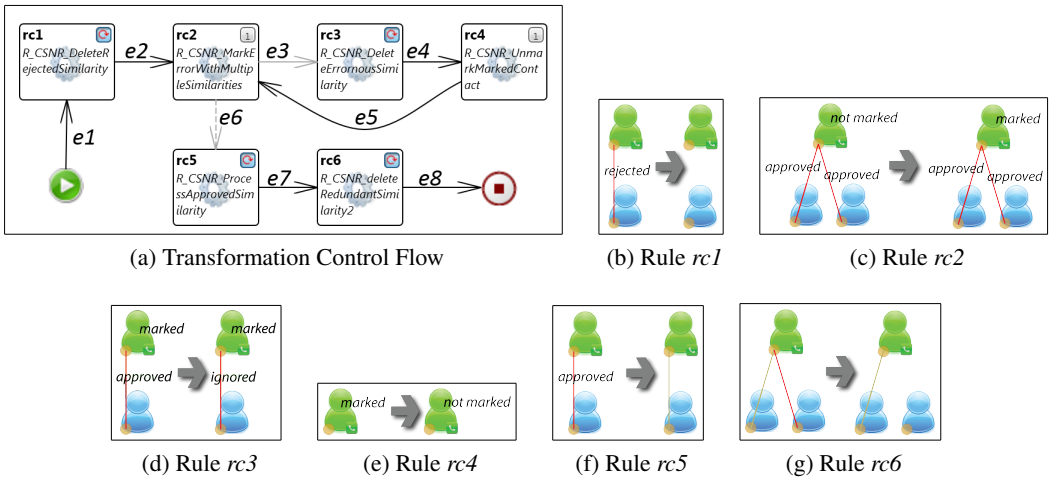
2.2 Similarity Handling Transformation

In VMTS, the graph rewriting-based transformations are defined with the use of two modeling languages: the Visual Control Flow Language (VCFL) and the Visual Transformation Definition Language (VTDL) [AAL+09]. The activity diagram-like VCFL models (control flow models) controls the execution order of the rewriting rules, while the rewriting rules are described with VTDL models. The application of the rewriting rules is based on the double pushout approach [EEPT06].

Phonebookmark provides a semi-automatic similarity detecting and resolving mechanism, which detects similarities between phone book contacts and the members of the network. Similarity means that the algorithm suggest to the user that the contact and the member represent the same person. In this case, a similarity edge is created between the contact and the appropriate member.

In the following, we present a model transformation (*Similarity Handling Transformation*) for the refactoring of *Phonebookmark* models. This model transformation will be used to demonstrate the verification system presented in the following section. The user can start this transformation manually after finishing the classification of the similarity edges, where classification means setting the value of the *ApprovalState* attributes of the edge. The transformation pro-

cesses the classified edges as follows: approved edges are converted into customization edges and rejected edges are removed. The control flow model of the transformation, is presented in Figure 2a. It has a start and an end node, and each other node corresponds to a rewriting rule. The dashed, gray control flow edges are followed, if the application of the source rules was unsuccessful, which happens when no matches of the left hand side can be found. The solid, gray edges are followed if the application of the previous rule was successful, while solid black edges are followed always. Rules with a circle in the top right bottom are executed exhaustively, which means that the rule is applied repeatedly, until it cannot be applied any more. For a more detailed specification of our model transformation language, see [AAL+09]. Figure 2 contains the definitions of the rules. Here, we use a concrete syntax-based formal representation of each rule for the specification of left hand side (LHS) and right hand side (RHS) of each of them. Informally, the transformation works as follows: (i) *rc*1 removes all rejected similarity edges. (ii) If there is a contact that has two approved similarity edges, *rc*2 marks the contact. Marking means setting an attribute of the entity. (iii) *rc*3 changes the approval state of an approved similarity edge of a marked node to ignored. This rule is reached only when *rc*2 has been applied successfully. (iv) *rc*4 removes the mark from a marked node. (v) *rc*5 replaces all approved similarity edges with customization edges. This rule is reached only when *rc*2 cannot be applied. (vi) *rc*6 removes all similarity edges which comes from a member that already has a customization edge.



