Semantics for Accurate Conflict Detection in SMoVer: Specification, Detection and Presentation by Example

Kerstin Altmanninger, Johannes Kepler University Linz, Austria

Wieland Schwinger, Johannes Kepler University Linz, Austria

Gabriele Kotsis, Johannes Kepler University Linz, Austria

ABSTRACT

In collaborative software development, the utilization of Version Control Systems (VCSs) is a must. For this, a multitude of pessimistic as well as optimistic VCSs for model artifacts emerged. Pessimistic approaches follow the lock-edit-unlock paradigm whereas optimistic approaches allow parallel editing of one resource, which are therefore the preferred ones. To be flexible for the ever increasing variety of modeling environments and languages such tools should be independent of the modeling environment and applicable to various modeling languages. Those VCS characteristics may 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. However, in optimistic VCSs concurrent changes can result in conflicts and inconsistencies. In environment and language independent VCSs inconsistencies would even arise more often due to information losses. Hence, accurate conflict detection methods are indispensable for the realization of such VCSs. To tackle this task, the "Semantically enhanced Model Version Control System" SMoVer is presented. With SMoVer it is possible to specify the semantics of a modeling language, needed for conflict detection in order to provide more accurate conflict reports than other current environment and language independent VCSs. In this work, it is exemplified how semantics of a specific modeling language can be specified in SMoVer, how those specifications can improve the accuracy of conflict reports and finally how those can be presented to modelers.

Keywords: Conflict Detection, Model Comparison, Model Consistency, Model-Driven Engineering, Model Evolution, Parallel Software Development, Version Control System (VCS)

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 are proper methods and tools which are available 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 Language (DSML) models.

Different systems use different strategies to provide collaborative editing. With the utilization of *pessimistic* VCSs, modelers 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 modeler 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 (2008) or Subversion (Tigris, 2008) are not applicable to model artifacts since they apply text-based comparison in a line-based manner and therefore cannot provide adequate conflict reports. To preserve the logical structure of model artifacts *graph-based* techniques need to be utilized instead.

Most current optimistic, graph-based VCSs for model artifacts are bounded to a specific modeling environment, e.g., the IBM Rational Software Architect (2008) also known as RSA. Such environment specific VCSs are therefore not widely applicable. Hence, modelers can not utilize the modeling environment of preference but need to use the one for which version control functionalities are provided and the whole model development team is using. So-called *environment independent* VCSs, like Odyssey-VCS (Murta et al., 2008), instead, are preferable. Such systems allow modelers to use their modeling environment of preference for editing their model versions 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. For example, if modelers evolve models for different application areas they might require to employ different modeling languages, for each application area the most appropriate one. To allow parallel editing in a team of the artifacts under development, language specific modeling environments with included version control functionalities can be utilized in some cases. Beside the drawbacks as already mentioned, for many modeling language no language specific VCS exists. Hence, often generic VCSs like Subversion (Tigris, 2008) are utilized. Examples for modeling language specific VCS approaches which provide solely versioning capabilities for e.g., UML models are Cicchetti and Rossini (2007), Oda and Saeki (2005) and RSA (2008). Therefore, the number of supported modeling languages of a VCS is an important characteristic. A *modeling language independent* (e.g., MOF-based) VCS like Odyssey-VCS (Murta et al., 2008) is desirable. Thus, the utilization of an environment and language independent VCS is of interest for modelers since they can choose their preferred modeling environment for editing model artifacts and furthermore can use the VCS for a number of modeling languages and different application areas.

Such a system needs to provide an *accurate conflict detection method* to achieve a merged, consistent model artifact. A conflict, however, represents inconsistencies between different parallel evolved model versions. To identify the accuracy of the conflict detection method, the definition of Leser and Naumann (2006, pp. 331-333), to determine the *effectiveness* of a method, can be conducted. Therefore, the results gained from the conflict detection method, and the actual perception of conflicts in 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 depict the optimum equate to accurate conflict detection. The accuracy of the method can be declined by *false-positive* and *false-negative* results. These are conflicts reported by the method and not conflicts in reality or those conflicts that 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-positive and false-negative results is particularly 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 modelers. Since environment specific VCSs receive an editing history, these systems dispose of more information for the conflict detection method than environment independent VCSs for which editing operations of modelers are often not present (Lippe & Oosterom, 1992). Secondly, VCSs for specific languages can provide language specific conflict reports since the conflict detection method dispose of language specific features. Hence, those systems gain more accuracy in conflict detection opposed to language independent VCS approaches. This insufficiency of language independent VCS can be ascribed to the fact that the conflict detection method has too little information about the meaning of the model artifacts under comparison. Thus, it is necessary not only to consider the logical structure of models in terms of a graph-based representation, but also to *understand t he model's semantics* in order to provide a more accurate identification of conflicts (cf. Edwards, 1997). Consequently, the focus of this work is laid on the description of how the accuracy of the conflict detection process, in an environment and language independent VCS called SMoVer (SMOVER, 2008), can be improved by the utilization of semantics. This description extends previous work (Altmanninger et al., 2008) in this context by a more in-depth discussion about semantic specifications for conflict detection in SMoVer and corresponding comprehensive examples.

