Towards Accurate Conflict Detection in a VCS for Model Artifacts: A Comparison of Two Semantically Enhanced Approaches

Kerstin Altmanninger **Gabriele Kotsis**

Department of Telecooperation
Johannes Kepler University Linz, Austria
Email: [kerstin.altmanninger|gabriele.kotsis]@jku.ac.at

Abstract

In collaborative software development the utilization of Version Control Systems (VCSs) is a must. For this important task some graph-based VCSs for model artifacts already emerged. Optimistic approaches, which are nowadays the designated ones, allow parallel editing of one resource and therefore changes can result in conflicts and inconsistencies. To be flexible for the ever increasing variety of modeling environments and languages VCSs should be independent of the modeling environment and applicable on any modeling language. Those VCS characteristics implicate a lack of information for the conflict detection method by virtue of firstly receiving solely the state of an artifact without concrete editing operations and secondly due to unavailable knowledge about the semantics of a modeling language. In such VCSs inconsistencies would even arise more often. Hence, accurate conflict detection methods are indispensable for the realization of optimistic, environment and language independent VCSs. This can be achieved by providing some understanding about the models's semantics which is possible by specifying machine interpretable formal semantics. Therefore, in this work, a comparison of two semantically enhanced conflict detection approaches is presented with respect to their suitability for the integration in an optimistic, environment and language independent VCS for model artifacts to achieve more accurate conflict reports.

Keywords: Version Control System, parallel model development, model evolution, model comparison, conflict detection, model consistency, model-driven engineering.

1 Introduction

The shift from code-centric to model-centric software development places models as first class artifacts in model-driven engineering (MDE). A major prerequisite for the wide acceptance of MDE is the availability of proper methods and tools for traditional software development, such as build tools, test frameworks or Version Control Systems (VCSs). Considering the latter, VCSs are particularly essential to enable collaborative editing and sharing of *model artifacts* like UML, ER or domain specific modeling languages (DSML) models.

Different systems use different strategies to provide collaborative editing. With the utilization of pessimistic VCSs, model developers can work on the same set of model artifacts. Parallel editing of the same artifact is prevented by locking. *Optimistic* VCSs instead are crucial when the development process proceeds in parallel. Those systems enable each model developer to work on a personal copy of a model artifact, which may result in conflicting modifications. Such conflicting modifications need to be resolved and finally merged by appropriate techniques for model comparison, conflict detection, conflict resolution and merging.

Generic VCSs like CVS[1] or Subversion[2] are not applicable to model artifacts since they apply text-based comparison in a line-based manner and therefore cannot provide adequate conflict reports. *Graph-based* techniques instead need to be utilized to preserve the logical structure of model artifacts.

Most current optimistic, graph-based VCSs for model artifacts are coupled to a specific modeling environment e.g., the IBM Rational Software Architect (RSA)[3]. Since modelers evolve models for different system stages and application areas they also utilize different modeling environments e.g., for each purpose or language the most appropriate or preferred one. Environment specific VCSs are not widely applicable and modelers are bounded to a specific modeling environment. Hence, the kind of coupling of the version control functionalities to a model environment is essential. Therefore, standalone VCSs, so-called *environment independent* VCSs, like Odyssey-VCS (Murta et al. 2008) are preferable. Such systems allow modelers to use their modeling environment of preference for editing their model artifacts which leads to a better acceptance of the VCS.

In view of the fact that MDE is not only about UML and in the light of a growing number of DSMLs, VCSs which are solely applicable on specific modeling languages are often not usable. E.g., approaches of VCSs like Cicchetti et al. (2008), Oda & Saeki (2005) and RSA provide solely versioning capabilities for UML models. Hence, the number of supported modeling language is an important characteristic of a VCS. A *modeling language independent* (e.g., MOF-based) VCS like Odyssey-VCS (Murta et al. 2008) is desirable. Summing up, the utilization of an environment and language independent VCS is of interest for model developers since they can choose their preferred modeling environment for editing model artifacts and furthermore can use the VCS for a number of modeling languages.

To achieve a merged, consistent model artifact an *accurate conflict detection method* is essential. To find out the accuracy of the conflict detection method the definition of Leser & Naumann (2006) to deter-

[1] http://www.nongnu.org/cvs/

[2] http://subversion.tigris.org/

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

mine the *effectiveness* of the method can be applied. Therefore, the results gained of the conflict detection method and the actual perception in the reality are considered. With the result *true-positive*, a conflict has been detected by the conflict detection method and in reality. Accordingly, the result *true-negative* states that a conflict has neither been detected by the method nor in reality. Those two results correspond to the best case for accurate conflict detection. The accuracy of the method can be reduced by *false-positive* and *false-negative* results. Those are conflicts reported by the method which are actually no conflicts in reality or those which have not been detected by the method.

For dealing with concurrent modifications on models, in an environment independent and language independent VCS, the concentration on the reduction of false-negative and false-positive results is particularly essential and more challenging. Firstly, environment independent VCS can only operate on the state of a model artifact whereas environment specific VCSs can trace the modification performed by the developers. Since environment specific VCSs receive a editing history those systems dispose of more information for the conflict detection method than environment independent VCSs (Antkiewicz & Czarnecki 2007, Dig et al. 2008). Secondly, VCSs for specific languages can provide language specific conflict reports and therefore gain more accuracy in conflict detection opposed to language independent VCS approaches. For language independent VCS the method to detect conflicts must be general for any modeling language and therefore can not rely on the model languages' semantics. Hence, 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* in order to provide more accurate identification of conflicts. This can be achieved with different techniques by specifying the modeling languages semantics for semantic conflict detection between concurrently edited model versions.

