Improving Conflict Resolution in Model Versioning Systems*

Petra Brosch†
Institute of Software Technology and Interactive Systems
Vienna University of Technology, Austria
brosch@big.tuwien.ac.at

Abstract

Collaborative software development is nowadays inconceivable without optimistic version control systems (VCSs). Without such systems the parallel modification of one artifact by multiple users is impracticable. VCSs proved successful in the versioning of code, but they are only conditionally appropriate to the management of model versions. Conflicts which may occur when two different versions of one model are merged are detected in an unsatisfactory manner and consequently, automatic conflict resolution is hardly offered. In this work, a generic framework allowing precise conflict detection and intelligent conflict resolution for models is proposed.

1. Introduction

This research proposal describes a generic framework for model versioning. When applied on models, current VCSs suffer from three deficiencies comprising *erroneous conflict detection*, *unsupportive conflict resolution*, and *inflexibility* with respect to various modeling languages and modeling tools. We present an adaptable approach to overcome these problems in order to realize user-friendly versioning of models.

Since models are a source of information which has become indispensable to software engineering—either for documentation purpose or for model driven engineering (MDE)—the need for a model management tool arises. Driving paradigms in software engineering are changing from "everything is an object" to "everything is a model" [3]. Source code and configuration files are usually managed by version control systems in an *optimistic way*, allowing developers parallel manipulation. In contrast to *pessimistic locking*, where only one developer is allowed to edit a file, multiple developers may check-out and modify the same artifact at the same time. The parallel modified files have to be compared, conflicting changes must be detected and resolved, and finally a merged, consistent version is checked-in into the repository. Standard VCSs for code versioning usually work on file-level and perform the conflict detection by line-oriented text comparison. Even when applied on the XMI-serializations of models, such systems are not suitable for the versioning of models. For example, the arrangement of the elements in the serialization is not unique which might result in the detection of conflicts which are no conflicts at all. Furthermore, lines of text as indivisible units are too coarse grained and parallel modifications of the same line cannot be handled [7]. Moreover, conflict detection is poor, when using the model's text representation, because the graph-based structure is destroyed and the associated syntactic and semantic information is lost. To preserve semantic meaning of models and to ensure the structural validity of the model in respect of its metamodel, the graph based representation should be retained and special model management tools are highly valuable [2].

We realize such a model management system with special focus on model versioning and genericity in the context of the AMOR project (Adaptable Model Versioning) [1]. The generic framework is independent of any Ecore[1] based modeling language and may be utilized out-of-the-box for arbitrary domain-specific modeling languages (DSLs).

2. Motivating Example

Once conflicts are found, they have to be resolved in order to receive a consistent merged version. Three types of conflicts, namely *syntactic*, *structural*, and *semantic conflicts*, may arise during a merge [7].

In the following, we present small examples which illustrate typical merge problems.

*This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and FFG under grant FIT-IT-819584

†Funding for this research was provided by the fFORTE WIT - Women in Technology Program of the Vienna University of Technology, and the Austrian Federal Ministry of Science and Research.

