Technology Support for Collaborative Inconsistency Management in Model Driven Engineering

Christian Bartelt
University of Clausthal
cbar@tu-clausthal.de

Björn Schindler
University of Clausthal
bschi@tu-clausthal.de

Abstract

Model driven engineering is one answer to increasing demands on software development and maintenance. Today’s software systems are often large, complex but also safety-critical and should be highly adaptable in life cycle. The efficient development of large and complex software systems needs a high degree of collaboration in the design and specification phases. Well-defined, (graphical) modeling languages provide therefore a matter of communication for software engineers. Further the distribution of development locations and the concurrency of work are typical in global software engineering projects. This kind of collaborative modeling needs reliable integration mechanisms for co-evolved models. However the syntactic and semantic correct (consistent) integration of concurrent evolved models is not satisfactorily supported by current tools. Especially the inconsistency resolution for merged architecture and design models is a communication-intensive, collaborative task. This paper proposes a technology support of automatic inconsistency analysis and visualization for distributed modelers whose synchronize their parallel work.

1. Introduction

Currently developed software systems are often large and/or highly dynamic. Among that, many software systems have to meet several demands as safety-critical reliability, high adaptability and maintainability at life cycle. These facts have huge impacts on the description techniques in the design and specification phases. A design or specification document should be intuitively understandable as a matter of communication in engineering team. But nevertheless the description should be precisely (formal/computable) formulated to enable a reliable verification. Model Driven Engineering (MDE) approaches are widely-discussed in the scientific community as one answer for the mentioned challenges. During the last years, the level of abstraction has been moving from source code to models that describe software systems on a higher level. As stated by Lin et al. [9], models are elevated to first class artifacts. This is reflected in many approaches currently used in industrial practice. MDE is a software engineering approach which focuses on models which based on modeling languages with well-defined (possibly formal) abstract syntax and semantics. Further they provide a graphical concrete syntax which semantics are intuitively and team-wide understandable or unified. That is one precondition for a collaborative team-modeling in the creative design phases. But today’s large software systems are often developed in global software engineering projects [3]. That kind of projects requires a high level of distributed, collaborative work. Therefore a lot of established configuration management systems are able to support the realization and integration phases on source code level. For the management of concurrent evolving text or source code documents, software configuration management systems (SCM) as CVS or Subversion [5,16] or collaboration development platforms like [7,11] are popular. But similar to source code, models are subject to a continuing evolution. In industrial practice a lot of versions, configurations or variants are created and account for evolution. The management of distributed and parallel, evolving models is an ambitious challenge. It can be divided in three aspects: 1. How is it possible to decompose a complex distributed model in versionable artifacts? 2. How is it possible to compute a difference or a merge of parallel evolved models respectively its versioned modeling artifacts? 3. How is it possible to preserve the syntactic and semantic consistency of automatically merged model versions? An answer for the first and second question is proposed in [2]. This paper explains a modeling support for the collaboration platform Jazz which provides the decomposition/composition of XMI-based models in versionable artifacts within an import/export mechanism. This mechanism enables the management (merge, tracing etc.) of modeling artifacts with Jazz-internal SCM-services. But a pure merge of two parallel evolved models is an incomplete integration because the consistency of merged models is not proven. Even if two parallel changed model
versions are correct regarding syntax and semantics, the merged result (which is computed by the both parallel change-traces) can be inconsistent. In these cases all inconsistencies must be resolved to finish a valid integration. A manual inconsistency resolving in large and complex model repositories is very time-consuming and error-prone. However this process can be supported by automated inconsistency analysis (retrieval for causing change actions or involved modelers etc.) and visualization. In general an automatic inconsistency resolving is not possible because the semantic intention of involved modelers is unknown.

The problem of model integration in a collaborative, parallel design process is known. As mentioned in [9] one way of concurrent design of models is the persisting of model data as structured text files (typically XML) and a management of these files using SCM systems as CVS or Subversion. An Example for such systems is 4everedit [10]. One disadvantage of this kind of systems is the absence of syntactical checking of the merge results.

