Models in Conflict – Towards a Semantically Enhanced Version Control System for Models

Kerstin Altmanninger

Department of Telecooperation, Johannes Kepler University Linz, Austria
kerstin.altmanninger@jku.at

Abstract. For a widespread success of the model-driven paradigm, appropriate tools such as "Version Control Systems" (VCS) allowing for consistency maintenance between concurrently edited model versions are required to adequately support a model-based development process. Initial attempts for graph-based versioning of model artifacts are either tightly coupled to the modeling environment, not flexible with respect to the used modeling language or cannot interpret the model's semantics. On basis of those characteristics, the goal of the outlined thesis presented in this paper is to provide mechanisms to detect conflicting modifications between parallel edited model versions more accurately. By reducing falsely indicated conflicts and by finding additional semantic conflicts, the resolution process can be simplified by means of appropriate techniques for comparison, conflict detection, conflict resolution and merge.

1 Introduction

The shift from code-centric to model-centric software development places models as first class entities in "Model-driven Software Development" (MDSD) processes. A major prerequisite for the wide acceptance of MDSD are proper methods and tools which are available for traditional software development, such as build tools, test frameworks or "Version Control Systems" (VCS). Considering the latter, *optimistic* VCS which do not rely on pessimistic methods (such as locking) are particularly essential when the development process proceeds in parallel such that different developers concurrently modify a model, which may result in concurrent, potentially conflicting modifications. Hence, such conflicting modifications need to be resolved in terms of a model check-in process of the VCS by appropriate techniques for model *comparison*, *conflict detection*, *conflict resolution* and *merging*.

In case model developers use different modeling environments to edit their model artifacts and hence the employed modeling tools are not tightly coupled to the VCS, certain approaches that rely on tracking model modifications (e.g., operation-based mechanisms) are not applicable. Instead, a *loosely-coupled* VCS for model artifacts has to be provided which operates in a state-based manner. However, in the light of a growing number of "Domain Specific Languages"

(DSLs), a *flexible* approach, which can be adapted to the used modeling language, is desirable since most of the VCS for models (e.g., like the commercial tool IBM Rational Software Architect[1] and Odyssey-VCS [1]) solely concentrate on versioning UML models.

For dealing with concurrent modifications on models and specifically for the identification of conflicts, it is necessary not only to consider the logical structure of models in terms of a graph-based representation but also to "understand" the *model's semantics*. For example, concurrent modifications on a model may not result in an obvious conflict when syntactically different parts of the model (e.g., different model elements) were edited. Nevertheless, they may interfere with each other due to side effects [2], thus yielding an actual conflict, which, without considering the model's semantics, would remain hidden. Furthermore, certain conflicts which would be detected by a structural difference computation are not necessarily conflicts because in modeling languages often more than one possibility exists to model a specific case. E.g., in UML activity diagrams, decision nodes as well as conditional nodes are two equivalent ways to express alternative branches in a process, which could in fact result in a conflict if two developers edit a model concurrently by using such different but semantically equivalent modeling concepts. Valuable conflict reports, however, are essential for model developers in order to ensure the correctness of the merged version and consequently to avoid finally merged model artifacts which are not in the model developers intent. Therefore, in this paper, an optimistic, loosely coupled and flexible "**S**emantically enhanced **Mo**del **Ver**sion Control System" (SMoVer)[2] is laid out which is able to provide some "understanding" of the model's semantics in order to achieve accurate conflict reports by reducing falsely indicated conflicts and by finding additional semantic conflicts.

The remainder of this paper is structured as follows: Section 2 identifies the problems encountered by existing approaches. In Section 3, the research hypotheses are given and the goal of the thesis, presented in this paper, is laid out. In Section 4, the conceptional design of SMoVer is explained. The actual realization status of SMoVer is presented in Section 5 and a comparison to existing VCS is given in Section 6. Finally, Section 7 discusses further prospects beyond the scope of the outlined thesis and Section 8 gives a conclusion.