(a) Transformation Control Flow (b) Rule *rc1* (c) Rule *rc2*

(d) Rule *rc3* (e) Rule *rc4* (f) Rule *rc5* (g) Rule *rc6*

Figure 2: Similarity Refactoring Transformation

3 Components of a verification framework

In this section, we summarize the fundamental components of our verification approach that have been presented in [ALL10, AELL10, AEL+10a, AEL+10b].

Model Condition Description Language (MCDL). MCDL is a language for writing expressions (formulae) to specify properties of models. These properties can state the existence or non-existence of *patterns* of elements extended by additional constraints.

Informally, patterns consist of model elements along with a set of additional attribute constraints. A pattern exists in a model, if the model elements can be matched such that the match satisfies the constraints as well. MCDL handles attribute constraints in an abstract level, therefore, the constraint language may vary through different implementations, e.g. in VMTS, constraint code is written in C# or OCL. LHS and RHS of rules are also defined as patterns. For example, in LHS of rule $rc1$, we specify that the similarity edge must be rejected, i.e. its *ApprovalState* attribute must have the value *rejected*, which is an example for attribute constraints in VMTS.

The simplest expression of MCDL can define that a match of a pattern P must exist in the model, moreover, MCDL expressions can be composed to more complex MCDL formulae by standard logical operators $(\neg, \wedge, \vee)$. For example, a model satisfies the MCDL condition $c = \exists \wedge \nexists$ if there exists a member in the model, but there are no contacts ($\nexists$ is the abbreviated form of $\neg\exists$). The presentation of more complex MCDL expression is beyond the scope of this paper (see [AELL10] for more details), however, we will show more examples for MCDL in Section 4.1.

Model Condition Inference Logic. Given two MCDL expressions ϕ_1, ϕ_2, we may want to prove or refute the logical implication $\phi_1 \Rightarrow \phi_2$. For example: $\exists \Rightarrow \exists$. Model Condition Inference Logic (MCIL) is an inference logic defined over expressions of MCDL. Deduction rules for the calculus have been proposed in [ALL10], and an automated reasoning system is implemented in VMTS. To illustrate the role deduction rules, we present two simple and intuitive examples for them: (i) if $P_1 \subseteq P_2$ then $\exists P_2 \Rightarrow \exists P_1$, (ii) if $P_1 \subseteq P_2$ then $\nexists P_1 \Rightarrow \exists P_2$. $P_1 \subseteq P_2$ denotes that the P_1 is a part of P_2, i.e. if a match of P_2 can be found in a model, then, a match of P_1 can also be found. $\exists P_1$ and $\nexists P_2$ are examples for MCDL expressions. The proof of the previous deduction rules are trivial and, therefore, not detailed in this paper.

MCIL is used during the verification of model transformations. For example, assume that we can prove that ϕ_{final} is true for each output model of the transformation, and let ϕ_v be a verifiable property. (MCDL formulae are usually denoted by Greek letters.) In this case, if the implication $\phi_{final} \Rightarrow \phi_v$ can be proved, then the property is verified.

Propagation. Propagation is a method to analyze a single rule and having some properties of the input model, it should derive some properties of the output models. In other words, assume that we know some properties of all possible models that can be verified before the application of the rule. These properties are described by MCDL and are called the *incoming formula*. We have the *incoming formula*, and by the very definition of the rule, we may derive certain properties that will be true after the application of the rule. These properties described by MCDL are called the *outgoing formula*. This method is called the *propagation of formulae through a rule*, which is a very complex task itself. The derivable information depends on the incoming formula and the definition of the rule. There are several propagation rules that resembles to deduction rules. Informally, they are defined in the following form: if a property is derivable from the incoming formula and another property is true for the rule, then a third property will always be true for the output model. The more propagation rules we have the more information can be derived and, therefore, the more properties of the output model can be proved. In the following, propagation

will be referred as a function $\texttt{Propagate}(r, \varphi_{in})$ that computes an outgoing formula given a rule r and an incoming formula φ_{in}. In the following, we show an example for the propagation method:

$\varphi_1 =$  and $\varphi_2 =$ 

By the definition of *rc*5 (Figure 2f), it can be proved that, if given a model M that satisfies φ_1 and M is modified by executing rule *rc*5, the resulting model will also satisfy φ_1 (since *rc*5 does not create rejected similarities). Moreover, we can infer that the modified model must satisfy φ_2 independently of the fact that M satisfied φ_2 before the application of the rule (because *rc5* is applied exhaustively).

