DOCTOR OF SCIENCES

A distributed collaborative model editing framework for domain specific modeling languages

Koshima, Amanuel

Award date:
2016

Awarding institution:
University of Namur

A Distributed Collaborative Model Editing Framework for Domain Specific Modeling Languages

Amanuel Alemayehu Koshima

Faculty of Computer Science

University of Namur

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the subject of Computer Science

PReCISE Research Center 12 January 2016

Doctoral Committee

Prof. Dr. Wim Vanhoof Faculty of Computer Science University of Namur	Chair
Prof. Dr. Vincent Englebert Faculty of Computer Science University of Namur	Promoter
Prof. Dr. Philippe Thiran Faculty of Computer Science University of Namur	Co-Promoter
Prof. Dr. Anthony Cleve Faculty of Computer Science University of Namur	Internal Reviewer
Prof. Dr. Tom Mens Faculty of Sciences University of Mons	External Reviewer
Prof. Dr. Pieter Van Gorp Faculteit Industrial Engineering & Innovation Sciences Eindhoven University of Technology	External Reviewer

Acknowledgements

First and foremost, I would like to thank God almighty for his abundance blessings and supports that have made me who I am today. I also express my sincere gratitude to my promoter Prof. Vincent Englebert and co-promoter Prof. Philippe Thiran for giving me an opportunity to work under their supervision and accepting me as their teaching assistant. I could not have succeeded in this PhD thesis work without their advice and support. I specially thank them for gave me much freedom and encouragement to define the scope of the PhD thesis and to become more autonomous and proactive in my research. I also appreciate their reviews and feedback that I have received during writing different articles and this PhD dissertation.

Next, I would like to thank the jury members of my PhD examination: Prof. Wim Vanhoof, Prof. Anthony Cleve, Prof. Tom Mens, and Prof. Pieter Van Gorp. Many thanks also to my current and old colleagues, in particular Prof. Patrick Heymans, Dr. Mohamed Boukhebouze, Waldemar P. Ferreira Neto, Dr. Erbin Lim, Dr. Gilles Perrouin, Mihail-Octavian Staicu, Dr. Fabian Gilson, and Dr. Ebrahim Khalil Abbasi. I would like to express my sincere gratitude to Reddy Nababushana and Sophie Colpaert.

I am also thankful to my father Alemayehu Koshima, my mother Aberash Wakene, my sisters and bothers for their encouragements and supports. Last but not the least, I would also like to thank my wife Miheret Mulatu Meless for her abundance love and care. You have been my encouragement and strength during this journey, I thank you so much for testing and debugging the DiCoMEF framework with me ☺.

Lastly, I would like to thank all reviewers who participated in reviewing our published articles, your feedback was crucial for the realization of this PhD thesis work.

Abstract

Nowadays software solutions are becoming complex due to inherent complexities of problems they are dealing with (for example, a safety critical systems like a flight controller). Besides time-to-market pressures, user requirement changes during and after a development of software products, and evolution of the underlying software platforms increase complexities to the already complex problems. To mitigate these complexities, the Model Driven Engineering approach has been adopted by the software engineering community. Model Driven Engineering adopts separation of concern principles that reduce complexity, improve reusability, and ensure simpler evolution of modeling languages. It raises the level of abstraction of software development from technological details (i.e., source code and underlying platforms) to the problem domain. Indeed, it brings a new era of software development by shifting trends of software engineering from code-centric to model-centric. Model Driven Engineering uses Domain Specific Modeling Languages to describe concepts in a specific domain.

Despite the fact that Domain Specific Modeling tools are becoming very powerful and more frequently used, the support for their cooperation has not reached its full strength, and demand for model management is growing. In cooperative work, the decision agents are semi-autonomous and therefore a solution for reconciliating DSM after a concurrent evolution is needed. Conflict detection and reconciliation are important steps for merging of concurrently evolved (meta-)models in order to ensure collaboration. In this PhD thesis, we present a distributed collaborative model editing framework that supports concurrent editing of models and meta-models. This framework also supports a hierarchical collaboration among members of a collaborative ensemble. It captures edit operations of (meta-)model

whenever users adapt (meta-)models using a modeling language defined to capture history of modifications. The sequence of edit operations are used as a means to communicate work among members of a cooperative ensemble. In addition, the framework uses these operations to compare (meta-)models and to detect conflicts. The framework detects syntactic and static semantic conflicts, and it provides facilities to capture rationale of modifications using multimedia files that could help the conflict reconciliation process. Besides, the framework supports a role-based conflict reconciliation mechanism, where the evolution of (meta-)models is supervised by a human controller. In this framework, roles are dynamic and easily assigned to different users.

Résumé

Aujourd'hui, les solutions logicielles sont de plus en plus complexes en raison de la complexité croissante des problèmes rencontrés (par exemple, la sécurité de systèmes critiques comme les systèmes de commande de vol). En outre, la pression du marché, les modifications des exigences des utilisateurs avant voire après la phase de développement, ou encore l'évolution des plates-formes logicielles ajoutent encore de nouvelles dimensions à cette complexité. L'Ingénierie Dirigée par les Modèles (IDM) a été adoptée par la communauté de l'ingénierie logicielle afin de mitiger cette complexité, en prônant la séparation des préoccupations. L'IDM élève le niveau d'abstraction, en passant des détails technologiques (c-à-d., les codes sources et les plates-formes sous-jacentes) vers des abstractions plus proches du domaine d'application. À cette fin, l'IDM préconise l'usage de langages de modélisation spécifiques (LMS) pour décrire des concepts dans un domaine spécifique.

Bien que les outils propres aux LMS deviennent de plus en plus puissants et fréquents, il n'est pas aisé de les utiliser dans un contexte collaboratif alors que la demande se fait de plus en plus pressante. Les agents impliqués dans des tâches coopératives prennent des décisions de manière semi-autonome et une solution pour réconcilier des modèles spécifiques est indispensable. La détection de conflits et la réconciliation de modèles sont des étapes importantes pour la fusion de (méta)modèles qui sont subi des évolutions concurrentes. Dans cette thèse, nous allons présenter une plate-forme d'édition collaborative et distribuée de modèles qui supporte l'édition concurrente de modèles et leurs méta-modèles. Il supporte également un mode de collaboration hiérarchique. Il capture les opérations d'édition des (méta-)modèles en utilisant un langage spécifique pour la représentation de ce type d'historique. Les séquences d'opérations sont ensuite exploitées afin d'échanger l'objet des tâches coopératives parmi les membres d'un groupe collaboratif. Ces opérations sont également utilisées afin de comparer et détecter les conflits entre les (méta-)modèles. Cette plate-forme est capable de détecter les conflits tant syntaxiques que sémantiques (statique) et facilite la capture des intentions des modifications par le recours à des annotations multimedia qui sont ensuite utilisées pour aider le processus de réconciliation. Les utilisa-

teurs de cette plate-forme sont caractérisés par des rôles dont l'attribution est dynamique et peut être modifiée. Ils sont supervisés par un acteur jouant le rôle de superviseur.

Table of contents

List of figures

List of tables

Nomenclature

API Application Programming Interface

BPMN Business Process Model and Notation

CASE Computer-Aided Software Engineering

CRUD Create, Read, Update or Delete

CSCW Computer Supported Cooperative Work

CVS Concurrent Versions System

DiCoMEF Distributed Collaborative Model Editing Framework

DMS Data Management System

DSL Domain Specific Language

DSM Domain Specific Model

DSML Domain Specific Modeling Language

ECL Epsilon Comparison Language

EMF Eclipse Modeling Framework

ER Entity Relationship

GXL Graph eXchange Language

Jess Java Expert System Shell

JMS Java Message Service

MDA Model Driven Architecture

MDE Model Driven Engineering

MOF Meta-Object Facility

OCL Object Constraint Language

OMG Object Management Group

PIM Platform Independent Model

PNML Petri Net Markup Language

PSM Platform Specific Model

SOA Service Oriented Architecture

SVN Subversion

UML Unified Modeling Language

UUID Universal Unique Identifier

WYSIWIS What You See Is What I See

XMI XML Metadata Interchange

Chapter 1

Introduction

This chapter presents the motivation of the thesis in Section 1.1 and the problem statement in Section 1.2. Section 1.2.1 and Section 1.2.2 describe research restrictions and research questions respectively. Next, Section 1.2.3 explains the methodology and Section 1.3 demonstrates contributions of the thesis. In Section 1.4 we describe how the thesis is organized.

1.1 Motivation

Nowadays software solutions are becoming complex due to inherent complexities of problems they are dealing with (for example, a safety critical systems like a flight controller). Besides, there is a need to produce a high quality software within a short period of time (time-to-market pressure) and user requirements could evolve during and after a development of software products [Sriplakich, 2007]. To mitigate the aforementioned complexities, Model Driven Engineering (MDE) approach has been adopted by the software engineering community. *"MDE is about the use of relevant abstractions that help people focus on key details of a complex problem or solution combined with automation to support the analysis of both the problem and solution, along with the mechanism for combining the information collected from the various abstractions to construct a system correctly"* [Blackburn, 2008]. Abstraction is a natural cognitive process by which humans get a mental representation of a reality [Brambilla et al., 2012]. It facilitates communication between different users through

distinct views with various levels of detail [Blackburn, 2008]. Abstraction has been widely used in science and technology for many years, and it is commonly known as modeling.

Modeling is an act and a science of creating an abstraction of parts of a system understudy (SUS), a model. A model is a simplified or partial representation of a reality, and defined for a particular purpose(s) [Brambilla et al., 2012; Gonzalez-Perez and Henderson-Sellers, 2008]. The Collins English language dictionary [1] defines a model as *"a model of an object is a physical representation that shows how it looks like or how it works"* [Sinclair et al., 1987]. For example, Figure 1.1 shows a physical model of an automobile, which is also another physical system. This definition has a limitation, since non-physical systems like software prototypes can also be considered as models of a system to be developed (SUS). The Collins English language dictionary also provides another definition of a model, which states *"a model is a system that is being used and that people might want to copy in order to achieve similar results"*. For instance, software design patterns are considered as models based on this definition. Software design patterns specify reusable solutions for frequently occurring problems, such that users can copy and adapt the solutions so as to achieve similar results. Furthermore, *"a model of a system or process is a theoretical description that can help you understand how the system or process works, or how it might work"* [Sinclair et al., 1987]. This definition matches quite well with the usual definition of models in software engineering: Class diagram model, ER model, BPMN model, Statechart model, Petri Nets model, Enterprise Architecture model.

The structure of the model should be homomorphic with the structure of the SUS at some relevant level of details in order to interpret the model [Gonzalez-Perez and Henderson-Sellers, 2005, 2008]. Gonzalez-Perez at al. identify three kinds of interpretive mapping between models and SUSs: *isotypical mappings*, *prototypical mapping*, and *metatypical mapping*. Isotypical mappings states that there is one-to-one mapping between a model entity and an SUS entity, hence, the correspondence between the model entity and the SUS entity is straightforward. Figure 1.2 illustrates an isotypical mapping between the deployment model and the actual system to be developed, there is a one-to-one mapping between

[1] http://www.collinsdictionary.com/

[2] http://i.ytimg.com/vi/rZPE6yD3gPs/hqdefault.jpg

Figure 1.1 A physical model of an automobile [2]

the deployment model and the actual physical deployment of artifacts on nodes. Prototypical mapping establishes a one-to-many relationship between a model entity and a set of SUS entities. The model entity specifies examples of the kind of SUS entities, which can be matched with it. Figure 1.3 and Figure 1.4 demonstrate a prototypical mapping. Moreover, metatypical mapping declaratively describes mappings between one model entity and a set of SUS entities, it is an intensional definition. The SUS entities should satisfy properties specified by the model so as to be mapped to it. Figure 1.5 shows an example of metatypical mapping. Readers can find a detailed description about model in Chapter 2.

"The term Model-Driven Engineering (MDE) is typically used to describe software development approaches in which abstract models of software systems are created and systematically transformed to concrete implementations" [France and Rumpe, 2007]. However, abstraction is not new in computer science, for instance, it has existed and evolved with computer programming languages [Blackburn, 2008]. Different abstractions of protocols of

[3] Course material: http://directory.unamur.be/teaching/courses/INFOM434
[4] source: http://www.agilemodeling.com/artifacts/uiPrototype.htm
[5] source: https://en.wikipedia.org/wiki/Prototype-based_programming
[6] Course material: http://directory.unamur.be/teaching/courses/INFOM434

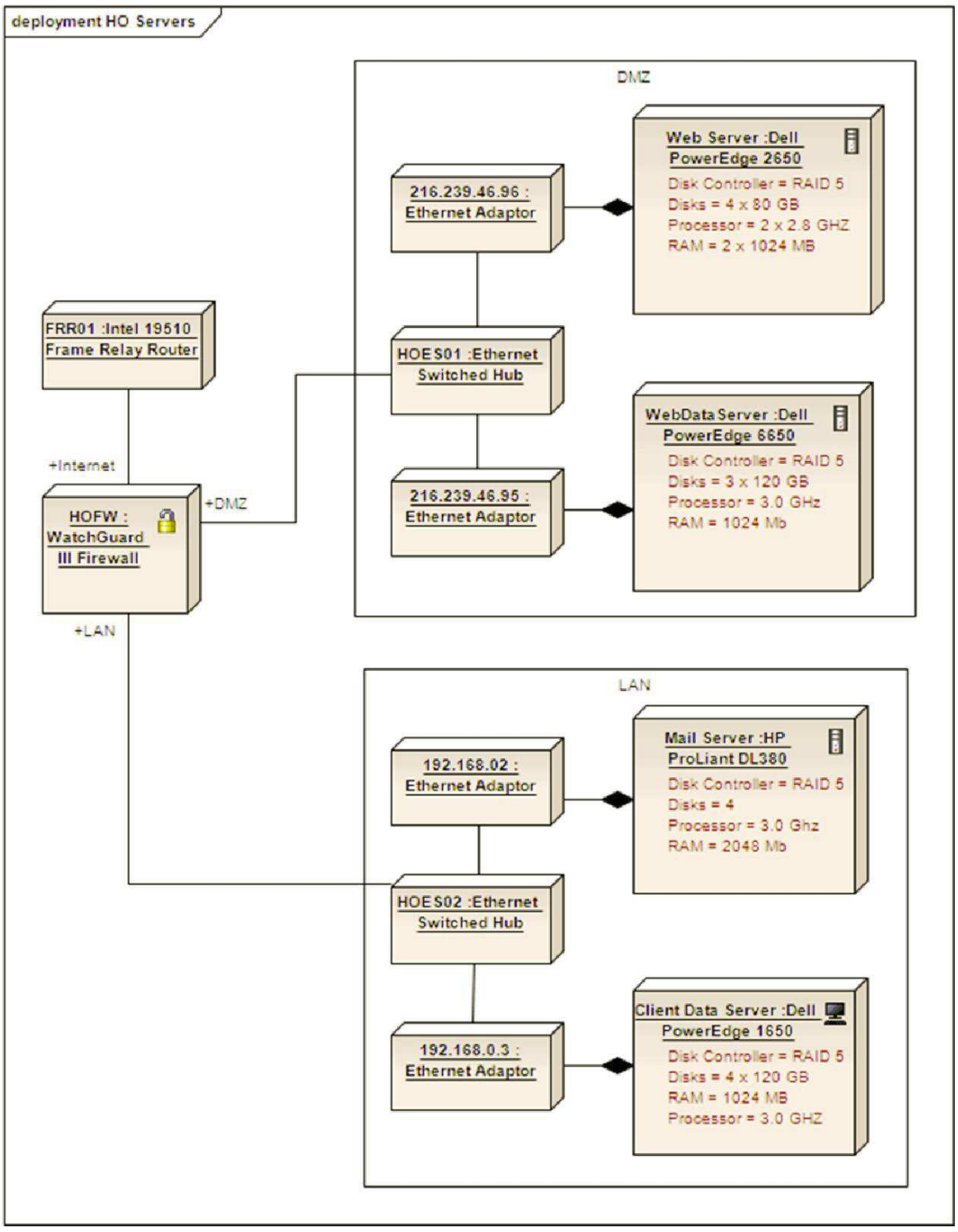

Figure 1.2 Isotypical mapping between the deployment model and the actual system to develop [3]

communication, concurrency concepts such as threads, specialized interfaces to hardware, and domain-specific functional details might be tangled together within a programming lan-

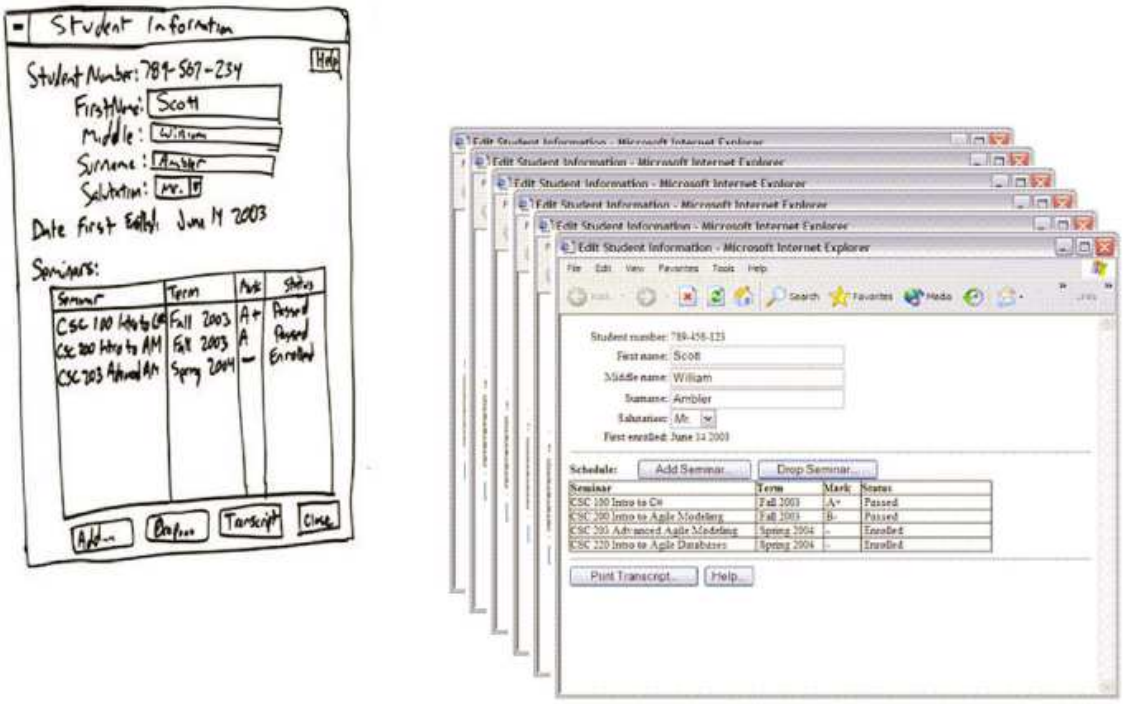

Figure 1.3 Example of prototypical mapping [4]

guages (e.g., spaghetti code), unless good development practices are applied to structure and layer the program. But, models give a way to systematically separate these views (abstractions) in MDE, since different types of models can only provide certain types of information. MDE also employs model transformation, model merging, automated model analysis, model simulation, and model execution so as to produce some type of computationally-based systems [Blackburn, 2008].