2 Problem Identification

The challenges emerging when realizing an optimistic, loosely coupled and flexible semantically enhanced VCS for model artifacts, for which an accurate conflict detection process has to be employed, span over the following issues.

Firstly, a check-in process which allows to detect and resolve conflicting modifications between parallel developed model versions [3,4] has to be provided. Secondly, to realize a loosely coupled VCS, the exchange of model artifacts between the VCS and the used modeling environments by model developers has to be

[1] http://www-306.ibm.com/software/awdtools/architect/swarchitect/
[2] http://smover.tk.uni-linz.ac.at/

enabled. Thirdly, techniques for a language independent and therefore flexible VCS and finally semantic enrichment techniques and strategies for the VCS's check-in process, for a more precise determination and resolution of conflicts, are needed.

Considering the check-in process in more detail, the succeeding challenges arise. Starting with the first phase of the check-in process, *model comparison* should not rely on text- or tree based approaches (like e.g., CVS[3], Subversion[4] and CoEd [5]) since they do not take the logical structure of models into account which is required for effective model comparison. Hence, existing graph-based approaches have to be employed. Furthermore, for the model comparison process techniques like the use of identifiers (IDs) for model elements or heuristics need to be considered in order to identify created, deleted and updated elements between model versions. In the *conflict detection* phase conflicts should not solely be identified due to the syntactical structure of models but additionally some "understanding" about the artifacts to be versioned should be provided to properly identify conflicts. This is already done by approaches [2,4] in the area of programming languages. They are, however, typically restricted to specific programming languages and therefore cannot immediately be reused in the realm of models. In addition, these approaches rely on formal semantics whereas existing modeling languages, such as UML, commonly do not exhibit a formal description of their semantics not least since being hard and costly to define [6]. Therefore, the *conflict detection* phase requires a more specific approach where semantics can be defined particularly for the purpose of detecting conflicts more precisely. *Conflict resolution*, however, commands for an appropriate identification of the reasons of conflicts, especially when going beyond just supporting syntactical conflict detection. Hence, conflicts need to be visualized and reported adequately to model developers. Model *merging*, finally, must produce a consistent new model which is based on the results of the previous phases and which can be facilitated by model transformations.

3 Research Hypotheses and Goal of the Approach

To tackle the identified challenges in order to achieve a merged model versions with the greatest possibility to be in all developers intents, the following hypotheses have been defined:

- *Model comparison* can be successfully achieved by applying existing graph-based comparison techniques and has to rely on 3-way comparison approaches comprising the concurrently edited model versions and their common ancestor. Therefore, comparison techniques must not solely rely on either using identifiers or heuristics and need to consider any possible change which can be undergone by a model element.
- *Conflict detection* can be conducted on top of the before calculated difference sets and can not rely on full formal specifications of the semantics underlying

[3] http://www.nongnu.org/cvs/
[4] http://subversion.tigris.org/

a model since only certain aspects are relevant. Therefore, conflict detection can benefit from the definition of semantics allowing to find conflicts more precisely i.e., by avoiding falsely indicated syntactic conflicts, by finding previously undiscovered semantic conflicts and by finding more precisely defined conflicts.
- *Conflict resolution* can be empowered by reasoning on the semantics of the conflicts detected. Furthermore, adequate visualization techniques have to be provided for model developers.

Hence, the goal of the thesis can be reflected in three areas:

- *Development* of *concepts and techniques* for establishing an optimistic, loosely coupled and flexible semantically enhanced VCS which enables finding and resolving conflicts between model versions more accurate by reducing falsely indicated conflicts and by detecting additional semantic conflicts.
- *Implementation* of *SMoVer* which incorporates the concepts and techniques established by using state of the art technologies and standards (cf. Subsection 5.2).
- *Evaluation* of the quality of the conflict detection and resolution process of SMoVer. Firstly, on basis of a comparison of the loosely coupled approach to other loosely coupled and tightly coupled VCSs for models and secondly, of the power of expressiveness of the technique for semantic enrichment introduced by this approach for different modeling languages.