The remainder of this paper is structured as follows: Section 2 motivates the need for semantic specifications in order to provide an accurate conflict detection method. In Section 3, a case study is given which depicts two elaborated semantically enhanced conflict detection approaches by means of a concrete example. Further on, the findings gained out of the case study are discussed. Section 4 presents related work and finally a conclusion and future prospects is given in Section 5.

2 Semantics for Accurate Conflict Detection

Conflict Detection. In the following, a *scenario in a VCS* in which two developers concurrently edit a copy of one and the same model artifact is considered. Thus, the developer who submits the edited model first can proceed with the check-in without any additional effort. The developer who submits the edited model version to the VCS secondly, needs to go through the basic VCS phases (comparison, conflict detection, conflict resolution and merge). After accomplishing this task (s)he can check-in a merged version of the edited model versions.

Because the history of editing operations is not available in an environment independent system the changes performed by the developers need to be identified in the first phase. In other words, the structural features of the model artifacts, namely the attributes and references of a model element, are inspected by applying a 3-way-comparison (Mens 2002). Therefore the differences between the model artifact (V'') which is going to be checked-in and its ancestor version (V) and the differences between the last revision (V') in the VCS repository and V are calculated. The input for the comparison phase constitutes a XMI serialization of the three different versions of the model artifacts. The result of this comparison phase, based on identifying attributes designated in the metamodel, is a `structural diff` comprising added, deleted, and updated elements (Altmanninger 2008). In the second phase, conflict detection, techniques to detect conflicting situations on basis of the two `structural diffs` resulting from the 3-way comparison phase are provided. Conflicts, however, are presented through concurrent operations of one and the same model element like update-update or update-delete operations. Furthermore concurrent create-create operations may also lead to a conflict if they differ in their properties. Those conflicts, also called syntactic conflicts (Altmanninger 2008) because they are detected through a structural comparison of the syntax of the model artifacts, need to be resolved by the model developer. Otherwise the model versions cannot be merged and stored in the repository.

Problems. Since editing operations between model versions are not available in an environment independent VCS and the method is not aligned to a specific modeling language, some conflicts between different model artifacts may remain hidden or are no actual conflicts in reality. This insufficiency can be ascribed to the fact that the conflict detection method has too little information about the meaning of the model artifacts under comparison.

For example, if a false-positive result is received it means that a conflict has been detected by the method which does not constitute a conflict in reality. The reason for such a result is raised by structural modifications performed by developers on model artifacts, which do not actually change the meaning of a model. More concrete, some modeling languages offer different ways to express one and the same circumstance. For example, 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. Hence, to avoid a false-positive result, a mechanism is needed to express the semantics of those constructs to identify them as *equivalent concepts*. If an equivalent concept, however, has been detected the model developer has to be informed about this circumstance and should be able to store both "semantically equivalent concepts".

Moreover, the situation can occur that the method does not detect a conflict which is actually one in reality (false-negative). 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 (Thione & Perry 2005, Shao et al. 2007, Mens 2002), thus yielding an actual conflict, which, without considering the model's semantics, would remain hidden. Reasons could be, firstly, the violation of constraints, relationships or context conditions (*static semantics*) which cannot be operated on by utilizing a solely structural difference computation algorithm. Secondly, also concurrent changes of the behavior of a model artifact could affect a merged model artifact not incorporating behavioral side-effects (*behavioral semantics*). Such, also called *semantic conflicts* (Altmanninger 2008) should additionally be reported to the model developer. Opposed to syntactic conflicts, semantic conflicts do not need to be resolved

