VOLT 2014 Workshop Report

Moussa Amrani[1], Eugene Syriani[2], Manuel Wimmer[3], Robert Bill[3], Martin Gogolla[4], Frank Hermann[5], and Kevin Lano[6]

[1] University of Namur, Belgium
Moussa.Amrani@unamur.be
[2] University of Montreal, Canada
syriani@iro.umontreal.ca
[3] Vienna University of Technology, Austria
{wimmer,bill}@big.tuwien.ac.at
[4] University of Bremen, Germany
gogolla@informatik.uni-bremen.de
[5] Université du Luxembourg, Luxembourg
frank.hermann@uni.lu
[6] King's College London, UK
kevin.lano@kcl.ac.uk

Abstract. This report is a summary of the Third International Workshop on the Verification Of modeL Transformation (VOLT 2014) held at the STAF 2014 conference. The workshop brought together researchers from model-driven engineering, in particular from model transformation language engineering and model-based verification. The major aims of VOLT 2014 were to identify motivations, problems, and requirements for model transformation verification as well as to present different proposals supporting different kinds of model transformations and verification techniques.

1 Introduction

Model transformations are everywhere in software development, implicitly or explicitly. They became first-class citizens with the advent of Model-Driven Development (MDD). Despite some recent activity in the field, the work on the verification of model transformations remains scattered and a clear perspective on the subject is still not in sight. Furthermore, current model transformation tools mostly lack verification techniques to support such activities.

The Third International Workshop on the Verification Of modeL Transformation (VOLT 2014) is one of the most accurate venues to offer researchers a dedicated forum to classify, discuss, propose, and advance verification techniques dedicated to model transformations.

VOLT 2014 promoted discussions between theoreticians and practitioners from academy and industry, given its ideal co-location with STAF. A significant part of the workshop included a forum for discussing practical applications of model transformations as well as their verification. In particular, the following topics have been considered in the scope of VOLT 2014:

- Application of formal verification, theorem proving, model checking or testing to model transformation;
- Verification techniques dedicated to model transformation;
- Taxonomies of techniques for model transformation verification;
- Properties relevant to specific model transformations;
- Verification of model transformations expressed in languages such as: ATL, QVT, TGG, VIATRA, Kermeta, Epsilon, etc.;
- Verification of domain-specific model transformations, in contrast to general-purpose transformations;
- Case studies and experience reports;
- Tools and automation.

The remainder of this report is based on the outcome of the two moderated presentation sessions and the discussion session at the end of the workshop. In Section 2, we summarize the presentations and individual discussions of the presentations, while in Section 3 we summarize the overall discussions regarding model transformation verification in general and about property languages for model transformations in particular. Finally conclusions are presented in Section 4 with an outlook on future efforts to further establish and advance the model transformation verification community.

2 Papers and Presentations

The papers and the associated presentations are available on the workshop web pages (http://volt2014.big.tuwien.ac.at).

Checking Transformation Model Properties with a UML and OCL Model Validator by Martin Gogolla, Lars Hamann, and Frank Hilken. This contribution discussed model transformations in the form of transformation models which connect source and target metamodels. A transformation model is a direction-neutral transformation characterization that specifies in a descriptive way by means of OCL [12] invariants the [source,target] pairs constituting the transformation. Transformation models are analyzed with (what we call) a UML and OCL model validator on the basis of an implementation of relational logic on top of Kodkod. Within this approach it is feasible to prove transformation model consistency, i.e., to automatically construct a valid metamodel instance. Transformation model consistency is discussed in various flavours, i.e., (a) weak consistency, (b) class instantiability, and (c) class and association instantiability. Certain properties implied by the transformation model, e.g., whether a particular property is preserved by the transformation, can be inspected as well.

The discussion at the workshop brought up (among other interesting questions) the following topics: (a) Transformation computation: The model validator may be used to effectively compute the transformation that is determined by the transformation model; it is possible to specify a partial object diagram that represents, for example, the source part of a [source,target] pair; the model validator can then complete the partial object diagram, if possible, and present the target part of the model transformation again in form of an object diagram; this also works in the opposite direction from the target to the

source; (b) Used OCL features: Transformation models and their properties typically employ only a subset of full OCL; in particular patterns involving nested occurrences of forAll and exists are frequently used; depending on the nature of the underlying source and target metamodel, the closure operation may be needed in connection with reflexive associations; (c) Nature of analyzable transformation properties: Any property that can be formulated as an OCL invariant and that can be added to the current transformation model can be checked; the added invariant is then negated, added to the present invariants and an adequate configuration must be prepared by the developer for the model validator; if the model validator does not find a valid object diagram under the given configuration for the given invariants and the added negated invariant, it is assumed that the newly introduced invariant is a consequence from the stated invariants.