MDE shifts the level of abstraction of a software development from code-centric to model-centric [Bézivin, 2005; Favre, 2004; Kent, 2002]. In early adoption of MDE, Unified Modeling Language (UML) [Alanen and Porres, 2003], which is a general purpose modeling language, has been used to unify many modeling practices, however, the language has grown to become extremely large and complex. Besides, *"UML cannot fully define the relationships between diagrams and detailed behavior is difficult to define in UML"* [France and Rumpe, 2007]. Domain Specific Modeling Languages (DSML) have emerged to describe concepts in a specific domain [Brambilla et al., 2012; Schmidt, 2006]. For instance,

```
// Example of true prototypal inheritance style
// in JavaScript.

// "ex nihilo" object creation using the literal
// object notation {}.
var foo = {name: "foo", one: 1, two: 2};

// Another "ex nihilo" object.
var bar = {three: 3};

// Gecko and Webkit JavaScript engines can directly
// manipulate the internal prototype link.
// For the sake of simplicity, let us pretend
// that the following line works regardless of the
// engine used:
bar.__proto__ = foo; // foo is now the prototype of bar.

// If we try to access foo's properties from bar
// from now on, we'll succeed.
bar.one // Resolves to 1.

// The child object's properties are also accessible.
bar.three // Resolves to 3.

// Own properties shadow prototype properties
bar.name = "bar";
foo.name; // unaffected, resolves to "foo"
bar.name; // Resolves to "bar"
```

Figure 1.4 Javascript example of prototypical mapping [5]

DSMLs are used for specifying structures, behaviors, and requirements as well as code generation, rapid prototyping, and automated testing within specific domains. DSML describes a solution by directly using domain concepts rather than generic modeling languages. This helps domain experts to focus on specific concepts which are related to their field of expertise instead of underlying platforms. As a result, this approach improves the quality of software products, reduces the development costs, and increases the longevity of software

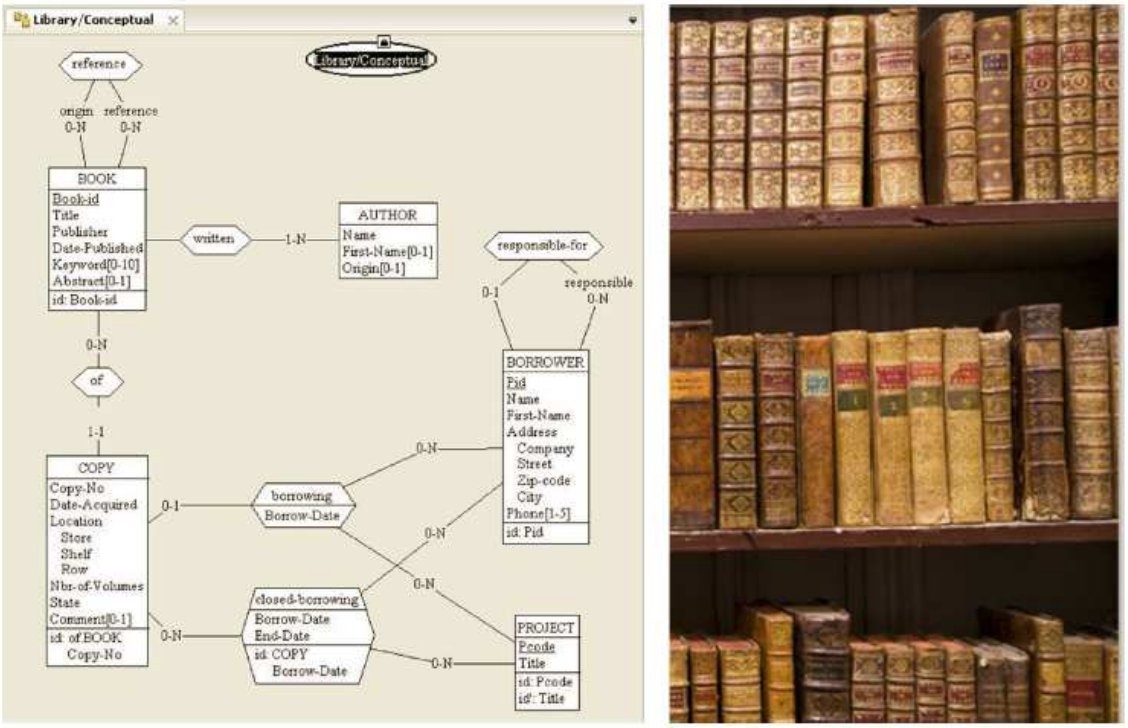

Figure 1.5 Example of metatypical mapping [6]

products [Frankel, 2003]. It reduces the cost of conception and development and improves the productivity five to ten times [Kelly and Tolvanen, 2008].

MDE describes concepts at different levels of abstractions using models, meta-models, and meta-meta-models. A model is an abstraction of a software system. A meta-model is a DSML oriented towards the representation of software development methodologies and endeavors [Gonzalez-Perez and Henderson-Sellers, 2008]. A meta-meta-model is a minimum set of concepts that define meta-models. Figure 1.6 illustrates the hierarchical relationships among model, meta-model, and meta-meta-model. A detailed description about meta-model and meta-meta-model is found in Chapter 2.

Modeling complex systems is far fetched for a single user to understand requirements and produce a quality product. Hence, there is a need for a group of users with different specializations to cooperate together. As stated by Zimmermann et al. [Zimmermann and Bird, 2012], whenever a complexity of a problem increases, then the diversity of users that are involved in a group increases. However, most of the DSM tools developed in the

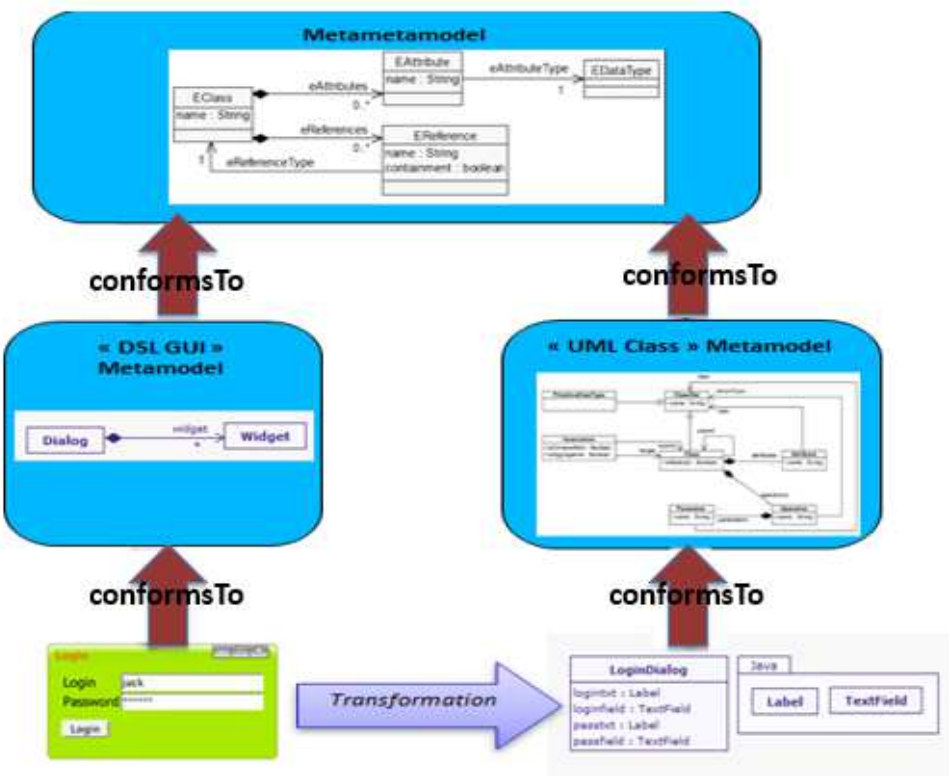

Figure 1.6 Example of MDE: model, meta-model, and meta-meta-model

past consider the modeling process as a single user task [Constantin et al., 2009]. This hypothesis is too restricting with regards to how projects are managed, hence, DSM tools should support collaborative modeling.

1.2 Problem Statement

As discussed in Section 1.1, modeling complex systems require collaboration among multiple users with different specializations. This means that (meta-)models need to be shared among multiple users (i.e., (meta-)modelers). Besides, these shared artifacts could be edited and could evolve concurrently throughout the development life cycle of the application. Eventually, these models become inconsistent with each other. Inconsistency is the main challenge that hinders collaborative modeling, because two inconsistent versions of (meta-)models cannot interoperate. Hence, conflicts that cause inconsistencies should be identified

and resolved. These conflicts could be *textual, syntactic, or semantic* [Altmanninger et al., 2009; Mens, 2002].

Textual conflict detection approach compares two or more text documents in order to detect conflicts between them. The granularity of the comparison might vary like a line of text, a paragraph, a sentence, a word, or a character [Altmanninger et al., 2009; Mens, 2002]. Syntactic conflict detection takes the syntax of the language (a tree or a graph structure) into account so as to detect conflicts [Altmanninger et al., 2009; Mens, 2002]. Refactoring of a software system might cause conflicts even though the changes are semantically equivalent. These conflicts need to be identified and resolved.

Static semantic conflicts could be specified with a formal language, afterwards, models are evaluated for conformance. For example, one static semantic constraint of a UML class diagram [OMG, 2011] requires that each class must have a name and there must be at most one class with a given name in the same package. This semantics is encoded as a constraint in UML class diagram using Object Constraint Language (OCL) [Warmer and Kleppe, 2003] and instances of the class (i.e., object) are evaluated to verify their conformity. Behavioral semantic conflicts are detected based on the execution behavior (runtime semantics) [Altmanninger et al., 2009; Mens, 2002]. In the literature, authors have used denotational semantics, operational semantics, program dependency graph, and program slicing to detect behavioral semantic conflicts.

Heterogeneity is another major factor that hampers interoperability (i.e. seamless cooperation among models, modeling tools) in collaborative modeling. In [Thiran et al., 1998], Thiran categorized heterogeneity at different levels such as *platform, data management system (DMS), location and semantics*. *Platform level* heterogeneity implies that the underlying technology such as hardware, operating system, or networking could be different. *DMS level* is about the technical detail of data implementation; data models might be defined using different conceptual modeling languages (i.e. with different expressive power). *Semantics level* heterogeneity occurs when equivalent concepts are modeled differently. *Location level* heterogeneity refers to the place where the data model resides (e.g. central or distributed).

Ouskel et al. identify the two main problems of interoperability as system heterogeneity and information heterogeneity [Ouksel and Sheth, 1999]. This classification is more elaborated in the context of model editing by Kühn et al. [Kühn and Murzek, 2006]. Model heterogeneity (i.e. meta-meta-model, meta-model, model) is classified as information heterogeneity. This heterogeneity is caused by using different modeling languages, implementation platforms, versions of models, etc. For instance, users exchange their work using method chunks or fragments [Ralyté and Roll, 2001] as a basis for communicating their works. These method chunks could be expressed in standard format like GXL [Holt et al., 2006], PNML [7], or XMI [OMG, 2007b]. Even though these tools define quite well the structure of data model, they vary in their semantics. This results in a problem of interoperability among different CASE tools. Most of the time, these types of interoperability problems are addressed using model transformations, specifically, exogenous model transformations, where the source and the target models are expressed in different modeling languages [Czarnecki and Helsen, 2006; Mens and Gorp, 2006; Mens et al., 2005a]. On the other hand, users can exchange models among the same family of CASE tools, specifically, it is a model transformation performed between models of the same modeling language (endogenous transformation) [Czarnecki and Helsen, 2006; Mens and Gorp, 2006; Mens et al., 2005a].

System heterogeneity concerns differences in access of services (i.e. API or files, standard format or proprietary), mechanisms (i.e. version management, multi language support, model analysis, and simulation), implementation platforms, and persistency services (i.e. durable storage). This PhD thesis considers a (meta-)model exchange between CASE tools of the same family, and system heterogeneity is not part of the interest of this work.

As discussed above, multiple users are involved in collaborative modeling who modify modeling artifacts in parallel. The collaborative modeling frameworks and environments should provide a concurrency support and handle conflicts. A central repository with merge mechanisms (optimistic approach) and locking facilities (pessimistic approach) are commonly used approaches to support concurrency, to handle inconsistency problems, and to

[7] http://www.pnml.org/

ensure collaboration [Mougenot et al., 2009]. Unfortunately, locking technique is inadequate for a large number of users who work in parallel [Altmanninger et al., 2009; Mens, 2002]. Besides, in practice, this technique takes much time for users to resolve conflicts [Altmanninger et al., 2009; Pilato et al., 2008]. In addition, this approach restricts users to be dependent on one repository. It could also introduce administration of access rights which might be cumbersome and cause user dissatisfaction.

Other modes of collaboration could consist in a group of people concerned by a cooperative task that is large, transient, not stable or even non deterministic [Schmidt and Bannon, 1992]. Besides, the interaction pattern among members of a group could be dynamic and users are semi-autonomous in their partial work. This type of collaboration allows each member to have his/her own copy of a shared work (i.e. (meta-)model) and carry on his/her activity in isolation with other users or a central authority. A user later communicates his/her work by sending messages to other members [Mougenot et al., 2009]. Implementing the exchange of method chunks [Ralyté and Roll, 2001] could serve as a basis for this mode of collaboration. This mode of collaboration gives users a better control over their data and addresses the problem of being dependent on a single repository. But, it is challenging to keep all copies of modeling artifacts consistent due to the fact that these modeling artifacts could be modified concurrently by users. In order to ensure collaborative modeling, communication among members needs to be managed and conflicts should be detected and reconciled.

A lot of research has been done in the past to mitigate complexities of software projects and ensure collaborative software development. Ignat et al. [Ignat et al., 2007a] compared different collaborative editing frameworks for text or tree based documents such as XML. In [Dewan and Hegde, 2007; Dewan and Riedl, 1993], a collaborative framework was proposed to edit source codes and collaboratively reconcile conflicts, and then merge the source codes into a new source code.

In the context of model editing, Saeki [Saeki, 2006] introduces the use of a versioning system to control and manage models and meta-models, which evolve independently. Saeki's work is mainly focused on keeping consistency between the current version of mod-

els with their respective meta-models. The author did not consider collaborative model editing in his work. EMFStore is a model repository for collaboratively editing EMF models which is implemented based on the premise of a central repository with copy-merge techniques [Koegel and Helming, 2010]. MetaEdit+ [Kelly, 1998] implements *Smart Mode Access Restricting Technology* (Smart Locks ©) to support concurrent access of shared modeling artifacts that are stored centrally.

In [De Lucia et al., 2007; Sriplakich et al., 2006], the authors present a collaborative model editing framework, which is based on a central server. As it was stated above, this type of collaboration limits developers to work on one central repository. In addition, developers sometimes prefer to work in isolation to avoid administrative hierarchy and interferences of other developers.

Constantin et al. propose a reconciliation framework for collaborative model editing [Constantin et al., 2009]. In their work, they suggest a weakly coupled mode of collaboration, where (meta-)models are managed in distributed fashion. But, the authors only provide a theoretical reconciliation framework to support collaborative work without providing a solution. In another work, Mougenot et al. [Mougenot et al., 2009] develop a peer-to-peer collaborative model editing framework called D-Praxis. D-Praxis adopts operation-based communication, where users exchange sequences of operations that adapt a meta-model. Besides, it implements automatic conflict resolution based on delete semantics and Lamport's clock [Lamport, 1978]. Nevertheless, this approach suffers from a similar problem of "lost-update". A user could lose his/her work because of a later modification performed by another member.

This PhD thesis has aimed at developing a distributed collaborative modeling framework. Of course, developing a collaborative modeling framework is complex [Benmouffok et al., 2009]. Specifically, ensuring a distributed collaborative framework where every member has his/her own local copy and working in isolation with other members makes the development even more complex. Models and meta-models need to be replicated at every member site and they could be edited concurrently. As a result, all replicas could be inconsistent. Therefore, the collaborative modeling framework should identify conflicts among

different versions of (meta-)models. These conflicts should also be resolved, but conflict resolution is a tedious and error prone task. Hence, the framework should assist users in resolving conflicts and merging conflicting versions of (meta-)models into a new version. Besides, since multiple engineers with different goals, strategies and experience levels are involved in collaborative modeling, their communication needs to be controlled so as to have effective collaborations. Users sometimes also need to work in a hierarchical mode of collaboration, where they work together to standardize (meta-)models through international standards, hence, the collaborative modeling framework should be flexible to support a hierarchical mode of collaboration.

1.2.1 Research Restrictions

In this PhD thesis, we take the following research restrictions that allow us to focus on the essence of the problem.

1. This work only considers models and meta-models. Source code files which could be generated from a model and other text files (i.e. configuration files) are not part of this study.

2. Models, which are supervised by different controllers, are considered as distinct (meta-)models, this means that there is no explicit dependency between them.

3. Each model element has a universal unique identifier.

4. This work studies only syntactic and static semantic conflicts.

1.2.2 Research Questions

We believe that the following questions must be answered to provide the envisioned distributed collaborative model editing framework.

1. *How can we manage concurrent modifications of model and meta-models?* Models and meta-models should be shared among members of a collaborative ensemble.

Besides, there should be a facility for users to edit and elaborate shared modeling artifacts concurrently. Modifications of shared artifacts could raise conflicts. Hence, conflicting modifications of models should be identified, reconciled and merged into a new version.

2. *How can we manage members of a cooperative group?* the organizational structure of the cooperative group should be modeled so as to manage activities of members and their communications.

1.2.3 Methodology

This section presents the methodology adopted by this work to answer the research questions. In order to answer the first research question, *"How can we manage concurrent modifications of model and meta-models?"*, we have used different model management activities such as model comparison, conflict detection, conflict reconciliation, model merging, and model versioning. To answer the second research question, *"How can we manage members of cooperative group?"*, we adopted a role-based organization of social structure to manage communication of members in the cooperative ensemble. The following subsections provide detail description about the methodology.

1.2.3.1 Model Management

Model comparison : Model comparison compares two models so as to drive their differences. *State-based comparison* and *change-base comparison* are the most commonly used approaches [Altmanninger et al., 2009; Conradi and Westfechtel, 1998; Mens, 2002]. State-based comparison takes states of two versions of models with a same ancestor as an input and derive their differences. This process is commonly referred to as differencing, and it is computationally expensive [Koegel et al., 2009c]. Change-based comparison keeps track of changes whenever they occur, and then it stores them into a repository. As such, there is no need to calculate deltas later [Altmanninger et al., 2009; Conradi and Westfechtel, 1998; Koegel et al., 2009c; Mens, 2002].

Operation-based comparison is a special type of change-based comparison where deltas are represented as a sequence of change-operations [Conradi and Westfechtel, 1998]. Operation-based comparison captures the exact time sequences of changes that could help to understand changes and detect conflicts [Mens, 2002]. Besides, it can also express sets of operations that occurred in a common context as composite operations. According to Koegel et al. [Koegel et al., 2009c], time sequences of changes and composite operations help users to understand changes more easily in operation-based comparison than in state-based comparison. In this work, we opted for the second approach. We defined a domain specific modeling language, DiCoMEF history meta-model, to capture edit operations of (meta-)model adaptations (See Chapter 5 Section 5.2.2). Chapter 4 Section 4.1.1 provides a detailed description of model comparison.

Conflict Detection and Resolution : According to [Altmanninger et al., 2009], conflicts are a set of operations that cause an inconsistency. These conflicts could be *textual, syntactic, composite, or semantic* conflicts [Mens, 2002]. In this work, we provide a formal definition of conflicts using Set theory (see Chapter 5 Section 5.2.5), and our conflict detection approach detects syntactic and static semantic conflicts. These conflicts might be solved manually, semi-automatically or automatically. Manual conflict resolution is time consuming and error prone to deal with large and complex models. Automatic conflict resolution is not applicable in most situations, because conflict detection and resolution is usually domain specific [Altmanninger et al., 2009]. Therefore, we adopt a semi-automatic conflict reconciliation approach in this work. Chapter 4 Section 4.1.2 and Section 4.1.3 explains in detail about conflict detection and reconciliation.

Model Merging : Merging is a process of integrating concurrently edited models that have the same ancestor (i.e. the same base model), into a new (meta-)model. The most commonly used merging techniques are *raw merge, two-way merge and three-way* merge [Altmanninger et al., 2009; Mens, 2002]. The last one gives the better result as compared to the other two approaches. Therefore, we adopted a three-way mergining technique in our work. Chapter 4 Section 4.1.4 presents a detailed description about model merging.

