Semantically Enhanced Conflict Detection between Model Versions in SMoVer by Example[⋆]

Kerstin Altmanninger, Alexander Bergmayr, Wieland Schwinger, and Gabriele Kotsis

Department of Telecooperation (TK)
Johannes Kepler University Linz, Austria
[kerstin|alexander|wieland|gabriele]@tk.uni-linz.ac.at

Abstract. For a widespread success of the model-driven paradigm, appropriate tools such as Version Control Systems allowing for consistency maintenance between concurrently edited model versions are required to adequately support a model-based development process. First attempts for graph-based versioning of model artifacts, however, perform conflict detection mainly on basis of the models' syntax without exploiting the models' semantics, are limited to specific modeling languages or are only able to compute a portion of potential semantic conflicts. In this paper, a flexible approach for semantically enhanced conflict detection is presented. By this approach, certain aspects of a modeling language's semantics, which are important for the conflict detection process, are explicated on the basis of view definitions, namely equivalent concepts, static semantics and behavioral semantics. It is shown how view definitions can be established for those three semantic aspects by means of Web Services Business Process Execution Language (WSBPEL) examples. Additionally it is demonstrated that the proposed conflict detection process allows to fine-tune the conflicts reported and to increase the effectiveness by reducing falsely indicated syntactic conflicts and by detecting undiscovered semantic conflicts.

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, VCS 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. Such conflicting modifications need to be resolved by appropriate techniques for model comparison, conflict detection, conflict resolution and merging.

⋆ This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and FFG under grant FIT-IT-810806.

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 [1], thus yielding an actual conflict, which, without considering the model's semantics, would remain hidden. E.g., if two developers concurrently edit different dataflows in a model on which a third one depends. Furthermore, certain conflicts which would be detected by a structural difference computation are not necessarily conflicts when considering the models' semantics because often more than one possibility exists in modeling languages to model a specific circumstance. 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 different, semantically equivalent modeling concepts.

Some approaches have already emerged providing some "understanding" about the artifacts to be versioned [1, 2] considering the field of programming languages. They, however, are 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 [3]. Furthermore, in the light of a growing number of domain specific languages a flexible approach is desirable. Hence, in previous work [4, 5] we presented a VCS called SMoVer[1] (**S**emantically enhanced **Mo**del **Ver**sion Control System), which is able to deal with semantic conflicts in the conflict detection process by defining update strategies and creating definitions of views of interest. In addition SMoVer is flexible to operate on any EMF[2]-based modeling language and does not rely on editing operations of concurrently modified model versions during the conflict detection process. The focus of this paper is to lay out the challenges for view definitions and according update strategies to enable semantically enhanced conflict detection.

The remainder of this paper is structured as follows: Section 2 shows how our approach makes use of a modeling language's semantics in the conflict detection process and lays out a characterization of semantic aspects which are important for view definitions. In Section 3 it is explained, by means of the "Web Services Business Process Execution Language" (WSBPEL) [6] and three semantic view definitions, in which way SMoVer can find conflicts with higher accuracy due to the fact that *falsely indicated conflicts are avoided* and *previously undiscovered ones are detected*. Section 4 discusses related work concerning conflict detection mechanisms in VCSs for models and finally Section 5 gives a conclusion and an overview of further prospects.

[1] http://smover.tk.uni-linz.ac.at/
[2] http://www.eclipse.org/modeling/emf/

2 Semantically Enhanced Conflict Detection

In the following a common scenario in a VCS is considered, where two developers A and B create personal working copies of a model V out of the repository and both want to check-in their version later back to the repository. However, if developer A commits her changed model V' to the repository first, the check-in process can proceed since the current revision V in the repository is the direct ancestor of the incoming working copy. The second developer B 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. This means, the two model versions V' and V" have to be compared with respect to their common ancestor version V, in order to ensure consistency between the parallel edited model versions. This comparison process, however, is based on a *structural difference* computation between the model versions. The interpretation of the resulting structural differences then yield to the identification of conflicts. To make explicit this process of the computation of conflicts between concurrently edited versions (V' and V") of a common model artifact (V), the following OCL expressions define the derivation of the 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. Whereas the *isUpdated* function determines updated model elements and the function *areNotEqual* checks for the equality (as opposed to the identity) of two model elements.

```
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.

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.
