Towards a Uniform Framework to Support the Evolution of Software Models

Magdalena Widl
widl@big.tuwien.ac.at

Vienna University of Technology, Favoritenstrasse 9-11, 1040 Wien, Austria

Abstract. Software models, as used in Model-Driven Engineering (MDE), are subject to continuous modification and extension in different evolution scenarios. Changes, often requiring propagation throughout the model, may cause undesired effects like inconsistencies and redundancies in different views of the model. Given the size of software models and the complexity of the reasoning tasks to detect relevant problems, automatic verification support is inevitable. We suggest to tackle this problem by first classifying the nature of changes in the course of software evolution and determining inconsistencies relevant for MDE, then finding one or more logic formalisms that are suitable to automate reasoning tasks, and finally establishing and evaluating an overall formal infrastructure to support model evolution.

1 Problem

One of the major challenges in modern software engineering is dealing with the increasing complexity of software systems. Model-Driven Engineering (MDE) promises to handle this complexity using the abstraction power of software models based on languages such as the Unified Modeling Language (UML) [4,21]. Like traditional program code, models are not resistant to change, but evolve over time by undergoing continuous extensions, corrections, and (possibly parallel) modifications. Model management tasks such as synchronization, versioning, and co-evolution describe multiple dimensions of model evolution. Change propagation, demanded by each of these tasks, bears the danger of infiltrating the models with inconsistencies and redundancies, both properties that render the models useless in the context of MDE. Model verification is thus indispensable.

Given the size of software models in practice, manual verification turns out a very tedious task, thus automated methods are needed. For some verification tasks, such as UML class diagram consistency, results on their computational complexity have been reported in [3,17], with many problems being EXPTIME-complete. This fact contributes significantly to the difficulty of developing such an automated method. Still, the situation is not hopeless. Using clever pruning techniques relying on model specific information, there is a good chance that despite the high complexity, feasible solutions can be found in practice. In the context of multi-view modeling, another way of reducing the search space is to consider the fact that one view provides constraints for others.

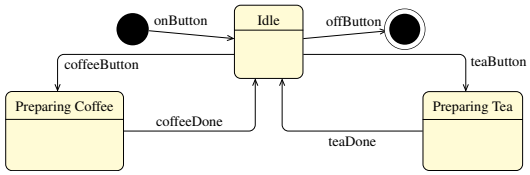

Fig. 1. State machine diagram for the class CoffeeMachine

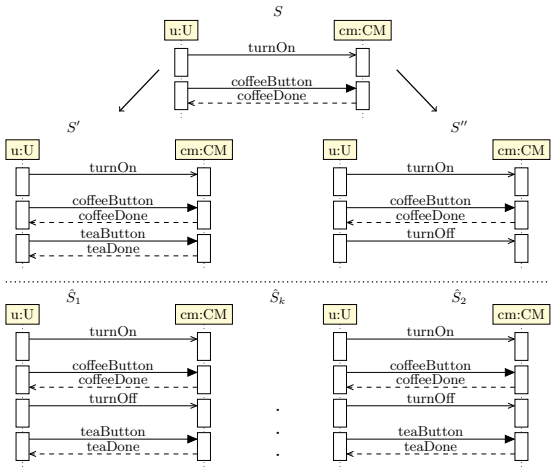

Fig. 2. Versioning scenario for a sequence diagram

The following example describes an inconsistency caused by an automatic merge of changes as it might occur in model versioning.

Figure 1 depicts a state machine diagram modeling a coffee machine and the upmost diagram S in Figure 2 a possible behavior of the same machine in terms of a sequence diagram. Two software engineers change the sequence diagram at the same time: one includes the messages *teaButton* and *teaDone*, resulting in S' the other adds the message *turnOff*, resulting in S''. Each change on its own results in a sequence diagram consistent with the state machine. The next step is to merge the changes into a new sequence diagram using an automatic versioning tool, e.g. as proposed in [6]. As the messages of a lifeline are represented as ordered list, a conflict is triggered by both newly added messages being stored at the same index of the list. A conceivable merging strategy is to consider all possible orders of the two added messages, which result in several syntactically correct diagrams. Figure 2 shows two possibilities $\hat{S}_1$ and $\hat{S}_2$, as $turnOff$ can be placed before or after *teaButton*. However, making tea after turning off the machine does not make much sense and a modeler would avoid such a solution in a manual merge process.