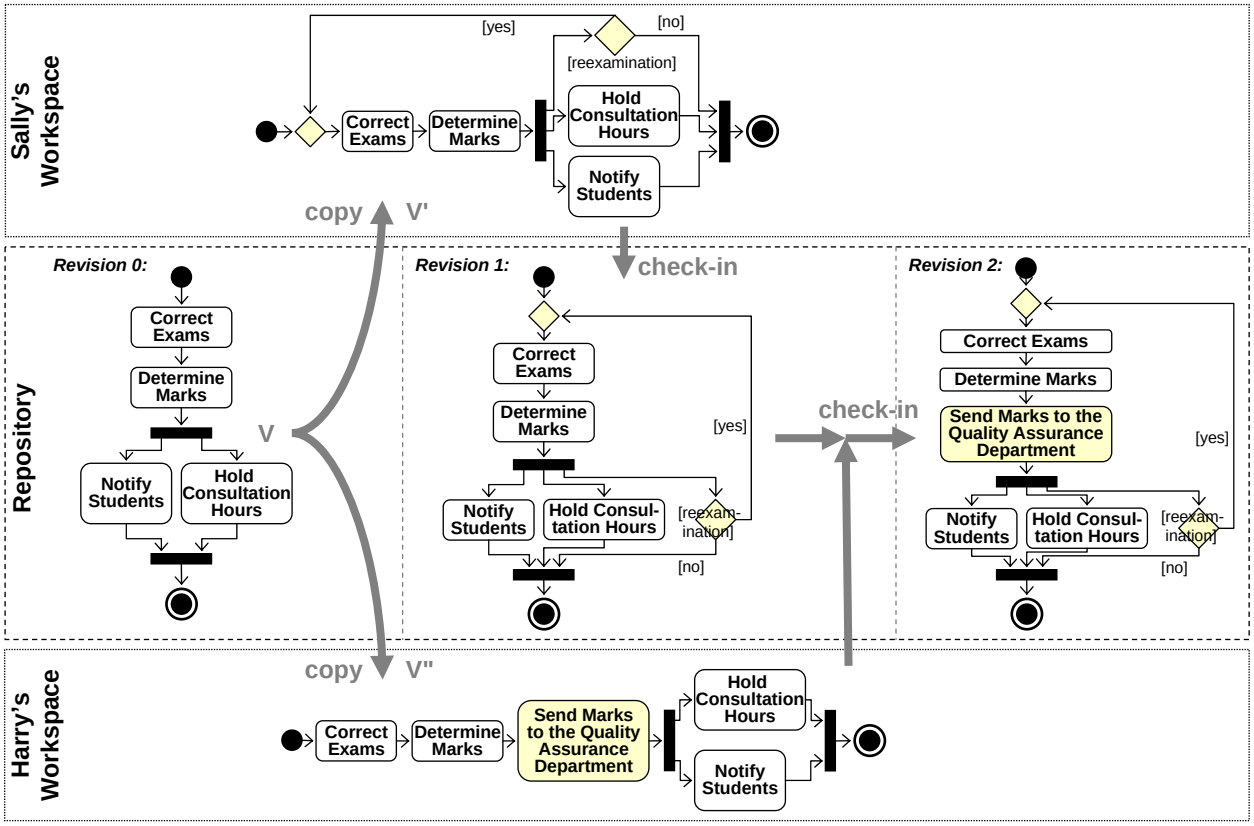

Figure 1: VCS check-in process of UML activity diagram versions.

by the model developer since the model versions can be merged if no syntactic conflict has been detected. Semantic conflicts, therefore, serve as additional notification mechanisms to model developers that the consistency of the resulting model version after the merge cannot be guaranteed without resolving the semantic conflict(s).

As described, to reduce false-positive and false-negative results to achieve an accurate conflict detection method, some understanding about the model's semantics is essential. Concretely, as identified previously, the specification of the three *semantic aspects* (Altmanninger et al. 2007), namely equivalent concepts, static, and behavioral semantics, of modeling languages need to be provided. In order to achieve this goal, a VCS needs an enhancement to cope with these issues. An algorithm, which checks the model versions for semantic interferences with pairwise comparison, would solve the previously mentioned issues to achieve more accurate conflict detection. This technique, however, is not feasible since it may need a lot of computing power if working on large and complex models. Moreover a more abstract approach is desirable to ease the maintenance of the conflict detection method.

Solution. To enhance the conflict detection method the meaning about the models semantics need to be specified in a formal manner. Neither implicit semantics, which are not formalized at all, nor informal semantics, solely understandable by humans, can be taken into account for conflict detection in a VCS since they cannot be processed by machines. Instead, *formal semantics*, with which it is possible to define a semantic mapping which is a functional or relational definition that relates both, the elements of the syntax and the elements of the semantic domain, need to be considered. Many different kinds of formal semantics evolved according to various application areas and disciplines (Harel & Rumpe 2004, Uschold 2003, Sheth et al. 2005, Sheth & Gomadam 2007, Slonneger & Kurtz 1995). Those different kinds of formal semantics vary mainly in the used *formalism* to express the semantic domain and the according mapping. Therefore, two possibilities to express semantics exist, which can also be found in a mixed presentation, namely in pure *mathematics* (Broy et al. 2006) or in the *metamodeling language* (Clark et al. 2008). Depending on the purpose of the semantic description, mathematics are utilized when the modeling language, for which semantics should be defined, does not conveniently provide the appropriate mechanisms ones need (e.g., Formal Semantics for UML[4]). To enable more accurate conflict detection both formalisms (mathematics and metamodeling languages) can be utilized.

3 Case Study

The case study was conducted solely on behavioral modeling languages since only those are capable to cover all three semantic aspects mentioned before. Therefore, the DSML Web Services Business Process Execution Language (WSBPEL) (OASIS 2007, Altmanninger et al. 2007) and a subset of the general purpose language UML activity diagram have been deployed. To receive comprehensive comparison results of the applicability of the two semantic formalisms both modeling languages have been applied and examples for all three semantic aspects have been established.