Language-independent model transformation verification by Kevin Lano, Shekoufeh Kolahdouz Rahimi, and Tony Clark. We present work on establishing a general verification framework for model transformations which is able to represent and analyse transformations in a range of model transformation languages. Specifications in different transformation languages are represented in a single transformation metamodel formalism. From this representation mappings to semantic models in specific verification formalisms, such as theorem provers and satisfaction checkers, can be defined.

This approach means that only one semantic mapping needs to be defined and verified for each target formalism, rather than semantic maps for each different transformation language and target formalism. The approach is illustrated by applying it to ATL.

Null considered harmful (for transformation verification) by Kevin Lano. OCL [12] is the official textual specification language used with the UML, and it is also widely used as a constraint language within model transformation languages, such as ATL, QVT, ETL, Kermeta and others to define transformation rules. The OCL standard defines two special values which may be used in specifications, $null$ and $invalid$: $null$ represents the absence of a valid value, in contrast to $invalid$, which represents an invalid evaluation.

This paper identifies problems with use of explicit null and invalid values in OCL, when OCL is used as part of a transformation specification language. The paper proposes an alternative restricted use of OCL which avoids these problems and facilitates transformation verification. Verification techniques are also described for a transformation language, UML-RSDS, based on this approach.

MocOCL: A Model Checker for CTL-Extended OCL Specifications by Sebastian Gabmeyer, Robert Bill, Petra Kaufmann, and Martina Seidl. This contribution discussed MoCOCL[7], a framework for the verification of CTL extended Essential OCL constraints on behavioral models defined using graph transformations specified by the tool HENSHIN and an initial Ecore state. Valid parts of CTL formulae like **Always Globally** are syntactically treated like regular boolean expressions requiring boolean parameters.

[7] Available at http://modelevolution.org/prototypes/mococl with a web demo for a Pacman example available on http://modelevolution.org/mococl

MOCOCL consists of a web interface for putting in metamodel, behavioral specification, initial model and the expression to be verified and an interspecting the result and an iterative, explicit state model checker integrated into the Xtext OCL Engine calculating the expression result and a *cause*, a generalized form of a counterexample providing all information sufficient to explain the result. The cause is visualized by an expression tree displaying subexpressions and informations about states and transitions as subtrees, the leaf nodes being single values or objects. The (sub)statespace corresponding to the cause of a subexpression of a CTL operation is vizualized as well. A click on a state gives the model for the state, a click on a transition gives displayed the two states in storyboard notation. Objects and values occuring in the cause are highlighted. The paper mainly discusses the visual interface and general use of the tool.

Since the tool could not yet be used for industrial-sized examples, a main topic of discussion was performance, especially (a) the state explosion problem typically occurring in model checkers which could be reduced by the use of symbolic model checking and/or differential states like in GROOVE and (b) that parallelization efforts could provide some speedup, both by parallelizing graph transformation applications and the model checking process in general.

Towards Domain Completeness for Model Transformations Based on Triple Graph Grammars by Nico Nachtigall, Frank Hermann, Benjamin Braatz, and Thomas Engel.

This presentation discussed the property of domain completeness for model transformations, which states that the model transformation can be executed for each valid input model. The main challenge here for analysing and ensuring this property is to bridge the gap between the specification formalism used for defining the source domain language and the specification formalism or technique used to define the model transformation.

The presentation focussed on model transformations that are based on triple graph grammars (TGGs) [1, 8, 13], which are a well-established concept for the specification and execution of bidirectional model transformations within model driven software engineering. Their main advantage is an automatic generation of operational rules for forward and backward model transformations, which simplifies specification and enhances usability as well as consistency. Several formal results for ensuring correctness and completeness have been published [4, 5, 7]. However, the result for ensuring completeness requires that the source domain language is a subset of the source component of the language generated by the TGG. Up to now, checking this condition was left to the domain expert.