In UML knowledge required for a semantically correct merge of sequence diagrams can be found in state machine diagrams. In the illustrated example,

the sequence of preparing tea after turning off the machine is not modeled in the state machine.

In this case, two different types of dynamic UML diagrams are involved in an inconsistency. However, the UML standard [21] also permits to define diagrams that are inconsistent within themselves. In dynamic diagrams, a possible inconsistency is the presence of deadlocks. Static diagrams are inconsistent if they contain one or more classes that cannot be instantiated due to overconstraining by contradicting cardinalities or disjointness/completeness attributes in generalizations.

Another category of inconsistencies comprises those between static and dynamic UML diagrams, e.g. a dynamic diagram using a method that is not defined by the static diagram.

Any of these inconsistencies may easily occur in the course of model evolution, which is a multidimensional, multi-view process. The success of MDE highly depends on consistent models, which are usually of considerable size and computationally very expensive to reason on. The development of automatic methods is thus inevitable.

We propose to exploit the information that is distributed over different UML diagrams to support automated, semantics-aware model management tasks present in the evolution of software models.

2 Related Work

The landscape of related work on this topic shows a very heterogeneous set of formal techniques, which is difficult to convert into an integrated MDE environment. Some verification tasks have not been tackled at all, or lack implementation and real-life experimentation.

An urgent need for supporting model evolution in the context of MDE has already been identified in 2005 by Mens et al. [20], and emphasized again by France et al. [13]. To date, different approaches tackling specific aspects of model evolution have been introduced [6,8,14,22], but to the best of our best knowledge, not all aspects have been covered yet and an overall framework is missing.

Some of the mentioned verification tasks have already been addressed separately in literature: Berardi et al. [3] and Borgida et al. [5] encode class diagrams in a description logic (DL), which allows to perform reasoning tasks using a DL solver. Using this encoding, they also prove that reasoning on class diagrams is EXPTIME-complete. Kaneiwa and Satoh [17] suggest some decision procedures to reason on different fragments of the UML class diagram, establishing some upper complexity bounds for reasoning on these restricted class diagrams.

Regarding the verification of dynamic diagrams, i.e. state-machine and activity diagrams, model checking has been a popular approach: Knapp et al. [18] and Schaefer et al. [23] suggest a verification tool that compiles state-machines into PROMELA models and collaborations into Buechi automata, which are fed into the SPIN [16] model checker to check whether the state-machine represents the system behavior described by the collaboration diagram. Model checking of

activity diagrams has been suggested by Eshuis [11]. A categorization of inter-diagram inconsistencies along with another description logic approach has been proposed in Van Der Straeten et al. [24].

Several works have tackled the issue of formal semantics of UML, such as [7,12,15,19] but each of them either only deals with single-view models or provides a formalization that is not suitable for reasoning tasks.

3 Proposed Solution

State of the art survey (WP1): The first step in the present project will be an in-depth survey of the state of the art in the areas of model evolution and model verification. We will broaden the scope of considered related work to software verification, given its close relation to model verification and the large amount of related work.

Taxonomy of change and definition of inconsistencies (WP2): Second, we will establish a taxonomy of change in model evolution. This will be done in a similar fashion to the work of Mens et al. [20], which tackles such a taxonomy for code evolution. We will also precisely define different types of inconsistencies in software models and analyze how particular types of changes are related to particular inconsistencies. This workpackage forms an important part of a precise problem definition.

Definition of a relevant subset of the UML metamodel (WP2): As mentioned in Section 1, inconsistencies may arise in different views of a software model where each view is represented by different diagrams. It is thus important to work on a multi-view model that includes both static and dynamic diagrams. We will define a multi-view metamodel as a subset of the UML metamodel [21], covering the UML class diagram to model the static view, and the UML state-machine and sequence diagrams to model the dynamic view. At first, we will work on a subset, possibly omitting constructs like generalization. The metamodel will then be iteratively extended in the course of the project.

Reasoning methods (WP3):

To automatically verify instances of our metamodel, we plan to use a logic formalism. The encoding to a formalism also gives it a formal semantics. We have several requirements to that formalism: It must be sufficiently expressive, yet still decidable, efficient reasoning tools should be available, and optimally, the formalism should be able to capture all views and all reasoning tasks.

To verify dynamic properties, we have considered to apply model checking techniques, similar to the approach by Knapp et al. [18]. So far, we developed a translation of the problem presented in Section 1 into Promela, the input language of the model checker Spin. However, the problem definition has been kept simple, e.g. it allows the sequence diagram to be checked against only one state machine. As next step we intend to extend the translation to multiple state machines.