This article is organized as follows. The following section describes the semantically enhanced model VCS SMoVer from a conceptual and implementation point of view. Subsequently it is exemplified, by means of WSBPEL (Web Services Business Process Execution Language, 2007), how semantics can be specified to improve the accuracy of conflict detection and finally how conflicts can be reported to the modeler in SMoVer. Related work concerning conflict detection mechanisms in VCSs for models is then discussed and finally a conclusion and an overview of further prospects are provided.

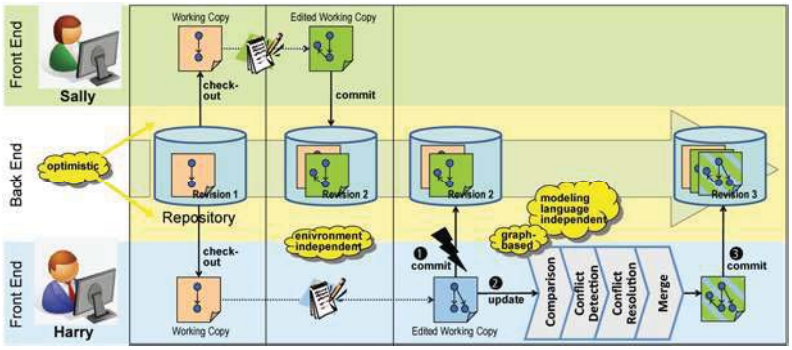
THE SEMANTICALLY ENHANCED MODEL VERSION CONTROL SYSTEM (SMOVER)

In the following the conceptual design of the semantically enhanced model VCS SMoVer is introduced starting with a scenario for the description of the workflow in an optimistic, graph-based, environment and language independent VCS for model artifacts. Finally, the implementation of SMoVer is described.

Conceptual Design

In Figure 1 a scenario is visualized in which two modelers, Sally and Harry are parallel editing the same model artifact by utilizing an optimistic VCS. Therefore Sally and Harry create personal working copies of a model artifact out of the repository and both want to commit their version later back to the repository. After editing the personal working copy in a modeling environment Sally commits her edited working copy of the model artifact to the repository first. The commit process can proceed since the current revision in the repository (Revision 1) is the direct ancestor of the incoming working copy. Harry edited his working copy with a different modeling environment and attempts to commit his model artifact later. Since the last revision in the repository is not the one he has checked-out previously his commit fails and returns with an "out of date" message. Hence he has to apply an update to retrieve a merged version of his edited model artifact and Sally's checked-in version which is now the last revision in the repository. If no other modeler committed a model artifact in the meanwhile Harry can commit the merged artifact to the repository.

Figure 1. Workflow showing the utilization of a VCS for model artifacts



The update Harry applied, however, consists of four phases: Comparison, conflict detection, conflict resolution and merge. In SMoVer, in the first phase, the edited model versions of Sally (V') and Harry (V") have to be compared with respect to their common ancestor version (V) which is called a 3-way comparison (Mens, 2002; Ohst et al., 2003). This comparison process is based on a graph-based structural difference computation (Lin et al., 2007; Rivera & Vallecillo, 2008; Toulmé, 2006) 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 Object Constraint Language (OCL) (Object Management Group, 2005) 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 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(eje.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 CONFLICTS 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 conflicts 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. Model elements have been either created or deleted by modelers 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 combinations of create and delete can be enumerated (CC, CD, DC, DD).
- Referenced element update (REF): A referenced model element has undergone an update.

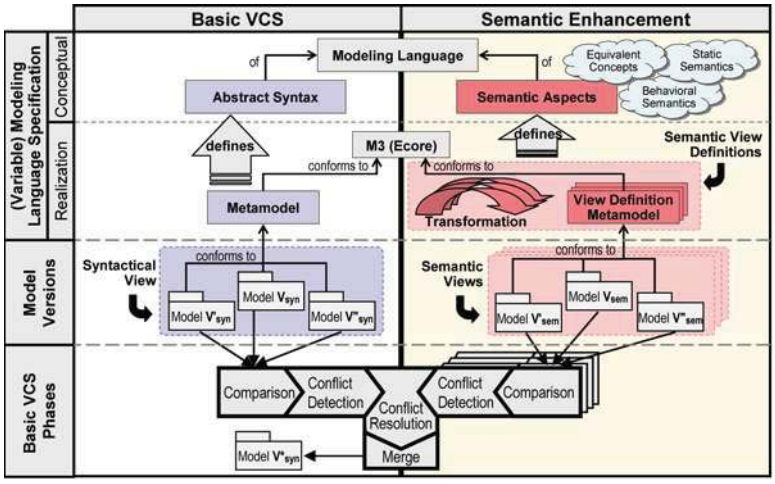
If all update strategies are regarded conjunctively on the concepts provided by the modeling language for the identification of structural differences, this may lead to a report of any structural changes leading to a huge amount of conflicts except particular ones of static and behavioral semantics. Moreover, the update-strategy ROL results out of REFS updates and the update-strategy REF results out of ATT, REFS and ROL updates which assist to define updates more precisely where needed.

Disabling update-strategies on specific model elements may restrict the reported conflicts to those which are valuable. For example, no conflict should be reported due to a REF update of a package element for which two modeler updated/added/deleted contained elements. Thus, the setting of those strategies on the elements of the modeling language which should be versioned is an essential task which needs to be defined in advance and thus 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 (cf. Figure 2) for semantically enhanced conflict detection. Therefore various possibilities exist with which false-positive and false-negative conflict detection results can be reduced. In the following a categorization according to three main semantic aspects important for accurate conflict detection, namely equivalent concepts, static semantics and behavioral semantics, is proposed.