In the following, an extract of the case study is presented in terms of an UML activity diagram example. To start with, the environment and language independent comparison and conflict detection method is

[4] http://www.cs.queensu.ca/~stl/internal/uml2/

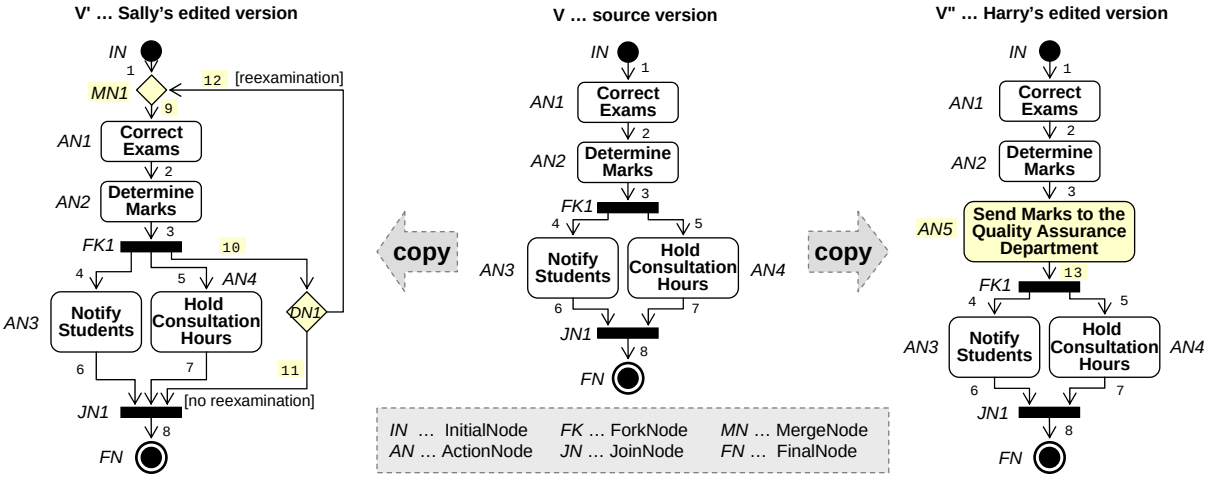

Figure 2: UML activity diagram versions with their according element names.

applied on the UML activity diagram model artifacts. Subsequently, two different semantically enhanced approaches are explained. Both of them can detect an additional semantic conflict between the parallel edited UML activity diagram model artifacts. To elaborate the advantages and disadvantages of those approaches the findings out of the case study are discussed.

3.1 Environment and Language Independent Conflict Detection

Fig. 1 illustrates a scenario for a typical check-in process of a VCS where two developers (Sally & Harry) concurrently edited copies of an UML activity diagram. The UML activity diagram in the scenario encapsulates the procedure for assessment of examinations. Sally edits her personal working copy by adding an additional *DecisionNode* reexamination. If a reexamination for the examination is provided, the control flow edge's target is a new inserted $MergeNode$ in front of the first activity in the procedure. Otherwise the control flow edge's target is the *JoinNode* at the end of the procedure. Sally submits her edited working copy (V') first to the repository. Since the last version in the repository (V) is the one she created a working copy of the check-in process can proceed. Harry has edited his personal working copy by adding an *ActionNode* after the determination of marks. Harry, then wants to check-in his version (V''). Since Sally submitted her version first, Harry needs to apply the VCS check-in phases to obtain a finally merged version. The conflict detection method operating on the abstract syntax of the model versions is not capable to understand the semantics e.g., of a control flow, concurrency or data flow which can be expressed by UML activity diagram. Hence, no conflict is reported and the editing operations of both developers are merged.

In Fig. 2 abbreviations for the $ControlNodes$ and $ActionNodes$ and numbers for the names of the $ActivityEdges$ of the UML activity diagrams are introduced for the description of the environment and language independent comparison and conflict detection method.