Discovery Algorithm. In the following, we detail the main concept of our verification approach that employs the components MCDL, MCIL and propagation. We assume that the control flow of a model transformation is a directed graph.

Given a model transformation, assume that we are able to assign MCDL formulae to each control flow edge of the transformation. This formula describes the property that can be proved at the current location of the transformation, i.e.: assuming that if the transformation stopped at the current location, and the model under transformation were considered the output model, the static conditions of the assigned formula would be satisfied by all possible output models. Similarly, the dynamic conditions of the assigned formula would be satisfied by all pairs of possible input and output models. Again, these conditions are independent from the concrete input models. During the analysis of a model transformation, our goal is to produce these *assignments* or, in other words, to *discover* the formulae on all flow edges. Assume that we have only one end node in the control flow, and it has only one incoming edge, and the formula φ_{final} is assigned to this edge. Therefore, φ_{final} is the formula that is satisfied by the transformation. Given a property that should be verified and is described as an MCDL expression φ_{ver}, if we can prove that $\varphi_{final} \Rightarrow \varphi_{ver}$, the property is verified. Moreover, we assign formulae not only to the edges before the end nodes, but to all flow edges, which helps locating the points where a significant property does not hold any more. The goal of our method is to collect the most information in the formulae that are assigned to the edges. However, if nothing can be propagated, it will not imply the failure of our algorithm, only that the assigned formulae will not contain relevant information, therefore, the verifiable properties cannot be proved. The pseudocode of the discovery algorithm is presented in Algorithm 1.

What are the benefits of the discovery algorithm outlined above? (i) Given a verifiable condition ϕ^{ver} specified as an MCDL formula, the verification is the proof of the expression $\phi^{final} \Rightarrow \phi^{ver}$. (ii) Formulae are discovered on all edges of the control flow graph, this can help to localize problematic points of the transformations during the testing phase. We will show examples for the propagation in Section 4.1, where the transformation handling the similarity is verified.

Algorithm 1 `Discovery`(**transformation** T, **initial MCDL formula** φ_{init})

1: initally there are no *processed* rules
2: let e_f be the unique flow edge of the start node: assign φ_{init} to e_f
3: **while** there are not *processed* rules **do**
4: $r \leftarrow$ choose a not yet *processed* rule randomly such that the formulae on all of its incoming edges have already been discovered. If no such rule exists choose a not yet *processed* rule randomly.
5: **if** all incoming edges of r are discovered **then**
6: $\varphi_{in} \leftarrow$ logical *or* of the formulae on the incoming edges of r
7: **else**
8: $\varphi_{in} \leftarrow \emptyset$
9: for all not yet discovered incoming edge e_{in} assign an empty formula $\emptyset$
10: $\varphi_{e_{out}} \leftarrow$ `Propagate`(r, φ_{in})
11: **for all** outoing edge e_{out} of r that has not yet been discovered **do**
12: assign $\varphi_{e_{out}}$ to e_{out}

4 Automated Verification Framework

The verification framework for model transformations has been implemented in VMTS. The main elements of the verification process and the components of the framework are presented in Figure 3. Rounded rectangles are artifacts that are created by the developers or by the verification framework automatically, while not rounded rectangles are the components of the verification system implemented in VMTS.

