Configurable Formal Methods for Extreme Modeling

Position Paper

Uli Fahrenberg and Axel Legay

IRISA / Inria Rennes

Abstract. Reliable model transformations are essential for agile modeling. We propose to employ a configurable-semantics approach to develop automatic model transformations which are correct by design and can be integrated smoothly into existing tools and work flows.

Model management is an essential activity in a model-driven development process. This is especially true in an agile context where models are continuously refined, iterated and combined.

It is a common misunderstanding that development by stepwise refinement, or use of component algebras, requires using a highly planned and waterfall-like development process. See for example the following quote:

> An important variant of the waterfall model is formal system development, where a mathematical model of a system specification is created. This model is then refined, using mathematical transformations that preserve its consistency, into executable code. Based on the assumption that your mathematical transformations are correct, you can therefore make a strong argument that a program generated in this way is consistent with its specification. [30, p.32]

We will argue below that methods from model management and formal system development, when combined with and inspired by approaches originating in the area of formal interface and specification theories, can play an important role also in an agile context.

Within the context of model-driven engineering, automated model transformations such as merging, differencing and slicing are of great importance. This is especially true in the industrial context where models can easily get so large that the engineers only see them through viewpoints, or slices; indeed, an explicit system model may not even exist, so that the general model is only implicitly given as a collection of viewpoints.

When the system model is so complex that no single engineer has a comprehensive view, it can be very challenging to ensure correctness of an applied model transformation by inspection. (Even when the system model is less complex, ensuring correctness by inspection may be a difficult and error-prone process and

require advanced tooling.) The use of model transformations which are correct-by-design, or at least checkable-by-design, hence becomes increasingly important. This point of view has also been argued in [1, 6, 24, 27].

A good example is given by one of the case studies in the MERgE ITEA2 project [28]. This consists of a large system model with multiple (more than twenty) viewpoints, each detailing a different aspect of the model. The involved engineers are only working with the model through these viewpoints, as the whole model is too complex to be worked with directly. Now when a model transformation is applied to the model (*e.g.* a new subsystem is added through model merging), the engineers can inspect at their respective viewpoints whether, *locally*, the model transformation has been applied correctly. But can we be sure that this implies that the transformation is also *globally* correct? Can we design a procedure which allows such kind of local-to-global reasoning?

To put it succinctly, it is an important problem in model-driven engineering to ensure that model transformations are *semantically correct* or that, at least, their semantic correctness can be inferred by a combination of slicing and local inspection.

One basic model transformation is the one of *differencing*, *i.e.* assessing differences between models. This is an important ingredient in version control and essential in three-way merging, but can also be applied, more elementarily, to inspect related models for their common points and differences. Semantically correct differencing procedures, for class diagrams and activity diagrams respectively, have been proposed in [24–26]. However, these procedures rely on a complete, formal semantics of the modeling formalism in question, which for most cases is unrealistic: in practice, engineers use modeling tools which do not have a formal semantics, or where an existing formal semantics is too complex to be practically useful.

It is our point of view that any "bottom-up" approach to correct model transformations which uses a complete formal semantics as a starting point, such as the above-cited, is of doubtful use in practice. In practice, engineers develop models according to an intuitive understanding of how things work, and not according to a complete formal semantics (if indeed it exists at all).

Apart from being correct-by-design or checkable-by-design, we have also argued in [18] that it is important that model transformations return an object *of the same type* as the inputs. Hence, the merge of two class diagrams, for instance, should again be a class diagram, the difference of two feature diagrams should again be a feature diagram, etc. This allows developers to visualize the transformation and to manipulate it using the usual tools for working with models; the transformation integrates smoothly into existing tools and work flows. Figure 1 shows an example of such a work flow which necessitates that the difference between two models is again a model of the same kind.

[24–26] propose semantic difference operators for class diagrams and activity diagrams, respectively, however, their approaches are *enumerative* in nature: their output is a (potentially infinite) list of object models which witness the difference between the input diagrams. The output language is thus different

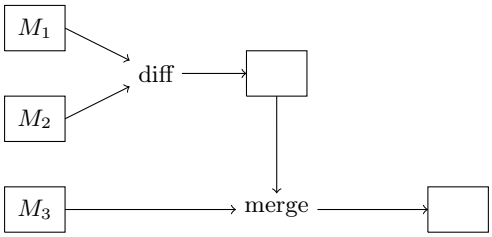

Fig. 1. Example of a work flow where the changes ("diff") between models M_1 and M_2 are automatically applied ("merge") to M_3. Note that a simplistic "diff" is not always the correct approach; it can be important to *e.g.*, tell the difference between renaming an entity and a deletion followed by an addition.

from the input language, which makes it impossible to integrate their tool into a standard work flow without additional processing steps.