The first category, *equivalent c oncepts*, 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 false-positive results may arise. This means that a conflict has been detected by the method which does not constitute a conflict in reality. For example, one modeler updates a part of a model and the resulting model holds the same meaning than the original ver-

Figure 2. Reference schema of SMoVer



sion. So (s)he used an equivalent way to express the semantics of constructs by using different, semantically equivalent modeling concepts. The second modeler edits the same model but adds, updates or deletes elements that are part of the elements edited by the first modeler and which would result in a conflict. Hence, view definitions for equivalent concepts can be defined in SMoVer to avoid false-positive results. Modelers, however, receive a notification about such avoided conflicts and can make a decision in the representation for storage. The definition for equivalent concepts, however, also serves to detect conflicts more precisely.

Whereas equivalent concepts may give rise to false-positive results, *static sema ntic* and *behavioral semantic conflicts* may not be detected by pure structural comparison and conflict detection between model versions. 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 (Mens, 2002; Shao et al., 2007; Thione & Perry, 2005), thus yielding an actual conflict, which, without considering the model's semantics, would remain hidden. Reasons could be, firstly, the violation of constraints (Object Management Group, 2005), 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 (e.g., data and control flow) of a model artifact could affect a merged model artifact not incorporating behavioral side-effects (*behavioral semantics*). For the detection of such conflicts, techniques such as denotational semantics, program slicing and dependence graphs are proven to be useful in software development (Mens, 2002; Shao et al., 2007; Thione & Perry, 2005). Hence, some of those techniques can be utilized for the creation of semantic view definitions.

A semantic view definition consists of two parts, namely a *view definition metamodel* and a corresponding *model transformation* (cf. Figure 2). The former, which defines the abstract syntax of the semantic view, can either be represented by a subset of the source metamodel (syntactical level) for expressing equivalent concepts, a domain specific view definition metamodel (like a metamodel of a dependency graph) or a metamodel of a different modeling language (Ryndina et al., 2007) to tackle static & behavioral semantics. 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 (Harel & Rumpe, 2004; Slonegger & Kurz, 1995), 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 *syntactical view* (V_{syn}, V'_{syn} and V''_{syn}) and *semantic views* (V_{sem}, V'_{sem} and V''_{sem}), by means of a structural comparison. Hence, a conflict 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.

Implementation DRAFT

In order to define the abstract syntax of a modeling language and a desired semantic view definition, a metamodeling architecture is needed. Therefore the "Eclipse Modeling Framework" (EMF) provides Ecore (Eclipse Foundation, 2008), which is a simplified version of MOF that constitutes the M3 layer, that has been chosen for the realization of SMoVer. 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) (Allilaire et al., 2006), which is a QVT-like model-to-model transformation language.

Accordingly, the top of Figure 2 shows the usage of this metamodeling stack in the context of the implementation architecture.

The comparison of the concurrently edited model versions (V' and V") with their common ancestor version (V) is carried out on a generic graph representation of the respective models and views by using persistent unique identifiers (IDs) and heuristics, e.g., for matching equivalent concepts. For model comparison, however, the EMF reference implementation of "Service Data Objects" (SDO) (Eclipse Foundation, 2008) is utilized. 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 offers a 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 aforementioned update strategies for the comparison phase and establishes the relevant sets of conflicting elements. The comparison, conflict detection and merge components of the implementation are realized using Java, SDO, EMF and ATL. ATL, however, serves for model transformations in the semantic view(s) and to produce a consistent merged model version.

SEMANTIC SPECIFICATION, CONFLICT DETECTION AND PRESENTATION BY EXAMPLE

In the following subsections the process of semantic specification, conflict detection and conflict report presentation for each of the previously mentioned semantic aspects (equivalent concepts, static & behavioral semantics) is exemplified by means of WSBPEL (2007) examples.

Semantic Specification

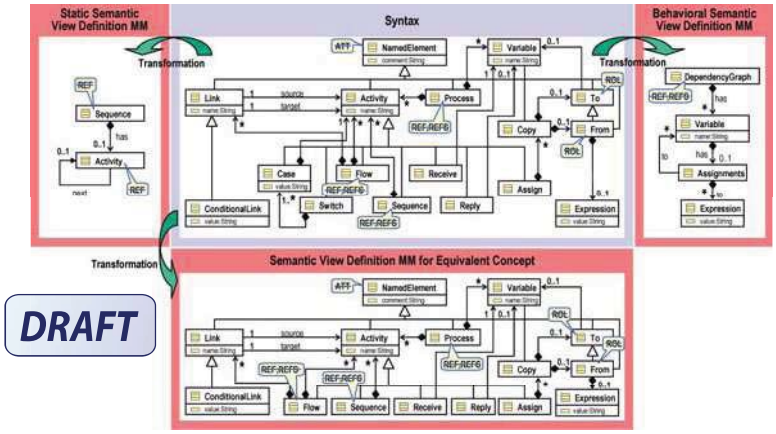
Before inspecting the defined semantic view definitions (cf. Figure 3), and the related examples in detail, the setting of the update strategies is explained in the following. Both tasks, however, are part of the semantic specification possibilities in SMoVer to adapt the conflict detection method to a specific modeling language to improve the accuracy of the method.

