Delta-driven collaborative modeling

Maik Appeldorn, Dilshod Kuryazov and Andreas Winter
University of Oldenburg
Uhlhornsweg 84, 26129 Oldenburg, Germany
maik.appeldorn@uni-oldenburg.de
{kuryazov,winter}@se.uni-oldenburg.de

ABSTRACT

Model-Driven Engineering has already become a significant means in software development activities, which is well-suited to design and develop large-scale software systems. Developing and maintaining large-scale model-driven software systems entails a need for collaborative modeling by a large number of software designers and developers. As the first-class entities of the model-driven engineering paradigm, software models are constantly changed during development and maintenance. Collaborative modeling support for frequently synchronizing model changes between collaborators is required. Thereby, a solid change representation support for model changes plays an essential role for collaborative modeling systems. This paper applies a meta-model generic, operation-based and textual difference language to existing domain-specific modeling tool UML Designer and demonstrates a collaborative modeling application – CoMo. It is validated for UML activity diagrams.

1 MOTIVATION

Model-Driven Engineering (MDE) has become a popular means for software development which supports appropriate abstraction concepts to software development activities. It aims at improving the productivity of software development, maintenance activities, and communication among various teams and stakeholders. In MDE, *software models* are the first-class artifacts of software development rather than the source code implementing the systems. This leads to the several main benefits of MDE: the productivity boost, models become a single point of truth and are automatically kept up-to-date with the code they specify [24, pp. 9ff].

Software models (e.g. in UML – Unified Modeling Language [31]) are the key artifacts in MDE activities. In order to cope with the constantly growing amount of software artifacts and their complexity, software systems to be developed or maintained are usually shifted to abstract forms using the modeling concepts. Software models focus on the most relevant aspects of the problem domain to be designed and developed. Simultaneously, software models are the *documentation and implementation* of the software systems being developed and maintained [24].

Like the source code of software systems, software models are constantly evolved and maintained in order to cope with various user changes such as improvements, extensions, and optimization. All development and maintenance activities contribute to the *evolution of software models*. These activities may include creating new artifacts, removing existing ones, or/and changing the attributes of existing artifacts. Any type of the aforementioned changes that may occur during development and evolution of software models are referred to as *model changes*.

Development and maintenance of software models requires a need for *collaborative modeling* of several collaborators like managers, designers, developers, testers, and maintainers in order to accomplish successful results [21]. Several collaborators involved in software modeling apply changes to the shared software models. Collaborative modeling approaches have to provide support for communication among collaborators by: *change representation* for storing the histories of the changed model elements, *synchronization* of the represented model changes, and *management* of software models and their revisions stored in the repositories. These core features that collaborative modeling approaches have to provide are also addressed in [18] as the result of a solid and sophisticated literature study.

Depending on the nature of interaction, collaborative modeling can be divided into two main forms, namely *concurrent* and *sequential* collaborative modeling [14], [27]:

- *Concurrent Collaboration.* The essential and widely used scenario of collaborative modeling is the development of software models in real-time by several collaborators in parallel. The concurrent collaborative modeling is usually dedicated to creating, modifying and maintaining the huge, shared and centralized software models. These changes are synchronized in real-time. The real-time collaborative editing approaches are extensively investigated in textual document writing, e.g., Google Docs [19], Etherpad [4], Firepad [16], and many more. Unlike textual document editing, the increased performance of change synchronization matters in collaborative modeling because of the graph-like complex structures of software models. Thus, model changes have to be identified continually, represented and synchronized using very simple notations. Typically, these changes are constantly detected by listening for users actions on a particular model instance and produced in form of the small set of changes contained in *Modeling Deltas* [26]. Thus, modeling deltas are rather small in real-time collaboration and they are usually not stored, yet synchronized between the parallel working copies of software models. The real-time synchronization of modeling deltas is enabled by *micro-versioning* [27].
- *Sequential Collaboration.* Another significant scenario of collaborative modeling is sequential collaboration which is also referred to as *model management* in some literature [18]. Sequential collaboration intends to identify the model differences between the subsequent revisions of modeling artifacts, store and reuse them when needed. There are several classical source code version control (sequential collaboration) approaches such as Subversion [8], Git [34], etc. These systems for the source code of software systems are the best aid in handling development and evolution of large-scale, complex and continually evolving software systems. The same support is needed for software models. For

instance, while designing software models in concurrent collaboration, collaborators intend to store the correct and complete revision of their model and open it after a while to continue development and maintenance. These model management activities are facilitated by *macro-versioning* [27]. In macro-versioning, the differences between subsequent model revisions are represented in *Modeling Deltas*, as well. Modeling deltas consist of the larger set of model changes in macro-versioning, whereas they represent the small set of model changes in micro-versioning.

In both micro- and macro-versioning scenarios, modeling deltas are the first-class entities and play an essential role in synchronizing the small model changes between the parallel instances of models, storing/representing model changes in efficient ways, and managing models and their revisions. The both collaborative modeling scenarios might rely on the same underlying change representation approach to deal with modeling deltas. Therefore, the efficient representation of modeling deltas is crucial for both scenarios.

A meta-model generic, operation-based and textual change representation approach entitled *Difference Language (DL)* for representing modeling deltas in both micro- and macro-versioning was introduced in [26]. A DL-based collaborative modeling infrastructure was developed and applied to UML class diagrams in [27]. The proposed DL is meta-model generic, operation-based, modeling tool generic, reusable, applicable, and extensible. Moreover, the associated technical support further provides a catalog of supplementary services which allow for producing and reusing modeling deltas represented by DL. As long as there already are several advanced model designing tools, this paper applies the DL approach to Sirius-based [11] domain-specific modeling tool UML Designer [30] which results in the **Co**llaborative **Mo**deling (CoMo) application. UML activity diagrams [31] are considered as the domain language throughout this paper.

The remainder of this paper is structured as follows: Section 2 investigates existing related approaches in the research domain. Several characteristics of DL and its collaborative modeling application are defined in Section 3. The core concepts behind the DL-based collaborative modeling are depicted in Section 4 and the same section explains the subset of services provided by DL. Section 5 explains the CoMo collaborative modeling application of DL. Section 6 discusses adaptability and extendability of the DL-based collaborative modeling application. This paper ends up in Section 7 by drawing some conclusions.

2 RELATED APPROACHES

The problem of collaborative modeling and its change representation is the actively discussed and extensively addressed topic among the research community of software engineering. There is a large number of research papers addressing to collaborative modeling (Section 2.1) and model difference/change representation (Section 2.2).

2.1 Collaborative Systems

This section briefly reviews some collaborative development, document editing and modeling approaches in order to derive some general concepts and principals.