Another approach is realized by the teamwork support in the popular CASE tool MagicDraw [12]. For consistency receiving it provides a pessimistic locking mechanism. But pessimistic locking mechanisms are unsatisfactory for modeling large models because they to strongly restrict concurrent work [4].

Some software prototypes which implement merge concepts for collaborative modeling are published [13,14]. However they do not provide a consistency solving support.

To sum up, there is not a technological support of optimistic model integration and consistency management in a distributed, collaborative modeling processes. In this paper a proposal for a tool support of collaborative inconsistency analyzing and visualization will be illustrated. The tool support was prototypically implemented as an extension of the Jazz modeling support [2] and realizes our published approach in [1,2].

2. Collaborative modeling within Jazz

Established configuration management (CM) tools for source code resources (CVS [5], SVN [15], or Clearcase [8]) enable an asynchronous collaboration of software engineers – (check-out – modify – check-in). Collaborative development platforms as Jazz integrate these technologies and provide additionally a team support for source code development (merging, change tracing, build management, test automation etc.). However they do not provide these services for models. In the SCM component of Jazz, the concept of branches serves for two purposes. The first one is the management of concurrently developed artifact versions (version management) and the second one the support of development organization (work item/task management). The aspect of task management will be focused is the following. As explained in [2] all concrete change operations in a modeling process are separated in so-called development tasks. There are two types of tasks – modification and integration tasks. Modification tasks (MT) contain functionally caused changes and integration tasks (IT) exclusively contain changes to resolve merge inconsistencies. An example modeling process is illustrated in Figure 1.

![Collaborative modeling process](image)

**Figure 1. Example task-based modeling process**

Each integration task is divided in two activities: model merge computation and inconsistency resolving. Figure 2 depicts an origin version \( V_0 \) and two concurrent developed versions \( V_A \) and \( V_B \). Further a temporal version \( V_M \) as a merge result is shown. This version must be checked regarding Consistency.

![Model integration](image)

**Figure 2. Steps of model integration**

If inconsistencies will be identified then they must be collaborative resolve by the involved modelers. All changes which are motivated by inconsistency resolving are encapsulated by a dedicated integration
task. The resulting model of each integration task is a consistent, integrated version \( VI \).

The following section illustrates the implemented collaborative modeling process with focus on integration and inconsistency resolving. Therefore a simple modeling scenario will be introduced.

3. Inconsistency management within model integration

The following section explains a sample modeling history for a illustration of integration tasks. Therefore a data model for a description of Graphical Primitives is changed in concurrent branches of a Jazz development line (Figure 3). The initial model which was developed in a modification task - Graph prims fundamentals - is depicted in Figure 4. It defines some shape types of graphical primitives and should be the basis for further concurrent development activities.

Figure 3. Task management within Jazz - new integration task

![Figure 3. Task management within Jazz - new integration task](image)

Figure 4. Initial example model

For example, one developer “Alice” opens a new modification task Upgrade structures (Figure 3) and adds a new generalization relation from Ellipse to Circle (Figure 3). Another developer “Bob” opens concurrently an own modification task Upgrade properties (Figure 3) and adds a new attribute upperLeftPt in the class Square and further he adds a generalization relation from Circle to Ellipse (Figure 6).

Figure 5. Alice's modifications

![Figure 5. Alice's modifications](image)
The depicted models were designed by using the Eclipse/EMF-based case tool Topcased [17] which was integrated in the Jazz TeamConcert client. The models are saved in the XMI [19] format. Furthermore it is possible to use other EMF-based modeling software to design models. Thereby unified modeling languages like UML are supported as well as domain specific languages (DSL) by the Jazz modeling extension because the implemented model management is independent of a specific meta-model.

For processing an integration of Alices’ and Bob’s modifications a new integration task (Figure 3) is created. After an instantiation of this integration task both models of Alice and Bob will be internally merged by the implemented modeling infrastructure. At first the integration infrastructure transforms the imported XMI-models in a meta-model independent representation [1]. This representation is a set of decoupled artifacts which can be merged by the optimistic merge algorithm of the Jazz platform.