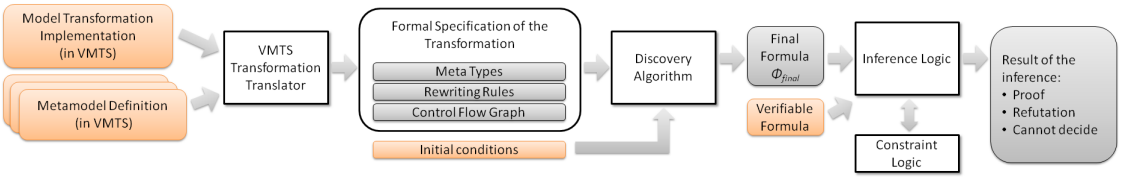


Figure 3: Components of the Verification Framework

The phases of a model-based development (i.e. the implementation of transformations and the verification process), and the roles of the components are as follows: 1) Domain experts define the metamodel of the modeled application domains. 2) Model transformation developers implement a model transformation. 3) From these artifacts, *VMTS Transformation Translator* automatically generates the formal specification of the model transformation. This specification is a formal, declarative description, which makes the further automated analysis of the control flow and rules possible. 4) The *Discovery Algorithm* traverses the specification of the transformation, propagates the initial conditions, discovers the formulae on all edges, and generates the final formula (ϕ_{final}). 5) MCIL is used to prove or refute the implication $\phi_{final} \Rightarrow \phi_{ver}$, where ϕ_{ver} is the verifiable formula provided by the developer or the tester of the transformation. The *Inference Logic* component of VMTS is the implementation of the MCIL. The result of the reasoning

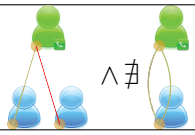
can be the proof, or the refutation of the implication, or that the algorithm cannot decide.

4.1 Verification of Model Transformation Handling Similarity

The complete, formal presentation of the verification of the similarity handling transformation would exceed the limits of this paper, and would need the more detailed introduction of MCDL and the other components, however, in this section, we present its main steps.

Primarily, we provide initial conditions. Informally, we assume that each input model satisfies the following conditions: (i1) there cannot exist parallel customization edges, (i2) parallel similarity edges cannot have the same target, and (i3) initially all contacts is *not marked* (a contact can be *marked* or *not marked*, which is expressed by the attribute *Processed*). The They are specified by MCDL in Table 1, let $\phi^{init} = \phi_1^{init} \wedge \phi_2^{init} \wedge \phi_3^{init}$.

Table 1: Initial Conditions

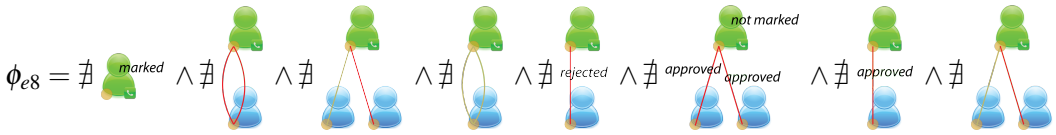
(i1) $\phi_1^{init} = \nexists$  $\wedge \nexists$	(i2) $\phi_2^{init} = \nexists$ 	(i3) $\phi_3^{init} = \nexists$ 

For the verification, we need to define the verifiable properties, which are as follows (the properties are formalized in Table 2): (v1) *After the application of the transformation, no approved similarity edge should be present in the model.* Each approved edge should be transformed to a customization edge, or should be deleted if there are more than one approved similarity edge from the same contact. (v2) *After the application of the transformation, no rejected similarity edge should be present in the model.* All rejected similarity edges should be deleted. (v3) *After the application of the transformation, it is forbidden that a contact has a similarity and a customization edge at a time.* In this case, the similarity edge should have been deleted. (v4) *After the application of the transformation, it is forbidden that a contact has two customization edge at a time, provided that before that transformation started this pattern was also forbidden.* This would result an inconsistent state.

Table 2: Verifiable Properties

(v1) $\nexists$ 	(v2) $\nexists$ 	(v3) $\nexists$ 	(v4) $\nexists$ 

After the initial conditions are provided and the discovery algorithm is executed, which means that for each flow edge e a formula ϕ_e is assigned. Hence, the final formula can be derived, which is $\phi^{final} = \phi_{e8}$ in our case. MCIL can derive all four verifiable properties from the final formula. The presentation of the assignments would exceed the limits of this paper, but we provide ϕ_{e8}.