4 SMoVer - Conceptual Design

In the light of the previously mentioned goals of the approach, SMoVer is proposed. In the following, the conceptual design of the system is explained.

Fig. 1 visualizes a common scenario in a VCS, where two model developers Sally & Harry create personal working copies of a model (V) out of the repository ❶. After they modified their personal working copies with their preferred modeling environment, both want to check-in their version later on to the repository. However, if Sally commits her changed model (V') to the repository first,

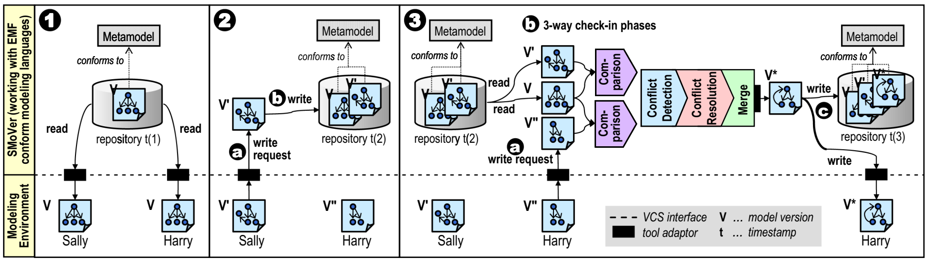

Fig. 1. Workflow in a loosely coupled and flexible VCS

the check-in process can proceed since the current revision in the repository is the direct ancestor of the incoming working copy (Fig. 1, ❷). Harry attempts to commit his changed model (V”) later whereas he has to apply a 3-way check-in process because the last revision in the repository is not the one he has checked-out previously (Fig. 1, ❸). This means, the two model versions of Sally and Harry have to be compared with respect to their common ancestor version in the repository, in order to ensure consistency between the parallel edited model versions. This comparison process, however, is based on a graph-based *structural difference* computation between the model versions. The actual comparison of model elements is based on an ID designated in the metamodel. Conflicts, however, may origin due to *creation*, *deletion* or *update* of model elements. By inspecting the structural features, namely the *attributes* and *references* of a model element, one can determine whether the model element as a whole has been updated. In particular four different *update strategies* to detect structural changes in a graph that are of interest for conflict detection are considered:

– **Attribute update (ATT):** The value of an attribute has been changed.
– **Reference update (REFS):** The set of referenced model elements has been changed. For example, new model elements have been created or deleted. Therefore the following possible combinations can be identified: Create-Create (CC), Create-Delete (CD), Delete-Create (DC), Delete-Delete (DD).
– **Role update (ROL):** A model element is referenced or de-referenced by another model element. Again, the four possible combinations of create and delete can be enumerated (CC, CD, DC, DD).
– **Referenced element update (REF):** A referenced model element has undergone an update (e.g., an attribute, reference or role update).

To make this process of detecting conflicts explicit, the following OCL expressions define the derivation of the corresponding conflict sets. In more detail, the conflict set (Con) contains all conflicting model elements and is a union of three further sets that represent update-update (UpdCon), create-create (CrCon) and update-delete (DelCon) conflicts accordingly. The *isUpdated* function determines updated model elements and the function *areNotEqual* checks for the equality (as opposed to the identity) of two model elements.