- Referenced element update (REF): A referenced model element has undergone an update.
- Reference update (REFS): The set of referenced model elements has been changed. Model elements have been either created or deleted by model developers whereas the following 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 combination of create and delete can be enumerated (CC, CD, DC, DD).

If all update strategies are regarded conjunctively for the identification of structural differences, this may lead to unintended side effects. In such a case this process would report any structural changes, due to parallel editing, between model versions and therefore a huge amount of conflicts except particular ones of static and behavioral semantics. Hence, disabling update-strategies on specific model elements may restrict the reported conflicts to those which are of interest for the developer who performed the check-in process. E.g., no conflict should be reported due to a REF update of a package element for which two developers updated/added/deleted contained elements. Thus, the *setting of* those *strategies* on the elements of the modeling language in which the models should be versioned is an essential task defined in advance of the check-in procedure and can be considered as the first step in providing semantically enhanced conflict detection. The second, and more powerful step is the *construction of semantic view definitions of interest.*

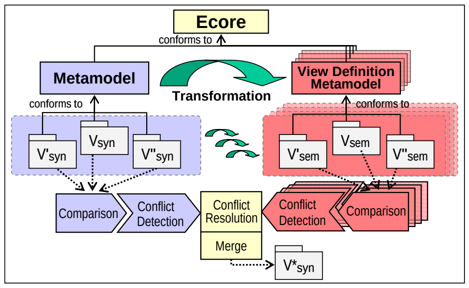

Fig. 1. Phases of a 3-way check-in process in SMoVer.

Such a semantic view definition consists of two parts, namely a *view definition metamodel* and a corresponding *model transformation* (cf. Fig. 1). The former, which defines the abstract syntax can either be represented through a *subset* of the source metamodel (syntactical level), a *domain specific view definition* metamodel (like e.g., a metamodel of a *dependency graph*) or a metamodel of a *different modeling language*. A semantic mapping between the source and the target metamodel (metamodel of the semantic view definition) is defined by a model transformation. Therefore, this approach is similar to the concept of translational semantics by [7], which maps the constructs of one language onto constructs of another, usually simpler language such as machine instructions. The output of such a transformation is another model which conforms to the metamodel representing the semantic view definition of interest. As a consequence of the transformation realizing a semantic mapping, conflict detection can be carried out now on both, model versions in the syntax (V_{syn}, V'_{syn} and V''_{syn}) and semantic views (V_{sem}, V'_{sem} and V''_{sem}), by means of a structural comparison. Therefore the conflicts determined purely upon the comparison of three versions of a model is called "syntactic conflict" whereas a "semantic conflict" is a conflict that is detected between model versions which have been transformed in a semantic view.

Anyway, various view definition possibilities exist for which a categorization is proposed according to three main semantic aspects important for versioning, namely: *Equivalent concepts*, *static semantics* and *behavioral semantics*.

The first category, "equivalent concepts", tackles apparent conflicts resulting from modeling diversities in that most modeling languages offer concepts allowing the expression of identical meaning in different ways to achieve convenient modeling and readability. Thus, through such modeling diversities, possible *falsely indicated conflicts* may arise. Besides the problem of equivalent concepts, the evolution of a concrete model may give rise to semantic inconsistencies between versions of a model artifact, which may lead to "static semantic" or "behavioral semantic conflicts". Static semantic conflicts may emerge through a violation of static characteristics (like e.g., inheritance, constraints [8], or relationships). In contrast a behavioral semantic conflict results due to concurrently editing of a model whereas both developer modified elements of the model which convey behavior and influences each other. For the detection of such conflicts techniques such as denotational semantics, program slicing and dependence graphs are proven to be useful in software development [2, 1, 9]. Hence, for conflict detection between model artifacts we can utilize some of those techniques e.g., by using a dependency graph as a view definition metamodel and a model transformation, which defines the model slices to be transformed. However, another option to detect behavioral semantic conflicts between model versions is a transformation of the models in a different modeling language. For example, model versions defining a business process can be transformed to models which explicates an object life cycle [10], which consequently provide a different view on the behavior of the model artifacts. Summing up, the detected static or behavioral semantic conflicts can be either *more accurate* ones than in the syntax (cf. subsection 3.2) or *previously undiscovered* ones (cf. subsection 3.3).