Concurrent collaboration systems record and synchronize change notifications during collaborative development. Collaborative document writing systems like Google Docs [19] and Etherpad [4] are widely used systems in concurrent document creation and editing. There are also several Web-based modeling tools e.g. GenMyModel [9] and Creately [7], which exchange changes over WebSockets. Their core ideas and underlying change representation techniques are not explicitly documented. Their modeling concepts and other services are not accessible, making them difficult to study and extend. The collaborative modeling approaches *emfCollab* [1] and Dawn [17] (the sub-component of CDO - Connected Data Objects [13]) provide collaborative development features for EMF models. However, they use custom serialization of changes for synchronization in concurrent modeling. Dawn utilizes standard relational databases to persist models under development.

Sequential collaboration systems (e.g., Git [34], Subversion [8]) are used for sequential revision control in source code-driven software development. There are also sequential collaborative modeling systems, e.g., EMF Store [20], SMoVer [3], AMOR [5] and Taentzer et. al. [35] that are discussed in Section 2.2.

For differentiating revisions and calculating differences, most of the collaborative systems for source code-driven software development use the Myer's LCS [28] algorithm, which is based on recursively finding the longest sequence of common lines in the list of lines of compared revisions. Software models can also be represented in textual formats using XMI exchange formats, but it is commonly agreed that the collaborative approaches for source code cannot sufficiently fit to MDE because of the paradigm shift between source code- and model-driven concepts [6], [33]. Differentiating the textual lines of the XMI-based software models can not provide sufficient information about the changes in associated and composite data structures of software models, as well as does not reflect the semantics of changes. As there are already outstanding collaborative systems for textual documents and the source code of software systems, MDE also requires support for generic, solid, configurable and extensible collaborative modeling applications for their development, maintenance, and evolution.

2.2 Delta Representation Approaches

As long as modeling delta representation is a decisively significant issue in developing collaborative environments, this section briefly reviews and classifies some delta representation approaches. The existing collaborative modeling approaches employ various techniques for model change representation in modeling deltas. Below, the existing approaches are classified and discussed according to their difference/change representation techniques.

The most approaches identify themselves as the operation-based difference representation. They usually use basic edit operations such as *create*, *delete* and *change* (or similar and more), which is a general concept being relevant to many difference representation approaches. Regardless of their difference representation techniques, they employ these basic operations only for recognizing the types of model changes, but information about model changes is generally stored in various forms as discussed below.

Model-based approaches represent modeling deltas using models. Cicchetti et al. [6] introduced a meta-model independent approach

that uses software models for representing model differences conforming to a difference meta-model. Cicchetti et al. also provides a service (incl. conflict resolution) to apply difference models to differentiated models in order to transform them from one revision to another. A fundamental approach to sequential model version control based on graph modifications introduced by Taentzer et. al. [35] is also used to represent model differences using models. The approach is validated in Adaptable Model Versioning System (AMOR) [5] for EMF models [33]. The major focus of the approach is differentiation and merging of software models that serve as the main foundations for sequential model version control.

Graph-based approaches represent model changes using internal graph-like structures. It is usually similar to model-based representation, but relying on the low-level graph-like structures. A generic approach to model difference calculation and representation using edit scripts is introduced in the SiDiff approach [23]. SiDiff consists of a chain of model differencing processes, including correspondence matching, difference derivation, and semantic lifting [22]. SiDiff does not rely on a specific modeling language. The SiDiff's difference calculator algorithm is also utilized by the approach in this paper to realized its delta calculator service (explained in Section 4.3).

Relational Database-based approaches represent model changes in classical relational databases. Likely, SMOVER [3] and Dawn [17] take advantage of relational databases to store model changes and to persist their models under development and evolution.

Text-based approaches represent model changes in modeling deltas by a sequence of edit operations in the textual forms embedding change-related difference information. An early text-based representation approach is introduced by Alanen and Porres [2]. EMF Store [20] is a model and data repository for EMF-based software models. The framework enables collaborative work of several modelers directly via peer-to-peer connection providing semantic version control of models. Dawn [17] and emfCollab [1] use their custom serialization of model changes for synchronizing them between collaborators. DeltaEcore [32] is a delta language generation framework and addresses the problem of generating delta modeling languages for software product lines and software ecosystems.

This section discusses the existing related approaches based on the criteria if representation by these approaches can provide small modeling deltas for both sequential and concurrent collaborative modeling. Besides, there are several approaches focusing only on some aspects of these collaborative modeling, and are not discussed in this paper.

The graph- and model-based representations of modeling deltas are more effective in case of the sequential collaboration and distributed concurrent collaboration. The modeling deltas represented by models or graphs usually consist of additional conceptual information for representing its modeling or graph concepts alongside actual change information. Thus, the model- and graph-based modeling deltas might not be as small (micro) as text-based modeling deltas. Modeling deltas have to be as small as possible to achieve higher performance (by rapid synchronization of modeling deltas) in concurrent collaborative modeling in real-time. During the evolutionary life-cycle, if all difference information is stored in a database, the database might become complex and large with an associated data set. Thus, the database-based representation approaches may require more implementation effort in identification and reuse of modeling deltas.

Modeling deltas represented in *textual forms* are the most likely to be small, especially well-suited to concurrent collaboration scenario. The textual modeling deltas are (1) directly executable descriptions of model changes; (2) easy to implement; (3) expressive, yet unambiguous providing necessary knowledge; (4) easy to synchronize with high performance; (5) easy serialization and deserialization by textual parser.

Literature review shows that research on modeling delta representation for collaborative modeling is still in its infancy. Considering the aforementioned discussions, this paper requests a textual *difference language (DL)* to represent modeling deltas in sequential and concurrent collaborative modeling.

3 CHARACTERISTICS

A meta-model generic, operation-based and textual *Difference Language (DL)* for representing modeling deltas in both micro- and macro-versioning was introduced in [26]. A collaborative modeling infrastructure based on DL was developed and applied to UML class diagram in [27], which resulted in a tool entitled *Kotelett*. DL further provides a catalog of supplementary services which allow to produce and reuse DL-based modeling deltas. The subset of these services *delta calculator (change listener feature), delta applier, model manager* and *delta synchronizer* are used in developing collaborative modeling application in this paper.

Several requirements *operation-based*, *meta-model generic*, *tool independent*, *delta-based*, *completeness*, *reusability* for DL and its services are described and fulfilled in [25]. Further requirements (*awareness of content and layout, genericness, supportiveness*) for the overall DL-collaborative modeling infrastructure are defined and satisfied in [27].

Since there are already several advanced domain-specific model designing tools, this paper applies the DL-based collaborative modeling infrastructure to Sirius-based [11] domain-specific modeling tool UML Designer [30]. The UML activity diagram [31] is considered as domain language throughout this paper. Before explaining the core application ideas, this section lists several further characteristics addressing the adaptability aspect of the approach.