In Listing 1 the differences and according conflict sets between the three different versions of the UML activity diagrams are stated. Therefore the method described in previous works (Altmanninger 2008, Altmanninger et al. 2007) is applied. The structural differences between the source version (V) and the edited versions (V', V'') are computed in the sets $Creates$, $Deletes$ and $Updates$. The computed sets of differences then serve as input to detect conflicts (Con) which origin through concurrent create-create ($CrCon$), update-update ($UpdCon$) and delete-update ($DelCon$) editing operations.

```
Creates'={V'–V}={MN1,9,10,DN1,11,12}
Updates'={V–>select(e|e.isUpdated(V,V'))}
         ={1(REFS),AN1(ROL),FK1(REFS),JN1(ROL)}
Deletes'={V–V'}={}
Creates"={V"–V}={AN5,13}
Updates"={V–>select(e|e.isUpdated(V,V"))}
         ={3(REFS),FK1(ROL)}
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: Conflict detection between the UML activity diagram versions.

Between the edited model versions of Sally & Harry no conflict could be detected, with the structural difference computation and conflict detection method, despite the fact that semantically interferences exist. A semantic conflict should be reported in order to make Harry aware of the situation that the source and the set of targets of $FK1$ have been concurrently edited. This means, if the two versions of Sally & Harry are merged, the resulting UML activity diagram provides a $ForkNode$ which has been concurrently updated (update-update conflict) by both developers. The model developer needs to obtain a notification about this semantic conflict in order to be able to react appropriately to prevent an incorrect merging of model versions as it would be the case for this example. Hence, a technique which incorporates the comparison of the control flow is essential. In the following two approaches to tackle this task are presented.

3.2 Semantically Enhanced Conflict Detection

Mathematical-based Approach: Dingel et al. (2006) present in their work a basic abstract syntax for activity diagrams using simple set-theoretical

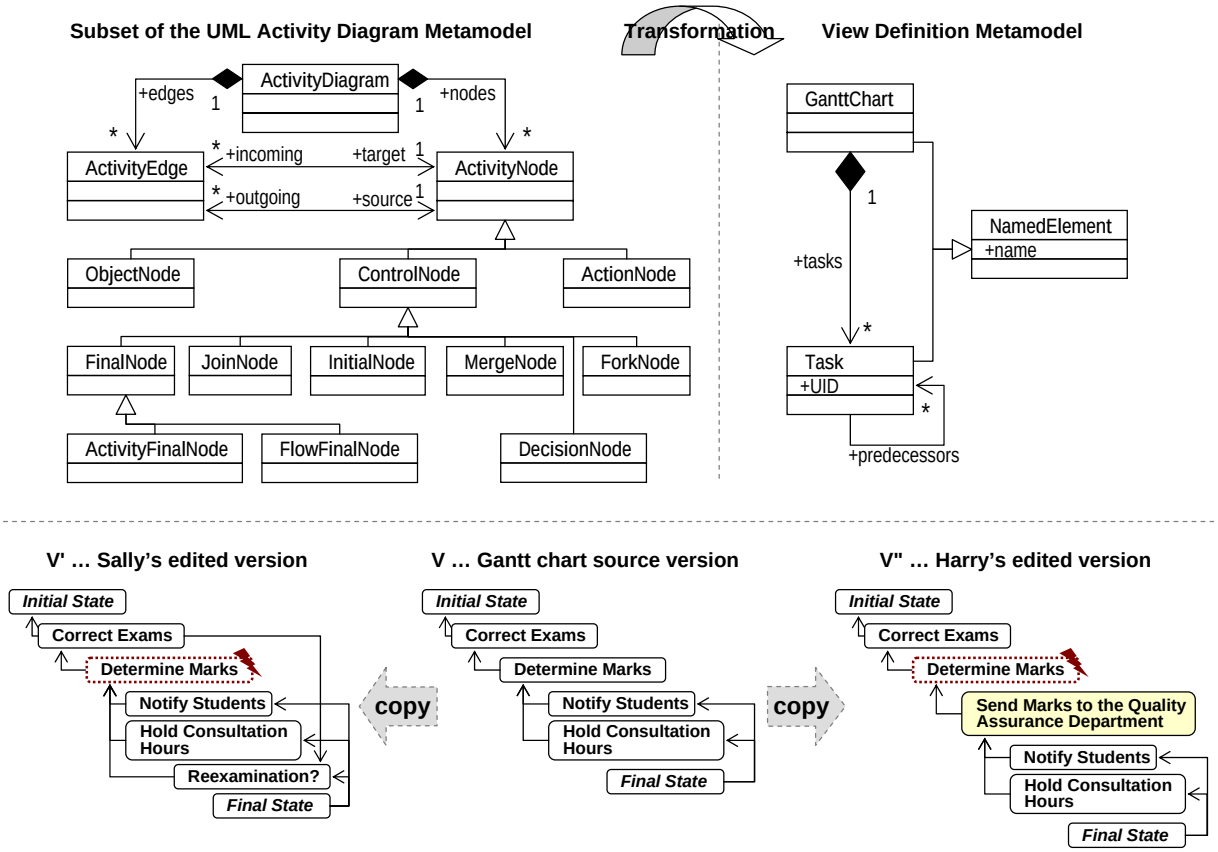

Figure 3: UML activity diagram and Gantt chart metamodels and according transformation to obtain the Gantt chart model versions.

Table 1: Mathematical instances of the UML activity diagram versions.

Sally's edited version	Source version	Harry's edited version
(1,IN,AN1)	(1,IN,AN1)	(1,IN,AN1)
(2,AN1,AN2)	(2,AN1,AN2)	(2,AN1,AN2)
(4,AN2,AN3)	(4,AN2,AN3)	**(3,AN2,AN5)→cr**
(5,AN2,AN4)	(5,AN2,AN4)	(4,AN5,AN3)→upd
(6,AN3,FN)	(6,AN3,FN)	(5,AN5,AN4)→upd
(7,AN4,FN)	(7,AN4,FN)	(6,AN3,FN)
(10,AN2,DN1)→cr		(7,AN4,FN)
(11,DN1,FN)→cr		
(12,DN1,AN1)→cr		

notation. To detect the conflict between the concurrently edited model versions of Sally & Harry the definition for the control flow from Dingel et al. (2006) can be utilized but need to be modified for the sake of detecting additional conflicts. A control flow is defined as an *ActivityEdge* that starts an *ActivityNode* after the previous one is finished. Neither objects nor data may pass along a control flow edge. Therefore, $cf \in ControlFlow$ is defined as a 3-tuple $cf = (name, source, target)$ where

1. $name \in String$ is the name/identifier of this *ActivityEdge*.
2. $source \in InitialNode \cup DecisionNode \cup ActionNode$ is the source of this control flow *ActivityEdge*.
3. $target \in FinalNode \cup DecisionNode \cup ActionNode$ is the target of this control flow *ActivityEdge*.

Applying the definition for the control flow on the three different model versions the results stated in Table 1 can be identified. A conflict detection algorithm computed on those 3-tuple instances of the mathematical definition can now detect a create-create conflict on basis of concurrent specified different targets for control flows for the source node $AN2$.

Metamodel-based Approach: To detect a semantic conflict due to concurrent updates of the control flow the problem domain can also be abstracted by means of a metamodel (e.g., a Gantt chart) and according transformation.

In the top of Fig. 3 both metamodels are stated with an associated transformation which maps the root *ActivityDiagram* element to the *GanttChart* element, the *InitialNode* element to a *Task* without predecessors and each *FinalNode*, *DecisionNode* and *ActionNode* element to a *Task* with their according predecessor elements.