3 Conflict Detection by Example

In the following subsections the process of semantically enhanced conflict detection for each of the previously mentioned semantic aspects (equivalent concepts, static & behavioral semantics) is exemplified by means of "Web Services Business Process Execution Language" (WSBPEL) [6] examples.

Before inspecting the defined view definitions (cf. Fig. 2) and the related examples in detail the WSBPEL metamodel is going to be observed in the following. Hence, Fig. 2 visualizes the metamodel of the WSBPEL modeling language in the center for which a possible setting of the update strategies, as the first step for semantic enrichment needed for conflict detection, has been defined. E.g., the attribute comment in the element *NamedElement* is neglected as conflicting value update of an attribute (ATT) because it does not influence the semantics of the model. Considering the REF update, for WSBPEL it is not essential to

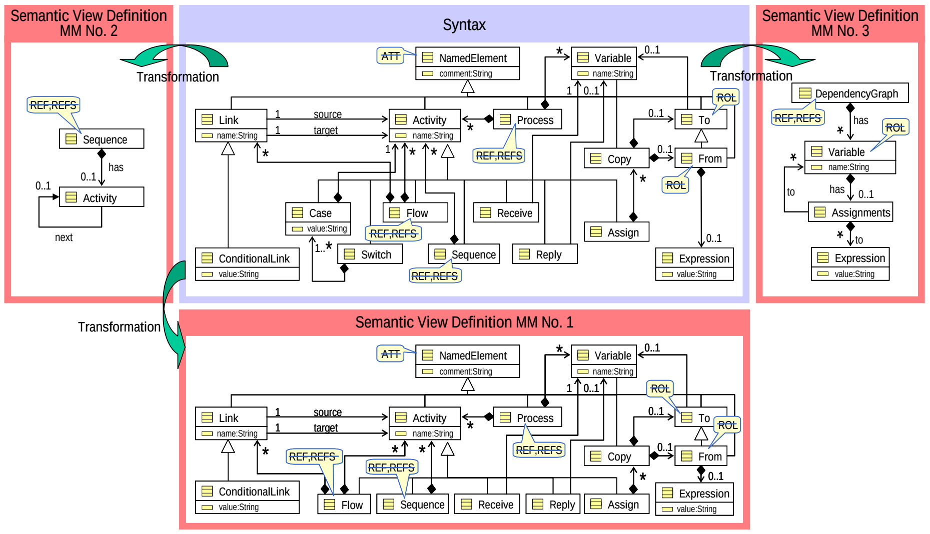

Fig. 2. Subset of a WSBPEL metamodel and associated semantic view definitions.

report an conflict in e.g., a *Process*, *Sequence* or *Flow* element because otherwise those elements would always lead to a conflict if concurrent modifications, which do not affect each other, have been performed in the same *Process*, *Sequence* or *Flow*. Similarly it is not of interest if two developer created or deleted *Activities* from/to a *Flow/Sequence* (REFS update) except in one case if two developers added an *Activity* to the same position in a *Sequence*. To restrict the amount of conflicts originally reported in the syntax therefore a semantic view definition has been established (cf. No.2 in Fig. 2). Finally, regarding ROL updates, they can be neglected for the computation of *To* and *From* elements because they cannot exist without the element *Copy* and in turn can only be referenced by a single *Copy* element.

3.1 Equivalent Concepts

To reduce the amount of previously falsely indicated syntactical conflicts raised by structural modifications, which do not actually change the meaning of a model, a semantic view definition can be established (cf. No.1 in Fig. 2).

