Delta Operation Language for Model Difference Representation

Dilshodbek Kuryazov
Software Engineering Group
University of Oldenburg, Germany
kuryazov@se.uni-oldenburg.de

Abstract: Software models evolve over time undergoing various changes and resulting in several versions during their lifetime. Models are differentiated during the evolution process and the differences between subsequent model versions are represented in differences documents for further analysis and manipulations as history information. Software models have rich data structures which differ from the code of software systems. That is why, a representation approach for the model differences has to provide effective handling and management of difference information. Furthermore, the model differences represented in the differences documents have to be easy to access and reuse.

This paper introduces the *Delta Operations Language (DOL)*, a meta-model independent and operation-based approach to model difference representations. The approach represents the model differences in terms of DOL and provides several *DOL Services* to access and reuse the DOL-based differences for further analysis and manipulations. To explain ideas behind the approach, it is applied to UML activity diagrams as a running example.

1 Motivation

Software models are designed using software modeling languages (cf. Unified Modeling Language (UML) [UML]). Models are widely used as abstractions of systems structural and behavioural artefacts. Abstraction of any software system is quite useful and practical to understand and trace system aspects from different viewpoints.

Software models follow varying concepts than the code of a software because of the paradigm shift between code and design levels. Software models have rich data structures with different syntax and semantics. Though, like the code of software projects, software models evolve over time undergoing various changes such extensions, corrections and improvements. These changes are applied to software models by a team of modellers using *Collaborative Modeling* and *Model Version Control* tools.

Several version control systems exist for code-based software systems (e.g. Subversion [CFP04], Git [Loe09] etc.). Software models can also be represented in the XMI (XML Metadata Interchange) serialization [Obj]. But, it is commonly agreed that code-centric version control systems can not completely handle the differences of software models [Cic08], [EMF].

Subsequently applying changes to software models results in several versions of the same model artefact. Since a model has several versions during its life time, the model

histories has to be handled and the differences between subsequent model versions have to be represented in appropriate ways.

Representing the model differences allows to store the histories of software models. Therefore, finding an appropriate approach for model difference representation is the best aid to build up various applications and tools such as model versioning, model history analysis and collaborative modeling. Considering aforementioned challenges, this thesis addresses the problem of model difference representation.

This paper introduces the general ***Delta Operations Language (DOL)***, meta-model generic and operation-based approach for model difference representations. Conceptually, DOL is a set of domain specific languages for model difference representation in terms of *operations*. A specific DOL for a specific modeling language is derived from the *meta-model* of a modeling language. A specific DOL is fully capable of representing the differences of models conforming the given meta-model in terms of DOL operations. Only changed elements between model versions are calculated and represented in difference documents referred to as *Modeling Deltas*. The operations in a modeling delta are called *Delta operations*.

Additionally, the DOL approach aims at providing several ***DOL Services*** which can access and reuse delta operations. These operative DOL services improve applicability and re-usability of the DOL-based modeling deltas in different application areas. The operation-based DOL approach addresses several use cases in this PhD proposal. These are *Generic Model Versioning System (GMoVerS)*, *Model History Analysis* and *Collaborative Modeling* that are discussed in Section 5.

The paper is structured as follows: The related approaches are discussed in Section 2. Section 3 explains a simplified difference representation example of the DOL approach. In Section 4, the main ideas behind the thesis are portrayed. Section 5 presents application areas of the approach. Current and ongoing works are expressed in Section 6. The paper ends up by defining expected contributions of the thesis in Section 7.

2 Related Work

There are several approaches to model difference representations providing some additional services. This section discusses some of related difference representation approaches which provide a support to deal with some aspects and principles of model difference representation.

An operation-based difference representation is introduced by Alanen and Porres [AP03] which is also meta-model independent. It detects differences and union of models and represents the differences as a sequence of difference operations. The approach supplies conflict resolver service that is extended by the combination of the difference operations. Another operation-based approach is *EMF Store* framework [HK13] introduced by Helming and Koegel. EMF Store is a data model repository for EMF (Eclipse Modeling Framework) models [EMF]. The framework supports collaborative modeling services such as semantic versioning, branching and conflict detection services. The EMF Store platform is extended by Krusche and Bruegge with Model-based Real-time Synchronization [KB14]. The EMF Store provides change tracking feature for atomic changes.