```
Creates'={DN1}
Deletes'={}
Updates'={AN1(REFS),FN(REFS),AN2(ROL)}
Creates"={AN5}
Deletes"={}
Updates"={AN3(REFS),AN4(REFS),AN2(ROL)}
CrCon=DelCon={}
UpdCon={AN2(ROL)}
Con={AN2(ROL)}
```

Listing 2: Conflict detection between the Gantt chart model versions.

Applying the transformation on the UML activity diagram versions (cf. bottom of Fig. 3) the conflict detection method, as given in Listing 1 on the UML activity diagram versions, can now be deployed on the Gantt chart models (cf. Listing 2).

Since those model versions make explicit the control flows between the actions/tasks consequently an additional semantic conflict can be reported. Specifically, the conflict occurred through a concurrent update of both developers of the references pointing to the node $AN2$.

This kind of semantic specification for more accurate conflict detection has been realized in preceding work in SMoVer, the "**S**emantically Enhanced **M**odel **Ver**sion Control System"[5] (Altmanninger 2008). Depending on the type of conflict, which should be avoided or detected, various "semantic view definitions" (target metamodels and transformations) can be specified. Each semantic view definition provided by SMoVer belongs to one of the three semantic aspects and holds a unique name which briefly describes the purpose. How complex and explicit such definitions are designed fully relies on the liability of the SMoVer administrator. He/She has to define a set of semantic view definitions for a specific modeling language all at once when setting-up the VCS or evolutionary so that a group of developers can collaborative edit model artifacts.

3.3 Discussion

As experiences and the above mentioned example showed, conflict detection is not restricted to mere structural difference detection, but with sophisticated methods also conflicts on a semantic level can be avoided and additional ones detected.

Comparison of the different approaches. For the comparison of the suitability of the approaches for the integration in an optimistic, environment and language independent VCS four important dimensions could be identified (cf. Table 2).

Starting with the first dimension, both approaches for semantically enhanced conflict detection are capable to define all **three semantic aspects** important for conflict detection in order to avoid conflicts due to the definition of equivalent concepts and to detect additional static and behavioral semantic conflicts. To tackle this task, mathematics can be utilized. Those are often called as more *precise* because people who communicate using mathematical terminology and notation tend to define things mathematically and therefore precise as well (Harel & Rumpe 2004, Broy et al. 2006). Nevertheless, sometimes mathematical techniques are not quite intuitive and thus need readers to cope with it. In the contrary, the utilization of transformations to different metamodels for the definition of semantics allows to directly execute those transformations using existing mechanisms. Furthermore no knowledge about a different notation for VCS administrators and model developers is needed. VCS administrators can specify new metamodels and according transformations to guarantee more accurate conflict detection with *well known techniques* (metamodels & transformations) and is therefore fast adaptable in an evolutional manner.

In order to be able to **compute semantic conflicts**, for the mathematical-based approach, an interpreter of the defined rules has to be developed. By transforming the source model versions in a different probably more abstract representation, with the metamodel-based approach, the target models provide essential advantages compared to the mathematical approach. With this approach, the *same comparison and conflict detection method* as utilized for the source model versions can be applied on the target models. Moreover, the conflicts computed between the target model versions can be simply traced back to the source models by the model elements IDs.

Table 2: Comparison of the two semantic specification approaches.

Approaches	Definition of Semantic Aspects	Semantic Conflict Detection Method	Visualization of Conflicts	Reuse for Sem. Specifications
Mathematical-based	+	−	−	partially
Metamodel-based	+	+	+	partially

For the mathematical-based approach a **presentation of the conflicting situation** as visualized in Table 1 would not be sufficient for model developers because it cannot be assumed that all model developers understand the mathematical instances of the model artifacts. Therefore an additional technique need to be developed to present the reason of the detected semantic conflict. In the metamodel-based approach, the model developers benefit from the automatically received presentation of the model versions in the semantic views which can be displayed to the developers without enormous additional effort. The model versions are represented in an abstract manner in each semantic view and therefore serve additionally for a better understanding about the conflicting situation.

For the definition of the semantic aspects for accurate conflict detection formal mathematical semantic specification, like those from Dingel et al. (2006) and Broy et al. (2006), can be reused to some extent as visualized in the preceding example. **Reuse** can also be enabled for the metamodel-based semantic specification approach. For instance, examples of the Atlas Transformation Language (ATL) Website[6] (Allilaire et al. 2006), a metamodel of a dependency graph (Altmanninger et al. 2007), a object life cycle (Ryndina et al. 2006) or Petri Nets (Farooq et al. 2007) can be applied for the purpose of making explicit specific aspects of a modeling language. In the UML activity diagram example out of the case study (cf. Section 3) e.g., the transformation of a UML activity diagram in a MSProject model of the ATL Transformation Website has been slightly adapted and reused for a different purpose, for making explicit the control flow for the sake of conflict detection.