4.2 Evaluation

The satisfaction of a property of a model transformation is undecidable in general, since, for example, the termination itself is undecidable in general [Plu98]. Our methods provide a way for the analysis of certain properties, but we cannot assure that the analysis will always lead to a successful verification. MCDL is used to define the properties of the model transformations that need to be analyzed. The main idea behind the analysis of a model transformation is that we follow the execution order of the rules and try to discover what properties can be proved at different locations of the control flow graph. We process each rule one-by-one and derive a set of properties that are true after the application of the rule exploiting that we know the properties that have already been proved to be true before the rule. If our algorithm fails, i.e. we cannot prove a certain property, it can have three reasons: (i) MCDL is not able to express the property to be analyzed, or (ii) it may happen that even a rule guarantees a certain property, we may not be able to automatically derive this property, because of the lack of appropriate propagation rules, or (iii) the control flow graph may contain too complex structures and the pieces of information should be collected from different points of the control flow. To summarize, the successfulness of the verification cannot be assured.

The main advantages of our verification methods are as follows: (i) MCIL terminates, i.e. if a property can be derived by the deduction rules, then, it can be proved by MCIL. (ii) The analysis provides information about each location of the transformation. This can help locate the problematic points and debug the transformation. (iii) The formal background of the verification methods is platform- and tool-independent.

The main disadvantage of the verification method at its current state is the lack of efficient control graph processing algorithms and the low number of propagation rules. We also need to take efficiency issues into account. Largely increasing the set of propagation and inference rules, the time cost of the algorithms may increase as well, even if the algorithm terminates. We believe that this issue is not a primary problem to solve, since the analysis needs to be performed only once.

5 Related Work

In this paper, we have outlined the formal background for the platform-independent verification of graph rewriting-based model transformations and presented a framework that can automatically analyze model transformations. In the following, work related to both components are evaluated. Note that the research of this field is still in initial phase, therefore, no fully functional tools are available.

Analysis of concrete transformations have been presented in several publications [Var02, BH07, BBG+06], but these approaches can usually be applied to only certain (class of) trans-

formations, or only for certain (type of) properties, and cannot be generalized. [Pen09] presents a formalism that is similar to our concept of conditions on models. Nested conditions that are based on traditional application conditions of graph rewriting systems for high-level structures are formalized. Additionally, a sound and complete satisfiability algorithm for graph conditions is investigated and a fragment of conditions is identified, for which the algorithm decides. One main difference between this work and ours is the handling of attribute constraints, our framework makes it easily integrate an external constraint logic. [Ore08] also introduces formalization for attributed graph constraints. The new notion of attributed constraint combines a (standard) graph constraint with a formula describing a condition on the attributes of the graphs involved in the constraint. Moreover, [Ore08] also presents inference rules for the classes of constraints considered, showing their soundness and completeness. In [Sch09], the authors introduce a formalism to describe a model transformation in a declarative way, hereby, verification of soundness conditions can be performed using an interactive theorem prover.

AGG [LT04] supports a consistency control mechanism which is able to check if a given graph satisfies certain consistency conditions specified for a graph grammar. Our approach does not rely on graph grammars with a specific start graph, but on a control flow-based ordering of rules. Consistency conditions describe basic properties of graphs as e.g. the existence of certain elements, independent of a particular rule. A graph grammar is consistent if the start graph satisfies the consistency conditions and the rules preserve this property. GROOVE [KR06] is also a tool to analyze consistency constraints on models and graph transformation systems. The language to specify these constraints is similar to MCDL, however, in this tool, the consistency checking rather resembles to validation methods, i.e. the output of specific input models are analyzed. [ABK07] presents an approach similar to ours: 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 the 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. [LBA10] presents a mathematical background for the analysis of model transformations. Similarly to our approach, [LBA10] also formalizes metamodels, models, and structural properties of the transformed models. Their implemented model transformation framework can validate certain type of properties by construction, because of several constraints of the control structure that is used to determine the execution order of the rules.