A meta-model independent and model-based difference representation is introduced by Cicchetti et al. [Cic08]. The approach uses a *difference model* for representing the dif-

ferences. The differences model conforms to the differences meta-model derived from the base meta-model by *automatic transformations*. The approach provides a *difference application* service which requires *model matching* between the differences and base models. Cicchetti et. al. has *conflict resolution* support for differences merging. Another model-based difference representation approach, EMF compare and merge [EMF] was introduced for differencing EMF models.

SMOVER (Semantically enhanced Model Version Control System) [ABSK07] uses *standard SQL database approach* to store the model differences and provides several standard versioning services such as *add*, *checkout*, *commit* and *update*. SMOVER mostly addresses flexible difference merging technique and requires to have an adapter which lies between external modeling tools and SMOVER.

DeltaEcore – A Model-Based Delta Language Generation Framework [SSA14] addresses engineering delta modeling languages for software product lines. The approach aims at automatically derive delta operations for software architectures of software product lines.

In contrast to the existing approaches, a number of additional contributions of the approach in this paper can be indicated. Deficiency of additional services to reuse and exploit representation information diminishes the applicability of an approach in a wide range of applications. Thus, the DOL-based difference representation approach aims at providing a number of the DOL-services which can access and reuse difference representation information. Meanwhile, the existing approaches provide only few services for exploitation and manipulation of model difference information. The DOL-based difference representation uses simple text-based notations instead of complex models to storage history by means of delta operations i.e. text-based difference representation for the graphical software models. Moreover, DOL provides the specific orchestration of the DOL-services extending application areas.

3 Example

This section describes a simple example of DOL-based model difference representation. To explain the idea of the DOL approach, a simplified UML activity diagram [UML] is employed in this section. The meta-model of a modeling language is required to represent the model differences in terms of DOL. The DOL operations are derived from the simplified meta-model of UML activity diagram depicted in Figure 1. A specific DOL is derived for this specific meta-model in this paper, but the DOL Generator is completely independent from meta-models.

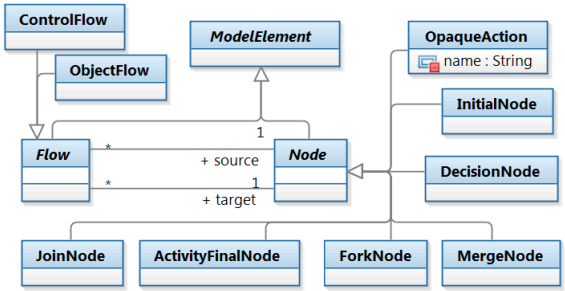

Figure 1: Simplified UML Activity Diagram meta-model

All meta-classes are of type *Node* or *Flow*. Each *Flow* has the target and source attributes and *Actions* have the *name* attribute whereas the other classes have no attributes.

Figure 2 portrays three subsequent versions of the same UML activity diagram performing an *Ordering System* example. All model versions conform to the same simplified meta-model shown in Figure 1.