[1] http://www.eclipse.org/modeling/emf/

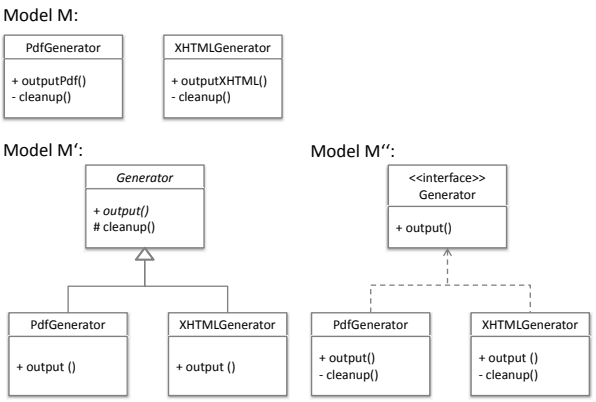

Figure 1. Semantic Conflict

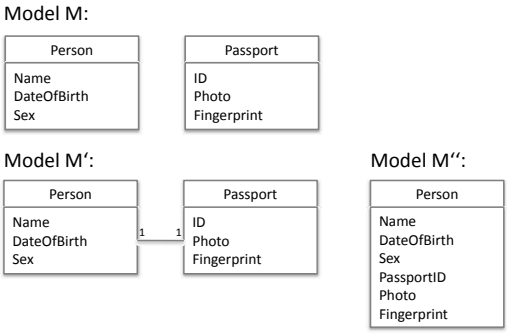

Figure 2. Syntactic Conflict

Figure 1 shows a semantic conflict, that might happen during the refactoring phase. Two classes in model M, `PdfGenerator` and `XHTMLGenerator`, do a similar job, that is generating output in a specific format. Two developers refactor the model in parallel, one introduces in model M' an abstract class `Generator` with the abstract method `output` and a default implementation of the method `cleanup`. The other developer extracts the public method `output` into an interface, as shown in model M". This semantic conflict may be resolved by ignoring the interface. Since the abstract class and the interface have similar semantics, and the abstract class provides a default implementation of a helper method, the changes in model M' may be suggested to take place in the merged version.

The second example depicted in figure 2 points out a syntactic conflict. Again, two developers edit the same model elements. Model M consists of two classes `Person` and `Passport`, but the relation among them is missing. Two parallel modifications take place, the first developer adds a one-to-one association, while the second developer deletes the class `Passport` and adds its attributes into the class `Person`. When the changes are merged, a syntactic conflict at the model layer has to be resolved. One end of the association added in model M' has been deleted in model M", and a dangling references problem occurs. Two suggestions for conflict resolution are possible; delete the association, or do not delete the class `Passport` and keep the association.

Interestingly, merging the same situation at the code layer, a semantic conflict arises, because declared variables are checked in the semantic analysis phase of a compiler, which is performed after syntax analysis. Conflict resolution strategies are nearly the same; do not introduce the attribute `passport` in class `Person`, or do not delete the class `Passport`.

Different representations of similar problems might need a special treatment, even though the underlying concepts are the same. Thus, generic VCSs, that are applicable to any modeling language, are often characterized by poor conflict resolution support. Modeling language dependent VCSs, in turn, have proper versioning support, but are very inflexible [1]. This leads to the need for an adaptable model versioning system.

3. Research Goals

For achieving a high quality model versioning system, we have identified three key research goals. Firstly, to overcome the problems of too generic, and too specific VCSs, an *adaptable framework* for model versioning should be provided.

Secondly, a *precise conflict detection* is needed. To enhance the precision of conflict detection and avoid wrongly indicated conflicts, the generic conflict detection component provides hooks for extensibility. The conflict detection may be enhanced by defining semantic equivalent concepts for specific modeling languages. Moreover, the state-based conflict detection may be enhanced by a specific operation-based versioning approach which may incorporate refactoring operations.

The third key research goal comprises *intelligent conflict resolution*. After conflict detection, the user is usually confronted with conflicting differences of two models and is forced to remodel alternatives to achieve a valid and consistent model. As changes in models often have several side-effects, manually resolving conflicts is error prone and cumbersome. The conflict resolution component illustrated in figure 3, supports the user in this repetitive task with a proper visualization of conflicts and suggestions of possible resolution strategies, verifying, that merged models are valid and consistent with respect to the according metamodel. Suggestions for conflict resolutions are looked-