Meta-model. DL is conceptually a family of domain-specific languages and generic with respect to the meta-models of modeling languages. As long as the modeling concepts of any modeling language can be recognized by looking at the meta-models of these languages, specific DLs for particular modeling languages are generated by importing their meta-models. Therefore, DL requires the meta-model of a modeling language, in this particular case, the meta-model of UML activity diagrams. DL supports the following characteristics to apply the DL-based collaborative modeling infrastructure to the modeling language:

- *DL Generation.* In order to apply the DL-based collaborative modeling infrastructure to a particular domain-specific language which is defined by its appropriate meta-model, the *DL generator service* (Section 4.3) generates a specific DL from the meta-model of a potential modeling language, UML activity diagram. Then,

the modeling deltas can be represented in terms of DL on the instance models conforming to that modeling language.
- *Service Adaptation.* DL supports several potential supplementary services (Section 4.3) for extending the application areas of DL. These services are capable of operating on the DL-based modeling deltas, e.g., calculating, applying, synchronizing, and managing modeling deltas. For applying the DL-based collaborative modeling infrastructure to domain-specific tool UML designer, delta calculator and delta applier services have to be adapted. The other services remain unchanged.

These characteristic are addressed in Section 4 to apply the DL-based collaborative modeling infrastructure to design UML activity diagrams in the UML Designer tool.

4 APPROACH

The DL-based collaborative modeling infrastructure considers model changes as the first-class entities for supporting concurrent collaboration by *micro-versioning* and sequential collaboration by *macro-versioning*. Thus, first of all, DL aims at representing model changes in modeling deltas in efficient ways.

DL is conceptually a family of domain-specific languages for representing model changes. A specific DL is derived from the meta-model of a modeling language. Since this paper applies the DL-based collaborative modeling infrastructure to designing UML activity diagrams in UML Designer, the approach requests the meta-model of activity diagram as initial prerequisite to generate a specific DL. Thereafter, the relevant DL services have to be adapted to be capable of handling new modeling languages in the new model designing tool.

Section 4.1 depicts the substructure of the UML activity diagram meta-model. Section 4.2 exemplifies a simplified running example for the DL-based change representation. Section 4.3 explains a subset of supplementary DL services required for applying collaborative modeling and how they are adapted to be handy in this particular case.

4.1 Meta-Model

The modeling concepts of any modeling language can be recognized by inspecting the meta-model of that language. Thus, a specific DL for UML activity diagram is generated by the *DL generator service* (explained in Section 4.3) importing the UML activity diagram meta-model. Then, the model changes in modeling deltas can be represented in terms of DL on the instance activity diagrams. However, the DL generator is generic with respect to the meta-models of modeling languages.

Figure 1 depicts the substructure of the UML activity diagram meta-model that is used throughout this section as a running example. The meta-model is separated into two parts by a dashed line. Below the line, it depicts the *content part* (i.e., abstract syntax) which is adapted from the standard UML activity diagram meta-model for EMF (Eclipse Modeling Framework) [33]. In graphical modeling, each modeling object has design information such as color, size, and position, also called *layout information*. Above the dashed line, Figure 1 portrays the *layout part* (i.e., concrete syntax). The layout part of the meta-model depicts the substructure of Graphical Modeling Framework (GMF) notation [12] which supports notation for developing visual modeling editors based on EMF. The *ActivityNode* and *ActivityEdge* of the content part are connected to the *Node* and *Edge* of the layout part through the *DNode* and *DEdge* artifacts of the Sirius *odesign* notation.

This way of designing meta-models allows for using the same collaborative modeling environment for different modeling contents. The complete meta-model is used for creating overall collaborative modeling application explained in Section 5.

4.2 Motivating Example

In order to express how DL is applied to change representation for activity diagrams, this section presents a simplified example for the DL-based change representation in modeling deltas. A very simple UML activity diagram is chosen as a running example to apply the DL approach. Figure 2 depicts three subsequent revisions namely `Rev_1`, `Rev_2` and `Rev_3` of the same UML activity diagram describing an `"Ordering System"` example. All model revisions conform to the same meta-model shown in Figure 1. Figure 2 further depicts two concurrent copies of the latest revision, in this case, `Rev_3`. Two designers, namely `Designer_1` and `Designer_2` are working on these parallel copies.

Globally Unique Identifiers. According to the *NamedElement* class of the meta-model in Figure 1, each modeling artifact has an attribute named *id*. In order to identify modeling artifacts and to represent referenced changes in modeling deltas, this identifier attribute is used in modeling delta operations. Assigning modeling artifacts to globally unique identifiers allows to identify and keep track of the modeling artifacts of evolving software models over time. The *inter-delta* references allow for tracing any particular modeling artifact by detecting the predecessor and successor artifacts of an initial modeling artifact. The *delta-model* references are used to refer to modeling artifacts from modeling deltas in applying modeling deltas to software models by the DL applier service (explained in Section 4.3).

Model Changes. In the first revision, the model consists of one *Opaque Action* named `"Receive"` as well as *Initial Node* and *Final Node*. All modeling artifacts are connected by *Control Flows*. While evolving from the first revision to the second, the following changes are made on the model: *Fork Node, Join Node*, two *Opaque Actions* named `"Fill Order"` and `"Send Invoice"` are created, the target end of the control flow `g5` is reconnected to the *Fork Node*, the name of the *Opaque Action* `g2` is changed, and several control flows are created connecting these nodes. The model again evolves into the third revision `Rev_3` after making the following changes: the target end of the control flow `g5` is reconnected to the *Opaque Action* `g7`, the target end of the control flow `g12` is reconnected to the *Activity Final Node*. The nodes *Fork Node*, *Join Node*, *Opaque Action* `"g8"` and the control flows `g10, g11, g13, g14` are deleted.

In the third, last revision, the model is then being further developed by two designers concurrently. `Designer_1` changes the names of the *Opaque Actions* `g2` and `g7` from `"Receive Order"` and `"Fill Order"` to `"Receive Orders"` and `"Fill Orders"`, respectively. These changes are then sent to the other instance in form of the DL-based modeling delta (Figure 6). On the other instance, `Designer_2` creates a new *Opaque Action* named `"Close Order"`. The same designer creates one *Control Flow* `g16` and reconnects