In the center of Figure 3 the metamodel of WSBPEL (2007) is visualized for which a possible setting of the update strategies, as the first step to adapt the conflict detection method in SMoVer to a specific modeling language, is shown. To start with, the attribute comment in the element NamedElement is neglected as conflicting value update of an attribute (ATT) because it supposably does not influence the semantics of the model. Considering the REF update, for WSBPEL it is not advisable to report a conflict in, for example, a Process, Sequence or Flow element. Otherwise those elements would always report a conflict if concurrent modifications, which do not affect each other, have been performed in the same Process, Sequence or Flow. Similarly, no conflict should be reported if two modelers created or deleted Activity elements from/to a Flow/Sequence (REFS update) except in one case if two modelers added an Activity to the same position in a Sequence. To limit the amount of falsely indicated conflicts (false-positive results) reported in the syntactical view, a semantic view definition has been established (cf. Static Semantic View Definition in Figure 3). Accordingly, a conflict is only reported if a user interaction is required. 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.

Semantic Conflict Detection

In the following for each of the three semantic aspects an example is given in order to describe

Figure 3. Subset of a WSBPEL metamodel and associated semantic view definitions



the semantically enhanced conflict detection process.

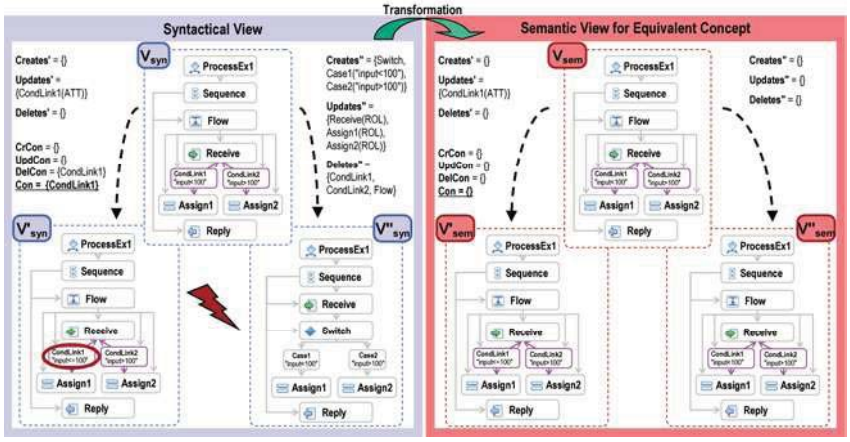
Equivalent Concepts

Considering the example as shown in Figure 4, it describes two activities (Assign1, Assign2) referenced by two ConditionalLink nodes (CondLink1, CondLink2) defining preconditions for the execution of an activity. Sally has changed her working copy by adapting the condition in CondLink1. Harry has replaced the concept of utilizing a Flow and according ConditionalLink nodes by a Switch with two Case statements in his working copy. Regarding the modified working copies, a purely structural comparison would report a conflict during the check-in process, since CondLink1 has been updated by Sally and deleted by Harry. A modeler, however, would identify the parallel edited model versions as consistent to each other after inspecting their semantics since the conflict in the syntactical view results due to the utilization of two different, semantically equivalent concepts.

Therefore, a semantic view definition (cf. Semantic View Definition for Equivalent Concept in Figure 3), constituting a subset of the WSBPEL metamodel is defined. Notice, the concept of Switch and Case has been excluded in the semantic view definition metamodel. Accordingly, modeling diversities of the WSBPEL language, since each Switch statement with Case nodes can also be expressed through a Flow with ConditionalLink nodes, are reduced. The above mentioned fact is covered by the semantic mapping, in that each Switch statement with Case nodes become transformed into a Flow with ConditionalLink nodes conveying the same meaning. 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 modeler, which results in a more effective conflict detection.

Reflecting the above explicated example, a syntactic but no semantic conflict is reported. The knowledge gained conveys that the different model versions comprise equivalent concepts and therefore are consistent to each other. Imagine a slightly different scenario in which Harry modified the Case1 element which is transformed to a CondLink1 element in the semantic view, a semantic conflict is reported as well additionally to the syntactic one. In this case, the semantic conflict detection conveys the information about the origin of the semantic conflict in the equivalent concept. Therefore, with the help of semantic view definitions

Figure 4. Syntactic conflict resulting from semantic equivalent concepts



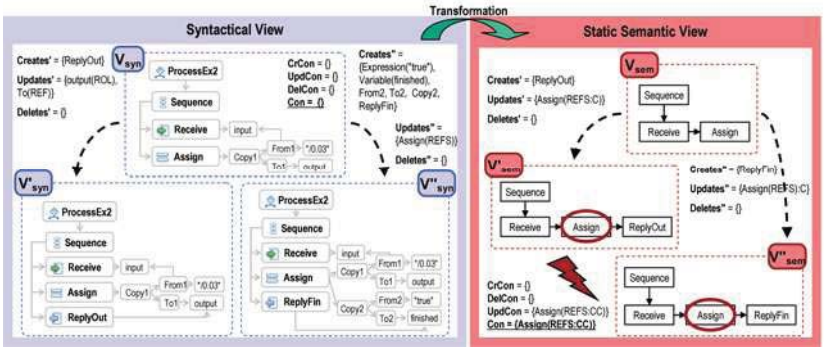
for equivalent concepts, semantic conflicts in equivalent concepts can be detected and appropriately reported to modelers.

