Generic vs. Language-Specific Model Versioning[*]

Adaptability to the Rescue

Petra Brosch
Business Informatics Group
Vienna University of Technology, Austria
brosch@big.tuwien.ac.at

Philip Langer
Business Informatics Group
Vienna University of Technology, Austria
langer@big.tuwien.ac.at

Martina Seidl
Institute for Formal Models and Verification
Johannes Kepler University
Linz, Austria
martina.seidl@jku.at

Manuel Wimmer
Software Engineering Group
Universidad de Málaga, Spain
mw@lcc.uma.es

Gerti Kappel
Business Informatics Group
Vienna University of Technology, Austria
gerti@big.tuwien.ac.at

ABSTRACT
In this paper, we discuss how to make a generic model versioning system language-specific by using various adaptation techniques. In particular, we recap some lessons learned during the AMOR project and outline some open challenges for adaptable model versioning systems.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management—*Software configuration management*

Keywords
model versioning, model management

1. INTRODUCTION
In code-centric software development, mainly language-independent versioning systems are employed [3]. Such systems do not pose any restrictions concerning programming languages and development environments. Also in the context of model-driven engineering (MDE), language-specific versioning systems are rare, when taking into consideration that domain-specific modeling languages are becoming more and more popular. However, language-specific versioning support provides better results regarding difference and conflict detection, as well as conflict visualization and resolution. The main reason for not adopting a language-specific versioning approach seems to be the following. In contrast to language-independent approaches, language-specific versioning systems currently have to be developed for each modeling language from scratch. While generic model versioning systems provide out-of-the-box versioning support for arbitrary modeling languages adhering to the same meta-metamodeling language, they might lack of precision as they are not aware of the language semantics. Hence, to combine the strengths of both approaches, a generic yet adaptable framework for model versioning seems promising.

In this paper, we discuss how such an adaptable versioning framework may be realized. To this end, we present AMOR[1] [2], an optimistic model versioning system which provides generic versioning support in order to be used out-of-the-box for any Ecore-based modeling language. For improving the overall versioning process, AMOR may be adapted with language-specific information. Based on our experiences with AMOR, we derive multiple challenges which adaptive model versioning systems have to face.

2. ADAPTABLE MODEL VERSIONING
The main goal of AMOR is to combine the advantages of both generic and language-specific model versioning by providing a generic, yet adaptable model versioning framework. The generic framework offers versioning support for all modeling languages conforming to a common meta-metamodel out-of-the-box and enables users to enhance the quality of the versioning capabilities by adapting the framework to specific modeling languages using well-defined adaptation points. Thereby, developers are empowered to balance flexibly between reasonable adaptation efforts and the required level for versioning support.

Model merging with AMOR. Given two concurrently evolved models V_{r1} and V_{r2} stemming from one common ancestor version V_o, AMOR realizes a multi-phase versioning process.

The first phase of the merge process concerns the *operation*

[*]This work has been partly funded by the Vienna Science and Technology Fund (WWTF) through project ICT10-018 and by the Austrian Science Fund (FWF) under grant J 3159-N23.

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

detection [6]. The goal of this phase of the process is to detect precisely which operations have been applied between V_o and V_{r1} as well as between V_o and V_{r2}. Since AMOR aims to be independent from the modeling editor, a state-based model comparison is performed, which is carried out in three steps. First, the revised models are *matched* with the common original model V_o. Therefrom, two *match models* are obtained, which describe the correspondences among the original model and the revised models. Next, the applied *atomic operations* are computed. Besides atomic operations, AMOR provides techniques for detecting *composite operations*, such as model refactorings. The output of this phase of the process are two difference models $D_{V_o,V_{r1}}$ and $D_{V_o,V_{r2}}$, which describe all operations performed in the concurrent modifications.