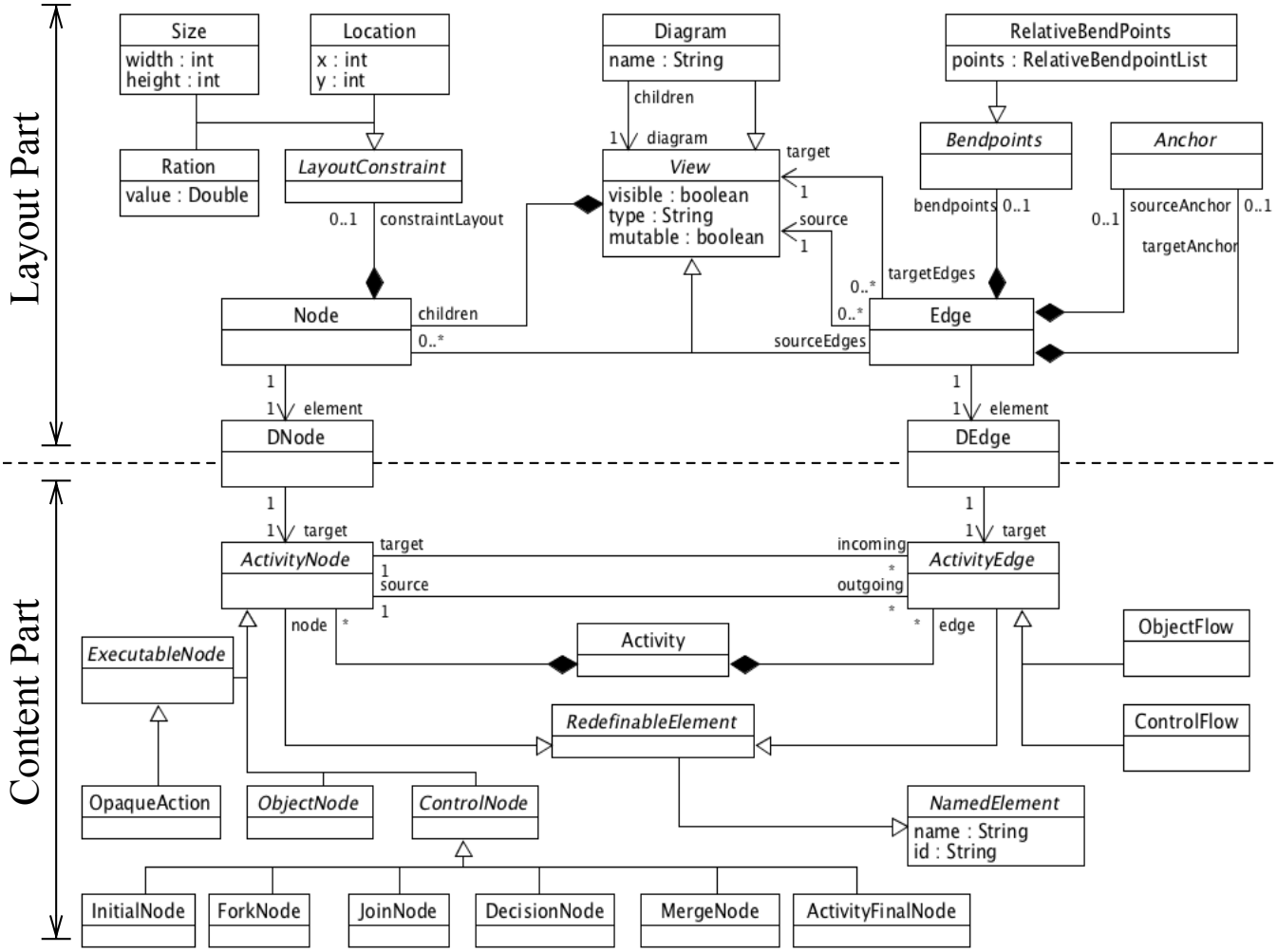

Figure 1: UML Activity Diagram Meta-model

the target end of the control flow g12 from the *Activity Final Node* g3 to the newly created node g15. These changes are then sent to the other instance, Designer_1 is working on, as the DL-based modeling delta (Figure 7).

Modeling Deltas in Sequential Collaboration. In sequential collaboration (cf. Subversion [8], Git [34]), the differences between subsequent revisions are usually identified and represented in reverse order, i.e., they represent changes in the *backward deltas* [26]. Because these systems intend to store differences as directly executable forms that is more practical in retrieving the earlier revisions of software systems. Since the latest revision is the most frequently accessed revision, they store the most recent revision of software systems and several differences deltas for tracing back to the earlier revisions. DL-based difference representation also follows the similar art of delta representation in its macro-versioning scenario.

The example depicted in Figure 2 describes the two backward deltas between three subsequent revisions (Rev_1, Rev_2 and Rev_3). These backward deltas (Figure 3) are directed in reverse order, i.e., application of the backward deltas to models transforms models into the earlier revisions. For instance, application of the backward delta in Figure 3 to Rev_3 results in Rev_2.

```
1  g6=createForkNode();
2  g8=createOpaqueAction("Send Invoice");
3  g9=createJoinNode();
4  g10=createControlFlow(g6,g7);
5  g11=createControlFlow(g6,g8);
6  g13=createControlFlow(g8,g9);
7  g14=createControlFlow(g9,g3);
8  g5.changeTarget(g6);
9  g12.changeTarget(g9);
```

Figure 3: Backward Delta between Rev_3 and Rev_2

According to the DL syntax, each delta operation contains a *Operator Part* (cf. *g6=createForkNode();*) which describes the *type of change* by means of *operations* (one of create, change, delete) [26] and an *Modeling Artifact* (cf. *g6=createForkNode();*) (with attributes

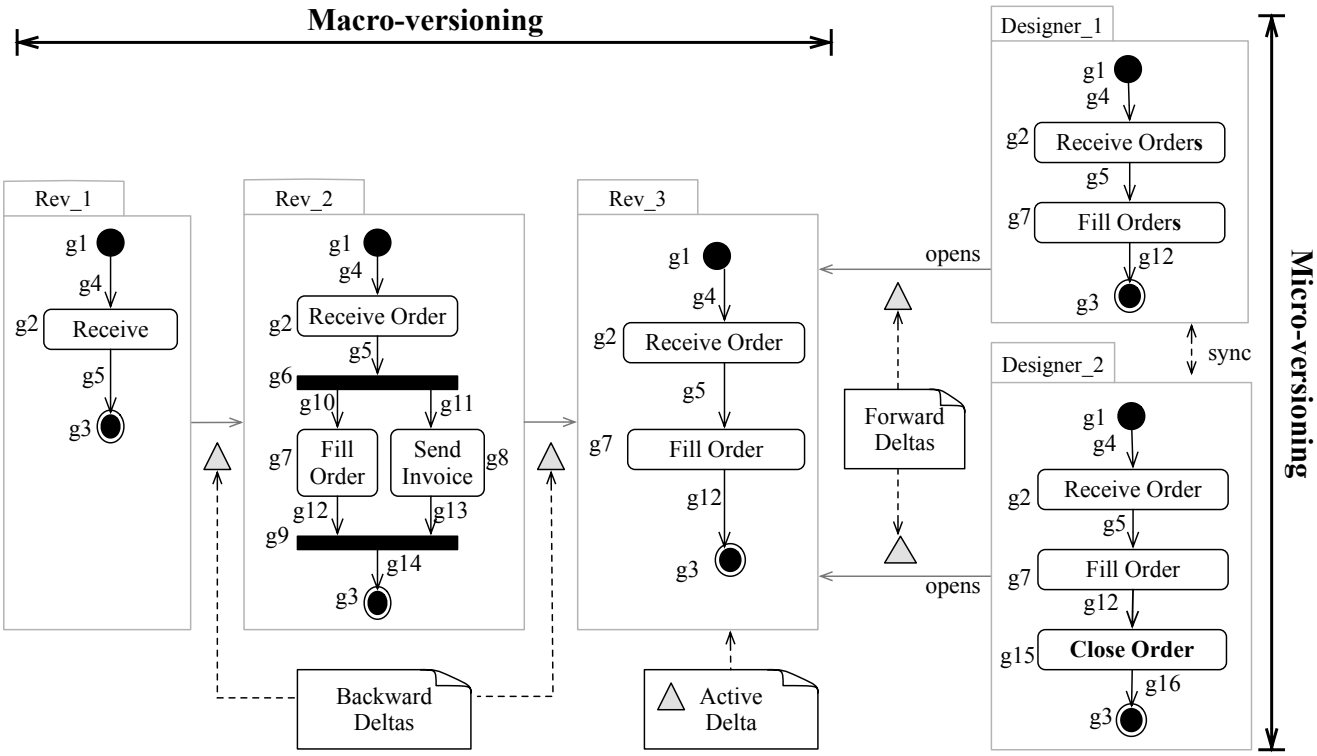

Figure 2: Simplified UML Activity Diagram

if required) which refers to model element. To refer to model elements from the DL operations in modeling deltas, *globally unique identifiers* of model elements are used as references (g_x).