Considering the example as shown in Fig. 3 it describes two *Activities* (Assign1, Assign2) referenced by *ConditionalLinks* (CondLink1, CondLink2) defining preconditions for the execution of an activity. Developer A has changed her working copy by adapting the condition in CondLink1 and Developer B has replaced the concept of *ConditionalLinks* by a *Switch* with two *Cases*. Regarding the modified working copies, a conventional conflict detection process would report a conflict during the check-in process, since CondLink1 has been updated by developer A and deleted by developer B. Domain experts would determine

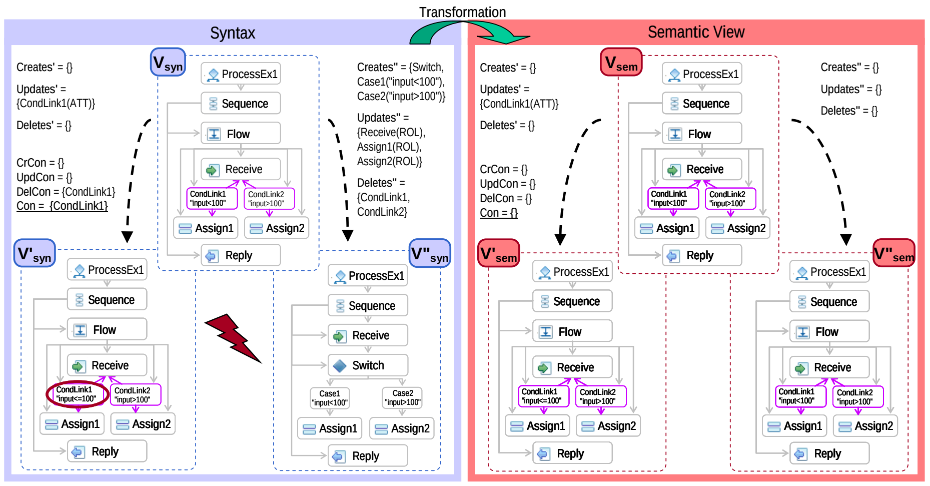

Fig. 3. Syntactic conflict which results due to semantic equivalent concepts.

the parallel modified model versions as consistent to each other after inspecting their semantics. Hence, a technique is essential with which such semantically equivalent concepts can be detected and reported to developers appropriately.

Therefore, a semantic view definition (cf. No.1 in Fig. 2), constituting a subset of the WSBPEL metamodel is utilized. Notice, the concept of *Switch-Case* has been excluded to reduce the modeling diversities of the WSBPEL language, since each *Switch-Case* statement can also be expressed through *ConditionalLinks*. Exactly the above mentioned fact is covered by the semantic mapping, in that each *Switch-Case* become transformed into a *Flow* with *ConditionalLinks* conveying the same meaning. As a consequence, after a transformation of the model versions from the syntax in the semantic view, the conflict detection process can be carried out in the semantic view again. Due to the fact that no semantic conflict can be found, a previously falsely indicated syntactic conflict has been avoided. Notice, this gives rise to a major benefit that employing semantic view definitions may be utilized to reduce the amount of reported conflicts to a developer, which results in a more effective conflict detection process.

Reflecting the above explicated example, a syntactic but no semantic conflict is reported, the knowledge gained conveys that the different model versions are consistent to each other. Imagine an adapted situation that developer B modified the Case1 element which is transformed into a CondLink1 element in the semantic view, a semantic conflict would be reported as well to the syntactic one. In case, the semantic conflict detection would convey the information about the origin of the semantic conflict in the equivalent concept. Therefore, with the help of semantic view definitions for equivalent concepts, semantic conflicts in equivalent concepts can be detected and appropriately reported to model developers.

3.2 Static Semantic Conflicts

While all parallel changes, performed by developers, are intended to affect a model semantically, it is clearly the case that many changes have unintended effects caused by either static or behavioral semantic aspects. Therefore, in this subsection the need for static semantic view definitions avoiding e.g., the violation of language constraints, is explained.

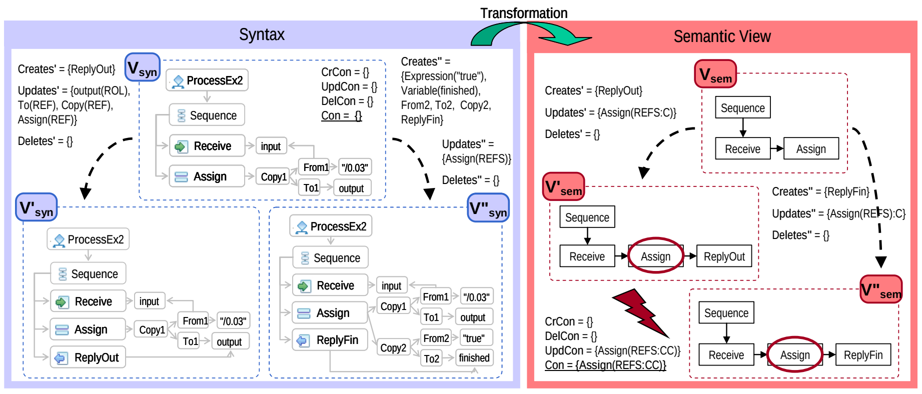
Transformation
Syntax
Semantic View
Creates' = {ReplyOut}
Updates' = {output(ROL), To(REF), Copy(REF), Assign(REF)}
Deletes' = {}
Creates'' = {Expression("true"), Variable(finished), From2, To2, Copy2, ReplyFin}
Updates'' = {Assign(REFS)}
Deletes'' = {}
CrCon = {}
UpdCon = {}
DelCon = {}
Con = {}
Creates' = {ReplyOut}
Updates' = {Assign(REFS:C)}
Deletes' = {}
Creates'' = {ReplyFin}
Updates'' = {Assign(REFS):C}
Deletes'' = {}
CrCon = {}
DelCon = {}
UpdCon = {Assign(REFS:CC)}
Con = {Assign(REFS:CC)}

Fig. 4. Detection of a static semantic conflict.