6 Conclusions

In this paper, we have outlined a formal, automated framework for the verification of graph rewriting-based model transformations. We have presented how the components of the framework work together in an implementation of our verification methods in a modeling tool, VMTS. We have demonstrated the usability of our methods on a case study of the verification of refactoring mobile-centric social network models. We provided a summary of the main advantages and disadvantages of our approach. In future work, we would like to complete the formalism behind each of presented components of our solution, and present more complex case studies. We

believe that our approach can provide the basis for industrial model transformation verification methods.

Acknowledgements: This paper was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. This work is connected to the scientific program of the "Development of quality-oriented and harmonized R+D+I strategy and functional model at BUTE" project. This project is supported by the New Hungary Development Plan (Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002).

Bibliography

[AAL+09] L. Angyal, M. Asztalos, L. Lengyel, T. Levendovszky, I. Madari, G. Mezei, T. Mészáros, L. Siroki, T. Vajk. Towards a Fast, Efficient and Customizable Domain-Specific Modeling Framework. In *Software Engineering*. 2009. Innsbruck, Austria.

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

[AEL+10a] M. Asztalos, P. Ekler, L. Lengyel, T. Levendovszky, T. Mészáros. Automated Verification by Declarative Description of Graph Rewriting-Based Model Transformations. In *MPM'10*. Oslo, Norway, October 2010.

[AEL+10b] M. Asztalos, P. Ekler, L. Lengyel, T. Levendovszky, T. Mészáros. Formalizing Models with Abstract Attribute Constraints. In *GCM'10*. Enschede, The Netherlands, September 2010.

[AELL10] M. Asztalos, P. Ekler, L. Lengyel, T. Levendovszky. MCDL: A Language for Specifying Graph Conditions with Attribute Constraints. In *MODEVVA'10*. Oslo, Norway, October 2010.

[ALL10] M. Asztalos, L. Lengyel, T. Levendovszky. Towards Automated, Formal Verification of Model Transformations. In *ICST*. Paris, France, April 2010.

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

[BH07] D. Bisztray, R. Heckel. Rule-Level Verification of Business Process Transformations using CSP. *ECEASST* 6, 2007.

[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.

[EL10] P. Ekler, T. Lukovszki. Experiences with phonebook-centric social networks. In *CCNC*. Las Vegas, USA, 2010.

[KR06] H. Kastenberg, A. Rensink. Model Checking Dynamic States in GROOVE. In *In Model Checking Software (SPIN*. Pp. 299–305. Springer, 2006.

[LBA10] L. Lucio, B. Barroca, V. Amaral. A Technique for Automatic Validation of Model Transformations. In *MoDELS (1)*. Pp. 136–150. 2010.

[Len06] L. Lengyel. *Online Validation of Visual Model Transformations*. PhD thesis, Budapest University of Technology and Economics, 2006.

[LT04] J. de Lara, G. Taentzer. Automated Model Transformation and Its Validation Using AToM3 and AGG. In *Diagrammatic Representation and Inference*. Pp. 182–198. 2004.

[Ore08] F. Orejas. Attributed Graph Constraints. In *ICGT*. Pp. 274–288. Springer-Verlag, Berlin, Heidelberg, 2008.

[Pen09] K.-H. Pennemann. *Development of Correct Graph Transformation Systems*. PhD thesis, Department of Computing Science, University of Oldenburg, Oldenburg, Germany, 2009.

[Plu98] D. Plump. Termination of graph rewriting is undecidable. *Fundam. Inf.* 33(2):201–209, 1998.

[Sch09] B. Schätz. Formalization and Rule-Based Transformation of EMF Ecore-Based Models. *SLE, Toulouse, France, September 29-30, 2008. Revised Selected Papers*, pp. 227–244, 2009.

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

[Vis] Visual Modeling and Transformation System (VMTS) website. http://vmts.aut.bme.hu/.