To represent the model's semantics, so-called *semantic view definitions* are introduced in order to make certain semantic aspects explicit. To start with, the

```
Creates'=(V'-V)
Creates"=(V"-V)
Updates'=V->select(e|e.isUpdated(V,V')
Updates"=V->select(e|e.isUpdated(V,V")
Deletes'=(V-V')
Deletes"=(V-V")

CrCon =Creates'->intersection(Creates")->select(e|e.areNotEqual(V',V"))
UpdCon=Updates'->intersection(Updates")->select(e|e.areNotEqual(V',V"))
DelCon=(Updates'->intersection(Deletes"))->union(Updates"->intersection(Deletes'))

Con=UpdCon->union(CrCon->union(DelCon))
```

Listing 1.1. OCL constraints for the determination of conflict sets

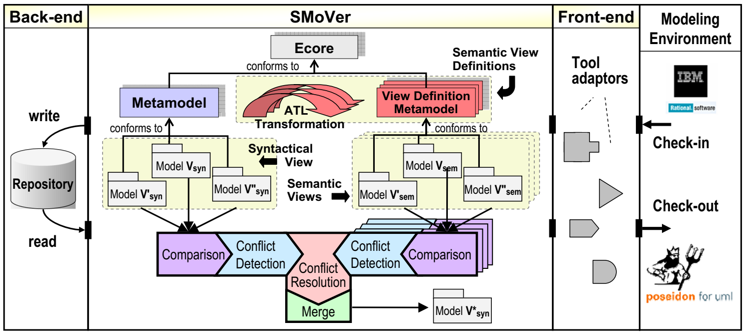

Fig. 2. Conceptual design of SMoVer

basis of the approach is the metamodel which describes the syntax of the models to be versioned. Additionally, to be able to provide semantic conflict detection, a *metamodel representing a certain view* of interest has to be defined. On basis of those metamodels, a *transformation* can be specified such that rules of a model transformation relate the elements of the metamodel (abstract syntax) to which the original model conforms to and the elements of the metamodel representing the definition of the view of interest , the so-called *semantic view.* As a consequence of the transformation realizing a semantic mapping, conflict detection can be carried out on both, model and semantic view (cf. Fig. 2). Conflicts that are determined purely upon the comparison of model versions are *syntactic conflicts* whereas a *semantic conflict* is detected between the representations of the model versions in a semantic view. The actual detection of conflicts in both the original model and the view works analogous to the graph-based detection of structural conflicts.

Compared to the definitions of semantics for programming languages [7] the translational approach, by means of semantic view definitions, is similar to a *translational semantics* specifications. In a translational approach, which can be considered as a special case of denotational semantics, constructs of one language map onto constructs of another, usually simpler language such as machine instructions. Similarly, in SMoVer, a translation into a semantic view that defines a certain facet of interest is proposed for the purpose of conflict detection.

In the following example (cf. Fig. 3) Sally & Harry are working concurrently on a WSBPEL [8] model. Therefore, a language developer previously defined the metamodel and the according IDs and update strategies in SMoVer. Additionally (s)he also set up a semantic view definition which purpose is to detect static semantic conflicts due to addition of “Activities” in a “Sequence” on the same position whereby the model versions cannot be merged because it is not clear which “Activity” comes first. This conflict could also be detected in the syntax if the update strategy `REFS:C` is considered but then all concurrent insert operations of “Activities” in a “Sequence” would be reported as a conflict whereas probably most of them are no actual conflicts. Therefore the language developer created a view of interest for this circumstance which allows to find the actual