As shown in Fig. 4, a WSBPEL model is given describing a *Sequence* of *Activities* which is subject to an order. Hence, in case two developers concurrently insert the *Activities* ReplyOut and ReplyFin at the same position in the *Sequence* gives rise to a conflict due to the fact that a decision is required about the order of the created *Activities* in the merged version.

However, the above mentioned specific case of a REFS:CC update can also be detected in the syntax by applying the REFS:CC strategy but using a semantic view definition is more powerful. The reason for that is twofold. Firstly, by simply using the REFS:CC strategy for the conflict detection in the syntax many falsely indicated conflicts may be detected as well (e.g., creation of *Activities* at different positions) and secondly, the conflicting model element can not be identified as precise as with a semantic view definition. Using a REFS:CC strategy in the syntax does not provide to make available the model element which actually causes the conflict, since the relationship of activities is merely expressed implicit through the *Sequence* element.

Therefore, to make explicit the relationship of *Activities* a semantic view definition is defined as shown in No.2 in Fig. 2, describing an ordering relation of *Activities*. Looking at the model transformation, realizing the semantic mapping, a *Sequence* of *Activities* become mapped to a connected chain of *Activities* each *Activity* pointing to its successor, whereas the *Sequence* element itself becomes mapped to a *Sequence* element in the semantic view metamodel, pointing to the first *Activity* in the chain. Coming back to Fig. 4 a static semantic conflict is detected in the semantic view due to the reference property *next* of the Assign *Activity*. Summing up, this example for a static semantic view definition strongly increases the effectiveness of the conflict detection process because a conflict is only reported if developer interaction is needed.

3.3 Behavioral Semantic Conflicts

To tackle the emergence of behavioral semantic conflicts, caused by potentially occurring side effects, an example for a view definition for detecting behavioral semantic conflicts is explained in the following.

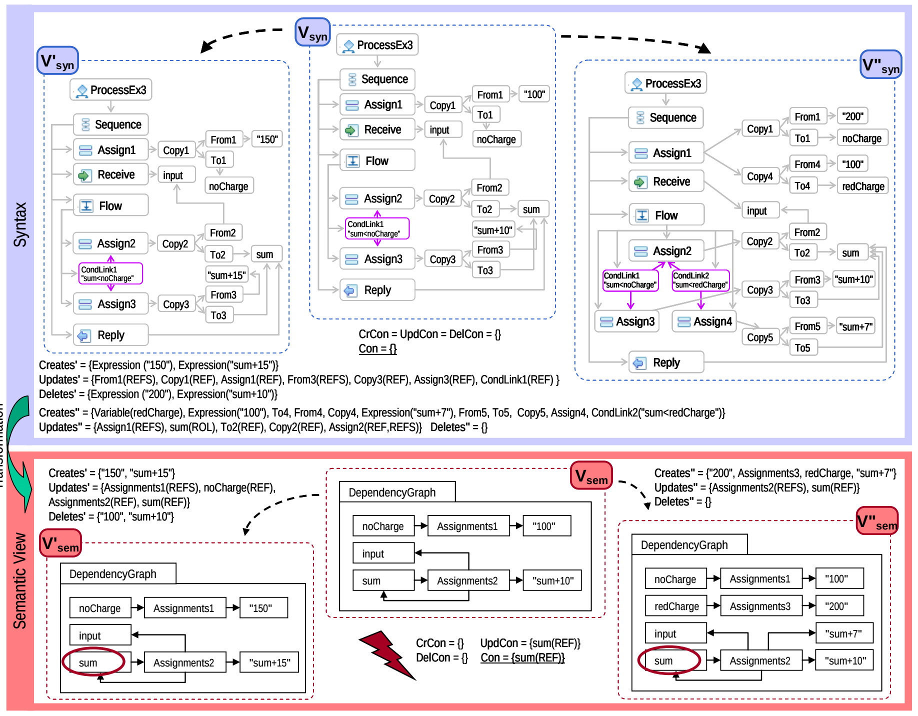
Syntax
Transformation
Semantic View
V'syn
Vsyn
V''syn
ProcessEx3
Sequence
Assign1
Receive
Flow
Assign2
Assign3
Assign4
Reply
CondLink1 "sum<noCharge"
CondLink2 "sum<redCharge"
Copy1
Copy2
Copy3
Copy4
Copy5
From1
From2
From3
From4
From5
To1
To2
To3
To4
To5
input
noCharge
redCharge
sum
"150"
"100"
"200"
"sum+15"
"sum+10"
"sum+7"
CrCon = UpdCon = DelCon = {}
Con = {}
Creates' = {Expression ("150"), Expression("sum+15")}
Updates' = {From1(REFS), Copy1(REF), Assign1(REF), From3(REFS), Copy3(REF), Assign3(REF), CondLink1(REF) }
Deletes' = {Expression ("200"), Expression("sum+10")}