Figure 7 depicts the exported result of the merge process. All changes Alice’s and Bob’s modification tasks were optimistically integrated. Note, both models (Figure 5, Figure 6) of both concurrent modification tasks were syntactical correct regarding the UML language specification but the automatically merged result model depicting in Figure 7 contains a serious inconsistency: Between Circle and Ellipse exists cyclic inheritance relations. But the UML 2.0 Superstructure forces acyclic inheritance relations. This constraint is formal described as OCL-expressions in the OMG specification and is implemented in the EMF-based meta-model of UML:

**Operations:**

Classifier::parents() : Set(Classifier);
parents = generalization.general

Classifier::allParents() : Set(Classifier);
allParents = self.parents() ->
union(self.parents()) ->
collect(p | p.allParents())

**Constraint:**

context Classifier

not self.allParents() -> includes(self)

The UML language specification contains a lot of further similar constraints. The following section explains a proposal for technology support of the identification, analyzing and visualization of inconsistencies to better a collaborative, manual resolving.
4. Collaborative inconsistency resolving

All model versions in the explained scenario have to be consistent to the UML specification. For consistency checking the Eclipse Modeling Project [6] provides a comprehensive UML 2.0 validation support. Furthermore the EMF-Framework enables a specification of any other (domain specific) metamodels (e.g. by OCL enriched ECore-models). Our implemented consistency management extension uses this generic validation functionality to identify inconsistencies in optimistically merged models. The consistency management plug-in is integrated in the Jazz client environment.

But what is a sophisticated inconsistency resolving support for the collaborative work of modelers? For this activity within an integration task three new consistency management views (Figure 8) were prototypical implemented as plugin. These views are the following:

- Inconsistency view
- Integrated model view
- Concurrent evolution view

The inconsistency view depicts a list of identified inconsistencies within an automatically merged model. Related to the introduced scenario two inconsistencies by cyclic inheritance relations between the classifiers Circle, Ellipse could be identified. The integrated model view shows the merged model as an EMF tree view. The displayed model is the same as in Figure 7. It is the merge result of the integration task Upgrade all. The concurrent evolution view displays three models: In the upper area the common origin model of the integrated, concurrent branches is shown (model of initial modification task Graph prims fundamentals). In the lower area the both concurrent model modifications of the origin model (results of modification tasks Upgrade structure/properties) are depicted. The integrated EMF Compare component [18] processes a three way diff operation of model versions. The required model versions are retrieved in the version history of the model repository on the Jazz server.

All three views are dynamically coupled. That means when a developer select an inconsistency item in the upper inconsistency view then all involved model elements will be marked in all other views. In
Figure 8 the second inconsistency item is selected in the inconsistency view. Because of that the concerned model element Circle in the merged model and in the origin model is dynamically marked. Especially Alice's and Bob's inconsistency-causing modifications are marked boldly in the split comparison window (insertion of an inheritance relation).

With help of the inconsistency identification and dynamic representation modelers are supported during analyzing of merge-related inconsistencies. So they can more rapidly find the modifications in concurrent branches which are causing for the identified syntactic inconsistencies.

5. Conclusion and further work

The presented technology support extents the Jazz modeling support [2] by services for inconsistency identification and analyzing. The whole implemented collaborative modeling extension includes now a mechanism for the merge of concurrently evolved model version and the core functionality of collaborative inconsistency management. It is necessary to resolve syntactical inconsistencies after an automatically merge of model versions. The collaborative modeling extension is independent of a specific modeling language. It can process UML models as well as any domain specific models.

In our further work we want to realize a more sophisticated illustration of an inconsistency analysis than the current front end (Figure 8). Therefore the tree views have to be replaced by graphical model views and the list of technical inconsistency descriptions must be translated in a more human readable text items. Figure 9 depicts our vision of this illustration. All necessary information to enable a generation of the enriched models which are depicted in Figure 9 can currently retrieved by our implemented prototype.

6. References