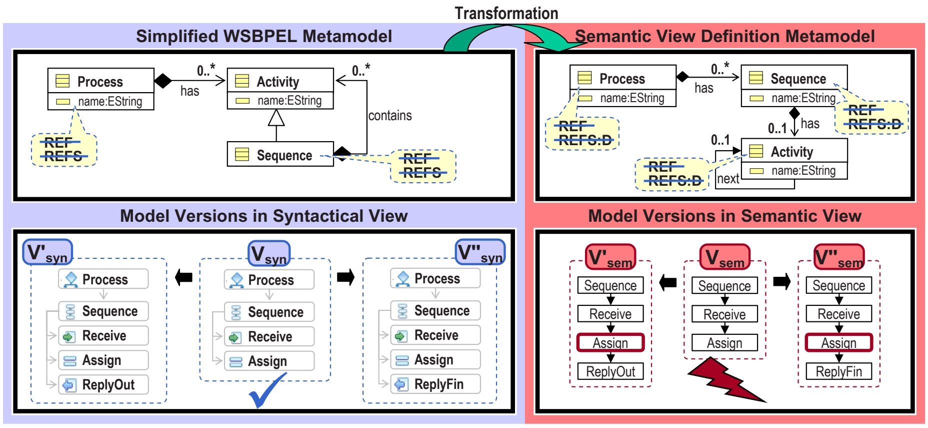

Fig. 3. Conflict detection example

conflicts more accurately than in the syntax. Hence, as shown in Fig. 3, Sally & Harry both insert an "Activity" on the end of the "Sequence" whereas a static semantic conflict arises (`Con={Assign(REFS:CC)}`) in the semantic view by applying the before mentioned conflict detection algorithm.

However, various view definition possibilities exist for which a categorization is proposed according to three semantic aspects important for versioning, namely: *Equivalent concepts*, *static semantics* and *behavioral semantics*. Through the definition of "equivalent concepts", which allow the expression of identical meaning in different ways to achieve convenient modeling and readability, falsely indicated conflicts can be avoided. Through the definition of static semantics, which describe static characteristics of a model (like inheritance, constraints [9], or relationships), additional "static semantic conflicts" can be detected. In contrast, through the definition of behavioral semantics, with which the ability arises to detect concurrent changes of the behavior of a model artifact (e.g., by using dependency graphs [2,4,10] or by transforming the model in a different modeling language [11]), additional "behavioral semantic conflicts" emerge.

5 SMoVer - Realization Status

In the following subsections, the realization status of the aforementioned goals are laid out to evince the stated hypotheses.

5.1 Concepts and Techniques

Considering the main purpose of SMoVer in providing accurate conflict reports and the previously defined characteristics of the system, the following concepts and techniques for the check-in phases can be identified.

In a loosely coupled context, the implementation of the algorithm for the *comparison phase* should be a metamodel independent approach to derive model differences. Therefore, the decision has been made to compare model elements using IDs for each element in order to detect model modifications [12]. The representation of changes is grouped in creation, deletion, and changes like in related work [13]. Moreover, the proposed comparison phase considers a detailed categorization of update strategies (cf. Section 4) with which it is possible to fine-grain the kind of modification and therefore also to provide a more detailed conflict report in the succeeding phase of the check-in process.

As mentioned in the previous section and in preceding works [12,14] the *conflict detection phase* is realized by a determination of conflict sets between three versions of a model artifact. Therefore the first work in this context [12] describes the techniques and strategies needed for conflict detection. The second one [14] gives an overview on how to define and work with multiple semantic view definitions exemplified by a specific modeling language and categorizes them in three semantic aspects for which semantic view definitions can be utilized. Additionally it is demonstrated that the proposed conflict detection process allows fine-tuning of the conflicts reported and an increase in effectiveness by reducing falsely indicated syntactic conflicts, by detecting undiscovered semantic conflicts and by more precisely defined semantic conflicts than reported in the syntax. Therefore, from a purely conceptual point of view, the activities needed to be covered by this phase are completed.

For the *conflict resolution phase*, two main conceptual decisions have to be made about the following two challenges. Firstly, how the semantic conflicts can be efficiently traced back from the semantic view and being reported in the syntactical representation and secondly, how the conflicts can be visualized in the VCS to fully support the model developer during the resolution process. Regarding the second activity "visualization", it has to be investigated if the VCS can make use of the concrete syntax of models during this phase and how this concrete syntax can be preserved in the system for specific modeling environments.