Creates'' = {Variable(redCharge), Expression("100"), To4, From4, Copy4, Expression("sum+7"), From5, To5, Copy5, Assign4, CondLink2("sum<redCharge")}
Updates'' = {Assign1(REFS), sum(ROL), To2(REF), Copy2(REF), Assign2(REF,REFS)} Deletes'' = {}
Creates' = {"150", "sum+15"}
Updates' = {Assignments1(REFS), noCharge(REF), Assignments2(REF), sum(REF)}
Deletes' = {"100", "sum+10"}
Vsem
V'sem
V''sem
Creates'' = {"200", Assignments3, redCharge, "sum+7"}
Updates'' = {Assignments2(REFS), sum(REF)}
Deletes'' = {}
DependencyGraph
Assignments1
Assignments2
Assignments3
CrCon = {}
DelCon = {}
UpdCon = {sum(REF)}
Con = {sum(REF)}

Fig. 5. Detection of a behavioral semantic conflict.

The presented example, as shown in Fig. 5 visualizes a *Process* for the derivation of postal charges for the payment (*Variable* sum) during online shopping. If the payment is smaller than the value 200 additional postal charges are added (Assign3). Developer A has changed the charging calculation process through adapting the noCharge limit and increasing the charge, whereas Developer B has modified the same process slightly different. Due to the fact that Developer

B has inserted a second charge limit (redCharge, Assign4) the modifications of both developers implicitly affect the *Variable* sum as well. Notice, the interference of the *Variable* sum can not be recognized by conflict detection on the syntax. However, it might still of interest for the developers, since a merged version would conform to none of the developers intents.

Therefore, a semantic view definition is used (cf. No.3 in Fig. 2), exploiting the concept of a dependency graph, to make explicit the interference of the *Variable* sum. Considering the semantic mapping, each *Variable* become mapped to a *Variable* referencing a container element *Assignments*, which stores references to all values (*Variables* and *Expressions*) relevant to the current *Variable*. Consequently, concurrent modifications concerning assigning values to one and the same *Variable*, can be detected. Thus, the REF update upon the *Variable* sum indicates that both developers performed implicit changes to this variable resulting in the detection of a behavioral semantic conflict. Reflecting the example the amount of reported conflicts can be increased by utilizing behavioral semantic views of interest. Therefore unintended side effects can be detected and therefore wrongly merged versions encapsulated with a time consuming bug fixing process can be avoided.

4 Related Work

The most relevant approach considering model versioning which provides semantic awareness during the conflict detection process is laid out by *Cicchetti et al.* [11]. They propose 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 are 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 whereas our approach for a semantically enhanced VCS for models can deal with all kinds of EMF-based model artifacts.