Model versioning : Model versioning is a crucial activity to manage the history of model evolution and to ensure collaborative modeling [Conradi and Westfechtel, 1997, 1998; Roebuck, 2011]. A version space defines all versions of (meta-)models and their relationships. The difference between two successive versions is represented as a delta. This delta could be a *symmetric delta* and a *directed delta* [Conradi and Westfechtel, 1997, 1998]. The symmetric delta represents differences between the state of two versions of (meta-)models, and it is a state-based versioning. On the other hand, the direct delta is an operation-based versioning that denotes a sequence of edit operations that adapt (meta-)models. In this work, we adopted an operation-based versioning approach. Chapter 4 Section 4.1.5 presents a detailed description about model versioning.

1.2.3.2 Communication Management

Member organization: We adopt a role based modeling of social structures and interactions among members of the cooperative group. Modeling of social structures is important to analyze users' collaboration. Besides, it is helpful to reconcile conflicting activities [Penichet et al., 2007]. Chapter 4 Section 4.2.1 describes in detail the organizational structure of member organization.

Communication of Members: In distributed collaborative environment, engineers can communicate their activities in a peer-to-peer communication mode, but it is difficult to keep all local models consistent due to concurrent modifications of models. Another mode of change propagation is where every developer sends sequences of changes to a controller. Afterwards, the controller supervises modifications and propagates accepted changes to other members. We adopted the second mode of communication and will investigate different policies on how to apply propagated changes on all local copies. See Chapter 4 Section 4.2.2 for a detailed description about communication management.

1.3 Contributions

This PhD thesis presents a distributed collaborative model editing framework called DiCoMEF. This framework manages any models based on EMF/Ecore meta-model. In DiCoMEF, both models and meta-models are freely distributed without having constraints like a central repository. Engineers will be able to carry out their work independently without prior consultation of other users of the same (meta-)model. DiCoMEF uses a role-based modeling to represent a social structure of cooperative ensembles, which could facilitate conflict reconciliation and avoid chaos. For instance, it relies on human agents (controllers) to manage evolution of (meta-)model depending of their role in the cooperative work. The model controller supervises evolution of models, whereas the meta-model controller manages evolution of meta-models. Model (respectively meta-model) controller roles are flexible meaning that they can be assigned (delegated) to other members of a group as long as there is one unique coordinator per group. This dynamic role assignment could lead people to implement more elaborated strategies on top of DiCoMEF, i.e., a user can delegate his/her role to another person. Although using a controller to manage collaborative modeling may limit the scalability, it could be possible to implement different method engineering techniques (e.g., delegation mechanisms, pooling) and strategies on top of DiCoMEF to address the problem.

DiCoMEF captures elementary change operations (create, delete, and update) locally whenever members of a group modify their local (meta-)models. It defines a DSML to capture histories of edit operations, by extending the history meta-model of Edapt [8]. Edapt is an operation-based model migration framework that migrates instance models after changing a meta-model. These edit operations are used later as a means of communication among members of a group. In DiCoMEF, a controller is a central hub of communication in a cooperative ensemble, but members could still communicate with other colleagues directly. Peer-to-peer communication could hinder convergences of all copies of (meta-)models. We also formally define model, meta-model, edit operations, and conflicts using Set theory. Besides, we provide a conflicting set table to detect structural conflicts (using edit operations).

[8] http://www.eclipse.org/edapt/

Edit operations can be annotated with multimedia files to describe rationale of modifications, which could help to facilitate conflict reconciliation process. The framework also detects static semantic conflicts by relaying on EMF validation framework.

DiCoMEF relies on two concepts (i.e., main-line and branches) to ensure the communication framework. The main-line stores different versions of a copy (meta-)model locally at each editors site. Editors cannot modify (meta-)models stored on the main-line; they can only adapt those stored on the branch and send then their local modifications to a controller as "change requests" so as to commit changes on the main-line.

The DiCoMEF framework could be extended to support a large community of users, where an editor acts as a virtual controller for other editors (side editors) working under her/his supervision. These new roles (i.e., virtual controller and side editor) are transparent for the DiCoMEF controller. Side editors could also modify (meta-)models concurrently (e.g. by using the Cloud) but these modifications would be out of the scope of DiCoMEF. Read Chapter 5 Section 5.1 for a detail description of DiCoMEF.

The DiCoMEF framework has been implemented as an Eclipse plugin (54K LOC) that fully supports collaborative meta-modeling tasks. The support of instance models is still under implementation. The plugin, screenshots and other publications of DiCoMEF can be found in the DiCoMEF Web site [9]. We evaluated the DiCoMEF framework with master students with regards to the following criteria: (1) the feasibility of collaborative methods and processes with DiCoMEF, (2) the correctness of conflict detection mechanisms (recall and precision), (3) the usability of the merge tool and DiCoMEF framework, (4) measuring user efforts (time) needed to merge concurrently edited meta-models either manually or by using the DiCoMEF merge tool. This preliminary evaluation reveals overall positive results. The results indicate that the collaborative process of DiCoMEF is feasible and that the merge tool is usable (user friendly), correct, and helpful in the resolution of conflicts. In future work, we will provide a full support for collaborative modeling of instance models. Besides, more advanced collaborative workflows will be investigated and defined on top of the DiCoMEF framework. Furthermore, we will continue to conduct more experiments and

[9] https://sites.google.com/site/dicomef

evaluations. We will also improve the framework based on feedback from the preliminary evaluation results.

1.4 Thesis Organization

The remaining part of the thesis is organized as follows: Chapter 2 and Chapter 3 present preliminaries of domain specific modeling languages and computer supported collaborative work, respectively. Next, Chapter 4 presents a detailed overview of state-of-the-arts of model management, members management, and collaborative modeling. Subsequently, Chapter 5 and Chapter 6 describes the DiCoMEF framework and the evaluation of the framework. Finally, Chapter 7 gives the direction of the future work and conclusions.

Chapter 2

Model Driven Engineering

This chapter presents Model Driven Engineering (MDE) in Section 2.1. Next Section 2.2 describes Model Driven Architecture (MDA), which is the OMG initiative of MDE. Finally, Section 2.3 presents briefly about other MDE architectures proposed by different authors.

2.1 Model Driven Engineering

In today's modern business, industries provide sophisticated enterprise-scale software solutions, which deal with complex business domains. According to Tan et al. [Tan et al., 2007] *"Building enterprise-scale software solutions has never been easy. The difficulties of understanding highly complex business domains are typically compounded with all the challenges of managing a development effort involving large teams of engineers over multiple phases of a project spanning many months. The time-to-market pressures inherent to many of today's product development efforts only serve to compound the problems. In addition to the scale and complexity of many of these efforts, there is also great complexity to the software platforms for which enterprise-scale software are targeted."*

Model Driven Engineering (MDE) is a software engineering methodology that is adopted to deal with an ever increasing complexity of software solutions. MDE adopts separation of concern principles that reduces complexity, improves reusability, and ensures simpler evolution of modeling languages [Tarr et al., 1999]. MDE raises the level of abstractions

of software development from technological details (i.e., source codes and underlying platforms) to the problem domain. Indeed, MDE brings a new era of software development by shifting trends of software engineering from code-centric to model-centric [Bézivin, 2005; Bézivin, 2004; Favre, 2004; Kent, 2002].

In MDE, models are the principal artifacts that give full descriptions of software systems and are used for analysis, simulation, and source code generation of a software system [Meyers and Vangheluwe, 2011]- *"everything is a model"* [Bézivin, 2005]. Domain concepts are defined using well-suited modeling languages at acceptable level of abstraction [Meyers and Vangheluwe, 2011]. Indeed, G. Booch et al. said *"the full value of MDA is only achieved when the modeling concepts map directly to domain concepts rather than computer technology concepts"* [Booch et al., 2004]. Then, specifying domain concepts using Domain Specific Language (DSL) reduces accidental complexities [Brooks, 1987] that could arises during development of software systems.

DSML specifies structures, behaviors and requirements of applications within a specific domain. DSMLs are more close to the domain concepts that need to be modeled [Schmidt, 2006]. Therefore, domain experts concentrate on particular concepts which are related to their field of expertise instead of underlying platforms (software/hardware). For instance, in Figure 2.1, a domain expert (i.e., in micro-controller, mobile application, or SOA architecture) more easily understands concepts expressed using DSMLs than using a general purpose languages like UML [OMG, 2007a]. As a result, the quality of the software product improves and the development cost decreases and longevity of the software product increases [Frankel, 2003].

2.2 Model Driven Architecture

Model Driven Architecture (MDA) is the MDE initiative of Object Management Group (OMG® [1]) [Frankel, 2003; Kleppe et al., 2003; OMG, 2001; Stahl et al., 2006]. MDA relies on OMG® standards to provide open and vendor-independent solutions, which alleviate the

[1] http://www.omg.org/

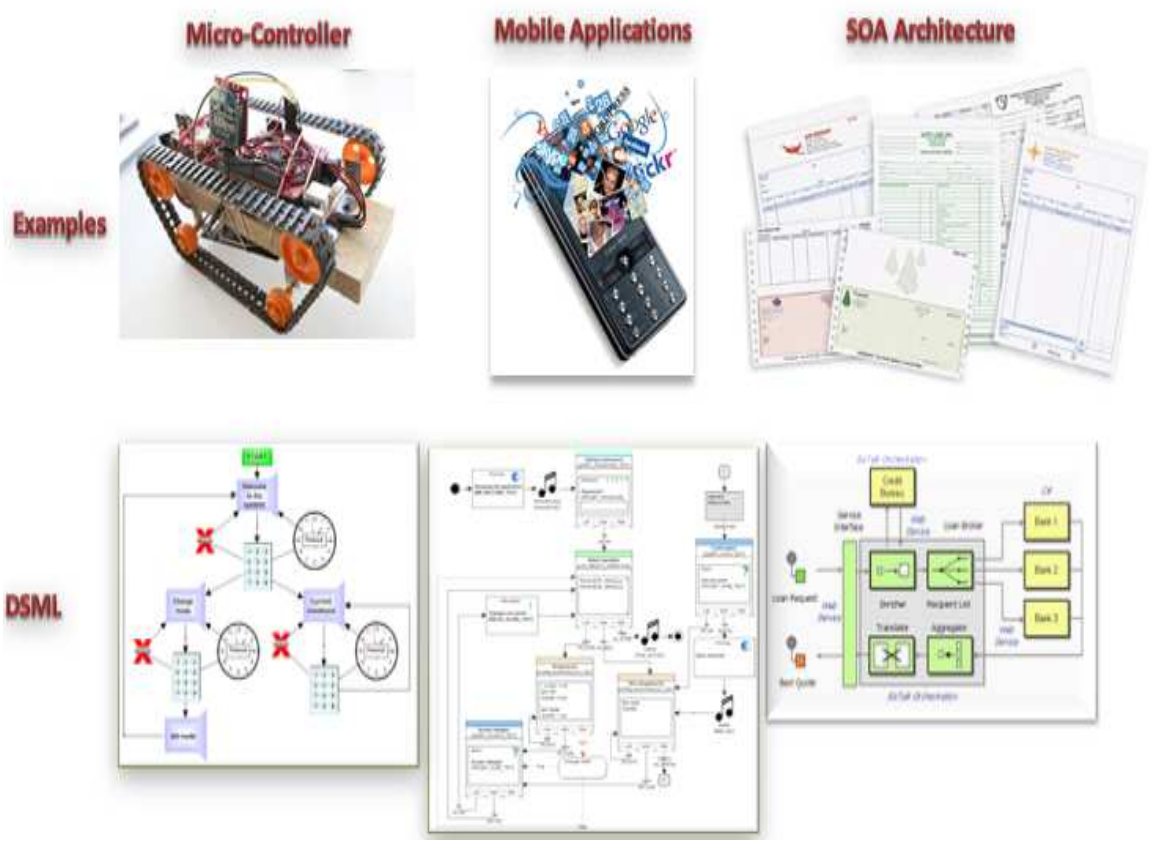

Figure 2.1 DSML Example [Micro-controller and Mobile application, Kelly and Tolvanen, 2008] [SOA architecture, Hohpe and Woolf, 2003]

aforementioned problems of complexities. The MDA uses a Platform Independent Model (PIM) to represent business logic and functionality of a system [Kent, 2002]. The PIM formally specifies the structure and behavior of an application without considering the underlying platform or implementation details. This means that the PIM insulates the core of an application from its technical details.

A Platform Specific Model (PSM) formally describes underlying platforms such as software and hardware [Kent, 2002]. The PSM is derived from PIM by applying one or more model transformations. Consequently, the source code is generated from PSM using model-to-code (model-to-text) transformation (see Figure 2.2). Model transformation automatically generates a target model from a source model based on a transformation definition [Gomes et al., 2014; Kleppe et al., 2003; Mens and Gorp, 2006; Mens et al., 2005a; Stahl et al., 2006]. The separation of specifications of system into PIM and PSM has advan-

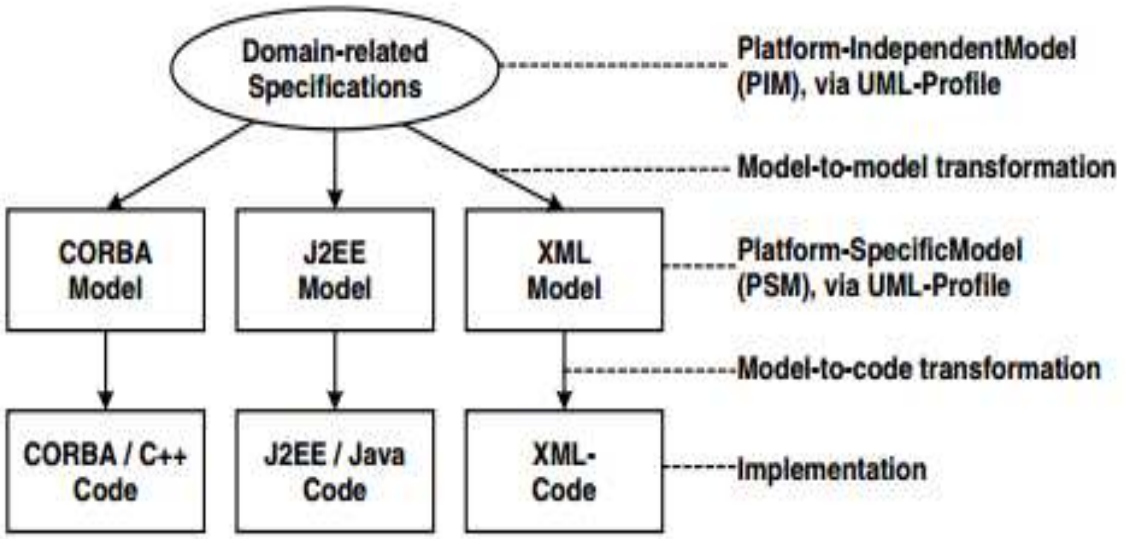

Figure 2.2 Basic principles of MDA, [Stahl et al., 2006]

tages, for instance, the business logic and the underlying technology evolve separately. This means that the business logic could be modified based on new requirements independent of the underlying platform. Likewise, the platform can also be changed in response to new technology arrivals.

Figure 2.3 demonstrates a four-layer architecture (i.e. meta-meta-model, meta-model, model and instance) of OMG® standard, where each layer represents a different level of model abstraction [Cicchetti, 2008; Gonzalez-Perez and Henderson-Sellers, 2008; OMG, 2002; Stahl et al., 2006]. According to Kühne *"a model is an abstraction of a (real or language-based) system allowing predictions or inferences to be made"* [Kühne, 2006]. A model is a description or specification of a system and its environment with an intended goal in mind, and it answers questions in place of the actual system [Belaunde et al., 2003; Bézivin and Gerbé, 2001].

Stachowiak identifies three basic features of a model such as *mapping*, *reduction*, and *pragmatic* features [Stachowiak, 1973]. The mapping feature states that a model is based on an original Subject Under Study (SUS). This feature holds for descriptive model, which makes statements about the SUS. A descriptive model captures some knowledge about an SUS so that it is used to perform a domain analysis [Kühne, 2006; Muller et al., 2012;

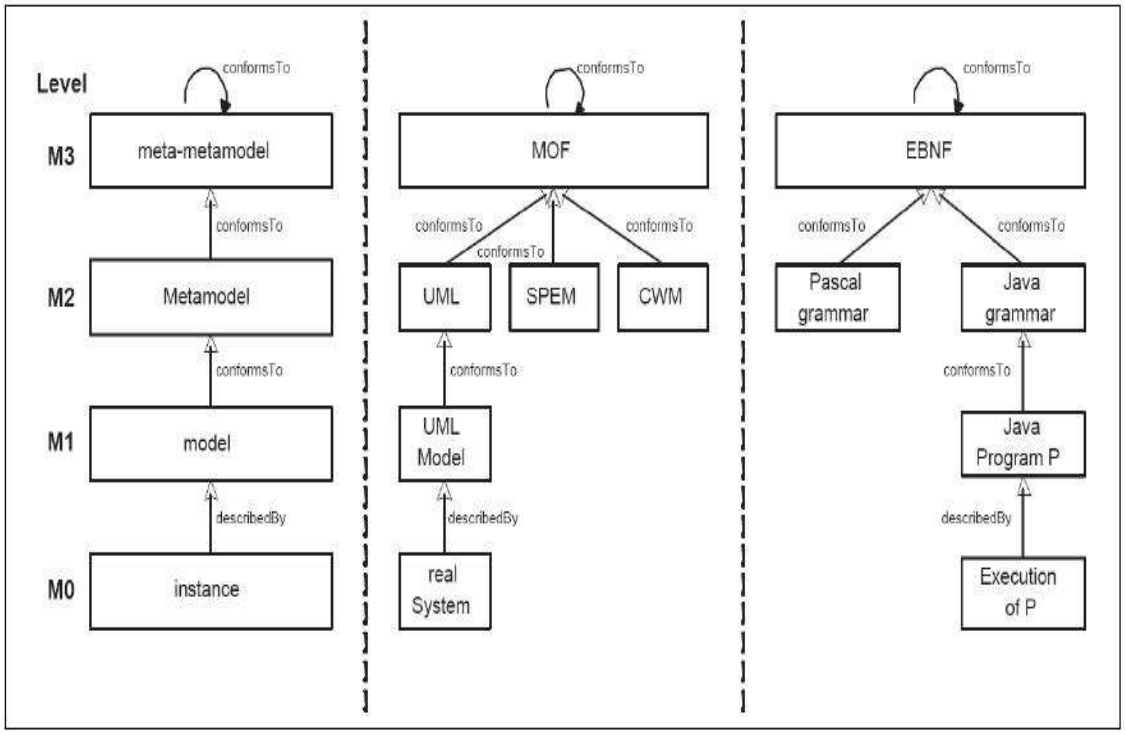

Figure 2.3 The four layered architecture of MDA, [Cicchetti, 2008]

Seidewitz, 2003]. In contrast, a specification model is a blueprint (construction plan) for the SUS, since there is no original SUS to map to the model. This work adopts the term "subject" instead of "original" as presented in [Kühne, 2006] to emphasis a model is based on an existing or imaginary system.

The *reduction* feature states that a model is an abstraction of an SUS, which only contains relevant features. According to Selic, *"a model is always a reduced rendering of the system that it represents"* [Selic, 2003].

The *pragmatic* feature is about the use of a model for some purpose in place of subject. Indeed, a model must be accurate and understandable to ensure the pragmatic feature [Selic, 2003]. An accurate model gives a true-to-life representation of an SUS. An understandable model remains in a form that directly draws our intuition. The model can also be used to correctly predict interesting features of an SUS that are not obvious for a user.

According to Gonzalez-Perez et al. [Gonzalez-Perez and Henderson-Sellers, 2005], a model must be homomorphic with its SUS to ensure the pragmatic feature. Specifically,

each entity in the SUS (at some relevant level of detail) should have a corresponding entity in the model that plays the same structural roles.