5.2 Implementation

In order to define the abstract syntax of a modeling language and a desired semantic view definition, a metamodeling architecture is needed. The "Eclipse Modeling Framework" (EMF)[5] provides Ecore, which is a simplified version of the OMG's metamodeling standard "Meta Object Facility" (MOF) that constitutes the M3 layer, has been chosen. EMF covers persistence support with an XMI serialization mechanism and a reflective API for manipulating EMF models. The creation of a semantic view from a model artifact is realized through the "Atlas Transformation Language" (ATL) [15], which is a QVT-like model-to-model transformation language. Accordingly, the top of Fig. 2 shows the usage of this metamodeling stack in the context of the implementation architecture.

The comparison of the concurrently edited model versions with their common ancestor version is carried out on a generic graph representation of the respective models and views. For this purpose, the EMF reference implementation of

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

"Service Data Objects" (SDO)[6] is used. SDO is a general framework to realize standardized access to potentially heterogeneous data sources such as databases, XML files or models serialized in XMI. SDO allows to create "datagraphs" from EMF models, which are convenient for comparison purposes as SDO's mechanism to establish the difference between two graphs. These so called "change summaries" are used in SMoVer to store modifications between versions, which are then used by the actual conflict detection mechanism. Hence, the underlying algorithm implements the comparison strategies mentioned in Section 4 and establishes the relevant sets of conflicting elements. Both, the comparison and merge component of the implementation are therefore carried out with Java on top of SDO, EMF and ATL for model transformations in the semantic view(s) and to produce a consistent merged model version.

Summing up, currently not implemented because no concepts and techniques have been defined yet are the tracing back of the computed semantic conflicts in the syntax and visualization techniques which will be focused on in the future.

5.3 Evaluation

For evaluating the feasibility of the approach it is planned to apply a series of case studies firstly, on the *effectiveness of the loosely coupled semantically enhanced check-in phases* applicable on various modeling languages and secondly, on the *power of expressiveness of semantic view definitions* in order to be able to identify semantic conflicts.

To start with, for the loosely coupled check-in phases it will be investigated how effective they are compared to other loosely coupled and tightly coupled VCS for models (cf. Section 6). In a first step, the evaluation is conducted on basis of a syntactic comparison and conflict detection techniques and in a second step with the help of semantic view definitions in order to derive semantic conflicts for exploring the approache's limitations utilized on a specific modeling language. On basis of this comparison between SMoVer and other loosely and tightly coupled VCS, a comprehensive statement about the effectiveness of the approach for a specific modeling language can be made. Nevertheless, an evaluation conducted on one specific modeling language is not sufficient. Because different modeling languages have different power of expressiveness, several modeling languages have to be analyzed for view definition possibilities. The knowledge derived from this evaluation is an overview for which modeling languages this semantically enhanced approach is more (eventually UML) or less (e.g., some DSLs) valuable.

6 Related Work

The most closely related graph-based approach considering model versioning which works in a state-based manner and provides semantic awareness during the conflict detection process is laid out by *Cicchetti et al.* [16]. They propose

[6] http://www.eclipse.org/modeling/emf/?project=sdo

to leverage conflict detection and resolution by adopting design-oriented descriptions endowed with custom conflict specifications. Hence, several conflicting situations, which can not be captured by a priori structural conflict detection mechanism can be specified that they refer to as "domain specific conflicts". The developers, however, are forced to enumerate all wrong cases in form of weaving models, which negatively affects the usability and scalability of the approach. Therefore, in the work of *Cicchetti et al.*, each modification, which is not allowed to preserve a design pattern and the design pattern itself have to be specified in a weaving pattern (as they exemplified for the singleton design pattern). Anyway, the approach of *Cicchetti et al.* focuses on the detection of previously undiscovered conflicts in terms of domain specific conflicts only, whereas behavioral semantic conflicts and the detection of previously falsely indicated conflicts as provided by SMoVer are not considered. In addition, so far, the work of *Cicchetti et al.* is solely applicable on UML models as opposed to SMoVer which is flexible by being able to deal with all kind of Ecore-based modeling languages.