Static Semantic Conflicts

In the syntactical view in Figure 5, a WSBPEL model is given (V_{syn}) describing a Sequence containing Activity nodes which are subject to an order. If Sally and Harry concurrently insert the Activities ReplyOut and ReplyFin at the same position in the Sequence, this gives rise to a conflict due to the fact that a decision is required about the order of the created activities to generate a 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 precisely as with a semantic view definition. Using a REFS:CC strategy in the syntax does not make available the model element which actually causes the conflict, since

Figure 5. Detection of a static semantic conflict



the relationship of activities is merely expressed implicitly through the Sequence element.

Therefore, to make explicit the relationship of Activity elements, a static semantic view definition is established (cf. Figure 3) describing an ordering relation of activities. Looking at the model transformation, by realizing the semantic mapping, a Sequence containing Activity elements is mapped to a connected chain of Activity elements 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 Figure 5 a static semantic conflict is detected in the semantic view due to the reference property next of the Assign Activity. Summing up, this static semantic view definition strongly increases the accuracy of the conflict detection report in this setting because a conflict is only reported if the modeler's interaction is needed.

Behavioral Semantic Conflicts

While all parallel changes, performed by modelers, are intended to affect a model semantically, it may occur that no syntactic conflict is reported but semantic interferences, caused by e.g., behavioral side effects, exist. To tackle the emergence of behavioral semantic conflicts an example for a view definition is explained in the following.

The presented example, as shown in Figure 6 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). Sally has changed the charging calculation process through adapting the noCharge limit and increasing the charge, whereas Harry has modified the same process slightly different. Due to the fact that Harry has inserted a second charge limit (redCharge, Assign4) the modifications of both modelers (Harry and Sally) 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 be of interest for Sally and Harry, since a merged version would conform to none of the modeler's intents.

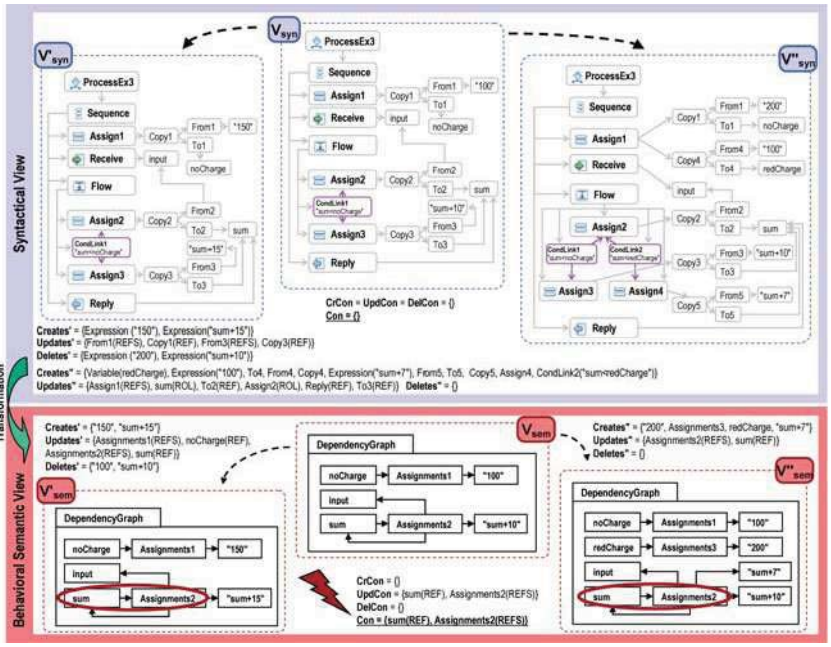
Therefore, a behavioral semantic view definition (cf. Figure 3) is used exploiting the concept of a dependency graph, to make explicit the data flow in the model versions. Considering the semantic mapping, each Variable is mapped to a Variable referencing a container element Assignments, which stores references to all values (Variable and Expression elements) 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 modelers performed implicit changes to this Variable resulting in the detection of a behavioral semantic conflict. Reflecting on the example, the amount of reported conflicts can be increased and therefore false-negative results are reduced realized by the utilization of behavioral semantic views. Hence unintended side effects can be detected and consequently wrongly merged versions encapsulated with a time consuming bug fixing process can be avoided.

Presentation of Conflict Reports

In order to assist modelers during the conflict resolution phase the result of the conflict detection phase should be presented as supportive as possible.