Within the subject of interface and specification theories in formal methods, semantic model transformations exist for many types of low-level behavioral models [3–5, 7–9, 11–14, 20, 22, 23]. In recent work, we have developed a generalization of these approaches in which the semantics of models is *configurable* [2,16,17]: When models and specifications contain quantities such as timing information or resource use, the precise behavior of model transformations depends on the type of quantities and on the application. Hence, the precise definitions and properties of model transformations depend on the quantitative semantics, and our generalization offers a generic way of configuring the transformations and properties according to the semantics.

Similarly in spirit, Maoz *et.al.* have in [10, 27] introduced techniques for semantically configurable analysis of some high-level models, *viz.* class diagrams and scenario specifications. They use feature diagrams [21,29] for configurability, so that the analysis depends on the selection of features.

We have in [15, 18] introduced semantically correct model transformations for feature diagrams and class diagrams, see Figures 2 and 3 for some examples. These operators' return types are the same as their input types, so that they can be integrated smoothly into existing tools and work flows, but they rely on a complete formal semantics. We believe that these approaches can be combined with the configurability of [10, 27] to yield model transformations which are automatic and correct-by-design, yet flexible enough to be practically useful.

Conclusion

We propose to employ a configurable-semantics approach, using feature diagrams, to develop automatic model transformations which are correct by design and can be integrated smoothly into existing tools and work flows. Such model transformations are important for agile modeling methods.

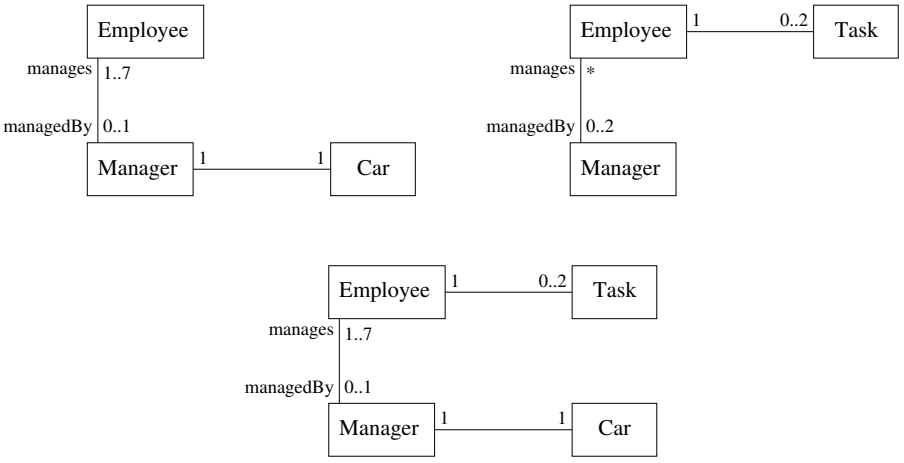

Fig. 2. Two class diagrams (above) and their merge (below), *cf.* [15].

As an example, consider the following scenario: An engineer wants to perform a three-way merge of two models which have evolved from a common ancestor. She will first use her development tools to attempt an entirely syntactic merge, which will detect some conflicts which are inessential because they only amount to syntactic differences while being semantically equivalent (given the developers' intuitive understanding of the semantics). These she can easily detect, and she can adjust her merge tool to take them into account (hence applying a partial semantics). However, due to semantic effects which propagate through the models, also the reverse may happen: there may be semantic conflicts which go undetected by the syntactic approach. These are more difficult to detect and require that even a merge which is syntactically correct be carefully checked by applying possibly several different semantics.

References

1. Kerstin Altmanninger. Models in conflict - towards a semantically enhanced version control system for models. In Holger Giese, editor, *MODELS Workshops*, volume 5002 of *LNCS*, pages 293–304. Springer, 2007.
2. Sebastian S. Bauer, Uli Fahrenberg, Axel Legay, and Claus R. Thrane. General quantitative specification theories with modalities. In Edward A. Hirsch, Juhani Karhumäki, Arto Lepistö, and Michail Prilutskii, editors, *CSR*, volume 7353 of *LNCS*, pages 18–30. Springer, 2012.
3. Sebastian S. Bauer, Line Juhl, Kim G. Larsen, Axel Legay, and Jiří Srba. Extending modal transition systems with structured labels. *Math. Struct. Comput. Sci.*, 22(4):581–617, 2012.
4. Sebastian S. Bauer, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wąsowski. A modal specification theory for components with data. *Sci. Comput. Program.*, 83:106–128, 2014.

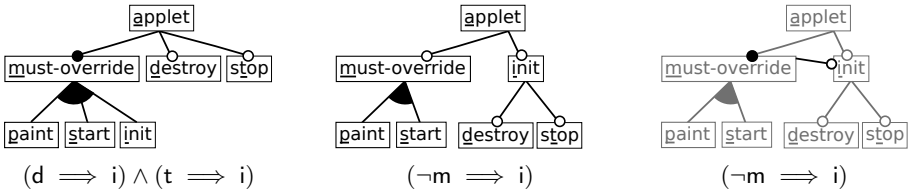

Fig. 3. Two feature diagrams and an over-approximation of their difference, *cf.* [18].