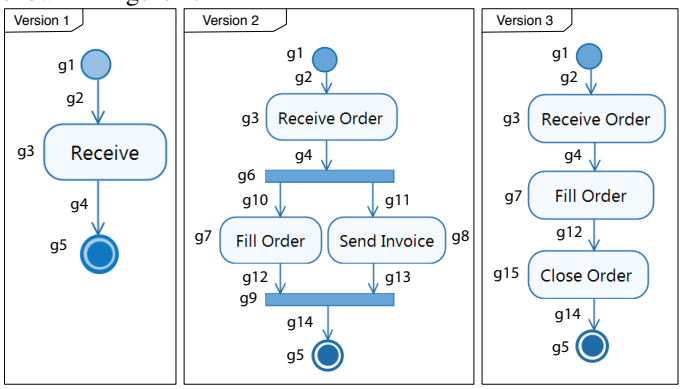

Figure 2: Example activity diagram in three concurrent versions

Each model element is assigned to a *persistent identifier* (g_x). The first version has an *Action* named *Receive*, an *Initial* and a *Final* node and *(Control) Flows* connecting these nodes. In the second version, the model has *Fork, Join nodes*, two *Actions* named *Fill Order* and *Send Invoice*. The target end of *Control Flow* `g4` is reconnected to the fork node, the name of the *Receive* action `g3` is changed and several control flows are created connecting these nodes. Finally, the model reaches into the third version. A new action with name *Close Order* is created, the target ends of `g4`, `g12`, and the source end of `g14` are reconnected. *Fork* and *Join*-nodes, the *Send Invoice* action, and the control flows `g10`, `g11` and `g13` are deleted.

In order to derive the specific DOL for UML activity diagrams, the meta-model in Figure 1 is used and three atomic operations `create`, `delete` and `change` are applied to the concepts of the meta-model (generation of a specific DOL is explained in Section 4.1). A delta-operation creates or deletes a model element or changes its attributes. The DOL approach considers that three basic operations are sufficient for deriving the complete set of the DOL operations for difference representations and the specific DOL is capable of representing all differences by these three operations. Other change operations like *moving* a group of elements from one place to another in a model can be achieved by changing one (or several) association(s) between a part that should be moved and the rest of a model.

Like classical version management systems for source code (cf. Subversion [CFP04], Git [Loe09]), the DOL approach follows a *backward delta* approach, where the most recent version of a model and several modeling deltas are stored directly. The last version (the third version in this example) is also represented by the DOL-operations which consists only of the creation operations.

```
1 g1=createInitialNode();
2 g3=createOpaqueAction("Receive Order");
3 g7=createOpaqueAction("Fill Order");
4 g15=createOpaqueAction("Close Order");
5 g5=createActivityFinalNode();
6 g2=createControlFlow(g1,g3);
7 g4=createControlFlow(g3,g7);
8 g12=createControlFlow(g7,g15);
9 g14=createControlFlow(g15,g5);
```

Figure 3: Active delta

Figure 3 depicts the modeling delta named *active delta* which gives directly the last model version. Each delta operation has a ***Do part*** (cf. *g1=createInitialNode();*) which describes the *kind of change* by means of *operations* (one of create, change, delete) and an ***Object part*** (with attributes if required) (cf. *g1=createInitialNode();*) which refers to the modeling concept. To refer to model elements from the delta operations, *unique persistent identifiers* are used as *references*.

Likewise, the differences deltas are directed in reverse order i.e. each change leads from the later version to the previous. Producing delta operations in reverse order is quite practical for implementation of applications.

```
g6 = createForkNode();
g7 = createOpaqueAction("Send Invoice");
g9 = createJoinNode();
g4.changeTarget(g6);
g12.changeTarget(g9);
g14.changeSource(g9);
g10 = createControlFlow(g6,g7);
g11 = createControlFlow(g6,g8);
g13 = createControlFlow(g8,g9);
g15.delete();
```

Figure 4: Delta between active and the second versions

```
g3.changeName("Receive");
g4.changeTarget(g5);
g6.delete();
g7.delete();
g8.delete();
g9.delete();
g10.delete();
g11.delete();
g12.delete();
g13.delete();
g14.delete();
```

Figure 5: Delta between the second and first versions

The differences delta in Figure 4 depicts the differences between the third and second versions. Finally, Figure 5 illustrates the differences delta between the second and first versions.

The modeling deltas in these figures are executable descriptions of the differences. Each of the difference deltas reverts the model to the previous versions. The modeling delta in Figure 4 reverts the model to the second version from the third and the modeling delta in Figure 5 reverts the model to the first version from the second.

4 Approach

This section discusses a general architecture of the proposed approach as depicted in Figure 6. It has three main levels such as ***DOL Generation*** (discussed in Section 4.1), ***DOL Services*** (Section 4.2) and ***DOL Applications*** (Section 5).