A model has a concrete syntax that facilitates communication among users. Fondement et al. defined concrete syntax as *"a concrete syntax is a surface language that acts as an interface between the instances of the concepts, and the human being supposed to produce or read them"* [Fondement, 2007]. Users do not generally edit models directly, but they interact with tools that present models with concrete syntax (i.e., text, graph, tree widget, ...). An abstract syntax of a model can be represented in one or more concrete syntaxes.

An abstract syntax of a model describes concepts of the languages and their relationships [Fondement, 2007; Meyers and Vangheluwe, 2011]. Software artifacts are generally governed by abstract descriptions (i.e, meta-model, grammar, ontology, ...). An abstract syntax of a model is specified by a meta-model. A meta-model is a model of models [Favre, 2005], it is a domain specific language oriented towards the representation of software development methodologies and endeavors [Gonzalez-Perez and Henderson-Sellers, 2008]. Similarly, as a grammar is used to specify a language, a meta-model constitutes a set of rules to construct all valid instance models. A meta-meta-model (i.e. MOF [OMG, 2002], MetaL [Englebert and Heymans, 2007], KM3 [Jouault and Bézivin, 2006], EMF/Ecore [Steinberg et al., 2009] is a minimum set of concepts that defines meta-models. It provides an unambiguous definition (i.e. syntax and semantics) for meta-models. Some mechanism has to be employed to terminate the hierarchy of meta-steps. A common approach is to define a meta-meta-model in terms of itself [Atkinson and Kühne, 2001]. For instance, MOF, EMF/Ecore, and MetaL are self-descriptive languages.

The semantics of a model describes the meaning of modeling concepts with respect to the domain [Rose, 2011]. It is defined using a semantic mapping function that maps each element of the abstract syntax to an element in a semantic domain [Harel and Rumpe, 2004; Meyers and Vangheluwe, 2011]. The semantic domain can be specified using formal languages such as Z [Woodcock and Davies, 1996] and Petri nets [Peterson, 1981], or it can be described in a semi-formal manner, which combines formal specification and natural

languages [Rose, 2011]. For instance, the static semantics of a UML model is specified using Object Constraint Language (OCL) [Warmer and Kleppe, 2003].

The meta-steps (meta- and meta-meta-) demonstrated in the four-layer architecture (see Figure 2.3) are relative, not absolute [Favre, 2005]. For instance, a meta-model is an instance model of a meta-meta-model if one considers the relationship between the meta-model and the meta-meta-model.

2.3 Other Model Driven Engineering Initiatives

There are other architectures of MDE proposed by different authors, for instance, Henderson-Sellers presents the three-layer architecture [Henderson-Sellers, 2006]. In [Gonzalez-Perez, 2005], Gonzalez-Perez demonstrates a new architecture that contains a modeling infrastructure orthogonal to the meta-level hierarchy. Atkinson et al. also present a deep-instantiation meta-modeling approach [Atkinson and Kühne, 2002]. Interested readers are referred to the aforementioned work and references for further information. This PhD thesis adopts a strict meta-modeling approach [Atkinson and Kühne, 2002] as specified by the four-layer architecture.

We will provide the formalization of model, meta-model, and meta-meta-model in Chapter 5. In the following chapters, we use the term “model” to refer both models and meta-models unless it is clearly specified.

Chapter 3

Computer Supported Cooperative Work

This chapter presents computer supported cooperative work (CSCW). The chapter presents the four different categories of CSCW; Section 3.2.1 and Section 3.2.2 presents a centralized approach with and without modification controller, respectively. Next Section 3.2.3 explains a decentralized approach with modification controller. Finally, Section 3.2.4 illustrates a decentralized approach without modification controller.

3.1 Computer Supported Cooperative Work

Software projects are naturally cooperative. They require collaboration among members of a group so as to produce a large software system [Whitehead et al., 2010]. *"Any software project with more than one person is created through a process of collaborative software engineering"* [Whitehead et al., 2010]. Indeed, modeling of software systems usually requires collaboration among members of a group with different scope and skills (i.e., middleware engineers, human interface designers, database experts, and business analysts). Schmidt et al. defines cooperative work as [Schmidt and Simone, 1996] *"cooperative work is constituted by the interdependence of multiple actors who, in their individual activities, in changing the state of their individual field of work, and change the state of field of work of others and who thus interact through changing the state of a common field of work"*.

In the above definition of cooperative work, the term *"common"* implies that members of the cooperative group share the same field of work (i.e. models, source codes, and documents). In this regard, software engineering collaboration can be considered as artifact-based or model-based collaboration, where users coordinate their activity to produce new models, to create shared meaning around models, and to remove errors and ambiguity within models [Whitehead et al., 2010]. Likewise, the word *"interdependence"* indicates dependencies between members of the cooperative group. Specifically, it states a work of one user positively relies on the quality and timeliness of other users. A user benefits from skills and knowledge of another user in the group.

Computer Supported Cooperative Work (CSCW) is a type of cooperative work in which computer systems are used to support mutually interdependent work. According to Borghoff et al., *"CSCW is a generic term which combines the understanding of the way people work in groups with enabling technologies of computer networking, and associated hardware, software, services and techniques"* [Borghoff and Schlichter, 2000]. On the contrary, groupware is a software system that is designed to support cooperative work [Johansen, 1988]. Collaborative modeling is a groupware in which computer systems are employed to support collaborative model editing, to manage users users interactions, and to help articulation of concurrently edited models.

Collaborative modeling is inherently distributed, where tasks are distributed among members of the cooperative group [Borghoff and Schlichter, 2000]. Hence, interactions among members of the group should be mediated and controlled in order to get the work done. This means that, the work has to be allocated to each member of the cooperative group, and a member has to be responsible and accountable to finish his/her work with a required criteria such as quality and time. The organizational structure ensures the allocation of precise description of responsibilities (roles) to every member of the group and it guides the relationship of the user with other members. According to Montes et al. [Montes et al., 2006], *"a key aspect for the development of cooperative system is to know how members of the cooperative group are organized to achieve the common goal"*.

The organization structure is built around roles [Montes et al., 2006], this structure changes over time due to different reasons. Schmidt at al. [Schmidt and Bannon, 1992] argue that a collaborative work group is transient, which is formed to handle a specific situation and dissolved later. Membership of collaborative work is also unstable and usually non-predictable, since users can freely join or leave the group. Moreover, the interaction pattern among members of a group could be dynamic depending on the requirements and the constraints of the situation, and members are semi-autonomous in their partial work. The roles are crucial to manage the information and process flows in the collaborative group [Zhu et al., 2006]. The organizational structure of members in cooperative group is discussed in detail in Chapter 4 Section 4.2.1.

"Awareness information is always required to coordinate group activities, whatever the task domain" [Dourish and Bellotti, 1992]. Awareness of individual and group activities is essential for a success of cooperative work [Dourish and Bellotti, 1992]. Knowledge about activities of others provides a context information about the common field of work and the way in which this field work is produced. A user can benefit from the context information to ensure relevance of his/her individual contributions, and to measure his/her contributions with respect to group goals and progress. The use of awareness information to manage communication is presented in Chapter 4 Section 4.2.3.

Communication is a means to create awareness among members of cooperative group. But, since multiple users are engaged in cooperative work, interactions between members and applications are not predictable. For instance, members could have their own goals, strategies, and experience levels that might lead to chaotic environment. Hence, policies or communication protocols are required to precisely define mode of interactions among members, and between members and applications in order to avoid conflicts and confusions [Edwards, 1996]. A role-based access rights and assignment of different responsibilities (or roles) to members of the cooperative group help to reduce chaos and improve coordination.

A shared and integrated repository is used to facilitate communication among members of CSCW. In shared information systems, members edit common models, and create common meaning and understanding around the the models [Whitehead et al., 2010]. Of course,

the shared information system needs to use a common data model to ensure communication among members. There are also other systems that use message exchange mechanisms to ensure communication among the group. For example, a service provider web service and a service consumer web service communicate by exchanging messages [Snell et al., 2001]. Likewise, members of a cooperative group may exchange messages to communicate their activities.

The mode of communication among members of a cooperative work is either synchronous or asynchronous. In synchronous collaboration, two or more members access and edit a shared model simultaneously, and communicate their work in real time. These concurrent modifications might conflict one another, hence, a protocol is required to ensure consistency. For instance, token-passing, locking schemes, or floor-passing schemes can be used to define the pessimistic communication protocol [Borghoff and Schlichter, 2000]. Some groupware applications integrate WYSIWIS (What You See Is What I See) interfaces that let users visualize parts of a concurrently edited model in exactly the same way in real time [Bani-Salameh, 2011; Gallardo et al., 2012; Minör and Magnusson, 1993]. Synchronous communication improves awareness of individual and group activities so that it helps to reduce conflicts and task completion times [Dewan and Hegde, 2007]. This style of working is usually suitable to accomplish urgent tasks that need high frequency of communication among members.

Members of a CSCW group may wish to have the option of disconnected workspaces to work privately [Dewan and Hegde, 2007]. This means that they edit models in their local workspace and integrate their activities later. This is an asynchronous mode of communication, where local activities are transparent from other users. In [Minör and Magnusson, 1993], the authors argue that complex software design and implementations are performed asynchronously by different members. Of course, the shared modeling artifacts could be edited and evolved concurrently throughout the development life cycle of a software application by different users. As a result, they might not seamlessly work together or the final result may not be what users want. In other words, modeling artifacts become inconsistent with each other.

Articulation work mediates and controls a set of distributed activities of the cooperative group [Schmidt and Bannon, 1992; Schmidt and Simone, 1996]. The coordination of interdependent activities is critical to achieve a goal [Borges and Pino, 1999]. Group communication can be managed using role assignment, division of tasks, and organizational structures, which can avoid confusions and conflicts among members. Besides, the reconciliation of conflicting activities requires conflict detection, resolving of conflicts, and merging. The reconciliation can be a *a priori* or *posteriori*. In a priori reconciliation mechanism, the cooperative group agrees on common terms and communication protocols beforehand to avoid confusion and disorder. But, it is practically impossible to anticipate all contingencies in advance so that the posteriori reconciliation is also required to deal with such incidents. Conflict detection, reconciliation work, and member organization are presented in Chapter 4 Section 4.1.2, Section 4.1.3, and Section 4.2.1.

3.2 Classification of CSCW

In [Boukhebouze et al., 2010], Boukhebouze et al. classify CSCW into four categories such as a centralized approach with modification controller, a centralized approach without modification controller, a decentralized approach with modification controller and a decentralized approach without modification controller. Their classification is based on the location of the repository and the management of modifications (coordination process).

3.2.1 Centralized approach with modification controller

The centralized approach with modification controller has a central repository, which is owned by a central authority who has a bird's-eye view and controls modifications of models (see Figure 3.1). The owner is the only one who can modify the model, whereas other members send messages (modification requests) to the owner so as to change the model. Modifications are performed in human-time. Moreover, this approach adopts roles like controller, editor, project leader, or expert so as to facilitate collaboration among members. For example, Barteit et al. provide a controller based collaborative task on top of the IBM

rational Jazz[1] [Barteit et al., 2009]. This mode of collaboration can also be implemented on top of EMFStore [Koegel and Helming, 2010], such that a model configuration manager manages the model evolution.

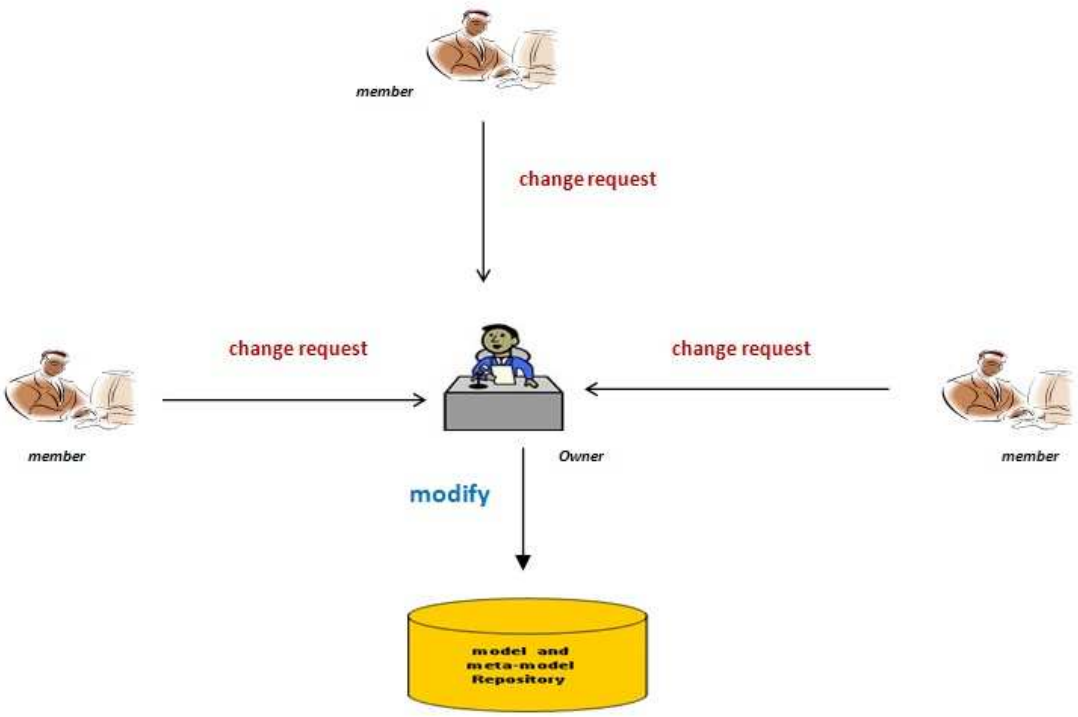

Figure 3.1 Centralized approach with modification control

3.2.2 Centralized approach without modification controller

As shown in Figure 3.2, the centralized approach without modification controller has one central repository, but there is no controller who manages modifications. In other words, all members have equal access rights (role) so that everybody can freely modify the model (without informing his/her colleague). The modifications are performed in a real-time and the central repository only stores the most recent modification (overwrite previous modification). In this type of CSCW, members are free to join or leave the group at any time without going through any organizational structure or authority. As a result, a dynamism of members of cooperative ensembles becomes very high as compared to the one with control.

[1] https://jazz.net/

Furthermore, the frequency of interactions among members is also high in order to reach on common agreement. More specifically, models are continuously modified by each member, therefore, it is difficult to have a stable version. Hence, members need to communicate (i.e. through shared information) until they agree on the semantics of model elements. Another solution could be to partition models into distinct parts and allow only one member to modify a part at a time. For example, Cacoo[2] and EMFStore [Koegel and Helming, 2010] are centralized collaborative tools that do not support a central modification controller role.

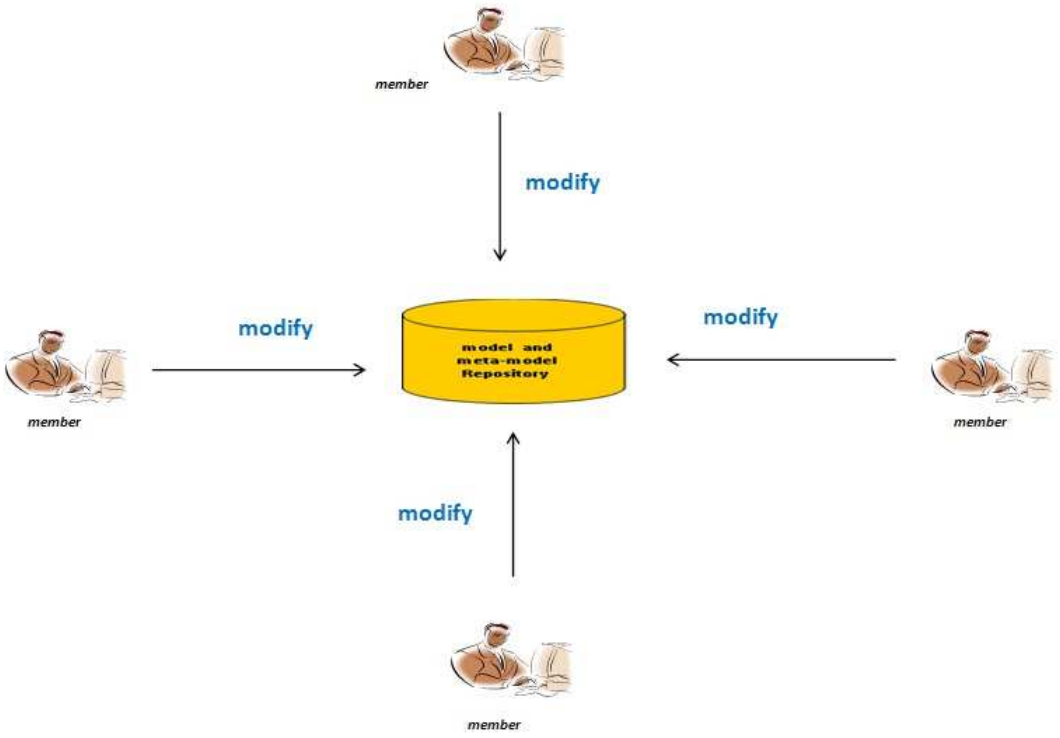

Figure 3.2 Centralized approach without modification control

3.2.3 Decentralized approach with modification controller

Decentralized approach with modification controller is a distributed system that replicates a clone of the master copy at each member site (see Figure 3.3). A mode of communication in decentralized approach is inherently asynchronous, members modify their local copies and communicate their activities later. Specifically, members communicate through a controller

[2]https://cacoo.com/home

who plays the role of communication hub and manage the evolution of models. In other words, a member cannot directly propagate his/her local modifications to other colleagues, rather, s/he first sends a message (i.e. change request) to the controller in order to get approval of modifications. For instance, this change request could be sequence of create, read, update or delete (CRUD) operations that are performed on the model elements [Koshima et al., 2013; Mougenot et al., 2009]. Afterwards, the controller inspects modifications proposed by a members and communicates accepted changes with other members. This mode of collaboration can also be implemented on top of Git [Chacon, 2009] and model versioning frameworks (e.g., EGit[3] and EMF Compare[4]).

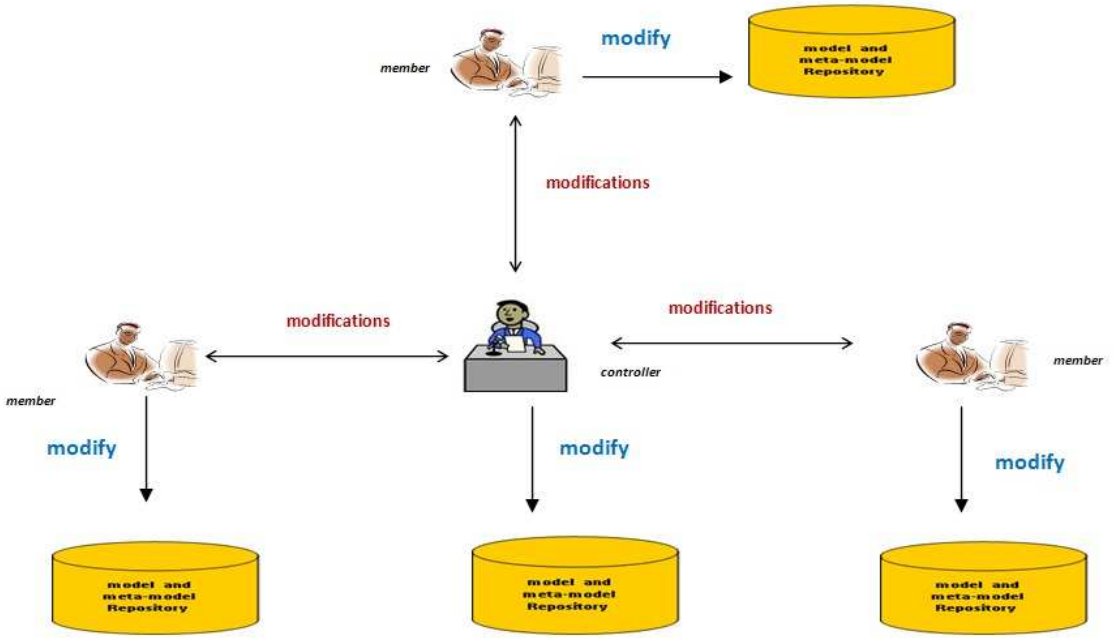

Figure 3.3 Decentralized approach with modification control

The controller may group and consolidate accepted change requests and share with all members, or s/he sends each change request to members in First In First Out (FIFO) order [Cormen et al., 2009]. The later one creates competition among members to submit modifications first (i.e. the first one gets high power to determines the overall design work). This might create inconvenience in collaborative work.