Another formalism we have in mind is Satisfiability Modulo Theories (SMT) [2]. We plan to attempt to express our model checking problem in SMT and apply SMT-based bounded model checking as proposed by Armando et al. [1] or Cordeiro et al. [9]: An SMT problem is expressed as a quantifier-free formula in first order logic and a background theory. The objective is to decide whether the formula is satisfiable with respect to the theory. Several theories, such as the theory of uninterpreted functions, the theory of bit-vectores or the theory of arrays, have been defined to date, most of them being decidable. The background theory of an SMT problem may be any theory, or a combination of theories.

Using SMT with one or perhaps more theories it may be possible to capture more than the state machine and sequence views and to perform other relevant reasoning tasks. We plan to also investigate on this possibility.

Handling complexity (WP4): Given that the described verification tasks will be of high computational complexity, we also consider investigating on less complex fragments of UML. The results presented by [17] and [3] provide a good foundation for this task.

Another way to tackle this problem is to consider the fact that in practice reasoning tasks need not always be executed on full models. Evolution of a software model often causes only small changes at a time. This could allow incremental verification, which has a considerable effect on the performance [10].

Evaluation (WP5): The implemented approach will be tested as described in the evaluation plan given in Section 6.

4 Preliminary Work

The described work is part of the recently started FAME project[1] and still in a very early stage. Through the model versioning project AMOR[2] we have already gained experience in the area of model merging and model visualization. We will also be able to use tools developed for AMOR, such as the Ecore Mutator[3] as well as model differencing and model merging components.

To date, we have been working on tasks described in WP1, WP2 and WP3. We have started surveys on model evolution and software verification and defined the scope of our work accordingly. As far as WP2 is concerned, we have defined possible evolution scenarios and inconsistencies within single diagrams and across different diagrams.

We have further defined a subset of the UML metamodel containing multiple views, namely the UML sequence, state-machine and class diagrams. Advanced concepts such as generalization are omitted for the moment. We plan to first work on this restricted metamodel and then gradually extend it.

Based on this metamodel, we have defined a translation of the problem described in Section 1 to Promela, the input language of the model checker Spin.

[1] http://www.modelevolution.org/

[2] http://www.modelversioning.org/

[3] http://www.modelversioning.org/index.php?option=com_content&view=article&id=65:ecore-mutator&catid=38:prototypes&Itemid=94

This way, SPIN can be used to detect inconsistencies between sequence diagrams and state machines, and its output integrated into an automated merging process.

5 Expected Contributions

The objective of our work is the establishment of an infrastructure to support model evolution in the context of MDE to facilitate several reasoning tasks on model level. Special emphasis will be set on techniques for semantic-aware model merging. This will unburden software designers from the tedious task of manually executing model management tasks such as inconcistency detection after model merging. Continuous integration will be well supported, allowing for quick error detection. More concretely, we will contribute:

- A thorough survey of the state of the art in model evolution and model verification
- A taxonomy of change in model evolution
- A precise definition of inconsistencies in multi-view models
- A formalization of a relevant, multi-view subset of UML using a well-established formalism
- Formal reasoning methods to detect inconsistencies

The results will also be relevant to the area of software testing. Diagrams, like sequence diagrams, can be used to model test cases and then be checked against a state-machine diagram. This allows automated support during the test phase of the software development cycle.

Further, our work will solve open problems for versioning tools like the one proposed in [6], i.e. automated merging of diagrams maintaining consistency with diagrams representing different views, as illustrated in the coffee machine example.

6 Plan for Evaluation and Validation (WP5)

The evaluation of our work will be done as follows. A prototypical intrastructure for model management will be implemented based on Eclipse open source technologies allowing the management of Ecore-based models. We will use the AMOR repository, which provides a powerful infrastructure for storing and managing evolving models and associated metadata.

To evaluate the expected improvements in the model management process, a quantitative evaluation of our approach is planned based on various types of experiments. We will reuse benchmarks containing model management scenarios that were previously developed for the AMOR project. A relevant set of software models will be generated using Ecore Mutator and obtained from industry. Using this input, we will evaluate the scalability of our logical reasoning method and verify if they indeed manage to fullfill the users' requirements, i.e. indeed cover all expected inconsistencies and integrate well within the modeling environment.

