By-Example Adaptation of the Generic Model Versioning System AMOR *

How to Include Language-Specific Features for Improving the Check-In Process

Petra Brosch † Martina Seidl
Konrad Wieland Manuel Wimmer
Business Informatics Group
Vienna University of Technology, Austria
lastname@big.tuwien.ac.at

Philip Langer
Department of Telecooperation
Johannes Kepler University Linz, Austria
philip.langer@jku.at

Abstract

We present configuration mechanisms based on by-example approaches for the adaptable model versioning system AMOR improving the complete versioning workflow. The *Operation Recorder* allows the specification of composite operations. Those operation definitions are used by the *Conflict Manager* supporting the specification of potential merge conflicts and suitable resolution strategies.

Categories and Subject Descriptors D.2.13 [*Software Engineering*]: Reusable Software—Reuse models

General Terms Design, Languages

Keywords model versioning, by-example configuration

1. Introduction

The development of software systems without version control systems (VCSs) is nowadays unimaginable. Optimistic VCSs are of particular importance because such systems effectively manage modifications on one software artifact performed by multiple developers at the same time. To cope with the complexity of modern software systems, model-driven development (MDD) has gained momentum. In this realm, software models are a valuable source of information, not only for traditional documentation purpose but also for the automatic generation of code from models (cf. [Bézivin 2005]). Like other software artifacts, models are developed in teams and evolve over time. Hence, they also have to be put under version control.

Standard VCSs for code usually work at the file-level and perform conflict detection by line-oriented text comparison. When applied to the textual serialization of models, the result is unsatisfactory because the information stemming from the graph-based structure is destroyed and the associated syntactic and semantic information gets lost. Consequently, dedicated VCSs for versioning models have been proposed. However, they either support generic model versioning and therefore do not consider language-specific aspects, or they are built for a specific language and thus are not suitable for other languages. The model versioning system AMOR [Altmanninger et al. 2008] aims at combining the advantages of both types of VCSs providing a generic framework with extension points for including language specific features.

2. Adaptation Techniques

To cope with the seeming contradiction of language independence and language specific features, the extensions points in AMOR support the following adaptation techniques.

Composite Operation Specification. The comparison of the modified model versions usually detects only the application of the primitive operations *insert*, *update*, and *delete*. Composite operations like *move* and refactorings are often not detected but represented as sets of primitive operations. As reported in [Dig et al. 2006], this loss of information is a disadvantage for merging the different versions of a model. Hence, the comparison algorithm has to be extended in order to detect composite operations. The technical specification of these recurrent composite operations should not only

* 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.

OOPSLA 2009, October 25–29, 2009, Orlando, Florida, USA.
ACM 978-1-60558-768-4/09/10.

be performed by experts demanding extensive programming effort and deep knowledge of the development environment, but also by modelers who finally apply the operations.

Conflict Specification. Conflicts are easily detected if the same element is modified in different versions of one model. If one conflict comprises multiple elements, standard systems are not able to detect this conflict at all, or they report multiple local conflicts, which further complicates conflict resolution. In AMOR, it is possible to specify potential conflicts by example, (i.e., without any programming effort), in order to enable the system to detect and to report complex conflicts in a concise representation.

Resolution Specification. The third extension point targets the user-friendly definition of resolution strategies. Recent VCSs indicate where conflicts interfere with the merge process, but they hardly provide any resolution support to the user. AMOR provides a simple recommender component, which offers resolution suggestions. In order to specify resolution suggestions, again a by-example approach is followed, which allows the modeler to specify how to resolve a conflict instead of implementing resolution scripts or defining model transformations.

3. Realization

We develop different tools, that improve the conflict detection and resolution capabilities by specifying composite operations, language-specific conflicts, and finally their resolution. In all of these steps the specification is derived from an example created directly in the concrete syntax of the modeling language and the preferred environment of the user. Therefore, no programming is necessary to adapt AMOR.

Step 1: Composite Operations. First, the modeler empowers the VCS to detect language-specific composite operations like refactorings. For this task we developed the Operation Recorder to design composite operation specifications. The user starts with modeling the initial situation in her preferred modeling environment. Then, this initial model is annotated with unique IDs and copied automatically by the Operation Recorder. On this working copy the modeler performs all operations the composite operation consists of, again in her preferred modeling environment. When the modeler confirms the revised working copy, the Operation Recorder precisely detects all performed operations by conducting a state-based comparison relying on the sound ID-based match. Finally, the Operation Recorder derives pre- and postconditions necessary for the application of the composite operation. The modeler may edit these conditions. For a more detailed description of the Operation Recorder please see the AMOR project homepage[1] and [Brosch et al. 2009a,b]. Completed operation specifications are henceforth used to detect occurrences of specified operations in generic model differences obtained by a state-based comparison. The detection mechanism is implemented by searching for the operation pattern contained in the operation specification. If the pattern is found and the model elements referenced by the matching operations fulfill the pre- and postconditions, an occurrence of the composite operation is at hand. This detection allows a more compact representation of the difference report by folding atomic operations that belong to a composite one. Thus, detecting composite operations enables a faster and better understanding of the original modeler's intention. Furthermore, it enables a smarter conflict detection and resolution.

Step 2: Language-Specific Conflicts. When two modelers perform overlapping refactorings, conflicts are often not detectable by solely applying generic conflict detection algorithms. In AMOR, the Conflict Manager allows to mark specific executions of refactorings as potentially conflicting changes. Since only the parallel execution of these refactorings on overlapping model elements leads to a conflict, a user-defined mapping is supported for specifying the commonly effected model elements. For this sake, the common model elements are simply connected to the respective elements of the refactorings. Using this conflict definition, the generic conflict detection algorithm is capable of marking language-specific conflicts.

Step 3: Resolution Strategies. In the last step a proper resolution strategy is set up to resolve conflicts. One of these strategies is to define an order in which the refactorings should be applied on the base model. Since the model versioning system is able to perform composite operations on arbitrary models by interpreting the operation specification obtained in Step 1, it is also capable of replaying them. Therefore, the replayed execution of a refactoring originally performed by one modeler encloses changes performed by the other modeler, which leads to a merged model combining both intentions and prohibiting merge problems [Brosch et al. 2009b, Dig et al. 2006].

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

References

Kerstin Altmanninger, Gerti Kappel, Angelika Kusel, Werner Retschitzegger, Martina Seidl, Wieland Schwinger, and Manuel Wimmer. AMOR—Towards Adaptable Model Versioning. In *Proc. of MCCM'08 @ MoDELS'08*, 2008.

Jean Bézivin. On the Unification Power of Models. *Journal on Software and Systems Modeling*, 4(2):171–188, 2005.

Petra Brosch, Philip Langer, Martina Seidl, Konrad Wieland, Manuel Wimmer, Gerti Kappel, Werner Retschitzegger, and Wieland Schwinger. An Example is Worth a Thousand Words: Composite Operation Modeling By-Example. *Accepted for MoDELS'09*, 2009a.

Petra Brosch, Philip Langer, Martina Seidl, and Manuel Wimmer. Towards End-User Adaptable Model Versioning: The By-Example Operation Recorder. In *Proc. of CVSM'09 @ ICSE'09*. IEEE, 2009b.

Danny Dig, Tien N. Nguyen, Kashif Manzoor, and Ralph Johnson. MolhadoRef: A Refactoring-aware Software Configuration Management Tool. In *Proc. of OOPSLA'06*. ACM, 2006.