[3]http://www.eclipse.org/egit/?gclid
[4]http://www.eclipse.org/emf/compare/

3.2.4 Decentralized approach without modification controller

Decentralized approach without modification control is characterized by a distributed model management. This means that, models are distributed among members. Moreover, every member can modify his/her local copy and send message (i.e. modifications) directly to other members without a supervision of central authority. This approach gives a total freedom for each member to work in terms of time and space. However, keeping models globally consistent is challenging due to concurrent modifications of models. Therefore, some policies (i.e. voting) could be implemented to facilitate the reconciliation of conflicting activities. For instance, if the majority vote is "YES" for a given change request, then it is applied on each local copy. Documentation of activities and partitioning of tasks could ease the collaboration among members, however, studies show that this does not work in practice [Dewan and Hegde, 2007]. See Figure 3.4 below. For example, D-Praxis [Mougenot et al., 2009].

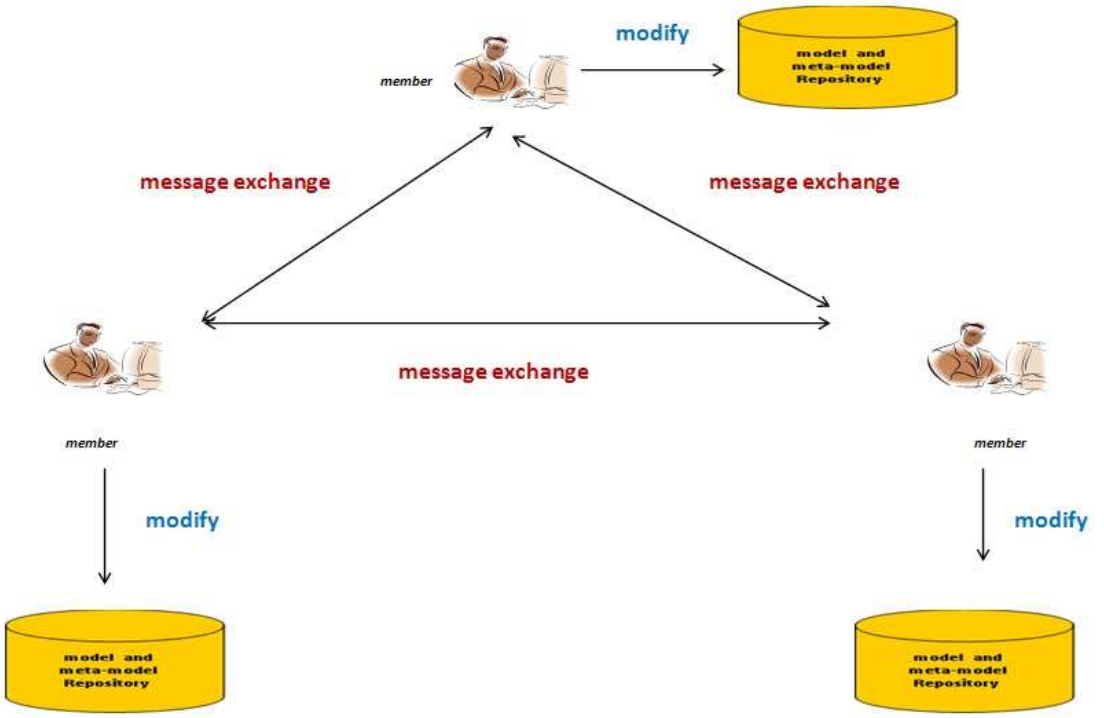

Figure 3.4 Decentralized approach without modification control

In general, CSCW with modification control has a more stable cooperative group than approaches without the controller. This is because, every member needs approval from the controller to join or leave the group.

There are also other classifications of CSCW in the literature and interested readers can refer to [Borghoff and Schlichter, 2000; Carstensen and Schmidt; Ellis and Wainer, 1999; Mills, 2003; Omoronyia et al., 2007]. Chapter 4 section 4.2.2 presents the state-of-the-art of collaborative modeling tools based on the categories presented in this chapter.

Chapter 4

State-of-the-art of Collaborative Modeling

This chapter presents the current state-of-the-art approaches and tools that support collaborative modeling. Besides, it provides a detailed analysis and limitations of these approaches.

4.1 Management of Models

As discussed in Chapter 1 Section 1.2 (problem statement), the management of shared modeling artifacts is important to ensure collaborative modeling. Firstly, concurrently edited models are compared to generate model differences and to identify conflicting modifications. Afterwards, conflicting modifications need to be visualized for users to facilitate a reconciliation process. Finally, the conflicting versions of the model should be merged into a new version.

Users edit models by interacting with the concrete syntax, which could be a textual or a graphical representation of the model. A concrete syntax usually contains more information than an abstract syntax. For instance, a graphical concrete syntax could contain design information, quasi-design information, and graphical information [Rho and Wu, 1998]. The design information stands for an abstract syntax of a model, which captures the essence of the structure of the software design. On the other hand, the quasi-design information is a

documentation about the design (e.g., description about UML class diagram). The graphical information is related to a graphical representation of an abstract syntax. It contains different shapes, relationship between shapes, positions, and layout information. During concurrent development of models, a design, quasi-design and graphical information could be modified by different users. In this work, we do not consider the evolution of quasi-design information and graphical information. We perform model comparison, conflict detection and merging activities to manage the evolution of design information of models.

4.1.1 Model Comparison

Model comparison compares (meta-)models so as to derive their differences [Altmanninger et al., 2009; Koegel et al., 2009c]. A two-way comparison approach compares two (meta-)models, whereas a three-way comparison approach compares two (meta-)models and their ancestor (meta-)model [Altmanninger et al., 2009; Koegel et al., 2009c]. In fact, in order to compare two models, they should be defined using the same modeling language (the meta-model definitions of both models need to be the same). The same holds true for meta-models, meaning that their meta-meta-models need to be the same so as to compare two meta-models. Model comparison can be classified into two different categories such as *state-based comparison* and *change-base comparison* depending on the information available for comparison [Altmanninger et al., 2009; Conradi and Westfechtel, 1997, 1998; Koegel et al., 2009c; Lippe and van Oosterom, 1992b; Mens, 2002].

4.1.1.1 State-based Comparison

State-based comparison compares the state of two or three revisions [1] (variants [2]) of models to derive their differences [Altmanninger et al., 2009; Conradi and Westfechtel, 1997, 1998; Koegel et al., 2009c; Lippe and van Oosterom, 1992b; Mens, 2002]. The comparison can be a two-way comparison that only takes the state of two models or a three-way comparison that considers the ancestor model in addition to the two models. The two-way comparison

[1] revisions: sequentially evolved models over a given time period to correct errors or improve the product
[2] variants: alternative (parallel) versions of models that are co-existing at a given time

cannot identify whether an element is created in one version or deleted in another version. Besides, it cannot differentiate between modifications performed in versions [Altmanninger et al., 2009; Mens, 2002]. Hence, two-way comparison is not suitable to support collaborative modeling.

The three-way comparison addresses these problems; it differentiates modifications, creations and deletions [Altmanninger et al., 2009; Mens, 2002]. Indeed, the comparison constitutes two steps such as matching and differencing. The matching step is performed to find correspondences between elements of two models. In [Kolovos et al., 2009], Kolovos et al. classify the matching approaches into four categories such as *static identity-based matching*, *signature-based matching*, *similarity-based matching*, and *custom language-specific matching algorithm*. The differencing step will be discussed later in this section. *Static identity-based matching* relies on the universal unique identifier (UUID). It assigns a UUID to each model element and to newly created model elements. It assumes that the UUID is non-volatile and persistent [Kolovos et al., 2009; Ohst et al., 2003; Treude et al., 2007]. That is, the UUID is not modified after it has been assigned to a model element.

Static identity-based matching approach creates a correspondence between two model elements if and only if they do have the same UUID. This approach has been adopted by Alanen et al. who propose a framework that uses UUIDs to perform a difference and union of models [Alanen and Porres, 2003]. In another work, Vernadat et al. also use UUIDs in TOPCASED to perform matching [Vernadat et al., 2006]. Furthermore, Sriplakich et al. performs a UUID based matching in ModelBus : An Open and Distributed Environment for Model Driven Engineering [Sriplakich, 2007]. The matching technique is fast and it has a time complexity of $O(1)$ [Koegel et al., 2009c; Treude et al., 2007]. Furthermore, it can find a match between model elements even though they have had fundamental changes [Altmanninger et al., 2009]. But, it cannot be applied for tools that do not provide facilities to manage UUIDs. Besides, it might not be suitable for models which are created independently or refactored [Kolovos et al., 2009; Treude et al., 2007]. It might also produce difference models with bad quality [Treude et al., 2007].

Signature-based matching dynamically computes the identity of model elements based on values of their features [Kolovos et al., 2009; Reddy et al., 2005] and signature types specified by a user. A signature type is a user defined function that specifies a subset of properties of a model element. For example, these sub properties could be attributes and association ends of a class diagram. This approach solves the limitation of static identity-based matching approach. It does not need each model element to have a global unique id. Hence, it seamlessly works with independently created and evolved models. But, this approach requires users to specify signature type functions manually. For instance, Reddy et al. use signature-based matching to perform model composition [Reddy et al., 2005].

Similarity-based matching uses heuristic search algorithms to find matches between two (or three) versions of models based on the aggregate similarity of their contents and their graph structure [Kolovos et al., 2009]. Unlike *Signature-based matching* approach that returns a boolean result, this approach returns a real number, which represents an aggregated similarity. But, it does not take the semantics of the language into consideration so that it might produce less quality results [Kolovos et al., 2009]. Besides, a matching algorithm is computationally expensive. In [Chawathe and Garcia-Molina, 1997; Koegel et al., 2009c; Kolovos et al., 2009], authors argue that a model matching problem can be reduced to the graph isomorphism problem, which is an NP-hard problem in its full generality. Most of model matching approaches deal with the computational complexity by employing a modeling language specific algorithm or providing an approximation of the exact solution [Kolovos et al., 2009]. For instance, UMLDiff [Xing and Stroulia, 2005], DSMDiff [Lin et al., 2007], SiDiff [Treude et al., 2007], Epsilon Comparison Language (ECL) [Kolovos et al., 2006], and EMF Compare [3] use a heuristic-based matching algorithm to find correspondences between two (three) versions of model.

Custom language-specific matching algorithm is a special type of similarity-based matching that uses domain specific information and tailored to a specific modeling language [Kolovos et al., 2009]. This approach considers the semantics of the modeling language to create matches. As a consequence, it produces more accurate results and reduces the

[3] http://www.eclipse.org/emf/compare/

search space by avoiding unnecessary comparisons [Kolovos et al., 2009]. For instance, in UML, this approach could use the domain specific knowledge of class diagram to compare two operations. It only compares these operations if it find a match between two classes that contain these operations. There is some work that employed this algorithm such as SiDiff [Treude et al., 2007], Epsilon Comparison Language (ECL) [Kolovos et al., 2006], and EMF Compare.

Some tools combine static identity-based matching approach with similarity-based matching or custom language-specific matching algorithm. As stated in EMF Compare — Developer Guide [4], the default behavior of EMF compare finds matching between model elements based on their identifiers, if model elements have identifiers. Afterwards, it uses a similarity-based matching or a custom language-specific matching algorithm to find matches for model elements that do not have identifiers. The approach could improve the accuracy of the matching result and reduce the time complexity of the computation.

After the matching step, the differencing step starts by taking the matching results as input and computes differences between two (or three) versions of models [Altmanninger et al., 2009; Conradi and Westfechtel, 1997, 1998; Koegel et al., 2009c; Kolovos et al., 2009; Lippe and van Oosterom, 1992b; Mens, 2002]. The difference between models can be represented as a model itself. In [Cicchetti et al., 2007], Cicchetti et al. propose a model-based, minimalistic, transformative, compositional, and meta-model independent way to represent differences between models. The comparison is performed in linear time, $O(n)$, for n model elements. Of course, the abstract syntax of software artifacts determines the type of comparison approaches to be used in order to generate matching and differences. For instance, SVN [Pilato et al., 2008] and Git [Chacon, 2009] use a line-based comparison approach to compare text files. Line-based comparison cannot handle multiple changes on the same line [Altmanninger et al., 2009; Koegel et al., 2009c; Lippe and van Oosterom, 1992b; Mens, 2002]. Modifications of a model are structural changes and a single change in the model could be reported as many lines of modification in the line-based comparison. Nguyen et al. express this incongruity as *impedance mismatch* [Nguyen et al., 2005]. Therefore, line-

[4] http://www.eclipse.org/emf/compare/documentation/latest/user/user-guide.html

based comparison is not suitable for models that are graphs in nature [Altmanninger et al., 2009; Barteit et al., 2009; Koegel et al., 2009c; Koshima and Englebert, 2014; Koshima et al., 2013; Nguyen et al., 2005; Westfechtel, 2010].

Figure 4.1 is an example of EMF compare that performed a three-way comparison among Petri net meta-models expressed using EMF/Ecore. The original version of the meta-model is labeled "(1)" in the top part. The two modifications are visualized on the left and right sub-sections of the diagram. They are labeled "(2)" and "(3)", respectively. The figure shows the matching and modifications among the three meta-models.

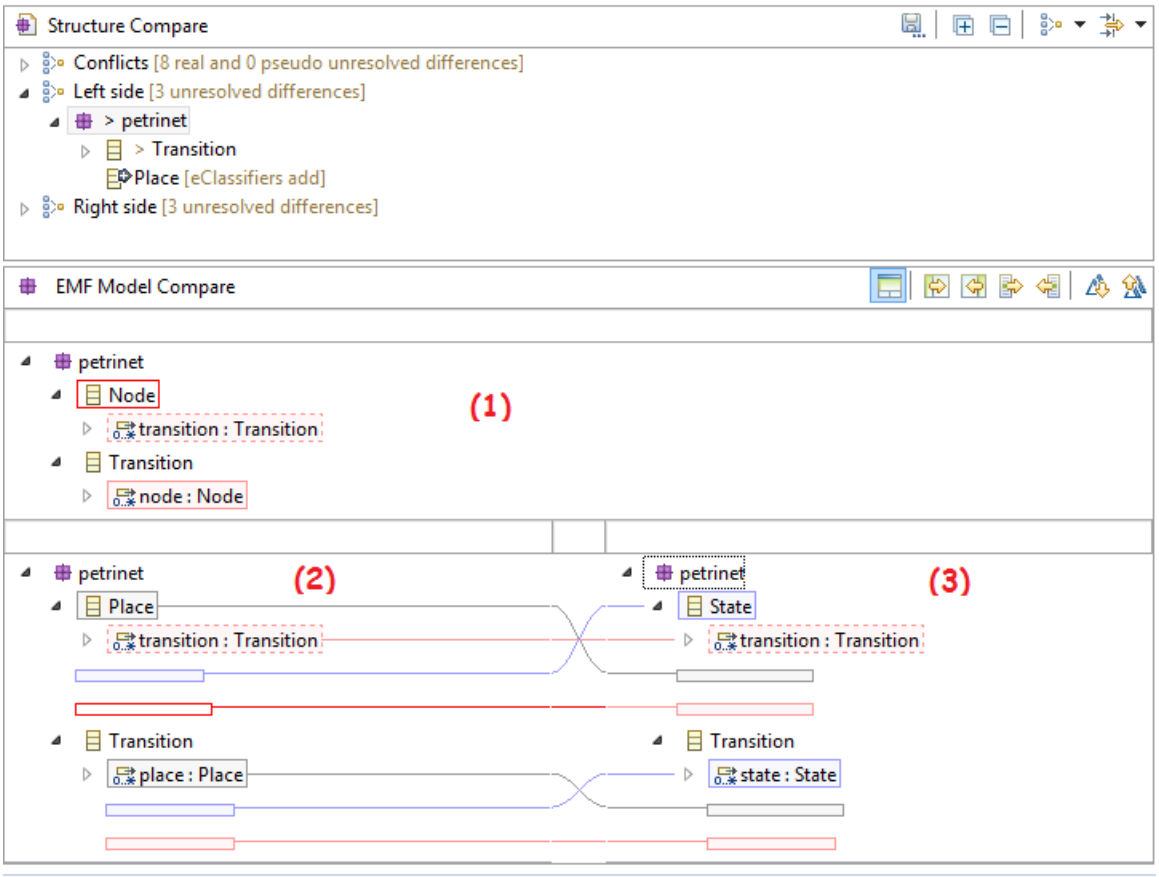

Figure 4.1 EMF Compare: Three-way comparison

Lessons Learned

State-based comparison makes the modeling tools independent from the version control system [Altmanninger et al., 2009; Conradi and Westfechtel, 1997, 1998; Koegel et al.,

2009c; Lippe and van Oosterom, 1992b; Mens, 2002]. As a result, the modeling tools are not required to observe or record changes while they occur. Independently developed and re-engineered models can also be used without difficulty. Hence, users can work with their choice of modeling tools. However, this technique cannot catch the time order of changes that are important to understand modifications, conflict detection, and facilitate merging process [Altmanninger et al., 2009; Koegel et al., 2009c; Lippe and van Oosterom, 1992b; Mens, 2002]. Besides, it might not derive the difference (delta) correctly and the computational complexity of computing deltas (i.e. matching and differencing) is expensive, specifically, if the model is large in size [Koegel et al., 2009c]. Furthermore, it is inadequate to derive group of changes that occur together (i.e. refactoring operations). Moreover, it is less obvious for users to review and understand modifications [Koegel et al., 2009c]. Last but not least, it is not possible to attach rationale of modifications to specific changes that evolve models, rather, change requests and rationale of modifications are attached to the final state of a model. Hence, users have to map the final state of a model to the rationale of modifications and the change requests manually.

4.1.1.2 Change-based Comparison

Unlike *state-based comparison* that ignores edit operations and only considers the end-result, *change-based comparison* keeps track of changes performed on software artifacts (i.e., models and meta-models) [Altmanninger et al., 2009; Asklund, 1994a,b; Conradi and Westfechtel, 1997, 1998; Koegel et al., 2009c; Lippe and van Oosterom, 1992b; Mens, 2002]. Changes are treated as first class concepts [Koegel et al., 2009c]. In [Conradi and Westfechtel, 1998; Mens, 2002], authors classify change-based approach into *intensional* and *extensional* versioning. *Extensional* change-based versioning approach uniquely identifies each version and annotates it with deltas (a sequence of change operations) relative to another version (base-line). Deltas specify differences between two versions and they are used to re-construct previous versions. This approach uses deltas as a documentation, hence, it is called a change package [Conradi and Westfechtel, 1998].

Asklund et al. develop a collaborative software development environment that embeds backward deltas so as to represent changes between two versions of a hierarchical structured document [Asklund, 1994a,b; Magnusson et al., 1993a,b]. The collaborative software development environment provides facilities for users to consult a revision graph (deltas) and re-construct previous versions. Deltas help users to know the current status of a hierarchical structured document and to identify a person who performed changes. In other work, Rho et al. also use a change package approach to specify deltas between two versions of graphs [Rho and Wu, 1998]. In [Steinberg et al., 2009], EMF framework also adopts a change package (i.e., change model) to store backward deltas for each modifications and it employs these backward deltas to perform undo operations.

Intensional change-based versioning lets changes to be assembled freely and independently of a version on which they are going to be applied [Conradi and Westfechtel, 1998; Mens, 2002]. This approach is flexible and helps to automatically construct different versions based on a set of change operations. In [Mens, 2002], the author refers this approach as *change set model*. Intensional change-based versioning could be used for parallel software development and versiong, and software maintenance [Mens, 2002].