Based on the two difference models, the next phase of the process of the merge process aims at detecting *conflicts* among the concurrently applied operations [6]. Thereby not only *atomic operation conflicts* like delete-update conflicts, but also conflicts among *composite operations* are revealed. All detected conflicts are saved into a *conflict model*.

The computed differences and detected conflicts serve then as input for the *conflict resolution* phase [1, 13]. AMOR's conflict resolution process provides two interchangeable strategies. Both strategies combine the strength of automatic merging and inconsistency toleration and calculate a *tentatively merged model*. The tentative merge acts as base for either *collaboration* [13] and/or *recommendation* supported conflict resolution [1].

Adaption in AMOR. Generic versioning is accomplished by using the reflective interfaces of the Eclipse Modeling Framework [12] (EMF) serving as reference implementation of OMG's MOF standard [11]. Thereby, all modeling languages can be handled immediately for which an EMF-based metamodel is available. AMOR is also independent of the used modeling editor and does not rely on specific features on the editor side. Therefore, we may not apply editor-specific operation recording to obtain the applied operations. Instead, AMOR works only with the states of a model before and after it has been changed and derives the applied operations using state-based model differencing. Further, AMOR is adaptable with language-specific knowledge by the users of the versioning system. AMOR's design enables these users to create and maintain the adaptation artifacts by themselves without requiring deep knowledge on the implementation of the versioning system and programming skills. Therefore, AMOR is adapted by providing descriptive adaptation artifacts and uses, as far as possible, modeling languages to specify the required language-specific knowledge. No programming effort is necessary to enhance AMOR's versioning capabilities with respect to language-specific aspects.

3. FUTURE CHALLENGES

Based on our experience gained during the AMOR project, we outline in this section some interesting challenges to be addressed in future.

Adaptation points. As mentioned above, one major goal of AMOR is to combine the benefits of generic and language-specific versioning systems by providing adaptation points that can be used to enhance the quality of the merge process with respect to the peculiarities of a specific modeling language. Therefore, we introduced adaptation points for adding language-specific knowledge such as match rules, composite operations, conflict types, and conflict resolution strategies in order to enable dedicated support for specific aspects of a modeling language. However, it is still an open question whether these adaptation points are *sufficient*. Thus, further research is necessary to investigate if the quality of certain phases of the merge process can be significantly improved when additional language-specific knowledge is incorporated.

Specification of adaptation artifacts. Another open challenge with respect to the adaptation of a versioning system is how to represent the language-specific knowledge in terms of adaptation artifacts. One way is obviously to replace the entire generic implementation of a merge phase with a language-specific implementation that encodes the language-specific knowledge. This, however, requires programming effort and specific knowledge on the mechanics of the versioning systems (cf. white-box adaptation vs. black-box adaptation [4]). A superior way would be to enable users to describe the respective language-specific knowledge in terms of declarative adaptation artifacts and provide generic algorithms that incorporate and interpret these adaptation artifacts dynamically. For instance, it seems to be more convenient to describe the *reasons for language-specific conflicts* for adapting a conflict detection mechanism in contrast to specifying an *implementation for detecting these conflicts*. However, designing such generic algorithms and finding a proper representation of the language-specific knowledge is a major challenge.

Derivation and reuse of adaptation artifacts. Adapting a model versioning system towards dedicated support for a specific modeling language can be a time-consuming task. Therefore, another challenge concerns techniques for *deriving the adaptation artifacts (semi-)automatically* from existing specifications of the modeling language, such as the abstract syntax definition (metamodel and validation rules), the concrete syntax definition, and, if available, the operational semantics definition or modeling operations such as refactorings. We believe that novel techniques exploiting these specifications of the modeling language for deriving adaptation artifacts bear a significant potential. Once adaptation artifacts exist, it would increase efficiency to allow for *reusing existing adaptation artifacts*, on the one hand, across different phases of the merge process and, on the other hand, for multiple similar modeling languages that share specific aspects. Thereby, we should follow the DRY principle ("don't repeat yourself") [5] when adapting model versioning systems.