DOL Generation defines generating a specific DOL by importing the meta-model of a modeling language. The resulting DOL is produced in form of Java Interface. The DOL approach also provides several services that lie at the *DOL Services* level. These services serve to manage, manipulate and reuse the DOL-based modeling deltas stored in the repository. The third level depicts *DOL Applications* developed by the specific orchestrations of the DOL-services.

Each DOL-service has a particular task and is involved in constructing the specific orchestrations in the framework of the DOL applications. For instance, models are designed in external modeling tools and parsed into the internal representations by the *Adapter*. The difference *calculator* is used to detect the differences between the compared models and produce the differences and active deltas by implementing the DOL interface. The difference calculator uses an *Optimizer* to produce the optimized modeling deltas and to write them into the repository. After all, other DOL-services such as a *Patcher* and a Change

Tracer utilize the DOL operations. The patcher reverts older model versions. The change tracer tracks a specific model element and reports the change history information of that element. Then, these change reports can be browsed to see and analyse the change history.

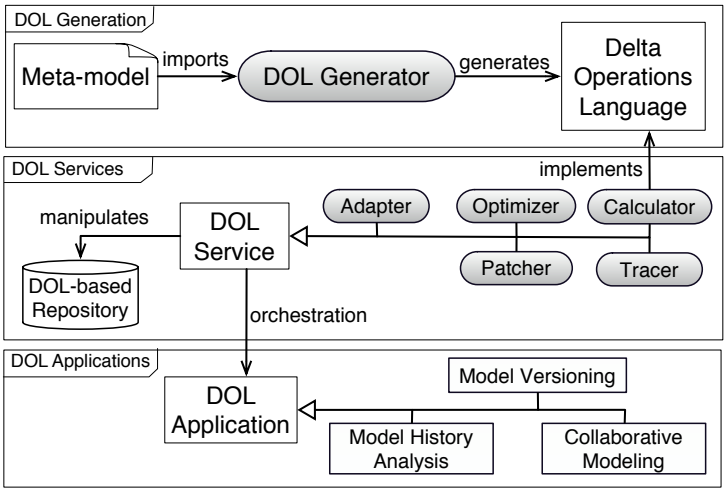

Figure 6: Overall architecture of the approach

These DOL-services are employed in developing the DOL applications such as *Model Versioning*, *Collaborative Modeling* and *Model History Analysis* (discussed in Section 5).

4.1 DOL Generation

The Delta Operations Language (DOL) is a family of the operation-based languages for model difference representations. This section shows how a specific DOL is generated for a specific modeling language. As mentioned above, the meta-model of a modeling language is required to derive a specific DOL. The DOL Generator imports the meta-model depicted in Figure 1 as the example and applies three basic operations to each concept of that meta-model.

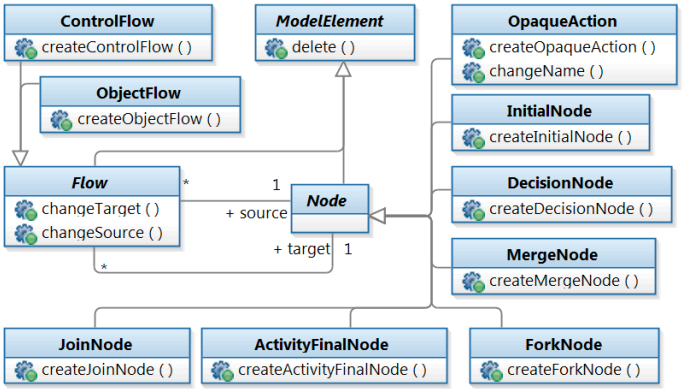

Figure 7: Meta-model of the specific DOL operations

The specific meta-model in Figure 7 depicts an abstraction of the specific DOL operations shown in Figure 8.