In practical scenarios, the source and target languages are given independently from the TGG. In particular, this is the case for an industrial application for satellite systems using the tool HenshinTGG[8] [6]. As main result, we provided a general method for analysing and showing that the source domain language $\mathcal{L}_S$ is included in the language $\mathcal{L}(TGG^{\mathrm{S}})$ that is generated by the source rules of the TGG. This provides the first of two components for verifying domain completeness.

Since the presented method does not yet fully bridge the described gap, the workshop discussion addressed questions on how effective and usable the approach already is

[8] Available at http://de-tu-berlin-tfs.github.io/Henshin-Editor/

and how it can be adapted to show full domain completeness. One solution for showing the remaining step for analysing full domain completeness is to show that $\mathcal{L}(TGG^{\mathrm{S}})$ is contained in $\mathcal{L}(TGG)^{\mathrm{S}}$, i.e., that the triple rules imply the same restriction on the source domain as it is the case for the derived source rules of the TGG. This would allow us to apply the general result of TGGs that ensure completeness for $\mathcal{L}(TGG)^{\mathrm{S}}$.

3 Discussions

The discussion session was structured as follows: first, we aimed at determining the current tool support of the formalisms used in the talks given during the workshop; second, we outlined a potential wish list for property languages for model transformations based on the results of the first discussion. This section summarizes each discussion.

3.1 Formalisms for Model Transformation Property Languages

We concluded that currently, two formalisms are mainly used for defining structural as well as temporal properties of model transformations, a fact that was also reflected by the presentations of the workshop:

Structural Properties are commonly expressed with either graph patterns or using the Object Constraint Language (OCL). It seems that each formalism corresponds to the underlying constructs for designing models (either graph-based or MOF-like models, respectively), and reflects the practice and the familiarity of modelers.

Temporal Properties exist in both styles as extensions in the literature to go beyond structural properties [11, 16].

Kevin Lano's paper [9] raised an interesting discussion about which intermediate language could act as a pivot model to bridge the current formalisms to formal methods and accompanying tool support.

Both property languages could be used to catch properties of interest for in-place and out-place transformations [10]. However, as noted by the audience, upcoming transformation paradigms such as streaming transformations [3] or approximate transformations [15] may challenge the current state-of-the-art, thus requiring to build new property languages more suited to these kinds of transformations. As a result, a general theory may be required to derive a mapping between property kinds and transformation kinds, e.g., it is not clear if temporal properties are of particular purpose for out-place transformations, while they seem perfectly applicable for in-place transformations.

Based on the discussions, the participants concluded that we are still in the exploration phase concerning property languages for model transformations. Therefore, more empirical studies are needed to compare, classify and potentially (partially) unify the different approaches currently available.

3.2 Wish List for Property Languages

We dedicated the second part of the discussion at exploring two research questions: (i) what kind of properties transformation engineers may like to specify? (ii) Which future can we foresee for property languages?

One major distinction made by the audience is the difference between *white-box* and *black-box* properties: the *black-box* approach only reasons about the input/output models pairs, while the *white-box* approach allows to reason about the state/transition systems induced by the application of the transformation rules.

Another important challenge for property languages is to find a acceptable balance between reusing well-known languages such as modelling standards, while reaching the performances of current mature mainstream model-checking tools available from the Computer-Aided Verification community. This challenge seems impossible to reach because it seems contradictory, even if dedicated transformations between transformation languages as well as the associated property languages towards the input languages of such tools may seems largely feasible. Furthermore, what has to be explored in this respect are the current boundaries of model transformation verification approaches concerning the impact of property specifications and transformation implementations on the memory consumption and execution time of verification approaches.

Another topic of discussion was the different ways of addressing non-functional properties of models and/or transformations. Currently, there is no standard way to represent model or transformation traces, because the abstraction level into which traces are expressed may highly depend on the verification purpose. One interesting approach allowing to define only the necessary traces is presented in [14]: it builds on the idea of observers that record certain information during transformations. Having more generalized observers which may be reusable in different transformations but which are specific for certain properties is considered as a promising research line to enable specific verification properties with keeping the trace models minimal. To summarise, the question about the relationship between property languages and trace languages has to be explored further in the context of model transformation verification.