Operation-based comparison is a special type of *change-based comparison* [Altmanninger et al., 2009; Conradi and Westfechtel, 1997, 1998; Koegel et al., 2009c; Lippe and van Oosterom, 1992b; Mens, 2002] that depends on *Intensional* change-based versioning [Conradi and Westfechtel, 1997, 1998; Mens, 2002]. It represents deltas as a sequence of change-operations [Altmanninger et al., 2009; Conradi and Westfechtel, 1997, 1998; Koegel et al., 2009c; Lippe and van Oosterom, 1992b; Mens, 2002]. So that, there is no need to perform differencing activity because deltas are already identified [Koegel et al., 2009c]. A set of operations are applied to the base version, V_0, so as to obtain a new version V_1. Complex operations (i.e. refactoring operations, transactions) can also be used to describe deltas [Altmanninger et al., 2009].

On the contrary to *state-based comparison*, *operation-based comparison* captures the exact time sequence of modifications [Altmanninger et al., 2009; Asklund, 1994a,b; Conradi and Westfechtel, 1997, 1998; Ignat and Norrie, 2004; Koegel et al., 2009c; Lippe and van

Oosterom, 1992b; Mens, 2002]. These traces of modifications could be helpful for users to understand changes and to keep the intention of another user who performed changes [Koegel et al., 2009c]. Besides, these change operations are used to detect conflicts and merge concurrently edited models [Altmanninger et al., 2009; Lippe and van Oosterom, 1992b; Mens, 2002]. On the top of that, this technique can capture refactoring or complex operations in the same context [Altmanninger et al., 2009; Koegel et al., 2009c; Mens, 2002]. Change operations can be annotated with multimedia files that describe the rationale of modifications [Koshima et al., 2011, 2013]. In addition, it is possible to explicitly associate change requests with a set of change operations (or composite operations) that implement the proposed modifications [Conradi and Westfechtel, 1998].

According to Koegel et. al. [Koegel et al., 2009c], changes are more easily reviewed and understood by users in *operation-based comparison* than *state-based comparison*. Moreover, *operation-based comparison* is more general than *state based-comparison* [Lippe and van Oosterom, 1992b]. In fact, one can use *operation-based comparison* to perform every possible operations that can be applied on *state-based comparison* by ignoring the stored extra information [Koegel et al., 2009c]. However, this approach has a limitation that necessitates a high coupling between modeling tools and change recorder [Altmanninger et al., 2009; Koegel et al., 2009c]. Like *state based-comparison*, *operation-based comparison* also needs to take into account the abstract syntax of software artifacts to capture edit operations. For instance, FORCE [Shen and Sun, 2002] is a flexible operation-based revision control environment that represents documents linearly; it does not keep in mind the abstract syntax of the software artifacts. Hence, FORCE is suitable for text documents, but it is insufficient to capture edit operations of tree or graph-based artifacts.

There are few operation based work that relies on the abstract syntax of software artifacts. For instance, Asklund et al. develop a collaborative software development environment for a hierarchically structured documents [Asklund, 1994a,b]. Ignat et al. also propose a flexible way to collaboratively edit XML documents [Ignat and Norrie, 2006]. In addition, Ignat et al. proposed a graphical editing framework for graph-based documents [Ignat and Norrie, 2004]. Furthermore, COPE/Edapt [Herrmannsdoerfer, 2009], D-Praxis [Mougenot et al.,

2009], and EMFStore [Koegel and Helming, 2010] are operation-based approaches that capture edit operations of graph-based software artifacts.

Lessons Learned

Operation-based comparison stores additional information (i.e., exact time sequence of modifications, complex operations, and rationale of modifications attached to changes) that is not available in the *state-based comparison* [Altmanninger et al., 2009; Asklund, 1994a,b; Conradi and Westfechtel, 1997, 1998; Ignat and Norrie, 2004; Koegel et al., 2009c; Koshima and Englebert, 2014; Koshima et al., 2011, 2013; Lippe and van Oosterom, 1992b; Mens, 2002]. It keeps traces of changes whenever they occur, hence, there is no need to calculate deltas [Koegel et al., 2009c]. This information is helpful to resolve conflicts, understand modifications, and preserve intention of operations. *Operation-based comparison* performs better than *state-based comparison* for large size models [Ignat and Norrie, 2004; Koegel et al., 2009c]. However, this technique requires tight coupling between modeling tools and change recorder. Besides, it cannot identified and match concurrently created equivalent concepts. For instance, $user_1$ and $user_2$ simultaneously create two attributes called *token* of type *Integer* in both *Node* and *State* classes in Figure 4.2. Although it creates matching between *Node* and *State* based on their unique identifier, it fails to create a mapping between two attributes named *token*. Hence, some work combine *state-based comparison* and *operation-based comparison* in order to benefit from the strongest side of the two worlds. This could improve conflict detection and merging. For instance, Brosch et al. [Brosch et al., 2009b] and Barrett et al. [Barrett, 2011] combine both model comparison approaches.

4.1.2 Conflict Detection

As discussed in Section 1.1 of Chapter 1, the main premise of the optimistic approach is that it encourages users to work independently and to synchronize their local work with other members later. In this approach, conflict detection and reconciliation are indispensable activities so as to facilitate collaborative work. In [Klein, 1991; Klein and Lu, 1989], Klein classifies conflicts into *competitive conflict* and *cooperative conflict*. A *competitive*

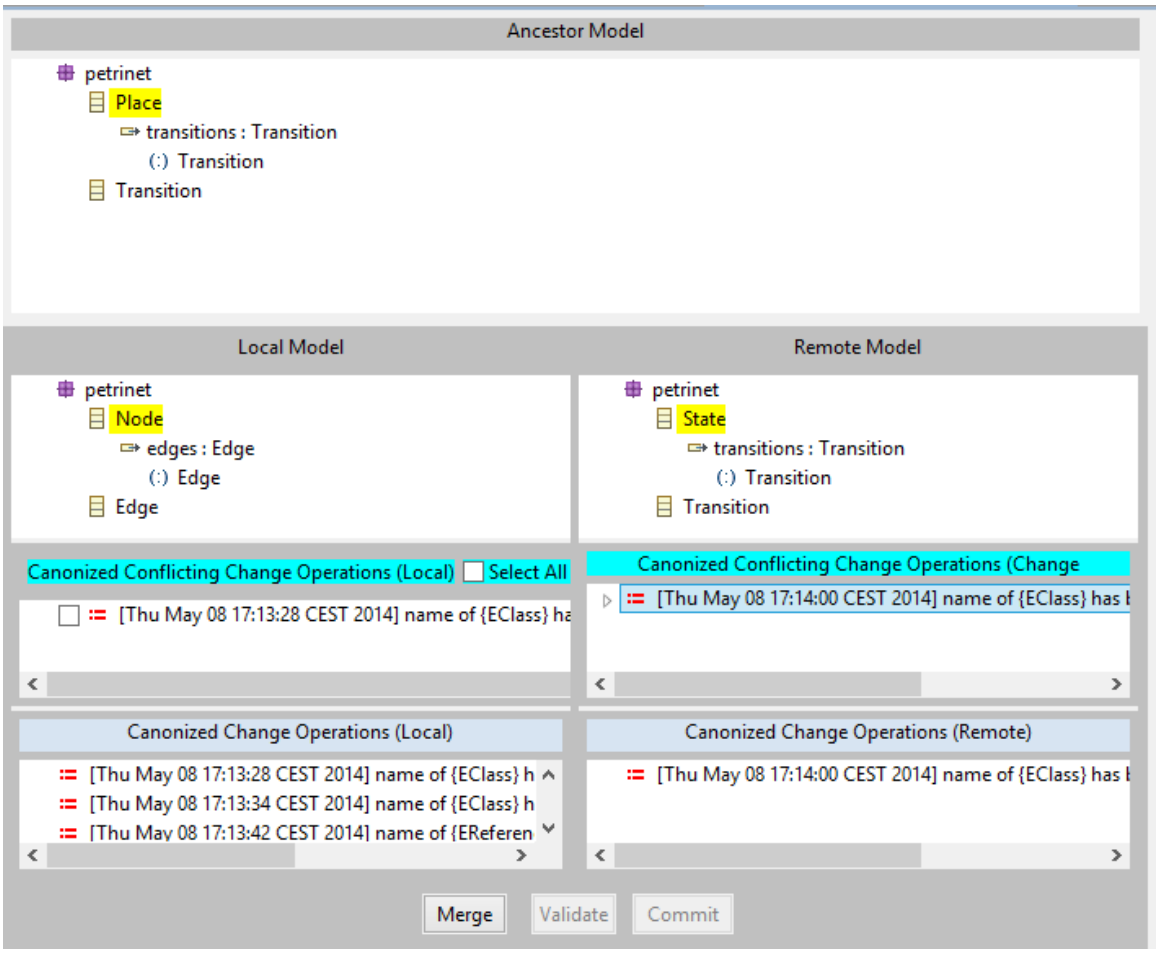

Figure 4.2 Three-way merge tool

conflict arises when each member of a group only thinks in terms of his/her own benefits rather than achieving a common objective of the group. Competitive conflict are difficult to resolve, since members are not willing to find a compromise solution that does not add personal benefit to them. On the other hand, a *cooperative conflict* occurs between a group of users who are collaboratively working to achieve a common goal. Users are willing to compromise less important goals and to find a mutual solution that satisfies their common goal. This PhD work only focuses on detecting and resolving cooperative conflicts.

A cooperative conflict is a common type of conflict that rises in the cooperative work. It occurs as a result of conflicting concurrent modifications (or overlapping changes) that cause inconsistent software artifacts [Altmanninger et al., 2009; Conradi and Westfechtel, 1998; Koegel et al., 2009a, 2010; Mens, 2002]. For example, two users concurrently modify

models locally and synchronize their activities later. In fact, their local modifications might conflicting each other, as a result, it might not be possible to merge their local modifications. Conflicting modifications might result in different models depending on the order of application of conflicting changes *(e.g. renaming the same class differently)*. Besides, applying one of the conflicting modification could violate the preconditions of the other modification, as a consequence, it is not possible to consolidate into the new version. These conflicts are refereed in the sequel as *merge conflicts* or *merge inconsistency* [Altmanninger et al., 2009; Mens, 2002]. Hence, these conflicting changes need to be identified and resolved so as to merge conflicting versions of software artifacts.

Mens classifies merge conflicts into *textual, syntactic,* and *semantic* conflicts [Mens, 2002]. Textual conflicts are detected between two or more text documents that are compared using line-based comparison technique [Altmanninger et al., 2009; Mens, 2002]. Because line-based comparison technique is too coarse-grained, it cannot handle concurrent modifications performed on the same-line of a text document [Altmanninger et al., 2009; Mens, 2002]. As a consequence, it might raise wrong textual conflicts. For example, suppose two users, Alice and Bob, are concurrently editing a text document and they have started from the base version that contains one line of text "Model Everything" [5]. Afterwards, Bob adds an exclamation mark at the end of the text ("Model Everything!"), whereas Alice adds a new word "Explicitly" at the end of the text as well ("Model Everything Explicitly"). These changes are not conflicting syntactically so that they can be consolidated together in the merged document that contains a line "Model Everything Explicitly!", however, the line-based merging raises conflicting flags.

Line-based comparison cannot capture semantic conflicts so that the merged result might not be semantically correct. For instance, Alice and Bob start editing a text document that contains one line of text "I saw Henry". Thereafter, Alice modifies the text to "I saw Henry with a beautiful lady" and Bob also evolves the text document to "I saw Henry in a bathroom". The merged text could be like "I saw Henry with a beautiful lady in a bathroom". The merge result is syntactically correct, but it might not reflect the intentions of Alice and

[5] "Model Everything Explicitly" is the motto of Hans Vangheluwe to advocate modeling of everything explicitly [Vangheluwe et al., 2007]

Bob (semantically incorrect). Therefore, the semantics of the software artifacts should be taken into consideration in order to produce a correct merged result. However, such type of semantics are difficult to capture and they reside in the users mind most of the time.

The granularity of the comparison might vary like a line of text, a paragraph, a sentence, a word, or a character [Altmanninger et al., 2009]. Selecting the right level of granularity could improve the result of conflict detection. Line-based merging technique is efficient, scalable, and provides satisfactory results [Altmanninger et al., 2009; Mens, 2002]. As a consequence, it is widely used and adopted technique by many version control systems such as Git [Chacon, 2009], SVN [Pilato et al., 2008], and CVS [Vesperman, 2006]. However, the textual merging fails to identify all conflicts, because it does not consider the syntax and semantics of software artifacts [Altmanninger et al., 2009; Mens, 2002]. For example, models are graph in nature and they have syntax and semantics that cannot be captured by text.

Syntactic conflicts are identified by analyzing the syntax and structure (i.e., tree or graph structure) of software artifacts [Altmanninger et al., 2009; Mens, 2002]. Syntactic merging raises merge conflicts if a merged result of concurrent modifications violates the abstract syntax of the language used to define the artifact [Altmanninger et al., 2009; Estublier et al., 2005; Mens, 2002]. This conflict detection approach gives a better result than the textual merging, because it considers the syntax of an artifact to compute conflicts [Altmanninger et al., 2009; Buffenbarger, 1995; Estublier et al., 2005; Mens, 2002; Westfechtel, 1991]. Syntactic conflict detection requires domain-specific information about the syntax of the language (abstract syntax) so as to identify conflicts [Altmanninger et al., 2009; Buffenbarger, 1995; Westfechtel, 1991]. For example, changing the order of attributes (properties) of a UML class does not cause any syntactic conflict, but textual merging could raise merge conflicts. The second example is that the EMF/Ecore meta-model specifies every *EReference* must have a type value assigned to an *EClass*. If a merged result generates a dangling reference (a reference which have a *null* value as a type), then it violates the constraints specified by the abstract syntax. Hence, the merge tool detects syntactic conflicts.

Composite operations (which are composed of atomic operations) can be used to represent refactoring or restructuring modifications performed on software artifacts [Altmanninger et al., 2009; Mens, 2002]. Fowler [Fowler, 1999] describes refactoring as *"a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior"*. Concurrently executed refactoring operations on a software artifact by different users might lead to syntactic conflicts even though the changes are semantically equivalent [Altmanninger et al., 2009; Mens, 2002]. For instance, a pair of refactoring operations that are applied on the same variable such as *move a variable* to a super-class and *encapsulate a variable* are conflicting. In [Mens, 2002], Mens refers such type of conflicts as *structural merge conflicts*. In another work, Altmanninger et al. consider such conflicts as syntactic conflicts. We also use the term syntactic conflicts to refer this type of conflicts.

Some work has been done in the past to detect syntactic conflicts between composite operations. For instance, Cicchetti et al. use a model-based pattern language to specify and detect conflicts between composite operations [Cicchetti et al., 2008a]. Brosch et al. also employ a language-specific information and refactoring operations to detect syntactic conflicts and improve the merge results [Brosch et al., 2009b]. Besides, Koegel et al. present a conflict detection mechanism between composite changes in EMFStore framework [Koegel et al., 2010]. Furthermore, Mens et. al. apply graph transformation and critical pair analysis to detect conflicts in refactoring [Mens et al., 2005b, 2006].

Syntactic conflicts do not take the semantics of the language into account while detecting conflicts, as a result, some conflicts could be hidden and undetected. Changes made by independent developers could be syntactically correct and semantically incorrect at the same time. In [Altmanninger et al., 2009; Mens, 2002], authors classify semantic conflicts into *static semantic conflicts* and *behavioral semantic conflicts*. Static semantic conflicts are conflicts that violate the constraints of the language. Concurrent modifications might interfere with each other and produce semantically inconsistent merge result. [Altmanninger et al., 2009; Mens, 2002]. Static semantics of the models can be specified using Object Constraint Language (OCL) [Warmer and Kleppe, 2003], afterwards, the models are evalu-

ated for conformance. For example, one of the static semantics specified an *EClass*, which is specified by the EMF/Ecore meta-meta-model, is that each class must have a name and there must be at most one class with a given name in the same package. Any class without name or classes that have the same name in the same package are reported as violating the constraints.

Altmanninger et al. perform a semantic mapping between a modeling language and a semantic domain [Altmanninger et al., 2010]. Model transformations are employed to transform instance models into another models conform to the meta-model of the semantic domain. Subsequently, the new models are analyzed to detect static semantic conflict. In another work, Saeki et al. use ontology and reasoners to detect semantic inconsistencies in models and meta-models [Saeki and Kaiya, 2006].

Behavioral semantic conflicts occur in software artifacts that have execution semantics. This conflict can be detected by analyzing the execution behavior of the software artifacts during the runtime. These types of semantic conflicts could occur in concurrent development of software artifacts, specifically, the merge result of two versions might produce unexpected results. For instance, a net salary calculation depends on income tax rate and social security contribution. If one user modifies the income tax rate and social security and another user uses these values to calculate the net income concurrently, then this might lead to behavioral semantic conflicts. Mens uses a conditional graph rewriting approach to detect behavioral semantic conflict [Mens, 1999a]. Maoz et al. develop an ADDiff framework that transforms activity diagrams into finite automata and analyze the traces to detect behavioral semantic conflicts [Maoz et al., 2011]. In the past, some work has been done to detect behavioral semantic conflicts by using different techniques such as denotational semantics, program slicing and dependency graph [Altmanninger et al., 2009; Mens, 2002].

Taentzer et al. classify conflict detection techniques into *state-based conflict detection* and *operation-based conflict detection* [Taentzer et al., 2010]. State-based conflict detection validates the well-formedness of the merging result of concurrently modified artifacts against a set of constraints [Taentzer et al., 2010]. Operation-based conflict detection identifies conflicts by analyzing change operations that have been modified the same part of an

artifact [Altmanninger et al., 2009; Mens, 2002; Taentzer et al., 2010]. It uses conflict tables and conflict sets to detect conflicts [Mens, 2002]. Alanen et al. apply a conflict table to detect syntactic conflicts between pair of modifications of models [Alanen and Porres, 2003]. In [Koegel et al., 2010; Koshima and Englebert, 2014], authors use conflict sets to detect syntactic conflicts between concurrent modifications of EMF models. Taentzer et al. adopt graph transformations and critical pair analysis to detect operation-based conflicts [Taentzer et al., 2010]. As discussed in Section 4.1.1.2, operations capture sequence of edit operations performed by users. These sequence of operations are traces of execution and they can be used to detect static semantic conflicts as well. For instance, the sequence might represent operational semantic of a statechart or Petri net model.

Cicchetti et al. use a difference model to represent a delta between two subsequent versions [Cicchetti et al., 2008a]. They firstly compute differences between two versions by employing model comparison and differencing activities, afterwards, the delta is expressed using Added, Deleted, and Changed operations in the difference model. The difference models that represent concurrent modifications can also be used to detect conflicts. Indeed, the Added, Deleted, and Changed operations are analyzed to identify overlapping modifications. Since the operations are used to identify conflicts, this type of conflict detection approach can also be regarded as operation-based conflict detection.

Eclipse validation framework employs a *state-based conflict detection* technique to check the consistency of EMF instance models [Steinberg et al., 2009]. EMF-IncQuery applies a rule based pattern language on state of EMF instance models so as to detect constrain violation [Bergmann et al., 2011]. Taentzer et al. use graph conditions and graph constraints to detect state based constraints resulted from graph modifications.