Each model element can be deleted and created with relevant parameters if they have attributes. Only attributes can be changed and all associations are associated with attributes. The specific DOL is generated as an Java Interface. The *methods* of the resulting interface are parametrized with the meta-model concepts including one of the *create*, *delete* and *change* operations (Figure 8). Implementations of these methods result in an analogous operation with relevant parameters.

```
1  //------ Creations ------
2  InitialNode createInitialNode();
3  OpaqueAction createOpaqueAction(String name);
4  ForkNode createForkNode();
5  JoinNode createJoinNode();
6  DecisionNode createDecisionNode();
7  MergeNode createMergeNode();
8  ActivityFinalNode createActivityFinalNode();
9  ControlFlow createControlFlow(Node source, Node Target);
10 ObjectFlow createObjectFlow(Node source, Node Target);
11 //------ Changes ------
12 void changeName(String newName);
13 void changeSource(Node newSource);
14 void changeTarget(Node newTarget);
15 //------ Deletion ------
16 void delete();
```

Figure 8: Interface generated from UML Activity Diagram meta-model

The DOL Generator is completely independent from modeling languages. Eventually, each interface method has the same structure: a *Do part* and an *Object part*. The delta operations are produced by implementations of that interface by assigning persistent identifiers.

4.2 DOL Services

In order to reuse and exploit the DOL-based modeling deltas, the difference representation approach introduces a set of reasonable *DOL-services* as depicted in Figure 6. The DOL-services consist of an *Adapter*, a difference *Calculator*, a delta *Optimizer*, a *Patcher* and a change *Tracer* which are discussed in detail, below.

Adapter. Software models are designed in model designing tools. Integrating version management or collaborative modeling tools with model designing tools is a challenge. But, these tools provide export/import feature for software models by XML Metadata Interchange (XMI) [Obj] format without layout information. Therefore, the *Adapter* is developed to exchange models between external modeling tools and the DOL applications.

In the DOL applications, models are represented as TGraphs internally [ERW08] to make them generally processable. The *Adapter* can parse models in the XMI serializations to TGraphs and vice versa.

Calculator. The difference calculator detects the differences between the differentiated model versions using state-based comparison and produces modeling deltas in terms of the DOL operations. Several approaches already exist for difference calculation. Therefore, Küpker [Kü13] investigated existing approaches such as UMLDiff [XS05] and SiDiff [SG08] [TBWK07] and combined them to *gDiff* (generic differentiating).

The difference calculator calculates the model similarities using the similarity metrics such as a name [XS05] and a structural similarity [TBWK07]. Created, deleted or changed elements are detected for each candidate element with the highest similarity.

The calculator produces two modeling deltas in the operation-based format by implementing a specific DOL Interface: the *active* and the *differences delta*. If the first version of a model is given, the calculator produces only the active delta without the differences delta. The persistent identifiers are assigned to delta operations by the difference calculator while matching model elements.

Optimizer. After the difference calculator produces the modeling deltas, they are optimized to improve their efficiency. For instance, if a particular model element is created and later deleted in the same delta, these two operations (create and delete) are registered for one element where both have no affect. Another example might be changing one element several times in one modeling delta. In this case, it is optimal to save only the last change instead of several change operations for that model element.

Moreover, the order of delta operations in modeling deltas is also important to avoid the lost and fuzziness of change information. The creation operations are placed on the first place in a modeling delta, changes come second and deletions are dropped to the end.

Patcher. The *patcher* reverts a model to earlier versions by *applying* (sequences of) modeling deltas to the last version. Modellers may need to revert a model to an older versions in the case of lost or damage of data. The inputs for the patcher are a *model* and several *modeling deltas*. An input delta is applied to an input model resulting in previous version of a model.

Since the delta operations are the executable descriptions, they are implemented by *in-place* model transformations [Kah06].

Tracer. To comprehend and analyse the histories of evolving models, the DOL approach provides a Change *Tracer* service which contributes to detect necessary information about each change history.

The change tracer receives the set of modeling deltas from the repository as input and searches for change information about a considered model element based on its persistent identifier. Detected change information is used by history analysis application (Section 5).