Likewise, Figure 4 illustrates the backward modeling delta consisting of the differences between the second and first revisions.

```
1  g2.changeName("Receive");
2  g5.changeTarget(g3);
3  g10.delete();
4  g11.delete();
5  g12.delete();
6  g13.delete();
7  g14.delete();
8  g6.delete();
9  g7.delete();
10 g8.delete();
11 g9.delete();
```

Figure 4: Backward Delta between Rev_2 and Rev_1

The modeling deltas in these figures are directly executable descriptions of the model differences. Each of these difference deltas allows for reverting the base model to the earlier revision from the latter. The modeling delta in Figure 3 reverts the model to the second revision from the third, whereas the modeling delta in Figure 4 reverts the same model to the first revision from the second. In the DL-based modeling deltas, the unchanged modeling artifacts are implicitly excluded simply not describing DL operations for them. Furthermore, the modeling deltas depicted in these figures consist of further DL operations for representing changes on layout information, as well. For the sake of simplicity, these change operations are not depicted in the backward modeling deltas. However, the forward delta depicted in Figure 7 consist of DL-based change operations for layout information.

Active Delta. The source code-driven sequential collaborative systems usually store the working copy of software project and several (backward) deltas representing the differences between software revisions in their software repositories. DL introduces a new term *active delta* to store the working copies of software models in form of the modeling deltas, as well. Active deltas are the DL-based descriptions of the base revision (working copy) of a complete model. Active deltas consist of only *creation* operations. Execution of an active delta creates a complete model out of empty model.

The third revision of the model depicted in Figure 2 is represented by the *active delta* in Figure 5 which consists of only creation operations. When this active delta is executed on an empty model, the third, base revision of the model depicted in Figure 2 is created.

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

Figure 5: Active Delta

Modeling Deltas in Concurrent Collaboration. In case of the micro-versioning scenario depicted in Figure 2, the modeling deltas are described in the *forward forms* where application of the modeling deltas to a model results in the newer revisions of the same model. There, the changes made on two parallel instances by `Designer_1` and `Designer_2` are represented in the forward modeling deltas. These deltas are synchronized with other parallel model instances in order to update these instances into the new

states by the change descriptions in the forward deltas. For instance, Figure 6 depicts the forward delta consisting of the changes made by `Designer_1`.

```
1  g2.changeName("Receive Orders");
2  g6.changeName("Fill Orders");
```

Figure 6: Forward Delta of `Designer_1`

In this delta, `Designer_1` changes the name of both *Opaque Actions* from `"Receive Order"` and `"Fill Order"` to `"Receive Orders"` and `"Fill Orders"`, respectively. This example represents the model revisions where their changes are not yet synchronized with other parallel instance of the model.

In the same vein, Figure 7 depicts the forward delta representing the changes made by `Designer_2`. Unlike the previous modeling deltas, this forward delta depicts DL operations for layout information starting from line 4. There, the new nodes *Location* (with the attribute values of *x, y*) and *Size* (with the attribute values of *width, height*) are created for the newly created *Opaque Action* `g15`. On the last line, the *RelativeBendpoints* (with the values of the point list *sourceX, sourceY, targetX, targetY*) of *Control Flow* `g12` is changed.

```
1  g15=createOpaqueAction("Close Order");
2  g12.changeTarget(g15);
3  g16=createControlFlow(g15,g3);
4  g18=createLocation(14,38,g15);
5  g19=createSize(18,6,g15);
6  g20.changePoints([22,28,22,38],g12);
```

Figure 7: Forward Delta of `Designer_2`

In the concurrent collaborative modeling enabled by micro-versioning, modeling deltas are represented by the DL operations in the forward forms. These forward deltas have the forward effect in the models, i.e., the models are updated with the change descriptions defined in the forward deltas. Because, the recent changes made by other parallel mates have to be propagated on this particular instance in order to keep them up-to-date and vice verse. The forward deltas are rather small than backward deltas and are not stored in the repository. The collaborative modeling approach distinguishes between *forward* and *backward* deltas because of their convenience for *micro-versioning* and *macro-versioning*, respectively. The forward propagation of the change operations in modeling deltas is more practical in case of micro-versioning, whereas the backward affect of modeling deltas to models is more convenient in macro-versioning.

These modeling deltas are represented by the specific DL for UML activity diagrams generated from the meta-model depicted in Figure 1.

4.3 DL Services

DL intends to extend its application areas by introducing several supplementary services, i.e., *DL generator* for generating specific DLs for the given modeling languages, *delta calculator* for calculating modeling deltas, *delta applier* for applying modeling deltas to models, *model manager* for managing models and their revisions, etc. As all DL services are explained in [26] in detail, this section briefly revisits the services which are involved in applying the DL-based infrastructure to design UML activity diagrams in UML Designer. These potential services are explained how they are adapted to be reusable in this section.

DL Generator. The DL generator generates specific DLs for modeling languages by importing their meta-models. While generating the specific DL, it inspects all concrete (i.e., non-abstract) meta-classes of given meta-models and the attributes of these meta-classes.

A specific DL is always generated in form of the Model API including Java Interfaces and Implementations (incl. constructors) to recognize modeling objects, as well as Model Utility to operate on instance models. The interfaces of Model API are parameterized with the elements of a given meta-model and the change operations like *create, delete, change*. While generating specific DLs, the creation and deletion operations are generated for each meta-class. It implies that model elements conforming to these meta-classes can be created or deleted on instance models. The change operations are generated for only meta-attributes, i.e., the attributes of each modeling artifact can only be changed on instance models. Meta-relations are associated with the attributes of meta-classes. The DL generator considers three atomic operations *create*, *delete* and *change* as the sufficient set of operations for representing all model changes.

The DL generator does not generate methods to modify an attribute that is fixed as an unique identifier. Modification of the values of the identifier attribute (e.g., `id`) on the instance level is not permitted as it specifies the identity of modeling artifacts. Thus, their values should not be changed as the part of model changes. For instance, all identifiers g_x in the example in Section 4.2 are stored using the attribute `id` of the class *NamedElement* in Figure 1.