In Figure 7 the possible outcomes of the conflict detection phase are visualized. The presentation of detected conflicts is by default reported for each view in which they are detected, in the syntactic and semantic views. Nevertheless, this presentation has some drawbacks. First, if a semantic view definition comprising an equivalent concept and a specific scenario has turned out as conceptually visualized in Figure 7, in the bottom left rectangle (C), the equivalent concept needs to be traced back to the syntactical view to eliminate the wrongly identified conflict. Secondly, semantic views are specific abstract views comprising checks for semantic interferences between the model

Figure 6. Detection of a behavioral semantic conflict



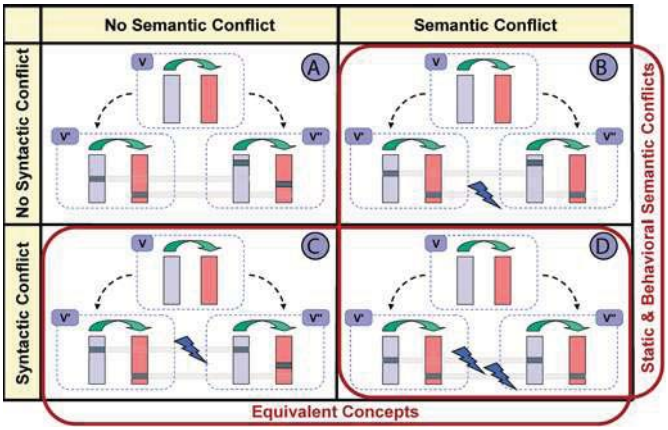
versions. If a conflict is detected in such a view it might be difficult for the modeler to figure out the changes he has to apply on his model in the syntactic view to resolve a conflict. Summing up, it is essential to collect all information retrieved out of the conflict detection phase in one view, the syntactical one to assist the modeler as best as possible in the conflict resolution phase. Therefore, a technique is essential to *trace back* semantic conflicts as well as information on equivalent concepts into the syntactical view.

Considering the previous example of the semantic view definition for an equivalent concept (cf. Figure 4) the elements in the semantic view which have been transformed in an equivalent presentation can be traced back by their ID to the elements in the syntactical view. This mechanism is needed to indicate the falsely detected conflicts due to equivalent concepts in the syntactic view. The modeler, however, can choose between the equivalent concepts. If (s)he does not choose one, the committed representation is taken for the storage in the repository.

In the second previously described example (cf. Figure 5), for a static semantic view definition the presentation of the static semantic conflict in the syntactical view is simply done by tracing back the conflicting elements in the semantic view by the elements IDs.

In the last example (visualized in Figure 6) the conflict for the element sum can be traced back according to a 1:1 mapping by the element's ID. The Assignment2 element in the behavioral semantic view is not present in the syntactical view (0:1) and therefore (a) WSBPEL metamodel element(s) can be specified for which a conflict should be displayed. Hence, it can be defined that all elements to which Assignments2 points should be traced back. Finally, behavioral semantic conflicts are displayed in the syntactical view in V′ for the elements sum, input and "sum+15" and in V" for the elements sum, input, "sum+7" and "sum+10".

Figure 7. Possible outcomes of the conflict detection phase



RELATED WORK

DRAFT

Versioning is a major subcategory of the broad field of *software c onfiguration management* (Conradi & Westfechtel, 1998), the discipline of handling the evolution of complex software systems. Emerging in the field of *software engineering*, the first versioning system was implemented in the early 1970s and was based on the simple idea that each time a file was changed, a new version was created. At that time, already multi-user management was supported, but only in a pessimistic manner. Since locking imposes limitations on concurrent editing *optimistic a pproaches* came up. Generic VCSs like CVS (2008) and Subversion (Tigris, 2008) provide comparison and merge algorithms which take neither the structure nor the semantics of the artifact under consideration into account. With those systems any kind of artifact can be versioned but they are too weak to support the underlying data model of e.g., model artifacts, program source code files and RDF graphs. Based on this observation, the quest for more expressive data models was guided not only in the area of software engineering but also in model-driven engineering and ontology engineering. Here, the shift from line-based to tree-based, and finally to graph-based data models took place. Since the natural representation of artifacts in software engineering, model-driven engineering and ontology engineering are graphs, the focus of the following related approaches (cf. Table 1) is laid on *graph-based* VCSs. Those related approaches are checked against the proposed VCS characteristics environment independent, language independent and accuracy improvement. VCSs with accuracy improvement techniques, like modeling environment specific VCSs which improve the accuracy by logging of editing operations or refactoring operations or VCSs which can solely provide accuracy improvements for a specific modeling language, are not considered.

As visualized in Table 1 the number of *environment independent* VCSs is rare. Solely Odyssey-VCS and CoObRA provide interfaces which allow the integration in arbitrary modeling environments. Odyssey-VCS provides a web service interface for the VCS functionalities check-out, commit and update by streaming the model artifacts XMI serialization. Thus, the integration in any modeling environment can be achieved easily. CoObRA is incorporated in the Fujaba tool (Schneider & Zündorf, 2007) and SMoVer is incorporated in Eclipse. Both VCS explicitly define interfaces which have to be implemented in any environment in order to use the backend. The rest of the approaches, mentioned in Table 1, are either not fully realized (cf. Cicchetti & Rossini and Alanen & Porres),

Table 1. Related Approaches checked against the proposed VCS characteristics

Area	Approach	Environment Independent	Language Independent	Accuracy Improvement
Model Engineering	Cicchetti & Rossini (2007)	×	×	✓
	Odyssey-VCS (Murta et al., 2008)	✓	✓	×
	CoObRA (Schneider & Zündorf, 2007)	✓	✓	×
	Alanen & Porres (2003)	×	✓	×
	CAME (Oda & Saeki, 2005)	×	×	×
	RSA (2008)	×	×	×
Software Engineering	MolhadoRef (Dig et al., 2008)	×	×	×
	Ekman & Asklund (2004)	×	×	×
Ontology Engineering	SemVersion (Völkel, 2006)	×	×	✓

are part of a modeling environment or provide plug-ins for specific modeling environments, IDEs and editors. For example, CAME and RSA have version control functionalities implemented, MolhadoRef as well as the approach of Ekman & Asklund are plug-ins for Eclipse and SemVersion can be used within Protégé.