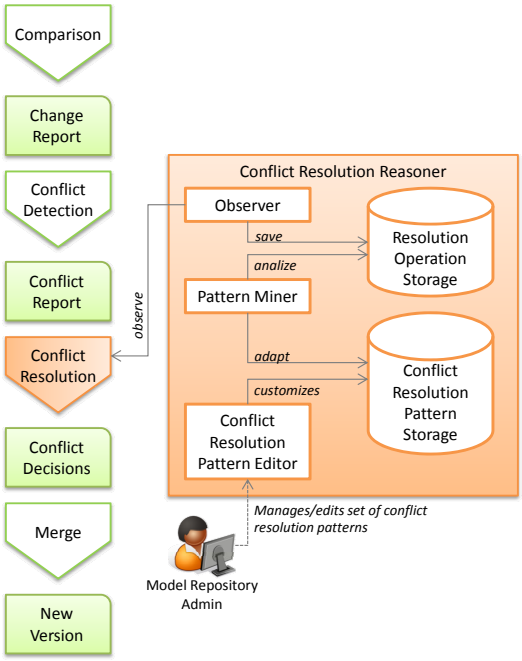

Figure 3. System Architecture

up in the conflict resolution pattern storage. Resolution patterns may be created by the model repository admin, and are derived by tracking user behavior when conflicts are manually resolved. Frequent conflicts are analyzed, whether they might be resolved automatically, or at least if the resolution may be supported with suggestions. As this information is language dependent, it is not part of the generic framework. Resolution rules will be added to the *Conflict Resolution Pattern Storage*.

4. Research Questions

The proposed work addresses multiple issues, which are investigated in the Ph.D. studies. The key issues are described in the following.

Classification of conflicts. One of the most urgent and probably challenging question is how to classify occurring conflicts with respect to their resolution. This classification is the basis for all the other research issues. Several approaches may be eligible reaching from distinguishing structural and methodological inconsistencies as proposed in [4] to a classification in pre- and post-merge conflicts. Another crucial concern is the difference between models and source code, meaning that models are either just another view of the same issue, or that conflict resolution differs due to the incorporated human interpretation in models.

Description and storage of conflict resolution patterns. Once conflicts are classified, their resolution patterns have to be described and stored. This can be done either by defining a dedicated conflict resolution language, or by graph transformation. An important issue is also how to provide a user friendly interface, which may be integrated in the pattern editor.

Automation of conflict resolution. To reduce merging errors occurred by incorrect decisions made in human interaction, as much conflicts as possible should be automatically resolved. Conflicts which occur very often, and may be resolved without human interaction have to be identified. A possible strategy is to learn conflict resolution patterns from the user, by tracking decisions made when conflicts are manually resolved. To ensure consistency of models and guarantee a successful merge, a test-driven approach already used in continuous integration tools for source code management may be applied—after merging, the model should be tested.

User support. Another key research question is how to present conflicts to the user in a supporting manner. That includes not only visualization of raw conflicts which have to be resolved totally by hand, but also possible suggestions for conflict resolution and the associated side-effects.

5. Research Methods

In order to investigate these research questions, the following research methods are applied.

Literature study and evaluation of existing systems. The first steps which currently take place, are a comprehensive study of the related work and an evaluation of state-of-the-art systems. For example, MolhadoRef [6] is a semantic and operation-based VCS for Java source code with respect to refactoring operations. MolhadoRef supports the user with concrete resolution suggestions when conflicts are found. SemVersion [11] is a RDF-centric versioning approach with structural and semantic based versioning for ontologies. CoObRa [10] is a modeling language independent tool, and provides a stepwise conflict resolution based on the latest version of the model in the repository and the change protocol. Also Odyssey-VCS [9] supports any modeling language, and allows the adaptation of the unit of comparison.

Questionnaire. One of the most forthcoming steps is preparing a questionnaire, which will be answered by interested software developers, that are already using VCSs and modeling tools. This questionnaire helps to identify the major requirements for model versioning systems, and how versioning is already used in practice.

Model analysis. For getting a better understanding of how models are used in practice, and how large differences between two versions are, models from industry projects will be analyzed.

Implementation of a prototype. Based on previous steps concerning change and conflict reports, an architecture as proposed in figure 3 is implemented.