Lessons Learned

Conflict detection is a corner stone for the realization of optimistic collaborative work. Concurrently modified software artifacts could raise different types of conflicts such as *textual, syntactic, static semantic and behavioral semantic* conflicts. Textual conflict detection approach is widely used for its simplicity and generality (tool and platform independent) [Alt-

manninger et al., 2009]. It does not take the syntax of the language into account so as to detect conflicts, as a result, it is not suitable for collaborative modeling. On the other hand, syntactic merging relies on the underlying structure of software artifacts to identify conflicts. It verifies the well-formedness of an instance model with respect to its meta-model. However, syntactic merging approach gives a false negative result due to its limitation to identify semantic conflicts. Semantic conflicts could be static semantic conflicts and behavioral semantics. Static semantic conflicts can be expressed using OCL or pattern languages and instance models are evaluated to verify their consistency against constraints. Behavioral semantic conflicts leads to unexpected behavior of the program or the model during runtime. These conflicts are identified using denotational semantic, program slicing and dependency graph [Altmanninger et al., 2009; Mens, 2002].

4.1.3 Conflict Resolution

Conflict resolution is a reconciliation phase that resolves conflicts identified using the methods discussed above. It is a critical requirement to merge two conflicting versions of models into a new version and also to ensure collaborative work in general. In [Klein and Lu, 1989], Klein classifies the conflict resolution work into a *computational model conflict resolution* and a *model of human conflict resolution*. The computational model conflict resolution provides a technical solution to reconcile merge conflicts, but it gives less emphasis on human conflict resolution behavior (i.e., group consensus, group interaction).

4.1.3.1 Computational model conflict resolution

Conflict resolution is usually done manually, but this process is time consuming and error prone to deal with large and complex models [Altmanninger et al., 2009; Mens, 2002]. Due to this, a tool support is required to assist users in reconciliation tasks. Different techniques that provide computational model conflict resolution are categorized into automization, conflict dependencies, and recommendations [Altmanninger et al., 2009]. An automization can be a semi-automatic conflict resolution or an automatic conflict resolution.

Automatic conflict resolution fully automates conflict reconciliation process [Altmanninger et al., 2009; Mens, 2002]. It detects and resolves conflicts automatically using different strategies. For instance, in multi-agent systems, autonomous agents resolve conflicts automatically [Emami and Narimanifar, 2012]. Wollkind et al. propose automated conflict resolution for air traffic management based on the negotiation model of multi-agent systems [Steven Wollkind, 2004]. Autonomous agents use a monotonic concession protocol to negotiate and find a middle ground, which can be agreed upon by all agents. In a monotonic concession protocol, each agent makes an initial proposal that is beneficiary for himself or herself, afterwards, s/he incrementally revises and compromises her/his earlier proposal to find a solution that is agreed by all agents [Endriss, 2006]. Edward proposes a *promotion strategy* that automatically resolves conflicts in collaborative applications. This strategy changes the order of sequences of operations in order to reduce dependencies among change operations and avoid conflicts. It promotes operations that depend on the previous operations into new operations that do not have any dependency on the earlier once. The promotion strategy may not be applicable in different situations [Edwards, 1997]. In [Badr, 2002], Badr et al. present a service-oriented conflict resolution control architecture that reconciles conflicts automatically in self-adaptive software.

Munson et al. [Munson and Dewan, 1994] present a flexible automatic three-way object merging framework using different merge policies. The merge policies are defined based on specific applications and collaboration contexts. In the context of model driven engineering, Mougenot et.el. propose a peer-to-peer collaborative model editing framework called D-Praxis [Mougenot et al., 2009]. D-Praxis resolves conflicts automatically based on delete semantics and Lamport's clock [Lamport, 1978]. For instance, if two operations are in conflict, then it keeps the recent operation and cancels the earlier operation, unless the earlier operation is a delete operation. This could result in lost-update problems meaning that changes performed by a user can be overwritten due to changes of the other user without his/her consent. Besides, the merged result might not reflect the intention of developers. Automatic conflict resolution is not applicable in most situations, because conflict detection and resolution is an intuitive process and domain specific [Altmanninger et al., 2009].

Semi-automatic conflict resolution provides facilities to resolve conflicts interactively [Altmanninger et al., 2009; Lippe and van Oosterom, 1992b; Mens, 2002]. There is a great deal of work that adopts the interactive conflict resolution approach. Mens et al. propose an approach to automate detection and resolution of model inconsistencies using graph transformation and dependency analysis [Mens et al., 2006]. The authors use graph transformation rules to express conflicts detection and resolutions. Besides, they apply a critical pair analysis to identify the potential dependencies between conflict detection and resolutions. This is because the resolution of one model inconsistency could cause new inconsistencies.

In [Edwards, 1997; Mens and Van Der Straeten, 2007], the authors present an interactive conflict resolution strategy called *recursive acceptance strategy* to reconcile conflicts in collaborative software applications. In this strategy, a user reconciles conflicting changes interactively and s/he removes either of conflicting operations in iterative fashion, until there is no conflict left. The authors also use a conflict tolerance strategy, which tolerates some type of conflicts. Hence, recursive acceptance strategy iterates until there is no conflict or the existing conflicts are tolerable conflicts. In [Sattler et al., 2003], Sattler et al. present an approach that interactively detects and resolves conflicts at the schema level and instance level in federated database.

Lippe et al. propose a semi-automatic conflict reconciliation approach that assist users in resolving conflicts interactively [Lippe and van Oosterom, 1992b]. SVN [Pilato et al., 2008], Git [Chacon, 2009], and CVS [Grune, 1986; Grune et al., 1989] are line-based versioning tools that provide interactive conflict reconciliation facilities. In the context of modeling, Saeki [Saeki, 2006], Saeki et al [Saeki and Oda, 2005], Brosch [Brosch, 2009], Bartelt [Bartelt, 2008], Sriplakich et al. [Sriplakich et al., 2006], Oda et al. [Oda and Saeki, 2005], Odyssey-VCS [Oliveira et al., 2005], CoObRA [Schneider et al., 2004], AMOR [Altmanninger et al., 2008], EMF Compare [6], and EMFStore [Koegel and Helming, 2010] present approaches that support interactive reconciliation facilities for conflicting versions of models. These interactive reconciliation tools return conflicts, and use icons and colors to highlight conflicts so that the user can resolve conflicts interactively.

[6] http://www.eclipse.org/emf/compare/

Some semi-automatic reconciliation techniques provide recommendation to resolve conflicts. For instance, the *explosion strategy* automatically explores all possible valid solutions that avoid conflicts, afterwards, it lets the user to select an appropriate solution. But, this approach is computationally expensive and not scalable [Mens, 2002]. In [Brosch, 2009], Brosch proposes a generic framework that guides the reconciliation process of model versioning by suggesting different resolution strategies. Zimmermann et al. apply data mining techniques [Han, 2005] to version histories to guide the software development process [Zimmermann et al., 2004b]. This tool recommends possible changes that need to be performed, and it shows warning messages for missing changes.

The merge tools should not display all conflicts to users in order to resolve conflicts interactively. Although semi-automatic conflict resolution is helpful in guiding the reconciliation process, it becomes tedious for the user to interactively resolve all detected conflicts. As a consequence, most versioning tools support both automatic and semi-automatic conflict reconciliation approaches. Edwards [Edwards, 1997] proposes a framework that provides both automatic and semi-automatic conflict reconciliation as discussed above. EMFStore [Koegel and Helming, 2010] also provides such a facility and it classifies conflicts based on their severity level as hard conflicts and soft conflicts. Hard conflicts occur when conflicting changes that cannot be applied on the model without losing the intention of the user [Koegel et al., 2010]. For example, a delete operation and an update operation result in a hard conflict. Hence, there is a need for users interaction and EMFStore provides a semi-automatic conflict resolution facility for hard conflicts. Whereas soft conflicts imply a set of conflicting changes that cause inconsistencies, but they can be applied to the model without loss of the intention of the users. For example, adding two different elements into a multi-valued feature adds the two elements, but indexes (positions) of the elements in the list depend on the serialization of the operations. EMFStore uses default or user defined strategies to resolve soft conflicts automatically.

Conflict tolerance is sometimes necessary in some specific situations [Mens, 2002], for example, at the early stage of development of a system, the requirements of the system are not clear enough and users might have limited understanding about the system as well,

hence, s/he might want to tolerate some set of conflicts. Therefore, the collaborative development tools should provide facilities to relax conflict detection and reconciliation process. Edwards proposes a conflict tolerance technique in collaborative software applications [Edwards, 1997].

Lessons Learned

Conflict reconciliation is an important activity to ensure model merging in particular and collaborative modeling in general. The manual conflict resolution approach is time consuming and error prone [Altmanninger et al., 2009; Mens, 2002]. On the other hand, the automatic conflict reconciliation approach automatically resolves conflicts based on different strategies. But this approach is not applicable in most situations due to the fact that conflict resolution is a domain specific activity. The semi-automatic conflict approach resolves conflicts interactively by asking users to encode domain knowledge. This approach is more suitable for collaborative software development (modeling) compared with the above two conflict resolution techniques.

4.1.3.2 Model of human conflict resolution

Much of the work on human conflict resolutions is dealing with competitive conflicts as well as psychology of human participants, who have high-level cognitive skills [Klein and Lu, 1989]. The psychological aspect of the participants and high-level cognitive skills are abstractions that are difficult to encode into machine-based design agents. In addition, as stated above, we are interested in cooperative conflicts that commonly occur during collaborative modeling. In this model, interactions among all concerned members of the collaborative group should be taken into account in order to identify and resolve conflicts (build group consensus). However, most collaborative development tools adopt a "checkout" $\rightarrow$ "modify" $\rightarrow$ "merge" workflow. This workflow puts all the burdens of conflict detection, reconciliation, and merging tasks on a single user who commits lastly. Barteit et. al. [Barteit et al., 2009] state conflict reconciliation tasks are too complex for a single user to merge inconsistent models. The user might wrongly understand intentions of other modelers, who

performed changes that were already committed to the repository. As a consequence, the integrated model might be inconsistent, hence, there is a strong need to participate other members during conflict reconciliation process.

Consensus building, voting, and leadership are different human conflict resolution approaches that are adopted by a wide range of disciplines [Michigan State University, 2007]. In case of conflicts, a decision has to be made to accept one and reject the other solution, or a compromise will be made to find an acceptable solution by all members. A consensus building approach takes concerns and ideas of all members in order to reach an agreement [Bressen, 2007]. This approach creates unity among members of a group rather than competition and polarizing effects (winner/loser). Moreover, the decision can be effectively implemented, because members are motivated to apply decisions that consider their inputs. Besides, this approach produces higher quality decisions, which integrate the wisdom of more people. However, this approach is time consuming, since it needs too much time and efforts to create consensus (agreement) among all members of the group. Besides, one member of a group could potentially block the whole reconciliation process. Hence, this approach could not be suitable for collaborative modeling, where decisions are made a lot of times throughout the development process.

Voting is one of a decision making approach, where users vote on conflicting decisions and the majority wins. This approach is faster than the consensus building approach to make decisions [Bressen, 2007]. However, voting creates friction and competition among members of the group due to the fact that the minority of the group are always overruled by the majority, regardless of the quality of their contributions. In collaborative modeling, users have different expertise and domain knowledge so that the minority of users might contribute a better proposal than the majority. But, the voting approach compromise the quality of the decision (solution). In [Michigan State University, 2007], the use of voting approach is discouraged for making decisions that are so important. Hence, the voting approach could not be suitable for collaborative modeling. COMA [Rittgen, 2008] and Collaboro [Cánovas Izquierdo and Cabot, 2012] are collaborative modeling tools that adopt voting technique to resolve conflicts.

Leader based decision making approach is usually fast compared with voting and consensus building techniques [Michigan State University, 2007]. Indeed, a leader of a group should consult and take inputs from other members to make a decision. This improves the quality of the solution and ensures effective collaboration. COMA [Rittgen, 2008] adopts a leader based decision making approach, specifically, it uses a seniority rule to facilitate the decision. Only the senior member of the group can make the decision during conflicting modifications of models, but other members can indirectly influence the decision by communicating their concerns and ideas. Since the leader based decision making process is fast and taking the seniority level of the user into consideration, it is suitable for collaborative modeling that has users with different expertise and domain knowledge.

Lessons Learned

A human conflict resolution approach, which is based on consensus building, produces solutions of good quality in their realization and increases members participation. Besides, it creates unity among member of the group. However, it takes too much time and one member of the group can veto the whole reconciliation process. As a result, it might not be a suitable approach for collaborative modeling, which has a series of decision making stages throughout the development life cycle. On the other hand, voting is fast, but it produces less quality results. Leader based reconciliation approach is faster than the two decision making approaches such as consensus building and voting. Moreover, it takes the seniority of the user into account, hence, it is a convenient approach for collaborative modeling.

4.1.4 Model Merging

Merging is a process of integrating concurrently edited software artifacts such as models into a new version of model. In [Altmanninger et al., 2009; Conradi and Westfechtel, 1998; Mens, 2002; Westfechtel, 1991; Wuu, 1994], authors classify merging into three categories such as *raw merge, two-way merge, or three-way* merge. Raw merge is a naive approach that combines change operations performed by two modelers sequentially. Sequential application of change operations could lead to conflicts. For example, suppose one user deletes

a model element, while another user renames the same model element. In raw merging, if we apply the delete operation followed by the update operation, then the conflict will be raised. Besides, this approach is susceptible to the "lost-update" problem, the previous modifications are lost as a result of recent modifications. The merged model does not reflect the intention of the users, therefore, this approach is not suitable for collaborative modeling.

Two-way merge compares two concurrently modified models and integrates them into a new version [Altmanninger et al., 2009; Conradi and Westfechtel, 1998; Mens, 2002; Westfechtel, 1991]. This merging technique does not consider the ancestor model during comparison phase, as a consequence, it is impossible to differentiate the modified model elements in the two versions. In addition, there is not enough information to know whether a new model element is created or deleted in these versions. Indeed, this information is crucial for conflict detection and reconciliation. Hence, the quality of the merged model might not be good. Therefore, this merging technique is not a good candidate for collaborative modeling.

Like the two-way merge, three-way merge compares two versions (i.e. models) in order to integrate them into a new version [Altmanninger et al., 2009; Conradi and Westfechtel, 1998; Mens, 2002; Westfechtel, 1991]. But, it also considers the base version (i.e. ancestor model) to calculate deltas. The common ancestor helps to identify edit operations between two versions of models more precisely. Moreover, this merging technique provides a better conflict detection facilities, as a result, the merged result has a better quality. In general, this merging technique mitigates the aforementioned drawbacks of two-way merge so that it gives a better result. Hence, it is widely adopted technique in collaborative software development and collaborative modeling. For example, EMF Compare, EMFStore [Koegel and Helming, 2010], and AMOR [Altmanninger et al., 2008] support three-way model merging.

Based on the underlying model comparison approach, merging technique can be classified into textual, syntactic, semantic, or structural merging [Buffenbarger, 1995; Mens, 2002]. Textual merging relies on text-based comparison, whereas syntactic merging uses syntactic model comparison. The semantic and structural merging techniques adopts semantic and structural comparison approaches, respectively. The reader can consult Section

4.1.1 for detailed descriptions about textual, syntactic, semantic, and structural comparisons. In [Ignat and Norrie, 2004; Kessentini et al., 2013; Lippe and van Oosterom, 1992b; Mansoor et al., 2015; Mens, 2002], the authors categorize merging techniques into state-based merging and change-based merging approaches. State-based and change-based merging techniques employ state-based comparison and change-based comparison, respectively.

Mansoor et al. and Kessentini et al. consider model merging as an optimization problem, and they apply search-based model merging techniques to produce a tentative merged model. The authors use genetic algorithm [Goldberg, 1989] as a global heuristic search technique [Kessentini et al., 2013; Mansoor et al., 2015]. However, we adopt the first classification (i.e., raw merge, two-way merge, and three-way merge) in this work.

Lessons Learned

Raw and two-way merge techniques do not ensure a good quality for the integrated model, therefore, they are not convenient approaches for collaborative modeling. In contrast, three-way merge derives more accurate deltas and identifies conflicts that cannot be captured by the previous ones. As a result, the quality of the merged result improves. Therefore, collaborative modeling tools and techniques should adopt the three-way merge to ensure better qualities of work.

4.1.5 Model Versioning

Model versioning is a crucial activity to manage the history of model evolution and to ensure collaborative modeling [Conradi and Westfechtel, 1997, 1998; Roebuck, 2011]. In [Conradi and Westfechtel, 1997, 1998], the authors use a *version model* which specifies software artifacts to be versioned and the organization of versions. Each version is uniquely identified in the version model. The version model also describes a sequence of operations that are necessary to construct a new version or to recover existing versions. The authors also classify the version model into *product space* and *version space*. The product space merely describes models (model elements and their structural relationship) without taking versioning into account. Of course, it assumes there exists only one version of the model. The

models should be uniquely identified in the product space using either user defined unique identifiers or system-generated unique identifiers. A coarse-grained model identification allocates a unique identifier to each model in the product space. In contrast, a fine-grained model identification uniquely identifies each model element in the product space.

Version space defines all versions and their relationships [Conradi and Westfechtel, 1997, 1998; Langer, 2011]. A version is defined by a tuple, $\upsilon = (ps, vs)$, where *ps* and *vs* represent a state of a model in the product space and a specific point in the version space, respectively [Conradi and Westfechtel, 1997, 1998]. The point in the version space corresponds to a specific point in the development time-line. Versions can be further decomposed into *revisions* and *variants*. The variant is one of the variability of the models that co-exist in parallel, whereas a revision represents the evolution of a variant overtime [Conradi and Westfechtel, 1997, 1998; Seiwald, 1996]. Like software artifacts, versions should be also distinguished using unique identifiers.

A delta specifies the relationship between two versions, specifically, it represents the differences between these versions. Conradi et al. identify two types of deltas such as *symmetric delta* and *directed delta* [Conradi and Westfechtel, 1997, 1998]. The symmetric delta denotes differences between the state of two versions, this delta is derived using state-based comparison technique, $(\upsilon_1 \backslash \upsilon_2 \ \cup \ \upsilon_2 \backslash \upsilon_1)$. In contrast, the directed delta represents a sequence of edit operations that adapt models from one version (υ_1) to another version (υ_2). For instance, EMFStore employs the directed delta technique to capture differences between two versions.

Conradi et al. classify versioning into extensional and intensional versioning [Conradi and Westfechtel, 1997, 1998]. In *extensional versioning*, each version is explicitly defined and has been sequentially checked-in to the version control system. Extensional versioning retrieves a specific version using its version identifier, afterwards, it apply changes to the retrieved version so as to reconstruct one of the previously created version. Version identification, immutability, and efficient storage facilities are important to ensure extensional versioning. *Intensional versioning* constructs different consistent versions automatically from a large version space. This versioning technique is flexible and it constructs different

versions on demand. A user annotates a specific version by its properties and queries the version space so as to construct a new version, which satisfies the query. Intensional versioning is suitable for situational method engineering, where the queries select method fragments which are later assembled together and produce situational methods [Brinkkemper et al., 1999; Gupta and Dwivedi, 2012; Mirbel and Ralyté, 2006; Ralyté et al., 2003, 2007; Saeki, 2006; Saeki and Oda, 2005].

Depending on the representation of modeling artifacts and deltas, model versioning is also categorized into *state-based versioning* and *change-based versioning* [Conradi and Westfechtel, 1997, 1998]. State-based versioning represents models as a set of entities and relations in the product space. It also describes the delta in terms of states of entities and relations in first and second version of model. The entities and relations are marked in the delta as present, if they exist in the second version. If not, they are noted as absent in the delta. Change-based versioning expresses models using change operations, which are necessary to produce the models. Like models, the delta is also represented using operations that modify the model from one version to another.

Lessons Learned

Model versioning is crucial to manage a history of model evolution, the history records modifications of models, dates of modifications, and identities of users who created revisions [Estublier, 2000] along with a comment (or rationale of modifications). Model versioning also facilitates collaborative modeling by providing a versioning support. A version can be a variant or revision, but we only consider revisions in this work. Furthermore, extensional versioning explicitly enumerates all available versions in the versions set. It reconstructs one of the previously created version by applying a sequence of changes to a reference version, which is selected from the version set. Intensional versioning automatically constructs new versions based on properties or queries. The list of versions are not known beforehand in intensional versioning. This work only focuses on extensional versioning, where each version is explicitly specified by the users. Moreover, state-based versioning approach is

suitable for frameworks that relies on state-based comparison, while change-based versioning is adopted by operation-based model collaborative modeling frameworks.