For instance, Figure 9 illustrates all history information of the control flow `g4` in the example in Section 3. The traced element was referred to in three versions. Information about these states is tracked by slicing three modeling deltas, the *active delta* in Figure 3 and two differences deltas in Figures 4 and 5.

```
1  g4 = createControlFlow(g3,g7);
2  g4.changeTarget(g6);
3  g4.changeTarget(g5);
```

Figure 9: History information of Control Flow `g4`.

5 DOL Applications

As a proof of concept, the DOL-approach is being prototypically implemented. To demonstrate its applicability, it is planned to use the approach in the context of three use cases namely *Modeling Versioning*, *Model History Analysis* and *Collaborative Modeling*. The architectures of these applications are built by the specific orchestrations of the DOL-

services.

Model Versioning. Model versioning aims at managing and manipulating models as well as storing and reusing the model differences. Model versioning systems are essential on handling models and their histories. To this end, the DOL difference representation approach is applied to develop ***G**eneric **Mo**del **Ver**sioning **S**ystem (GMoVerS)*. GMoVerS uses the DOL operations to represent the model differences.

Current implementations of GMoVerS support *adding* models to the versioning system, *committing* changes and *reverting* older versions. In each of these activities, the relevant DOL-services are involved in a certain order based on data-flow. Figure 10 displays a screen-shot of the GMoVerS development environment.

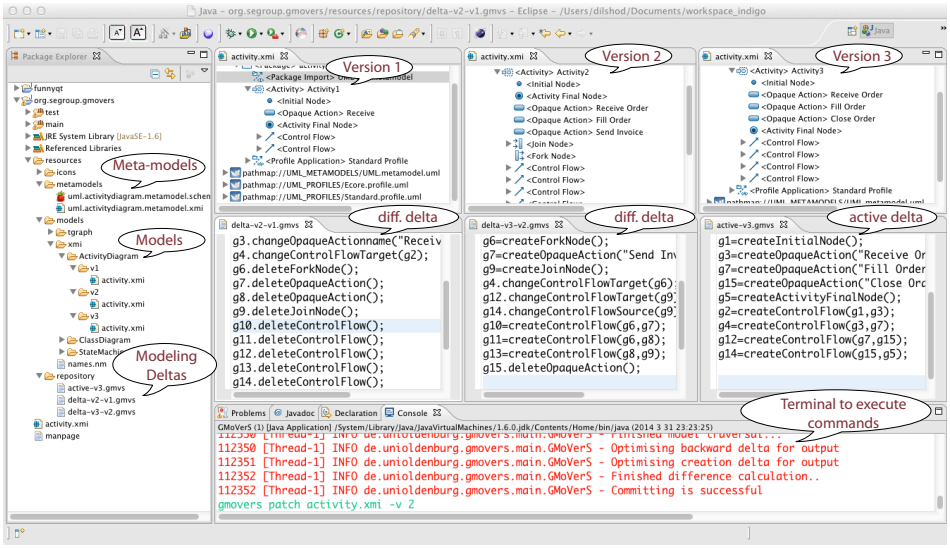

Figure 10: A screen-shot of GMoVerS

First of all, a model has to be ***added*** under version control to start model versioning. The *Adapter* is required to parse a model in the exchange format to internal representation, then the *calculator* is involved to produce the *active delta* for the new model. Similarly, ***committing*** changes requires the *adapter* to parse new versions to internal representation, the *patcher* to revert the previous model version, again the *calculator* to calculate the differences between the previous and committed versions and produce the differences and active deltas. ***Reverting*** needs the *patcher* firstly to revert the recent model version, then it is again called to revert requested version.

The explorer on the left side of Figure 10 displays the workspace including the meta-model in Figure 1, the models and modeling deltas in Section 3. Here, all DOL-based modeling deltas belong to the GMoVerS repository. On the upper row of the right side, Figure 10 displays three subsequent versions of the example model in Figure 2 in exchange formats. The central part of the right shows two differences deltas and the active delta. Finally, the most bottom of the screen-shot is a terminal to type and execute aforementioned version control commands such as add, commit and patch.