We further plan to conduct an empirical study within the context of educational activities. In particular, we will involve students of our institute's "Model Engineering" lab. It is usually attended by 80-100 students who work on different assignments in model engineering. We will split the students in two groups, one of them being supported by our new framework, the other serving as control group. At the end of the course we will draw conclusions based on the quality of the works, log-files, set-up questionnaires and personal feedback in order to learn about the practical applicability of our developed techniques.

7 Current status

The estimated time from submission of this proposal to completion is 30 months. The timeline is in accordance with the working plan of the FAME project. The outcomes will result in a Phd thesis. For a detailed description of the workpackages please refer to Sections 3 and 6.

References

1. A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. *International Journal on Software Tools for Technology Transfer*, 11:69–83, January 2009.
2. C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*. IOS Press, 2009.
3. D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1-2):70–118, 2005.
4. J. Bézivin. On the unification power of models. *Software and System Modeling*, 4(2):171–188, 2005.
5. A. Borgida and R. J. Brachman. Conceptual modeling with description logics. In *The description logic handbook: Theory, Implementation and Applications*, pages 349–372. Cambridge University Press, New York, NY, USA, 2003.
6. P. Brosch, G. Kappel, M. Seidl, K. Wieland, M. Wimmer, H. Kargl, and P. Langer. Adaptable model versioning in action. In *Modellierung*, pages 221–236, 2010.
7. M. Broy, M. V. Cengarle, H. Grönniger, and B. Rumpe. *Definition of the System Model*, pages 61–93. John Wiley & Sons, Inc., 2009.

8. A. Cicchetti, D. Di Ruscio, and A. Pierantonio. Managing dependent changes in coupled evolution. In *Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations (ICMT '09)*, pages 35–51, 2009.
9. L. Cordeiro, B. Fischer, and J. Marques-Silva. SMT-based bounded model checking for embedded ANSI-C software. *IEEE Trans. on Software Engineering*, May 2011.
10. A. Egyed. Instant consistency checking for the uml. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 381–390, New York, NY, USA, 2006. ACM.
11. R. Eshuis. Symbolic model checking of UML activity diagrams. *ACM Transactions on Software Engineering Methodology*, 15(1):1–38, 2006.
12. R. Eshuis and R. Wieringa. A formal semantics for UML activity diagrams - formalising workflow models. Technical Report TR-CTIT-01-04, Enschede, 2001.
13. R. B. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *Proceedings of Future of Software Engineering (FOSE '07)*, pages 37–54, 2007.
14. H. Giese and R. Wagner. From model transformation to incremental bidirectional model synchronization. *Software and System Modeling*, 8(1):21–43, 2009.
15. D. Harel and S. Maoz. Assert and negate revisited: Modal semantics for uml sequence diagrams. *Software and Systems Modeling*, 7(2):237–252, 2008.
16. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
17. K. Kaneiwa and K. Satoh. On the complexities of consistency checking for restricted UML class diagrams. *Theoretical Computer Science*, 411(2):301–323, 2010.
18. A. Knapp and S. Merz. Model checking and code generation for UML state machines and collaborations. In *5th Workshop on Tools for System Design and Verification*, 2002.
19. G. Lüttgen and M. Mendler. Fully-abstract statecharts semantics via intuitionistic kripke models. In U. Montanari, J. Rolim, and E. Welzl, editors, *27th International Colloquium on Automata, Languages and Programming (ICALP 2000)*, volume 1853 of *Lecture Notes in Computer Science*, pages 163–174, Geneva, Switzerland, July 2000. Springer-Verlag.
20. T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri. Challenges in software evolution. In *Proceedings of the 8th International Workshop on Principles of Software Evolution*, pages 13–22, Washington, DC, USA, 2005. IEEE Computer Society.
21. Object Management Group. UML superstructure specification. `http://www.omg.org/spec/UML/2.3/`, 5 2010.
22. R. Salay, J. Mylopoulos, and S. M. Easterbrook. Using macromodels to manage collections of related models. In *Proceedings of the 21st International Conference on Advanced Information (CAiSE '09)*, volume 5565/2009 of *Lecture Notes in Computer Science*, pages 141–155, 2009.
23. T. Schäfer, A. Knapp, and S. Merz. Model checking UML state machines and collaborations. *Electronic Notes in Theoretical Computer Science*, 55(3), 2001.
24. R. Van Der Straeten, T. Mens, J. Simmonds, and V. Jonckers. Using description logic to maintain consistency between UML models. In *UML*, pages 326–340, 2003.