5. Nikola Beneš, Jan Křetínský, Kim G. Larsen, Mikael H. Møller, and Jiří Srba. Dual-priced modal transition systems with time durations. In Nikolaj Bjørner and Andrei Voronkov, editors, *LPAR*, volume 7180 of *LNCS*, pages 122–137. Springer, 2012.
6. Greg Brunet, Marsha Chechik, Steve Easterbrook, Shiva Nejati, Nan Niu, and Mehrdad Sabetzadeh. A manifesto for model merging. In *GAMMA*, pages 5–12. ACM, 2006.
7. Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wąsowski. Constraint Markov chains. *Theor. Comput. Sci.*, 412(34):4373–4404, 2011.
8. Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Freddy Y. C. Mang. Synchronous and bidirectional component interfaces. In Ed Brinksma and Kim Guldstrand Larsen, editors, *CAV*, volume 2404 of *LNCS*, pages 414–427. Springer, 2002.
9. Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *EMSOFT*, volume 2855 of *LNCS*, pages 117–133. Springer, 2003.
10. Barak Cohen and Shahar Maoz. Semantically configurable analysis of scenario-based specifications. In Gnesi and Rensink [19], pages 185–199.
11. Alexandre David, Kim G. Larsen, Axel Legay, Mikael H. Møller, Ulrik Nyman, Anders P. Ravn, Arne Skou, and Andrzej Wąsowski. Compositional verification of real-time systems using Ecdar. *J. Softw. Tools Techn. Transfer*, 14(6):703–720, 2012.
12. Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wąsowski. Timed I/O automata: a complete specification theory for real-time systems. In Karl Henrik Johansson and Wang Yi, editors, *HSCC*, pages 91–100. ACM, 2010.
13. Benoît Delahaye, Benoît Caillaud, and Axel Legay. Probabilistic contracts: a compositional reasoning methodology for the design of systems with stochastic and/or non-deterministic aspects. *Formal Meth. Syst. Design*, 38(1):1–32, 2011.
14. Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher, and Andrzej Wąsowski. Abstract probabilistic automata. In Ranjit Jhala and David A. Schmidt, editors, *VMCAI*, volume 6538 of *LNCS*, pages 324–339. Springer, 2011.
15. Uli Fahrenberg, Mathieu Acher, Axel Legay, and Andrzej Wąsowski. Sound merging and differencing for class diagrams. In Gnesi and Rensink [19], pages 63–78.
16. Uli Fahrenberg and Axel Legay. General quantitative specification theories with modal transition systems. *Acta Inf.*, 51(5):261–295, 2014.
17. Uli Fahrenberg and Axel Legay. The quantitative linear-time-branching-time spectrum. *Theor. Comput. Sci.*, 538:54–69, 2014.

18. Uli Fahrenberg, Axel Legay, and Andrzej Wąsowski. Make a difference! (Semantically). In Whittle et al. [31], pages 490–500.
19. Stefania Gnesi and Arend Rensink, editors. *Fundamental Approaches to Software Engineering - 17th Int. Conf., FASE 2014. Proceedings*, volume 8411 of *LNCS*. Springer, 2014.
20. Line Juhl, Kim G. Larsen, and Jiří Srba. Modal transition systems with weight intervals. *J. Log. Algebr. Program.*, 81(4):408–421, 2012.
21. Kyo Kang, Sholom Cohen, James Hess, William Nowak, and Spencer Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, CMU, 1990.
22. Kim G. Larsen. Modal specifications. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 232–246. Springer, 1989.
23. Kim G. Larsen, Axel Legay, Louis-Marie Traonouez, and Andrzej Wąsowski. Robust synthesis for real-time systems. *Theor. Comput. Sci.*, 515:96–122, 2014.
24. Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. A manifesto for semantic model differencing. In Jürgen Dingel and Arnor Solberg, editors, *MODELS Workshops*, volume 6627 of *LNCS*, pages 194–203. Springer, 2010.
25. Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. ADDiff: Semantic differencing for activity diagrams. In Tibor Gyimóthy and Andreas Zeller, editors, *SIGSOFT FSE*, pages 179–189. ACM, 2011.
26. Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. CDDiff: Semantic differencing for class diagrams. In Mira Mezini, editor, *ECOOP*, volume 6813 of *LNCS*, pages 230–254. Springer, 2011.
27. Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Semantically configurable consistency analysis for class and object diagrams. In Whittle et al. [31], pages 153–167.
28. MERgE ITEA2 project. http://merge-itea-project.irisa.fr/.
29. Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. Feature diagrams: A survey and a formal semantics. In *RE*, pages 136–145. IEEE Computer Society, 2006.
30. Ian Sommerville. *Software Engineering*. International computer science series. Addison-Wesley, 9th edition, 2010.
31. Jon Whittle, Tony Clark, and Thomas Kühne, editors. *Model Driven Engineering Languages and Systems, 14th Int. Conf., MODELS 2011. Proceedings*, volume 6981 of *LNCS*. Springer, 2011.