Summing up, the metamodel-based approach, to gain accurate conflict detection, can be identified as *less time-consuming and complex to implement*. No additional comparison and conflict detection algorithm has to be invented for the models in the semantic view. Furthermore, the representation of the conflicts can be visualized on the models in the semantic view and can also be easily traced back to the original model versions contrary to the mathematical-based approach. Moreover, the *model developers gain more support* by the metamodel-based approach since they can view the detected semantic conflicts in the according semantic views, without the need of additional knowledge like the understanding of mathematical rules.

Expedient employment of the semantically enhanced approaches. The case study on different modeling languages (DSMLs and a subset of UML)

[5] http://smover.tk.uni-linz.ac.at/

[6] http://www.eclipse.org/m2m/atl/atlTransformations/

has revealed that different modeling languages provide different results of the conflict detection method without incorporating the semantics of a model. For a small DSML with few concepts like a metamodel of a Gantt chart (comprising tasks and references on predecessor tasks) no equivalent concepts can be defined to avoid conflicts and the abstraction level of the metamodel cannot be raised to find additional conflicts. Hence, the conflict detection method already computes almost solely true-positive results and therefore cannot be made more accurate. In the opposite if utilizing a language independent VCS with e.g., the general purpose language UML, the effectiveness of the language independent conflict detection method may not be satisfactory. UML, compared to DSMLs, provides much richer concepts and is more ambiguous. The more concepts a languages provides the more likely it is to have equivalent concepts. Furthermore constraints need to be considered to control the models conforming to the language. Consequently, an effective conflict detection method for complex DSMLs or UML models needs information about the models' semantics unlike for simple DSMLs.

4 Related Work

Besides SMoVer currently no optimistic, environment and language independent VCS with semantic enhancements for more accurate conflict detection on model artifacts exists.

Only two optimistic, environment independent but language specific approaches support semantically enhanced conflict detection to gain more accurate conflict reports. In the area of model engineering, Cicchetti et al. (2008) present an approach, which has not yet been implemented, for providing semantic awareness for the conflict detection method for UML models. Concretely, Cicchetti et al. (2008) 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. (2008), 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). The approach of Cicchetti et al. (2008) 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 the area of ontology engineering, *SemVersion* (Völkel 2006) performs semantic difference calculations on the basis of the semantics of the used ontology language. *SemVersion* is based on the Resource Description Framework (RDF), proposing the separation of language specific features (e.g., semantic difference) from general features (e.g., structural difference or branch and merge). Therefore, assuming using an RDF Schema as the ontology language and two versions (A and B) of an RDF Schema 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. *SemVersion*, however, is limited to RDF based languages and therefore does not provide the flexibility of being reused in the modeling domain.

5 Conclusion and Future Work

In this paper, different semantically enhanced conflict detection approaches (for an optimistic, environment and modeling language independent VCS) which are capable to reduce the occurance of false-positive and false-negative results have been elaborated, exemplified, and discussed.

The case study showed that the metamodel-based opposed to the mathematical-based approach needs less effort for implementation and also provides adequate support for model developers. In more detail the advantages of the metamodel-based approach opposed to purely mathematical specification of formal semantics are firstly the utilization of well known MDE constructs (metamodels & transformations) for the semantic enhancement. Secondly, the use of one and the same comparison and conflict detection method for the model versions in the syntax as well as semantic views. Thirdly, as a consequence of this transformation, developers are provided with a graphical presentation of the model versions in the respective semantic views. Hence, the conflict resolution phase can be completed by model developers with a better understanding about the conflicts detected.

Future work in this area will focus on a comprehensive evaluation of the effectiveness of the conflict detection method in terms of accuracy, which makes use of metamodel-based semantic specifications, realized in SMoVer. Therefore SMoVer is going to be compared to other VCSs for model artifacts like Odyssey-VCS (Murta et al. 2008) and RSA. Since the comparison of the effectiveness of the conflict detection method of SMoVer to other VCSs for models should be applied on as many tools as possible the subset of the general purpose modeling language UML activity diagram was chosen for the evaluation. Firstly because the majority of language specific VCSs provide versioning support for UML and secondly UML activity diagrams belong to behavioral languages and therefore all three semantic aspects, proposed in the context of semantically enhanced versioning (Altmanninger et al. 2007), can be exploited. Furthermore, it will be investigated in realizing a second release of the implementation of SMoVer which encapsulates an advanced representation of the result of the conflict detection phase. This is done by tracing back the falsely indicated syntactic conflicts and semantic conflicts from the semantic views.