Another loosely coupled, semantically enhanced approach called *SemVersion* is presented by *Völkel* [17], which is based on RDF, proposing the separation of language specific features (e.g., semantic difference) from general features (e.g., structural difference or branch and merge). To perform the semantic difference, the semantics of the used ontology language are taken into account. Therefore, assuming using an RDF Schema as the ontology language and two versions (A and B) of an RDFS ontology, *SemVersion* uses RDF Schema entailment on model A and B and infers all possible triples. Now, a structural difference on A and B can be calculated in order to obtain the semantic difference. The approach of *Völkel*, however, does not consider behavioral semantic conflicts and is not flexible to operate on any modeling language.

VCSs which detect conflicts solely due to structural comparison of concurrent edited model versions without incorporating semantics are numerous [18,1,19]. To start with, *Alanen & Porres* [18] provide state-based difference calculation and merging algorithms with which the functionality of a VCS for MOF-based models can be realized. This approach is therefore not tightly coupled to a specific modeling environment and enables developers the parallel editing of model artifacts with their preferred tooling. *Oliveira et al.* [1] presents a graph-based VCS for versioning UML models called *Odyssey-CVS*, aiming to support different UML-based CASE tools in evolving their artifacts. However, *Oliveira et al.* is not flexible in the used modeling language because it can only be applied to UML models. Similarly the tightly coupled approach of *Oda & Saeki* [19] and the commercial tool *IBM Rational Software Architect* are also limited to UML models by the *IBM Rational Software Architect* and additionally ER models by *Oda & Saeki*.

7 Future Challenges

Future challenges are numerous but since current researches in this area are still in the beginning not encountered in context of the outlined thesis presented in this paper. Firstly, as the proposed VCS is loosely coupled to the modeling environment XMI is used to exchange models. Because of the fact that different modeling environments export different XMI representations, so-called tool adaptors for a common XMI representation are essential. Secondly, in a longer prospect, a fully functional semantically enhanced VCS also needs to support versioning capabilities for huge model artifacts which have associations to other models in the same context. Therefore functionalities have to be provided to support versioning beyond one artifact. Thirdly, as metamodels evolve over time, for industrial settings the defined metamodels in the VCS also need to be defined as being able to be versioned. Hence, a smoothly technique has to be invented with which this can be realized considering that all models and according transformations have to be adapted to the new metamodel version as well. The migration of instances is in fact a well known problem from the area of schema evolution [20] e.g., in the field of database systems. Fourthly, an important prerequisite of a VCS for models are visualization techniques needed for the conflict resolution process in order to provide the developer with an adequate overview on the model elements. The challenges which has to be dealt with is how to work with the concrete syntaxes of the different model environments and how those data can be versioned in order to satisfy the the demands of developers.

8 Conclusion

In this paper an optimistic, loosely coupled and flexible VCS called SMoVer, which is extensible to incorporate the semantics needed for the conflict detection process between model versions, is presented. By means of transforming a model into a semantic view, conflicts due to equivalent concepts can be eliminated and hidden static and behavioral semantic aspects can be explicated. Therefore, various semantic view definitions can be established, consequently all of them covering a different semantic aspect. The conflict detection algorithm is applicable on the syntax and all semantic views in the same way. Hence, the joint use of semantic view definitions expressing certain semantic aspects of a modeling language and the employment of graph-based comparison techniques on models and views allows for an accurate conflict detection between versions of model artifacts. This is archived by reducing falsely indicated conflicts and by finding additional semantic conflicts.