Finally, we discussed about transformations that may be used for Models@Runtime which may have quite strict requirements on performance and timing. In this context, the question came up if probabilistic properties may be of interest for model transformation verification as well compared to the state-of-the-art of having flat properties.

4 Conclusion

At the end of the workshop many participants agreed that there is a need for further working on the foundations as well as application of model transformation verification. A future research line for an upcoming VOLT workshop may include to propose a common model verification example to compare different model transformation verification approaches; similar as it has been done in 2005 to compare different approaches for implementing model transformations [2].

Acknowledgement

The authors would like to thank all the authors and participants of VOLT 2014 for their contributions.

References

1. Andy Schürr and Felix Klar. 15 Years of Triple Graph Grammars. In *Proc. ICGT'08*, volume 5214 of *LNCS*, pages 411–425, 2008.
2. J. Bézivin, B. Rumpe, A. Schürr, and L. Tratt. Model transformations in practice workshop. In *Satellite Events at the MoDELS 2005 Conference*, pages 120–127, 2005.
3. J. S. Cuadrado and J. de Lara. Streaming model transformations: Scenarios, challenges and initial solutions. In *6th International Conference on the Theory and Practice of Model Transformations (ICMT)*, pages 1–16, 2013.
4. Fernando Orejas, Esther Guerra, Juan de Lara, and Hartmut Ehrig. Correctness, Completeness and Termination of Pattern-Based Model-to-Model Transformation. In *Int. Conf. on Algebra and Coalgebra in Computer Science (CALCO'09)*, volume 5728 of *LNCS*, pages 383–397. Springer, 2009.
5. Frank Hermann, Hartmut Ehrig, Ulrike Golas, and Fernando Orejas. Formal analysis of model transformations based on triple graph grammars. *Mathematical Structures in Computer Science*, 24(4):1–57, 2014.
6. Frank Hermann, Susann Gottmann, Nico Nachtigall, Hartmut Ehrig, Benjamin Braatz, Gianluigi Morelli, Alain Pierre, Thomas Engel, and Claudia Ermel. Triple Graph Grammars in the Large for Translating Satellite Procedures. In *Proc. Int. Conf. on Model Transformations (ICMT 2014)*, number 8568 in LNCS, pages 122–137. Springer, 2014.
7. H. Giese, S. Hildebrandt, and L. Lambers. Bridging the gap between formal semantics and implementation of triple graph grammars. *Software & Systems Modeling*, 13(1):273–299, 2014.
8. Hartmut Ehrig, Karsten Ehrig, Claudia Ermel, Frank Hermann, and Gabriele Taentzer. Information Preserving Bidirectional Model Transformations. In *Fundamental Approaches to Software Engineering*, volume 4422 of *LNCS*, pages 72–86. Springer, 2007.
9. K. Lano. Null Considered Harmful (for Transformation Verification). In *Third International Workshop on Verification of Model Transformations (*VOLT*)*, 2014.
10. T. Mens and P. Van Gorp. A Taxonomy Of Model Transformation. *Electronic Notes in Theoretical Computer Science (*ENTCS*)*, 152:125–142, 2006.
11. B. Meyers, R. Deshayes, L. Lucio, E. Syriani, H. Vangheluwe, and M. Wimmer. Promobox: A framework for generating domain-specific property languages. In *7th International Conference on Software Language Engineering (SLE)*, pages 1–20, 2014.
12. Object Management Group. Object Constraint Language (OCL) Specification (Version 2.2, formal/2010-02-01). Technical report, Object Management Group, 2010.
13. A. Schürr. Specification of graph translators with triple graph grammars. In *Graph-Theoretic Concepts in Computer Science*, volume 903 of *LNCS*, pages 151–163. Springer, 1994.
14. J. Troya, J. E. Rivera, and A. Vallecillo. Simulating domain specific visual models by observation. In *Spring Simulation Multiconference (SpringSim)*, page 128, 2010.
15. J. Troya, M. Wimmer, L. Burgueño, and A. Vallecillo. Towards approximate model transformations. In *Proceedings of the 3rd Workshop on the Analysis of Model Transformations (AMT) @ MODELS*, pages 1–10, 2014.
16. P. Ziemann and M. Gogolla. OCL extended with temporal logic. In *5th International Andrei Ershov Memorial Conference - Perspectives of Systems Informatics (PSI)*, pages 351–357, 2003.