In a longer prospect, it is planned to integrate the functionality to version a model in different languages (Störrle 2007) and support for metamodel versioning in SMoVer. Moreover, research in the area of VCSs for model artifacts will focus on building a VCS called AMOR (Adaptable Model Versioning) which comprises the characteristics of SMoVer and additional mechanisms (Altmanninger et al. 2008). Firstly, to further improve the accuracy of the conflict detection method, AMOR will provide supplementary to semantically enhanced conflict detection an operation-based conflict detection mechanism with which logged operations can be imported to the environment independent VCS. Secondly, AMOR will also support intelligent conflict resolution support, specifically aiming at techniques for the representation of differences between model versions and relieving modelers from repetitive tasks by suggesting proper resolution strategies, thus enhancing productivity and consistency of versioning.

References

Allilaire, F., Bézivin, J., Jouault, F. & Kurtev, I. (2006), ATL – eclipse support for model transformation, *in* 'Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France'.

Altmanninger, K. (2008), Models in conflict – towards a semantically enhanced version control system for models, *in* H. Giese, ed., 'Models in Software Engineering, Workshop and Symposia at MoDELS 2007, Nashville, TN, Reports and Revised Selected Papers', number 5002 *in* 'LNCS', Springer, pp. 293–304.

Altmanninger, K., Bergmayr, A., Kotsis, G. & Schwinger, W. (2007), 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)'.

Altmanninger, K., Kappel, G., Kusel, A., Retschitzegger, W., Schwinger, W., Seidl, M. & Wimmer, M. (2008), AMOR - towards adaptable model versioning, *in* '1st Int. Workshop on Model Co-Evolution and Consistency Management (MCCM) at the ACM/IEEE 11th Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)'.

Antkiewicz, M. & Czarnecki, K. (2007), Design space of heterogeneous synchronization, *in* 'Submitted to post-proceedings of Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE)'.

Broy, M., Crane, M. L., Dingel, J., Hartman, A., Rumpe, B. & Selic, B. (2006), 2nd UML 2 semantics symposium: Formal semantics for UML, *in* T. Kühne, ed., 'MoDELS 2006 Workshops', Vol. 4364 of *LNCS*, Springer, pp. 318–323.

Cicchetti, A., Ruscio, D. D. & Pierantonio, A. (2008), Managing model conflicts in distributed development, *in* K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl & M. Völter, eds, 'Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008, Toulouse, France, September 28 - October 3', number 5301 *in* 'LNCS', Springer, pp. 311–325.

Clark, T., Sammut, P. & Willans, J. (2008), *Applied Metamodelling: A Foundation for Language Driven Development*, second edition edn, Ceteva.

Dig, D., Manzoor, K., Johnson, R. & Nguyen, T. (2008), 'Effective software merging in the presence of object-oriented refactorings', *IEEE Transactions on Software Engineering* **34**(3), 321–335.

Dingel, J., Crane, M. L. & Diskin, Z. (2006), Activity diagrams: Abstract syntax and mapping to system model, Technical report, School of Computing, Queen's University, Kingston, Ontario, Canada. Draft – Version 0.0.

Farooq, U., Lam, C. P. & Li, H. (2007), Transformation methodology for UML 2.0 activity diagram into colored petri nets, *in* '3rd Int. Conf. on Advances in Computer Science and Technology (ACST), Phuket, Thailand', ACTA Press, pp. 128–133.

Harel, D. & Rumpe, B. (2004), 'Meaningful modeling: What's the semantics of "semantics"?', *Computer* **37**(10), 64–72.

Leser, U. & Naumann, F. (2006), *Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen*, Dpunkt Verlag.

Mens, T. (2002), 'A state-of-the-art survey on software merging', *IEEE Transactions on Software Engineering* **28**(5), 449–462.

Murta, L., Corrêa, C., Prudêncio, J. & Werner, C. (2008), Towards Odyssey-VCS 2: Improvements over a UML-based version control system, *in* 'Int. Workshop on Comparison and Versioning of Software Models (CVSM)', ACM, pp. 25–30.

OASIS (2007), 'Web Services Business Process Execution Language (WSBPEL) Standard Version 2.0', http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

Oda, T. & Saeki, M. (2005), Generative technique of version control systems for software diagrams, *in* '21st IEEE Int. Conf. on Software Maintenance'.

Ryndina, K., Küster, J. M. & Gall, H. (2006), Consistency of business process models and object life cycles, *in* T. Kühne, ed., 'Models in Software Engineering, Workshop and Symposia at MoDELS 2006, Genova, Italy, Reports and Revised Selected Papers', number 4364 *in* 'LNCS', Springer, pp. 80–90.

Shao, D., Khurshid, S. & Perry, D. E. (2007), Evaluation of semantic interference detection in parallel changes: an exploratory experiment, *in* '23rd IEEE Int. Conf. on Software Maintenance, Paris, France'.

Sheth, A. P. & Gomadam, K. (2007), The 4x4 semantic model: Exploiting data, functional, non-functional and execution semantics across business process, workflow, partner services and middleware services tiers, *in* '9th Int. Conf. on Enterprise Information Systems, Volume DISI, Funchal, Madeira, Portugal', pp. 5–12.

Sheth, A., Ramakrishnan, C. & Thomas, C. (2005), 'Semantics for the semantic web: The implicit, the formal and the powerful', *Int. Journal on Semantic Web & Information Systems* **1**(1), 1–18.

Slonneger, K. & Kurtz, B. (1995), *Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Störrle, H. (2007), A formal approach to the cross-language version management of models, *in* 'Nordic Workshop on Model Driven Engineering'.

Thione, G. L. & Perry, D. E. (2005), Parallel changes: Detecting semantic interferences, *in* '29th Annual Int. Computer Software and Applications Conf. (COMPSAC)', Vol. 1, IEEE Computer Society, pp. 47–56.

Uschold, M. (2003), 'Where are the semantics in the semantic web?', *AI Magazine* **24**(3), 25–36.

Völkel, M. (2006), 'D2.3.3.v2 SemVersion – versioning RDF and ontologies', $http://www.aifb.uni-karlsruhe.de/Publikationen/showPublikation?publ_id = 1163$.