Collaborative Modeling Architecture. After generating a specific DL for the given meta-model, namely the UML activity diagram meta-model (incl. Content and Layout parts) in this case, the DL-based collaborative modeling infrastructure can be used to handle collaboration activities. It is built by the specific amalgamation of the several DL services as depicted in Figure 8. It shows the reference architecture for collaborative modeling including *server* and *client* sides. The same underlying reference architecture was also utilized in [26] to develop the `Kotelett` collaborative modeling application for UML class diagrams.

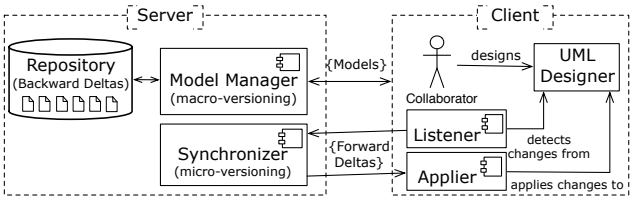

Figure 8: Reference Architecture for Collaborative Modeling

The server side consists of the *synchronizer* service to support *micro-versioning* and *model manager* service for *macro-versioning* which uses the DL-based modeling delta repository. On the client side, the changes made by collaborators are constantly detected by the change *listener* service while they are made using the model editor, in this case, UML Designer. These micro-versions are represented using forward modeling deltas and constantly sent to other parallel clients through the *synchronizer* service on the server. Once

these deltas arrive at other clients, they are applied to models by the *applier* service. As long as the synchronization of modeling deltas among clients carried out by the synchronizer on the server, the communication between collaborators is provided based on *star-topology*.

During collaboration, designers may store the particular state of their models whenever they are complete and correct. Collaborators are able to load models and their revisions that are saved earlier. The macro-versioning feature is provided by the *model manager* on the server.

Listener. The change listener service is the part of the DL delta calculator service. Calculation of modeling deltas depends on the scenario of collaboration whether it is macro-versioning or micro-versioning. In micro-versioning, the modeling deltas are produced by listening for changes in models by the change *listener*. Because changes have to be synchronized in real-time providing sufficiently high performance. In macro-versioning, modeling deltas between subsequent model revisions are calculated using the *state-based* comparison of subsequent revisions [23]. The state-based comparison and change listener are the two different features of the DL *delta calculator* service.

In case of the EMF-hosted UML Designer, the listener listens for *Notifications (org.eclipse.emf.common.notify.Notification)* which consists of information about the changed model elements and the change types. The change types in these notifications are mapped to the DL change types. The change types `ADD, ADD_MANY` are mapped to the `create` operation, `REMOVE, REMOVE_MANY` are mapped to `delete`, `SET, UNSET` are mapped to `change`, and `MOVE` is handled by the combination of the `create` and `delete` operations. The DL change listener is realized using the *Resource Set Listener* which listens for the *Transactional Editing Domain* of Sirius sessions. The use of transactional editing domain provides to perform the *redo/undo* operations without extra implementation effort.

Applier. In collaborative modeling, modeling deltas are applied to the base models in order to transform them from one revision to another. Application of modeling deltas to the base models is provided by the DL *delta applier* service [26]. In macro-versioning, the *model manager* utilizes the DL delta applier service to revert the older revisions of models by applying backward deltas. In case of the loss or damage of information on the working copies of models, designers may revert the earlier revisions or undo recent changes they made. The applier is also employed in micro-versioning in order to propagate model changes on the concurrent instances of shared models by applying forward deltas. The delta applier is further used by the model manager to load the working copies of models by applying active deltas to empty models. Like the listener, the DL delta applier is adapted to be useful in the EMF technical space. It converts the DL operations in modeling deltas to executable commands in the *Recording Command*. Then, these operations are executed in the *Transactional Command Stack* of *Transactional Editing Domain* of EMF.

Synchronizer. The server side of Figure 8 depicts the DL *synchronizer service* which allows for synchronization of micro-deltas between collaborators. Its primary task is to send modeling deltas to all connected clients except the sender. The synchronizer service is realized using the *KryoNet* API [15].

Model Manager. The repository on the server stores several backward deltas and one active delta for each software model under collaboration. The server provides *model manager* feature to operate on that repository. For instance, new models can be created in the repository, existing ones can be opened by collaborators to join collaboration, existing models can be deleted, revisions can be stored and loaded, etc.

5 APPLICATION

The collaborative modeling applications are developed by the specific orchestrations of the DL services explained in Section 4.3 and on the top of the DL-based delta representation. This section explains the collaborative modeling application `CoMo` (**Co**llaborative **Mo**deling) of DL.

The collaborative modeling application entitled `CoMo` is developed as an extension for Sirius-based domain-specific modeling tool UML Designer. `CoMo` takes advantage of the DL-based modeling deltas for synchronizing modeling deltas among the collaborators of the shared models. Figure 9 depicts a screenshot of `CoMo`. It displays two different tool instances working on the same model concurrently. These tool instances describe the exemplary micro-versioning scenario depicted in Figure 2 after the changes are synchronized. Each `CoMo` tool instance consists of several windows as explained below.

Figure 9 actually depicts the modeling user interface of UML Designer. As long as the DL-based collaborative modeling approach is applied to UML Designer which is an EMF- and Sirius-based domain-specific modeling tool [30], `CoMo` is completely realized using the EMF technical space.

After installing the `CoMo` support, two buttons appear in the tool as shown on the left instance under the indicator `CoMo`. When the first button (*Kol*) is clicked, the list of models currently available in the repository is displayed asking the user which model to join as collaborator. From the displayed dialog window, users can either select an existing model from the list or create a new model in the repository. If they select to join an existing model as a collaborator, that model is opened in the editor (D) and they can continue further developing that model. The users can open multiple models at once during collaboration.

In micro-versioning, the forward modeling deltas are not stored in the repository. However, reversion of changes in micro-versioning happens on the client side of the editor and provided by the *Redo/Undo* features of *Transactional Command Stack*. `CoMo` can save the model when the save button is clicked whenever the model is complete and correct. When the tool is asked to save the model by clicking the second button under the indicator `CoMo`, it calculates the differences (backward deltas) between the last and base revisions. Furthermore, new active deltas are also generated when the new revisions of models are stored. As the result of each click, one backward delta (representing differences between base and previous revisions) and one active delta (representing base working copy) are stored in the repository. This feature of the `CoMo` tool is currently under development.