Evaluation of the prototype. The implemented prototype will be tested and evaluated in a university course on model engineering with about 200 students.

6. Related Work

With dissemination of MDE and DSLs, also the demand for model versioning systems increases. This accompanies with the raising number of different approaches for model versioning. Besides the versioning tools mentioned above, some of the most related are shortly presented in the following.

Conradi et al. [5] point out, that model versioning systems will not provide "the" model, instead they should be customizable to come up to the specific needs of an application. Mens et al. [8] focus on inconsistency detection and resolution as graph transformation rules. Since the resolution of conflicts might result again in conflicts, detecting possible cycles or other incompatible strategies may improve the resolution process.

7. Conclusion

Within MDE, models are leveraged to first-class entities to benefit from their high level of abstraction, their rich semantics, and their powerful visualization capabilities. In order to establish MDE for a broad spectrum of application areas, a reliable infrastructure for model management is a primary prerequisite. As in traditional software engineering, MDE based projects are seldom conducted by one single developer, but by teams. Consequently, the deployment of version control systems is inevitable. When following an optimistic versioning approach, which allows concurrent modifications on one artifact by multiple developers, standard VCSs are inappropriate for model versioning as code and models differ in many aspects. When applying standard VCSs on models, the result is a very complex check-in process. Efficient model versioning demands for different methods and techniques than code versioning. In this work, we proposed a generic approach which aims at precise conflict detection and intelligent conflict resolution for the realization of user-supportive model versioning.

References

[1] K. Altmanninger, G. Kappel, A. Kusel, W. Retschitzegger, M. Seidl, W. Schwinger, and M. Wimmer. AMOR - Towards Adaptable Model Versioning. In *MCCM'08, Workshop of Models'08*, Toulouse, France, 2008.

[2] P. A. Bernstein and S. Melnik. Model Management 2.0: Manipulating Richer Mappings. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1–12, New York, NY, USA, 2007. ACM.

[3] J. Bézivin. MDA: From Hype to Hope, and Reality. http://www.sciences.univ-nantes.fr/info/perso/permanents/bezivin/UML.2003/UML.SF.JB.GT.ppt, October 2003. The 6th International Conference on the Unified Modeling Language, San Francisco, California, Keynote talk.

[4] X. Blanc, I. Mounier, A. Mougenot, and T. Mens. Detecting Model Inconsistency through Operation-Based Model Construction. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 511–520, New York, NY, USA, 2008. ACM.

[5] R. Conradi and B. Westfechtel. Version Models for Software Configuration Management. *ACM Comput. Surv.*, 30(2):232–282, 1998.

[6] D. Dig, K. Manzoor, T. N. Nguyen, and R. Johnson. MolhadoRef: A Refactoring-Aware Infrastructure for OO Programs. In *eclipse '06: Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange*, pages 25–29, New York, NY, USA, 2006. ACM.

[7] T. Mens. A State-of-the-Art Survey on Software Merging. *IEEE Transactions on Software Engineering*, 28(5):449–462, 2002.

[8] T. Mens, R. Van Der Straeten, and M. D'Hondt. Detecting and Resolving Model Inconsistencies Using Transformation Dependency Analysis. In *Model Driven Engineering Languages and Systems*, pages 200–214. Springer, LNCS 4199, October 2006.

[9] L. Murta, C. Corrêa, a. G. P. Jo and C. Werner. Towards Odyssey-VCS 2: Improvements over a UML-Based Version Control System. In *CVSM '08: Proceedings of the 2008 international workshop on Comparison and versioning of software models*, pages 25–30, New York, NY, USA, 2008. ACM.

[10] C. Schneider and A. Zündorf. Experiences in using Optimisitic Locking in Fujaba. *Softwaretechnik Trends*, 27, May 2007.

[11] M. Völkel. SemVersion Versioning RDF and Ontologies. Deliverable D2.3.3.v1, University of Karlsruhe, July 2006. KWEB EU-IST-2004-507482.