Modeling language independent VCS are rare. To start with, Odyssey-VCS as well as the presented VCS SMoVer support versioning functionalities for MOF-based model artifacts and therefore offer much flexibility in the utilization of the VCS. Another language flexible system is CoObRA which works with object-oriented data models. Over and above, Alanen & Porres do not provide difference calculation and merging algorithms with which the functionality of a VCS for MOF-based models can be realized. By contrast, the approach of Cicchetti & Rossini and RSA solely provide versioning functionalities for UML model artifacts and the modeling environment CAME supports additionally ER models. Additionally, MolhadoRef and the approach of Ekman & Asklund are solely applicable on the programming language Java. Finally, SemVersion is limited to RDS based languages and therefore also does not provide the flexibility of being reused in the modeling domain.

Only two approaches exist which provide semantic specification possibilities to *improve the accuracy* of the conflict detection method. In the area of model engineering Cicchetti & Rossini propose to leverage conflict detection and resolution by adopting design-oriented descriptions endowed with custom conflict specifications. Hence, several conflicting situations, which cannot be captured by a priori structural conflict detection mechanisms, can be specified that they refer to as "domain specific conflicts". The VCS administrators, however, are forced to enumerate all wrong cases in the form of weaving models, which negatively affects the usability and scalability of the approach. Therefore, in the work of Cicchetti & Rossini each modification is not allowed to preserve a design pattern and the design pattern itself has to be specified in a weaving pattern (as they exemplified for the singleton design pattern). Anyway, this approach 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 performs semantic difference calculations on the basis of the semantics of the used ontology language. SemVersion is proposing the

separation of language specific features (e.g., semantic difference) from general features (e.g., structural difference or branch and merge). Therefore, assuming the use of 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.

Summarizing, currently (2008) no optimistic, graph-based, environment and language independent VCS (besides SMoVer) exist which provides techniques to improve the accuracy of the conflict detection method. Nevertheless, accuracy improvement techniques are especially essential for such systems to tackle eventual information lacks opposed to environment and language specific VCSs (cf. Lippe & Oosterom, 1992).

CONCLUSION AND FUTURE WORK

In this paper a semantically enhanced VCS called SMoVer is presented in an exemple-based manner. SMoVer encapsulates the characteristics of an optimistic, graph-based, environment and language independent VCS which provides semantic specification possibilities to ensure more accurate conflict detection during the check-in process. This process of semantic specification in SMoVer is exemplified in terms of the modeling language WSBPEL. Therefore, firstly, the semantic specification possibilities by means of three semantic aspects (equivalent concepts, static and behavioral semantics) are elaborated. Secondly, the semantic conflict detection method and finally the presentation of the result of the conflict detection phase are exemplified and explained. 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 finalization of possible semantic view definitions in the form of a catalogue for the modeling languages WSBPEL and UML Activity Diagrams (AD). Both selected modeling languages are behavioral modeling languages and therefore all three semantic aspects can be exploited. Those two modeling languages have been selected since an *evaluation* conducted on solely one specific general purpose language or DSML is not significant. Thus, the evaluation will encompass an inspection of the expressiveness and number for semantic view definition possibilities for the DSML WSBPEL and the subset of the general purpose language UML AD. After the completion of this catalogue a comprehensive evaluation of the effectiveness of the conflict detection method, in terms of accuracy, realized in SMoVer will be conducted. Therefore SMoVer is going to be compared to other VCSs for model artifacts like Odyssey-VCS (Murta et al., 2008) and RSA (2008).

In the current implementation of SMoVer (2008) no graphical conflict resolution component exists and conflict reports are only available in text-based manner, separately for all views. Hence, it will be investigated in the implementation of advanced techniques for the presentation of the result of the conflict detection phase in SMoVer.

In a longer prospect, it is planned to integrate the support for metamodel versioning in SMoVer. Moreover, research in the area of VCSs for model artifacts will focus on building a VCS called AMOR (Altmanninger et al., 2008) which comprises the characteristics of SMoVer and additional mechanisms. 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

Alanen, M., & Porres, I. (2003). Difference and union of models. In P. Stevens, J. Whittle, and G. Booch (Ed.), *UML 2003 – The Unified Modeling Language* (LNCS 2863, pp. 2-17).

Allilaire, F., Bézivin, J., Jouault, F., & Kurtev, I. (2006). *ATL – Eclipse support for model transformation*. Paper presented at the Eclipse Technology eXchange Workshop (eTX) at the ECOOP Conference.

Altmanninger, K. (2008). Models in conflict – towards a semantically enhanced version control system for models. In H. Giese (Ed.), *Models in Software Engineering* (LNCS 5002, pp. 293-304).

Altmanninger, K., Kappel, G., Kusel, A., Retschitzegger, W., Schwinger, W., Seidl, M., et al. (2008). *AMOR – towards adaptable model versioning*. Paper presented at the 1st International Workshop on Model Co-Evolution and Consistency Management (MCCM) at the ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS).