The *model tree (A)* shows the list of models and diagrams (incl. the elements of these diagrams) the users are currently working on. The both instances display the modeling concepts of the UML

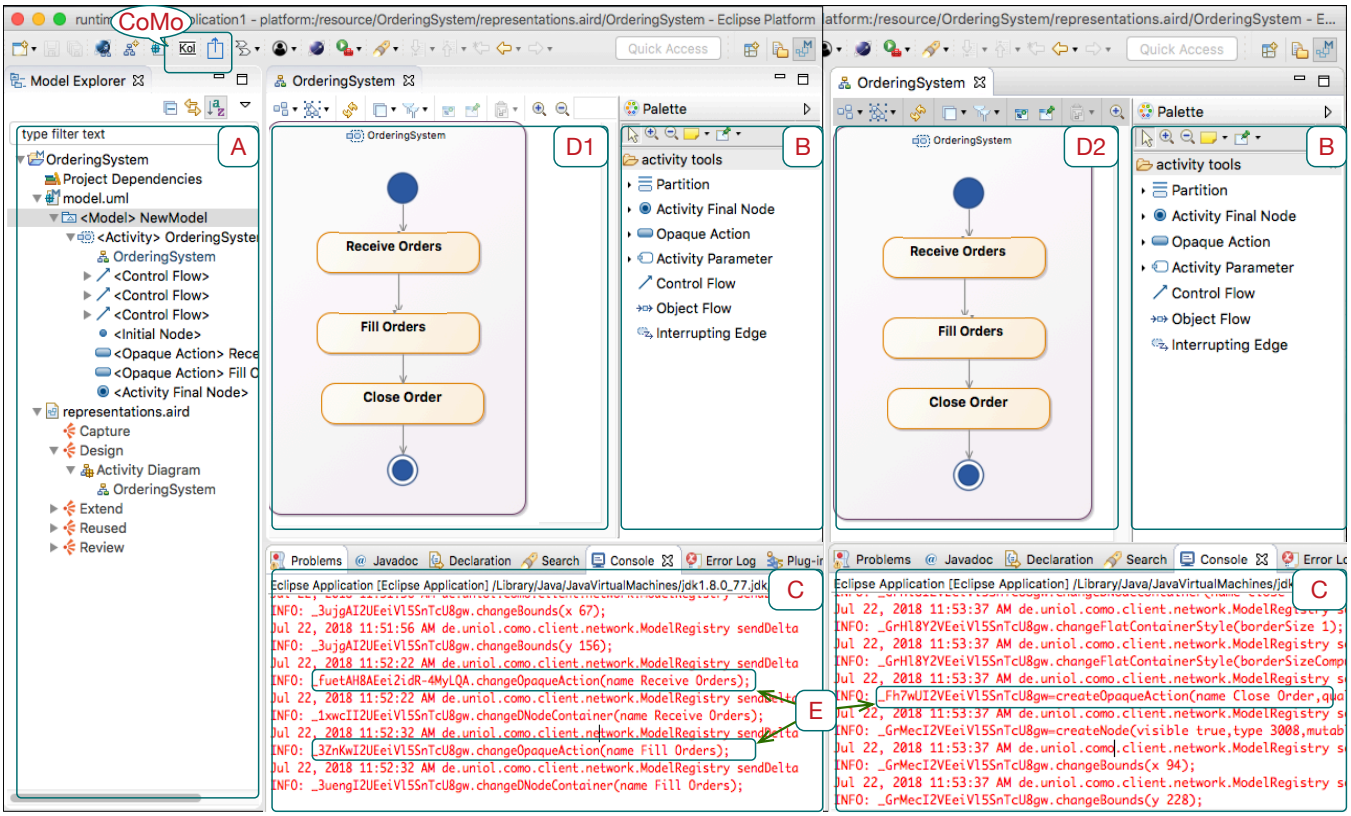

Figure 9: CoMo Screenshot

activity diagram (B). These concepts conform to the meta-model depicted in Figure 1. The logger window (C) constantly displays the modeling deltas that are exchanged among collaborators once changes are made in any instance. Creating one modeling artifact on the graphical modeling editor may result in one or many change operations that are contained in one modeling delta that is synchronized between collaborators. The *model editor* (D1 and D2) areas on the both instances show the exemplary activity diagrams that `Designer_1` and `Designer_2` are developing as depicted in Figure 2. In this case, these both instances are displayed after their changes are synchronized. The letter E indicates three changes made by two designers. These changes are: the name changes from `"Receive Order"` and `"Fill Order"` to `"Receive Orders"` and `"Fill Orders"` respectively (highlighted on the left logger), as well as the creation of new *Opaque Action* named `"Close Order"` (highlighted on the right logger).

Modeling deltas on the logger windows (C) are represented by the specific DL generated from the combined meta-model in Figure 1 including the standard UML profiles (content part, i.e., abstract syntax) and GMF notation (layout part, i.e., concrete syntax). The DL change listener and applier services are extended using EMF technical space features such as the command stack and resource set listener extensions for the editing domains. All other underlying technologies such as the DL synchronizer and model manager remain unchanged.

During experiments, the both DL applications `Kotelett` [27] and `CoMo` have shown sufficiently high performance by synchronization of small DL-based modeling deltas. So far, they have not faced any change conflicts in *micro-versioning*. This probably is attributed to the rapid synchronization of small modeling deltas. Thus, concurrent collaborative modeling enabled by the micro-versioning currently does not focus on the issue of conflict resolution. In the current implementations of concurrent collaborative modeling applications, the most recent changes are propagated on all parallel instances of models.

For merging various model revisions in *macro-versioning*, the `Kotelett` tool employs the existing merge technique provided by the `JGraLab` technical space [10]. The merge technique of `JGraLab` offers the semi-automated conflict resolution feature. The `CoMo` application aims at utilizing the existing EMF Diff/Merge approach [29] for model merging and conflict resolution in macro-versioning. However, the latter is still under development.

6 ADAPTABILITY

The DL-based collaborative modeling infrastructure can be adaptable by generating specific DLs and extending the relevant DL services.

- *Difference Language.* The approach provides the DL generator service for generating specific DLs. It is generic with respect to the meta-models of modeling languages, i.e., it is not language or tool specific. As depicted in Figure 1, the layout notation part (above the dashed line) of the meta-models enables adaptability of the approach for further modeling languages with the same layout notation. The modeling concept part (below the dashed

line) of the meta-model should be replaced by the meta-model of the appropriate modeling language. Eventually, the same layout information can be reused for further modeling languages.

- *DL Services.* The DL services are developed by following the service-oriented development principles. This allows to extend and adapt each independent service without affecting the rest of the underlying concepts and technologies of collaborative modeling. This enables tool developers to develop further services or adapt the existing ones as they want, and reuse the collaborative modeling environment without any further development effort. The DL services can also be replaced by other implementations and extended with additional features. The only prerequisite for these services is to recognize the syntax of DL. Eventually, these services can again be involved in service orchestrations for establishing the collaborative modeling applications.
- *DL Applications.* The DL applications, former `Kotelett` and current `CoMo`, are developed based on the same underlying reference architecture depicted in Figure 8. The DL services are orchestrated according to this reference architecture to develop the aforementioned DL applications. The same underlying reference architecture can be adapted and serve as a common blueprint for developing collaborative modeling environments for any (EMF-based) open source tools with lesser development effort.