Model History Analysis. Analyzing the model histories is crucial to understand what

changes are made to models and to know how a model evolves during its life-time. History analysis is built on the top of the change history *tracer*. The change tracer fetches a set of modeling deltas from the repository and detects history information from these deltas based on persistent identifiers. After detecting necessary history information, it is browsed as depicted in Figure 11.

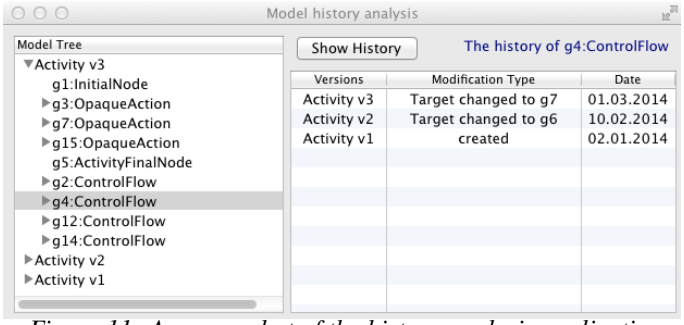

Figure 11: A screen-shot of the history analysis application

The screen-shot in Figure 11 displays the example model in Section 3. The change tracer firstly builds a tree for each model version running throughout all modeling deltas and outlines them in *Model Tree* as shown on the left side of Figure 11 including all model elements. If a specific element is selected and show history button is clicked, the tracer detects history information of the selected model element based on its persisted identifier and shows it in *Tabular View*. For example, the table on the right side shows the history of the *g4:Control Flow* in Figure 9.

Collaborative Modeling. Another prominent application of DOL is collaborative modeling which facilitates a teamwork of several designers on a shared model repository at runtime.

The implementation of the application is planned to rely on representing changes in terms of the DOL operations and exchanging these changes by the appropriate modeling deltas. To this end, it is planned to develop extra DOL-services named *runtime operation recorder* to detect operations embodying changes. The operation recorder will be integrated into end-user model designing tools and it detects changes made by modellers. While manipulating a model, the DOL-based change operations are recorded in modeling deltas. An evolving model has several development branches and *synchronization* is required among various development branches. The synchronization of changes basically is synchronization of the DOL-based modeling deltas among various branches.

6 Current and Ongoing Work

Extensive literature research has been done to become familiar with the existing difference representation techniques and additional services related to this thesis. A first sketch of the DOL approach was published in 2012 [KJW12].

The DOL-based difference representation approach is applied to versioning Sustainability Reports at companies in [KSW13]. As sustainability reports at companies are also subject to constant changes and evolution, companies intend to report and analyse sustainability information to provide the sustainable future. Thus, reports have to be versioned to

analyse the histories of sustainability reports.

The DOL generator and the DOL services such as the *adapter*, the difference *calculator*, the delta *optimizer*, the delta *patcher* and the change *tracer* (depicted in Figure 6) are implemented so far. *GMoVerS* and *Model History Analysis* applications are elaborated by the specific orchestrations of these DOL-services.

Building the specific orchestration of *Collaborative Modeling* is remaining as ongoing work. Further services are requested to build up the collaborative modeling application. Therefore, the DOL-services will be extended with *a runtime operation recorder* and a *synchronizer* for collaborative modeling. A runtime operation recorder facilitates the end-users with runtime operation registration. These modeling deltas will be synchronized by a *DOL-synchronizer*.

Finalizing extension and evaluation of the model versioning and history analysis applications on the large scaled software models is expected till August 2014. Implementation of further DOL-services and writing actual thesis is planned within the year 2014. Proof read and submission of the thesis is scheduled for the beginning of 2015.

7 Expected Contributions

This paper proposed the PhD thesis focusing on development of an approach towards model difference representation. The DOL approach is designed using existing software engineering technologies. The operation-based difference representation approach facilitates several additional DOL services to employ and reuse the DOL-based modeling deltas. The approach satisfies several reasonable principles and requirements: (1) *Meta-model Generic* – applicable to various modeling languages with respect to their meta-models, (2) *Tool Independent* – has own *Adapter*, (3) *Operation-based* – embodies all necessary information about a change using a meaningful syntax and completely follows compound modeling concepts, (4) *Delta-based* – only changed model objects are referred to in modeling deltas, (5) *Complete* – carries precise information about each change, (6) *Reusable* – provides several DOL-services which can directly access and manage the DOL-based modeling deltas making modeling deltas straightforward and accessible.