Performance vs. precision. As the overall merge process consists of several computation intensive steps, committing a new version may get extremely time-consuming for the user, who has to wait and to provide manual conflict resolution. In some cases performance may be increased by a more explicit representation of the language-specific adaptation artifacts. For example, evaluating OCL [10] constraints and tracing back to the respective changes may take much longer, than searching for forbidden change patterns, which are explicitly specified. However, this way, violations due to unspecified change patterns may not be revealed. Therefore, the trade-off between performance and precision has to be investigated in more detail.

Standardization of differences, conflicts, and resolutions. Another important practical issue refers to the interoperability between model versioning systems. In accordance with OMG's core mission of providing vendor and tool independent industry standards, interoperability, reusability, and portability are the major design criteria manifested in the model-driven architecture initiative [7]. However, while for the design of domain-specific modeling languages, those characteristics are preserved by OMG's MOF standard [11], a commonly accepted standard for representing language evolution is still missing, although some incentives [8, 9] have been provided by the OMG in this context. Hence, current model versioning systems use proprietary solutions for capturing model differences, detected conflicts, and their respective conflict resolution patterns, which impedes comparison and interoperability thereof.

Versioning and quality assurance. In the context of model versioning, syntactically non overlapping differences may lead to inconsistent models in respect of the modeling language's semantics or the modeled system's semantics. Therefore, some state-of-the-art model versioning systems aim at detecting and reporting such inconsistencies after the merged version is computed. However, it is controversial, whether inconsistency detection shall be included within versioning support, as such problems may be also introduced by one single modeler. Further, as efficient tools for inconsistency detection and quality assurance already exist, it has to be investigated how appropriate adaptation points for plugging such tools into the versioning systems may be designed.

4. REFERENCES

[1] P. Brosch. *Conflict Resolution in Model Versioning.* PhD thesis, Vienna University of Technology, 2012.

[2] P. Brosch, G. Kappel, M. Seidl, K. Wieland, M. Wimmer, H. Kargl, and P. Langer. Adaptable Model Versioning in Action. In *Modellierung*, volume 161 of *LNI*, pages 221–236. GI, 2010.

[3] J. Estublier, D. Leblang, A. Hoek, R. Conradi, G. Clemm, W. Tichy, and D. Wiborg-Weber. Impact of Software Engineering Research on the Practice of Software Configuration Management. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 14(4):383–430, 2005.

[4] T. Gschwind. *Adaptation and Composition Techniques for Component-based Software Engineering.* PhD thesis, Vienna University of Technology, 2002.

[5] A. Hunt and D. Thomas. *The Pragmatic Programmer: From Journeyman to Master.* Addison-Wesley Professional, 2000.

[6] P. Langer. *Adaptable Model Versioning based on Model Transformation by Demonstration.* PhD thesis, Vienna University of Technology, 2011.

[7] Object Management Group. Model-driven Architecture (MDA). `http://www.omg.org/mda/specs.htm`, 04 2005.

[8] Object Management Group. MOF 2.0 Versioning and Development Lifecycle (MOFVD). `http://www.omg.org/spec/MOFVD/2.0`, 05 2007.

[9] Object Management Group. MOF 2.0 Facility and Object Lifecycle (MOFFOL). `http://www.omg.org/spec/MOFFOL/2.0/`, 03 2010.

[10] Object Management Group. Object Constraint Language (OCL), Version 2.2. `http://www.omg.org/spec/OCL/2.2`, 02 2010.

[11] Object Management Group. OMG Meta Object Facility (MOF) Core Specification. `http://www.omg.org/spec/MOF/2.4.1/`, 08 2011.

[12] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *Eclipse Modeling Framework 2.0.* Addison-Wesley Professional, 2008.

[13] K. Wieland. *Conflict-tolerant Model Versioning.* PhD thesis, Vienna University of Technology, 2011.