7 CONCLUSION

A meta-model generic DL introduced in [26] is applied to UML class diagrams in [27], which resulted in the *Kotelett* tool. To demonstrate applicability of the DL-based collaborative modeling infrastructure, it is applied to UML Designer which resulted in the `CoMo` tool in this paper. DL is aware of both content (i.e., abstract syntax) and layout (i.e., concrete syntax) of modeling languages and provides sequential and concurrent collaborative modeling support, simultaneously. The proposed DL serves as a common change representation and exchange format for storing and synchronizing model changes between the subsequent and concurrent revisions of evolving models. The both sequential and concurrent collaborative modeling rely on the same base difference representation language (DL) for representing modeling deltas.

There are several EMF-based domain-specific modeling editors that can be used as a model designing tool for various modeling languages. However, they either lack collaborative modeling features or provide commercial solutions. They usually require open-source and operational collaborative modeling features with model repositories as a single point of truth. Thereby, the DL-based collaborative modeling infrastructure with its difference representation, services and reference architecture can be adapted and utilized for collaborative MDE.

REFERENCES

[1] A. Schmidt et al. 2011. visited on 22.07.2018. emfCollab: Collaborative Editing for EMF models. http://qgears.com/products/emfcollab/.

[2] M. Alanen and I. Porres. 2003. Difference and Union of Models. *In P.Stevens, J.Whittle, and G. Booch, editors, Proc. 6th Int. Conf. on the UML, Springer* LNCS 2863 (2003), 2–17.

[3] K. Altmanninger, A. Bergmayr, W. Schwinger, and G. Kotsis. 2007. Semantically enhanced conflict detection between model versions in SMoVer by example. In *Procs of the Int. Workshop on Semantic-Based Software Development at OOPSLA.*

[4] AppJet Inc. visited on 22.02.2018. Etherpad. Project Web Site: http://www.etherpad.com.

[5] P. Brosch, G. Kappel, M. Seidl, K. Wieland, M. Wimmer, H. Kargl, and P. Langer. 2010. Adaptable Model Versioning in Action. *in: Proc. Modellierung 2010, Klagenfurt, Austria* LNI 161 (March 24-26 2010), 221–236.

[6] A. Cicchetti, D. Di Ruscio, and A. Pierantonio. 2007. A Metamodel independent approach to difference representation. *journal of Object Technology* 6:9 (October 2007), 165–185.

[7] Cinergix Pty. visited on 22.07.2018. CreateLy. http://www.creately.com.

[8] B. Collins-Sussman, B. Fitzpatrick, and M. Pilato. 2004. Version Control with Subversion. *O'Reilly Media* (June 2004).

[9] M. Dirix, A. Muller, and V. Aranega. 2013. GenMyModel: UML case tool. In *ECOOP.*

[10] J. Ebert, V. Riediger, and A. Winter. 2008. Graph technology in reverse engineering. The TGraph approach. In *Proc. 10th Workshop Software Reengineering. GI Lecture Notes in Informatics.* Citeseer, 23–24.

[11] Eclipse Foundation. visited on 22.07.2018. Sirius. Project website: https://www.eclipse.org/sirius/.

[12] Eclipse Foundation, Project Web site. visited on 22.07.2018. Graphical Modeling Project (GMP). http://www.eclipse.org/modeling/gmp/.

[13] Eclipse Project Website. visited on 22.07.2018. EMF-based Model Repository: Corrected Data Objects (CDO). http://eclipse.org/cdo.

[14] C. Ellis, G. Simon, and R. Gail. 1991. Groupware: Some Issues and Experiences. *ACM* 34, 1 (1991), 39–58.

[15] Esoteric Software. visited on 22.07.2018. KryoNet. https://github.com/EsotericSoftware/kryonet.

[16] Firebase Inc. visited on 07.07.2018. Firepad. https://firepad.io.

[17] M. Fluegge. 2009. Entwicklung einer kollaborativen Erweiterung fuer GMF-Editoren auf Basis modellgetriebener und webbasierter Technologien. *Master's thesis, University of Applied Sciences Berlin* (2009). http://wiki.eclipse.org/Dawn

[18] M. Franzago, D. D. Ruscio, I. Malavolta, and H. Muccini. 2017. Collaborative Model-Driven Software Engineering: a Classification Framework and a Research Map. *IEEE Transactions on Software Engineering* (September 2017). https://doi.org/10.1109/TSE.2017.2755039

[19] Google Inc. visited on 07.07.2018. Google Docs. http://docs.google.com.

[20] J. Helming and M. Koegel. last accessed on 22.07.2018. EMFStore. Project web site. http://eclipse.org/emfstore.

[21] J. Herbsleb and D. Moitra. 2001. Global software development. *IEEE software* 18, 2 (2001), 16–20.

[22] T. Kehrer, U. Kelter, M. Ohrndorf, and T. Sollbach. 2012. Understanding model evolution through semantically lifting model differences with SiLift. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on.* IEEE, 638–641.

[23] T. Kehrer, M. Rindt, P. Pietsch, and U. Kelter. 2013. Generating Edit Operations for Profiled UML Models. In *MoDELS.* 30–39.

[24] A. Kleppe, J. Warmer, and W. Bast. 2003. *MDA Explained: The Model Driven Architecture: Practice and Promise.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[25] D. Kuryazov and A. Winter. 2014. Representing Model Differences by Delta Operations. In *18th International Enterprise Distributed Object Oriented Computing Conference, Worshops and Demonstrations (EDOCW), IEEE Computer Society Press, 2014, ISBN 978-1-4799-5467-4*, Manfred Reichert, Stefanie Rinderle-Ma, and Georg Grossmann (Eds.). Ulm, Germany, 211–220.

[26] D. Kuryazov and A. Winter. 2015. Collaborative Modeling Empowered By Modeling Deltas. In *Proceedings of the 3rd International Workshop on (Document) Changes: modeling, detection, storage and visualization.* ACM, 1–6.

[27] D. Kuryazov, A. Winter, and R. Reussner. 2018. Collaborative Modeling Enabled by Version Control. In *Modellierung 2018*, Ina Schaefer, Dimitris Karagiannis, and Andreas Vogelsang (Eds.), Vol. P-280. Gesellschaft für Informatik (GI), Bonn, 183–198. ISBN: 978-3-88579-674-9.

[28] E. W. Myers. 1986. An O (ND) difference algorithm and its variations. *Algorithmica* 1, 1 (1986), 251–266.

[29] O. Constant. visited on 22.07.2018. EMF Diff/Merge, Project Website. http://eclipse.org/diffmerge/.

[30] Obeo Network. visited on 22.07.2018. UML Designer. Project website: http://www.umldesigner.org.

[31] J. Raumbaugh, I. Jacobson, and G. Booch. 2004. *Unified Modeling Language Reference Manual.* Pearson Higher Education.

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

[33] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. 2008. *EMF: Eclipse Modeling Framework.* Addison-Wesley Longman Publishing Co., Inc.

[34] T. Swicegood. 2008. *Pragmatic version control using Git.* Pragmatic Bookshelf.

[35] G. Taentzer, C. Ermel, P. Langer, and M. Wimmer. 2012. A fundamental approach to model versioning based on graph modifications: from theory to implementation. *journal: Software and Systems Modeling* (April 25 2012).