Another semantically enhanced approach called *SemVersion* is presented by *Völkel* [12], 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 [13–15]. To start with, *Alanen & Porres* [13] provide difference calculation and merging algorithms with which the functionality of a VCS for MOF-based models can be realized. This approach is not tightly coupled to a specific modeling environment and therefore enables developers the parallel editing of model artifacts with their preferred tooling. *Oliveira et al.* [14] presents a graph-based VCS for versioning UML models called *Odyssey-CVS*, aiming to support 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 approach of *Oda & Saeki* [15] and the commercial tool *IBM Rational Software Architect*[3] are also limited to UML models by the *IBM Rational Software Architect* and additionally ER models by *Oda & Saeki*.

5 Conclusion and Future Work

In this paper a flexible and open semantically enhanced VCS called SMoVer, which is able to incorporate the semantics needed for the conflict detection process between model versions is proposed. By means of transforming a model into a semantic view, equivalent concepts, static semantics and behavioral semantics can be taken into account for conflict detection. As exemplified in section 3 the joint use of model transformations expressing certain semantic aspects of a modeling language, and the employment of existing graph-based comparison techniques on models and views, allows for a more accurate conflict detection between versions of models. This approach, however, benefits by establishing the necessary update strategies and view definitions (transformations and according view definition metamodels) besides increasing effectiveness in the conflict detection phase also in enabling to maintain consistency between concurrently edited model versions in syntax and semantics.

Future research, in a short distant prospect, will focus on the analysis of the semantics of further modeling languages, as exemplified in this paper with the WSBPEL language, for view definitions. Because diverse modeling languages have different power of expressiveness for equivalent concepts, static and behavioral semantics, an evaluation will address a diversity of different modeling languages in order to derive a comprehensive evaluation of the accuracy and according effectiveness of the approach. Furthermore, it is necessary to investigate into the realization of an extension of SMoVer, to not only support syntactic and semantic conflict detection but also conflict resolution and merging of

[3] http://www-306.ibm.com/software/awdtools/architect/swarchitect/

model versions. Therefore visualization techniques for conflict reports and resolution and additionally merge mechanisms have to be addressed. However, in a longer prospect, we focus on support for metamodel versioning and versioning of a model in different languages as proposed by the ModelCVS system [16].

References

1. Thione, G.L., Perry, D.E.: Parallel changes: Detecting semantic interferences. In: Proc. of the 29th Annual Int. Computer Software and Applications Conf. (COMPSAC). Volume 1., IEEE Computer Society (2005) 47–56
2. Mens, T.: A state-of-the-art survey on software merging. IEEE Transactions on Software Engineering **28**(5) (May 2002) 449–462
3. Harel, D., Rumpe, B.: Meaningful modeling: What's the semantics of "semantics"? Computer **37**(10) (2004) 64–72
4. Altmanninger, K.: Models in conflict – towards a semantically enhanced version control system for models. In Pons, C., ed.: Proc. of the MoDELS 2007 Doctoral Symposium. (2007) Nashville, Tennessee.
5. Reiter, T., Altmanninger, K., Bergmayr, A., Schwinger, W., Kotsis, G.: 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). (June 2007) Funchal, Madeira.
6. OASIS: Web Services Business Process Execution Language (WSBPEL) Standard Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf (April 2007)
7. Slonneger, K., Slonneger, K., Kurtz, B.: Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)
8. Object Management Group (OMG): OCL 2.0 Specification (June 2005)
9. 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. (October 2007) Paris, France.
10. Ryndina, K., Kster, J.M., Gall, H.: Consistency of business process models and object life cycles. In: Proc. of the 1st Workshop on Quality in Modeling. (2006)
11. Cicchetti, A., Rossini, A.: Weaving models in conflict detection specifications. In: Proc. of the 2007 ACM symposium on Applied computing (SAC), ACM Press (2007) 1035–1036 Seoul, Korea.
12. Völkel, M.: D2.3.3.v2 SemVersion – versioning RDF and ontologies. $http : //www.aifb.uni - karlsruhe.de/Publikationen/showPublikation?publ_id = 1163$ (January 2006)
13. Alanen, M., Porres, I.: Difference and union of models. In: UML 2003 – The Unified Modeling Language. Volume 2863 of LNCS., Springer (October 2003) 2–17
14. 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'05), ACM Press (2005) 1–16
15. 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)
16. Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: On models and ontologies – a layered approach for model-based tool integration. In Mayr, H., Breu, R., eds.: Proc. of Modellierung, Lecture Notes in Informatics. (2006)