With these principles, the DOL approach has the following contributions for its applications:

GMoVerS. The model differences are represented in terms of DOL and stored in small modeling deltas. Only changes are referred to in modeling deltas. The delta operations are directly executable descriptions of the model differences allowing model manipulations when needed.

Model history analysis. The delta operations assigned to persistent identifiers allow the change tracer to rapidly detect all necessary information about the change histories by verifying small modeling deltas.

Collaborative modeling. Synchronization of changes will be eased by exchanging small DOL-based deltas. The various model designing tools which are running on different platforms can communicate with each other in terms of DOL.

References

[ABSK07] K. Altmanninger, A. Bergmayr, W. Schwinger, and G. Kotsis. Semantically Enhanced Conflict Detection between Model Versions in SMoVer by example. In *Proceedings of the Int. Workshop on Semantic-Based Software Development at OOPSLA*, 2007.

[AP03] M. Alanen and I. Porres. Difference and union of models. In *UML 2003. LNCS*, pages 2–17. Springer, 2003.

[CFP04] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato. *Version Control with Subversion.* O'Reilly Media, 2004.

[Cic08] A. Cicchetti. *Difference Representation and Conflict*. PhD thesis, University of L'Aquila, (Italy), April 2008.

[EMF] EMF: Eclipse Modeling Framework (Compare). http://www.eclipse.org/emf/compare.

[ERW08] J. Ebert, V. Riediger, and A. Winter. Graph Technology in Reverse Engineering, The TGraph Approach. In *10th Workshop Software Reengineering (WSR 2008)*, volume 126, pages 67–81. GI (LNI), 2008.

[HK13] J. Helming and M. Koegel. EMFStore., 2013. http://eclipse.org/emfstore.

[Kah06] S. Kahle. JGraLab: Konzeption. *Entwurf und Implementierung einer Java-Klassenbibliothek für TGraphen*, 2006.

[KB14] S. Krusche and B. Bruegge. Model-based Real-time Synchronization. In *Inter. Workshop on Comparison and Versioning of Software Models (CVSM'14)*, Feb. 2014.

[KJW12] D. Kuryazov, J. Jelschen, and A. Winter. Describing Modeling Delta By Model Transformation. In *Softwaretechnik Trends (Issue on International Workshop on Comparison and Versioning of Software Models (CVSM 2012))*, no. Band 32 Heft 4. Gesellschaft für Informatik, November 2012.

[KSW13] D. Kuryazov, A. Solsbach, and A. Winter. Versioning Sustainability Reports. In *5.BUIS-Tage: IT-gestütztes Ressourcen- und Energiemanagement*, pages 409–419. Springer-Verlag, 2013.

[Kü13] C. Küpker. General Model Difference Calculation. Bachelor Thesis, Carl von Ossietzky University of Oldenburg, June 2013.

[Loe09] J. Loeliger. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. O'Reilly Media, 2009.

[Obj] Object Management Group. XMI Specification, v1. 2.

[SG08] M. Schmidt and T. Gloetzner. Constructing Difference Tools for Models Using the SiDiff Framework. *ICSE 2008*, pages 947–948, May 10-18 2008.

[SSA14] C. Seidl, I. Schaefer, and U. Aßmann. DeltaEcore - A Model-Based Delta Language Generation Framework. In *Modellierung*, pages 81–96, 2014.

[TBWK07] C. Treude, S. Berlik, S. Wenzel, and U. Kelter. Difference Computation of Large Models. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference*, pages 295–304. ACM Press, 2007.

[UML] UML: Unified Modeling Language. http://www.uml.org.

[XS05] Z. Xing and E. Stroulia. *UMLDiff: An Algorithm for Object-Oriented Design Differencing.*, pages 54–65. 6. ACM, 2005.