References

1. Oliveira, H., Murta, L., Werner, C.: Odyssey-VCS: a flexible version control system for UML model elements. In: Proc. of the 12th Int. Workshop on Software Configuration Management (SCM), ACM Press, New York (2005)
2. Thione, G.L., Perry, D.E.: Parallel changes: Detecting semantic interferences. In: Proc. of the 29th Annual Int. Computer Software and Applications Conf (COMPSAC), vol. 1, pp. 47–56. IEEE Computer Society, Los Alamitos (2005)
3. Westfechtel, B.: Structure-oriented merging of revisions of software. In: SCM, pp. 68–79 (1991)

4. Mens, T.: A state-of-the-art survey on software merging. IEEE Trans. Software Eng. 28(5), 449–462 (2002)
5. Bendix, L., Larsen, P.N., Nielsen, A.I., Petersen, J.L.S.: CoEd – a tool for versioning of hierarchical documents. In: Magnusson, B. (ed.) ECOOP 1998 and SCM 1998. LNCS, vol. 1439, Springer, Heidelberg (1998)
6. Harel, D., Rumpe, B.: Meaningful modeling: What's the semantics of "semantics"? Computer 37(10), 64–72 (2004)
7. Slonneger, K., Kurtz, B.: Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach. Addison-Wesley Longman Publishing Co., Inc., Boston (1995)
8. OASIS: Web services business process execution language (WSBPEL) standard version 2.0 (April 2007), http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf
9. Object Management Group (OMG): OCL 2.0 specification (June 2005)
10. Shao, D., Khurshid, S., Perry, D.E.: Evaluation of semantic interference detection in parallel changes: an exploratory experiment. In: Proc. of the 23rd IEEE Int. Conf. on Software Maintenance, Paris, France (2007)
11. Ryndina, K., Küster, J.M., Gall, H.: Consistency of business process models and object life cycles. In: Proc. of the 1st Workshop on Quality in Modeling (2006)
12. Altmanninger, K., Bergmayr, A., Kotsis, G., Reiter, T., Schwinger, W.: Models in conflict – detection of semantic conflicts in model-based development. In: Proc. of the 3rd Int. Workshop on Model-Driven Enterprise Information Systems (MDEIS), pp. 29–40. INSTICC Press (2007)
13. Toulmé, A.: Presentation of EMF compare utility. In: Eclipse Modeling Symposium (2006)
14. Altmanninger, K., Bergmayr, A., Kotsis, G., Schwinger, W.: Semantically enhanced conflict detection between model versions in SMoVer by example. In: Int. Workshop on Semantic-Based Software Development in conjunction with the Int. Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) (2007)
15. Allilaire, F., Bézivin, J., Jouault, F., Kurtev, I.: ATL – eclipse support for model transformation. In: Proc. of the Eclipse Technology eXchange Workshop (eTX) of the European Conf. on Object-Oriented Programming (ECOOP) (2006)
16. Cicchetti, A., Rossini, A.: Weaving models in conflict detection specifications. In: Proc. of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, pp. 1035–1036. ACM Press, New York (2007)
17. Völkel, M.: D2.3.3.v2 SemVersion – versioning RDF and ontologies (2006), http://www.aifb.uni-karlsruhe.de/Publikationen/showPublikation?publ_id=1163
18. Alanen, M., Porres, I.: Version control of software models. In: Yang, H. (ed.) Advances in UML and XML-Based Software Evolution, Idea Group Publishing (2005)
19. Oda, T., Saeki, M.: Generative technique of version control systems for software diagrams. In: Proc. of the 21st IEEE Int. Conf. on Software Maintenance (2005)
20. Roddick, J.F., de Vries, D.: Reduce, reuse, recycle: Practical approaches to schema integration, evolution and versioning. In: Roddick, J.F., Benjamins, V.R., Si-said Cherfi, S., Chiang, R., Claramunt, C., Elmasri, R.A., Grandi, F., Han, H., Hepp, M., Lytras, M., Mišić, V.B., Poels, G., Song, I.-Y., Trujillo, J., Vangenot, C. (eds.) ER Workshops 2006. LNCS, vol. 4231, pp. 209–216. Springer, Heidelberg (2006)