Cicchetti, A., Di Ruscio, D., & Pierantonio, A. (2007). A metamodel independent approach to difference representation. *Journal of Object Technology . Special Issue on TOOLS EUROPE*, *6*(9), 165–185.

Cicchetti, A., & Rossini, A. (2007). Weaving models in conflicts detection specifications. In *Proceedings of the ACM Symposium on Applied Computing* (pp. 1035-1036). ACM Publishing.

Conradi, R., & Westfechtel, B. (1998). Version models for software configuration management. *ACM Computing Surveys*, *30*(2), 232–282. doi:10.1145/280277.280280

Dig, D., Manzoor, K., Johnson, R., & Nguyen, T. N. (2008). Effective software merging in the presence of object-oriented refactorings. *IEEE Transactions on Software Engineering*, *34*(3), 321–335. doi:10.1109/TSE.2008.29

Eclipse Foundation. (2008). *Eclipse Modeling Framework Project (EMF)*. Retrieved August 29, 2008, from http://www.eclipse.org/modeling/emf/

Edwards, W. (1997). Flexible conflict detection and management in collaborative applications. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology* (pp. 139-148).

Ekman, T., & Asklund, U. (2004). Refactoring-aware versioning in Eclispe. *Electronic Notes in Theoretical Computer Science*, *107*, 57–69. doi:10.1016/j.entcs.2004.02.048

Harel, D., & Rumpe, B. (2004). Meaningful modeling: What's the semantics of "semantics"? *Computer*, *37*(10), 64–72. doi:10.1109/MC.2004.172

IBM. (2008). *IBM Rational Software Architect*. Retrieved August 29, 2008, from http://www-306.ibm.com/software/awdtools/architect/swarchitect

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

Lin, Y., Gray, J., & Jouault, F. (2007). DSMDiff: A differentiation tool for domain-specific models. *European Journal of Information Systems*, *16*, 349–361. doi:10.1057/palgrave.ejis.3000685

Lippe, E., & Oosterom, N. (1992). Operation-based merging. *ACM SIGSOFT Software Engineering Notes*, *17*(5), 78–87. doi:10.1145/142882.143753

Mens, T. (2002). A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering*, *28*(5), 449–462. doi:10.1109/TSE.2002.1000449

Murta, L., Corrêa, C., Prudêncio, J. G., & Werner, C. (2008). Towards Odyssey-VCS 2: improvements over a UML-based version control system. In *Proceedings of the International Workshop on Comparison and Versioning of Software Models* (pp. 25-30).

OASIS. (2007). *Web Services Business Process Execution Language Version 2.0*. Retrieved August 29, 2008, from http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

Object Management Group. (2005). *OCL 2.0 Specification*. Retrieved August 29, 2008, from http://www.omg.org/docs/ptc/05-06-06.pdf

Oda, T., & Saeki, M. (2005). Generative technique of version control systems for software diagrams. In *Proceedings of the 21st IEEE International Conference on Software Maintenance* (pp. 515-524). Washington, DC: IEEE Computer Society.

Ohst, D., Welle, M., & Kelter, U. (2003). Differences between versions of UML diagrams. *ACM SIGSOFT Software Engineering Notes*, *28*(5), 227–236. doi:10.1145/949952.940102

Rivera, J. E., & Vallecillo, A. (2008). Representing and operating with model differences. In R. F. Paige & B. Meyer (Eds.), *Proceedings of Objects, Components, Models and Patterns: 46th International Conference, TOOLS EUROPE*' (LNBIP 11, pp. 141-160).

Ryndina, K., Küster, J. M., & Gall, H. (2007). Consistency of business process models and object life cycles. In T. Kühne (Ed.) *Models in Software Engineering* (LNCS 4364, pp. 80-90).

Savannah. (2008). *Concurrent Versions System (CVS)*. Retrieved August 29, 2008, from http://www.nongnu.org/cvs/

Schneider, C., & Zündorf, A. (2007). *Experiences in using optimistic locking in Fujaba*. Paper presented at the Workshop on Comparison and Versioning of UML Models.

Shao, D., Khurshid, S., & Perry, D. E. (2007). Evaluation of semantic interference detection in parallel changes: An exploratory experiment. In *Proceedings of the 23rd IEEE International Conference on Software Maintenance (ICSM)* (pp. 74-83).

Slonegger, K., & Kurtz, B. (1995). *Formal syntax and semantics of programming languages: a laboratory based approach.* Boston: Addison-Wesley Longman.

SMOVER. (2008). *SMoVer: A Semantically Enhanced Version Control System for Models*. Retrieved August 29, 2008, from http://smover.tk.uni-linz.ac.at/

Thione, G. L., & Perry, D. E. (2005). Parallel changes: Detecting semantic interferences. In *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC)* (Vol. 1, pp. 47-56). Washington, DC: IEEE Computer Society.

Tigris. (2008). *Subversion*. Retrieved August 29, 2008, from http://subversion.tigris.org/

Toulmé, A. (2006). *Presentation of EMF compare utility*. Paper presented at the Eclipse Modeling Symposium.

Völkel, M. (2006). *D2.3.3.v2 SemVersion – versioning RDF and ontologies*. Retrieved August 29, 2008, from http://www.aifb.uni-karlsruhe.de/Publikationen/showPublikation?publ_id=1163