4.1.6 Model Transformation

Model transformation is the *heart* and *soul* of model driven software development [Sendall and Kozaczynski, 2003]. A model transformation automatically generates a target model from an input model based on a transformation definition [Gomes et al., 2014; Kleppe et al., 2003; Mens and Gorp, 2006; Mens et al., 2005a]. Of course, both input and target models conform to their respective meta-model (see Figure 4.3). The transformation definition (TD) specifies a set of transformation rules that define how the source model (Ma) is transformed into the target model (Mb). The transformation definition is also a model, and it is defined using the transformation language (MMt). For instance, ATL [Jouault et al., 2008], ETL [Kolovos et al., 2008], and Henshin [Arendt et al., 2010] are model transformation languages. The transformation rules are applied on either an abstract syntax or a concrete syntax of the model [Syriani and Vangheluwe, 2009]. These rules are executed on the transformation engine [Jouault et al., 2008]. Moreover, MMa, MMb, and MMt conform to MMM (meta-meta-model e.g., EMF, MOF, . . .).

According to Mens et al. [Mens and Gorp, 2006; Mens et al., 2005a], *endogenous model transformation* is a model transformation which has the same source (input) and target (output) meta-models. For instance, a sequence of change operations adapt a model from one version to another, while the meta-model remains the same, is an endogenous model transformation. In *exogenous model transformation*, the source and target models are expressed using different languages. For example, code generation, reverse engineering, and migration are exogenous model transformations. Furthermore, the authors also classify model transformations based on the level of abstraction as *horizontal transformation* and *vertical transformation*.

A horizontal transformation is a model transformation between models at the same level of abstraction. For example, an EMF/Ecore model should be transformed to a MOF model so as to edit the model using MOF aware tools. This type of transformation is a horizontal

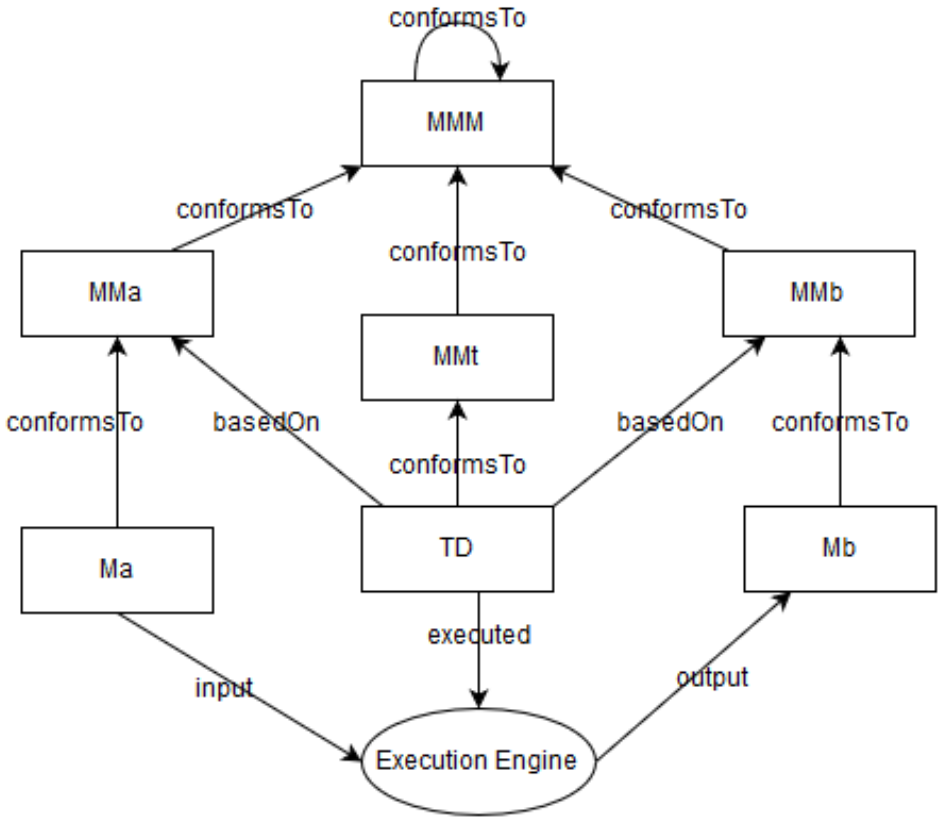

Figure 4.3 Model Transformation, [Jouault et al., 2008]

transformation. In contrast, a vertical transformation is a transformation between source and target models that are at different level of abstractions. For instance, a transformation between platform independent model and platform specific model is a vertical transformation. Moreover, model transformations further decompose into *in-place* and *out-place* transformations [Czarnecki and Helsen, 2006; Mens and Gorp, 2006; Mens et al., 2005a]. In-place transformation is an endogenous model transformation, where the source and the target models are the same. On the other hand, out-place transformation is an exogenous or an endogenous model transformation, which uses different source and target models.

Model transformation languages adopt either a declarative, an operational, or a combination of operational and declarative approaches to specify transformations [Czarnecki and Helsen, 2006; Mens and Gorp, 2006; Mens et al., 2005a]. The declarative approach only describes what needs to be transformed into what without giving any details about how the transformation is performed. Activities like navigation of the source model, creation of

target model, and managing the order of rule execution are handled by the transformation engine. It is easier to write and understand by users, sine they only needs to define relations between source and target models. In contrast, the operational approaches specify each steps necessary to produce target models from source models. It is a convenient approach for controlling sequences of execution of transformations explicitly. However, an implementation of transformations using imperative languages is very complex task.

On the other hand, the hybrid approach, which combines both declarative and operational approaches, lets users to specify model transformations using mixtures of declarative and operational approaches. This approach combines the strength of a declarative approach (i.e., easy to write and understand transformations) and an operational approaches (i.e., controlling the execution of model transformations). For instance, ATL [Jouault and Kurtev, 2006], QVT [OMG, 2005], and ETL [Kolovos et al., 2008] are hybrid model transformation languages.

A model transformation has a specific intent. According to Amrani et al., *"a model transformation intent is a description of the goal behind the model transformation and the reason for using it"* [Amrani et al., 2012; Lúcio et al., 2014]. The authors identify different catalogs of intents like refinement (i.e., synthesis and serialization), abstractions (i.e., abstraction, reverse engineering, restrictive query, and approximation), semantic definition (i.e., translational semantics and simulation), language translation (i.e., translation and migration), constraint satisfaction (i.e., model finding and model generation), analysis, editing (i.e., model editing, optimization, model refactoring, normalization, and canonicalization), model visualization (i.e., animation, rendering, and parsing), and model composition (i.e., model merging, model matching, model synchronization).

A synthesis transformation produces an executable output which is specified using a well-defined language format. For instance, a model-to-code generation is a synthesis transformation. Likewise, a serialization transformation is also a refinement and its mere objective is to store models into the storage medium (e.g., serialization of an Ecore model to XMI so as to store on hard drive) [Amrani et al., 2012; Lúcio et al., 2014]. Abstraction is a vertical model transformation that abstracts models by hiding some detail information and

revealing other information. For example, a reverse engineering and an approximation transformation are abstraction model transformations. An approximation transformation refines a model m_1 into a model m_2 with some degree of errors, indeed, the error margin decreases as long as the distance measure between the two models decreases [Amrani et al., 2012].

A translational semantic definition is a model transformation which expresses the meaning of the source models in terms of other modeling languages with a wellknown semantics [Amrani et al., 2012; Lúcio et al., 2014]. In other words, it transforms the abstract syntax of a source model to the target semantic domain with a well-defined semantics. The output of a translational semantic transformation is used to perform a model simulation. Language translation is a model transformation that expresses concepts and semantics of one modeling language in terms of another target language [Amrani et al., 2012; Lúcio et al., 2014]. For example, a mapping from a UML class diagram to a relational database schema is a translation model transformation. A model migration transforms models expressed in one language to models conform to another language (new version language). Model migration is a crucial activity in model management, for example, due to language evolution, models might not conform to a new version of the language, hence, instance models should be migrated.

Model Migration

As the name suggests, DSML specifies systems in specific domains, therefore, the language evolves due to new requirements, error corrections, and improvement of understanding about the domain [Gruschko et al., 2007; Meyers and Vangheluwe, 2011; Wachsmuth, 2007; Zhang, 2005]. Indeed, DSMLs evolve by modifying their respective meta-models in order to satisfy new requirements. As a result, instance models might not anymore satisfy the structures and constraints specified by new versions of their respective languages. It could also be impossible to edit models using editors built for an old meta-model. Therefore, models should be co-evolved with their respective meta-models in order to preserve the conformance relation with the language [Herrmannsdoerfer et al., 2008]. For example, as shown in *(Figure 4.4)*, a model version V_{m_0} conforms with meta-model version V_0, but it vi-

olates constraints defined by meta-model version V_1. Therefore, a model version Vm_0 needs to be migrated to version V_{m_1} so as to conform with evolved meta-model, V_{m_1}.

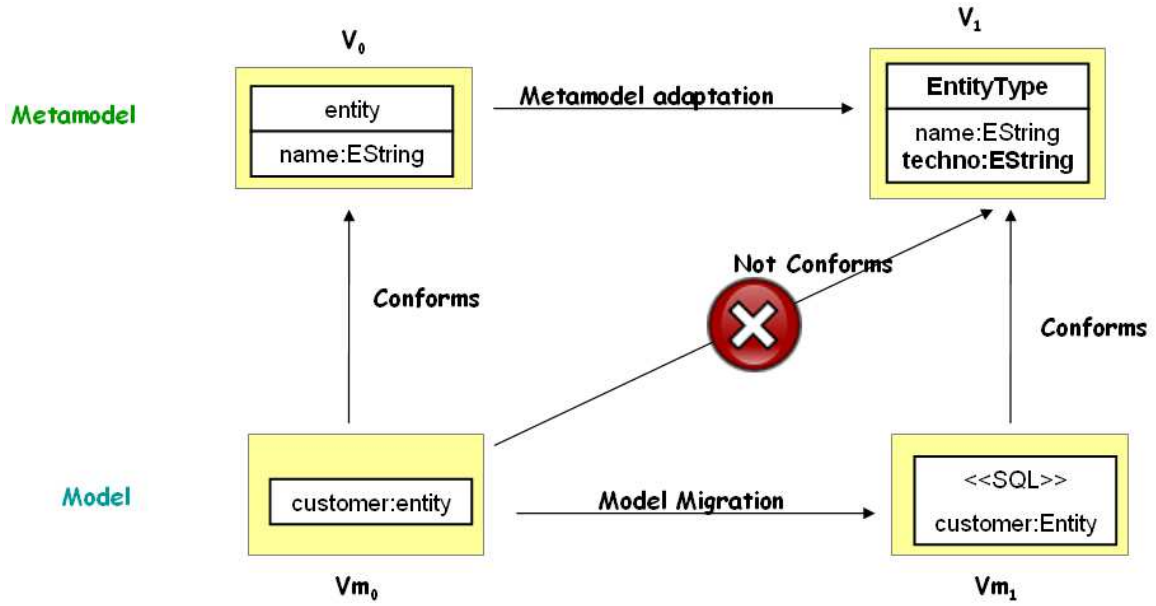

Figure 4.4 Example of Model Migration

Gruschko et. al. classify meta-model changes as *"not breaking"*, *"breaking and resolvable"*,and *"breaking and unresolvable"* changes [Gruschko et al., 2007]. Not breaking changes are meta-model modifications that do not break the conformance relationship between a meta-model and its instance models. For example, adding optional attributes to meta-model elements does not affect its instance model. On the other hand, renaming a meta-model element has an impact on its model. This type of change is called breaking and resolvable changes. The instance models can be automatically migrated in response to resolvable changed. Furthermore, some changes require user inputs in order to resolve conflicts and migrate instance models. This type of changes are breaking and unresolvable changes. For example, introducing mandatory attributes to meta-model elements requires a user to give initial values, since they cannot be derived from the information available in models. It is even not possible to assign default values to these attributes sometimes.

In [Rose et al., 2009], authors identify three types of model migration strategies such as *manual specification*, *operator-based approach*, and *meta-model matching approach*.

Graph transformation languages could also be used to migrate models, but as shown in [Rose et al., 2012], they are not easy to understand and results of migration could be incorrect. The meta-model matching approach adopts a state-based model comparison technique to drive the *diff* between the source and target meta-models, afterwards, it computes model migration instructions automatically for breaking and resolvable changes. In addition, it provides facilities to specify model migration instructions for unresolvable modifications. In [Cicchetti et al., 2008b] and [Garcés et al., 2009], the authors present model migration strategies based on a meta-model matching approach. However, this approach can not always automatically generate correct migration instructions [Rose et al., 2009]. Hence, it is not suitable for domains that require completeness, correctness, and predictability.

The operation-based approach uses a set of reusable coupled operations, which are integrated with the modeling tools to adapt meta-models and to derive a migration strategy at the model level. Coupled operations are high level operations (not CRUD operations) that are enriched by model migration instructions, such that they capture model migrations while adapting the meta-model [Herrmannsdoerfer, 2009; Herrmannsdoerfer et al., 2008]. The operation-based approach records a time sequence of modifications and keeps change operations in a same context (change operations which are contained in a composite operation) while incrementally adapting meta-models. Hence, it captures intentions of a user. For instance, COPE (recently called Edapt [7]) captures sequences of meta-model adaptation operations and couples them with model migration instructions [Herrmannsdoerfer et al., 2008].

As discussed in Chapter 1 in Section 1.1, users most of the time develop models in collaboration. They communicate their activities by exchanging models or edit operations performed on models. Specifically, operation-based collaborative modeling frameworks mostly normalize (canonize) sequences of modification operations, which are used as a means of communication among members, so as to speed up the transfer and to reduce the complexity of the merging process. Hence, operations can be superseded by new ones and can thus be cleaned from the history. For example, a create operation will be removed if it

[7] http://www.eclipse.org/edapt/

is followed by a delete operation on the same object. Canonization could also be applied for a library of coupled-operations defined [Koshima et al., 2013].

A coupled operation could be composed of one or many primitive operations (i.e. create, delete, set, add and remove) and model migration instructions. For example, in the Edapt framework [Herrmannsdoerfer et al., 2008], a *Create Attribute* coupled-operation is composed of primitive operations that create an attribute and set values for name, type, minimum and maximum cardinality, and default value of a newly created attribute. For instance, if a *Create Attribute* coupled-operation is followed by a *Change Attribute Type* coupled-operation, a primitive operation that sets a type of an attribute in *Create Attribute* coupled-operation needs to be deleted due to canonization. As a result, the *Create Attribute* coupled-operation becomes invalid; the canonized set of primitive operations do not satisfy constraints that state a *Create Attribute* coupled- operation must set a type value of a newly created attribute. Hence, sets of canonized primitive operations that are composed by coupled-operations need to be re-grouped into new sets of valid coupled-operations. The re-grouping of primitive operations are required to reuse existing model migration instructions of coupled operations or derive new model migration strategies.

Users should manually specify model migration instructions for breaking primitive change operations that are not composed by coupled operations. However, this is a tedious and error prone task [Koshima et al., 2013]. Furthermore, some of primitive change operations might need to be re-grouped into a set of coupled operations manually. This requires users to know the structure and constraints of each coupled operation in order to create a correct coupled operation. Besides, canonization also makes re-grouping more complex by removing some primitive change operations. Removing of primitive operations could invalidate constraints of a coupled operation. For example, a *Create Attribute* coupled-operation becomes inconsistent, if a *Set Attribute Type* operation is removed from it. Hence, model migration cannot be applied due to inconsistencies.

In manual model migration, users specify model migration instructions manually [Rose et al., 2009]. This approach does not require a library of coupled-operations so that users define model migrations instructions using tools of their preferences. In addition, it gives a

better control for users to manage model migration. But, manual specification needs more effort from a user to write model migration instructions and it could be error prone as well. Besides, it does not reuse recurring patterns in the model migration. For example, Epsilon Flock is a domain specific language to specify model migration instructions manually [Rose et al., 2010], but it reduces manual efforts by employing a conservative copying algorithm, which automatically copies only model elements that conform to a new meta-model. Besides, it provides facilities to execute migration instructions. In the 2010 transformation tool contest, it performed better in terms of correctness, conciseness, understandability, extensions, and appropriateness than other model migration techniques that participated in the contest [Rose et al., 2012]. ATL [Jouault and Kurtev, 2006] and QVT [OMG, 2005] are also manual model migration languages.

Last but not least, meta-model adaptation could also cause inconsistency in histories of instance models [Koshima et al., 2013]. For example, if a class is deleted from a meta-model, then instances of the same class type need to be deleted from the migrated model. As a result, these history elements (i.e. change operations create, set and add) that manipulate instance model elements are referring to classifiers that do not exist anymore in the modeling language. Hence, that history also needs to be migrated along with its instance model. A model migration instruction used to migrate instance model can also be used to migrate a history as well. A change operations that refer a classifier or a feature that does not exist should be removed.

Lessons Learned

Model transformation is the heart and soul of model driven software development [Sendall and Kozaczynski, 2003]. It encompasses a wide range of model management activities such as model refinements, abstraction, migration, refactoring, editing, simulation, visualization and composition. Mens et al. identify classification criteria for model transformation as *endogenous vs exogenous* transformation, *horizontal vs vertical* transformation, and *in-place vs out-place* transformation. In collaborative modeling, users usually edit models in parallel, while their respective meta-models do not change. These modifications are endogenous, hor-

izontal, and in-place model transformation. Sometimes, users also apply exogenous, vertical, and out-place model transformations. For example, source code generation, refinement, and abstraction activities are exogenous, vertical, and out-place model transformations.

Model migration is a special type of model transformation, which restores the conformance relationships between models and their respective meta-models. Model migration are classified as *manual specification* based model migration, *operator-based* model migration, and *meta-model matching* based model migration [Rose et al., 2009]. Meta-model matching based model migration automatically derives model migration instructions from deltas, which is the difference between the source and target meta-models. However, it might generate incorrect migration instructions [Rose et al., 2009].

Operator-based model migration relies on library of coupled operations, which are integrated with modeling tools. A user adapts meta-models using coupled operations, afterwards, model migration instructions are generated from these operations. In operation based collaborative modeling, canonization of change operations is important to reduce complexities of merging process. However, canonization could result in a sequence of inconsistent coupled operations, which hinder model migration. For example, in Edapt [8] [Herrmannsdoerfer, 2009; Herrmannsdoerfer et al., 2008], a *Create Attribute* coupled-operation is composed of primitive operations that create an attribute and set values for name, type, minimum and maximum cardinality, and default value of a newly created attribute. For instance, if a *Create Attribute* coupled-operation is followed by a *Change Attribute Type* coupled-operation, a primitive operation that sets a type of an attribute in *Create Attribute* coupled-operation needs to be deleted due to canonization. As a result, the *Create Attribute* coupled-operation becomes invalid; the canonized set of primitive operations do not satisfy constraints that state a *Create Attribute* coupled-operation must set a type value of a newly created attribute. Hence, sets of canonized primitive operations that are composed by coupled-operations need to be re-grouped into new sets of valid coupled-operations. Primitive operations are re-grouped into a coupled-operation in order to use model migration instructions defined by the coupled-operation. For primitive operations that are not composed

[8] https://www.eclipse.org/edapt/

by coupled operations and make instance models inconsistent, a model migration instruction should be written manually. Manually incorporating model migration instructions is a tedious and error prone task [Koshima et al., 2013].

Manual model migration does not need a dedicated modeling tool, rather users use any convenient modeling tools to update meta-models and to write model migration instructions manually. This approach is flexible and provides a better control to manage model migration. However, it is time consuming and error prone. Therefore, a tool support is mandatory to write correct and useful model migration instructions. Epsilon Flock [Rose et al., 2009] is a model migration language, which provides tool support to specify model migration instructions manually. This language outperforms other languages in terms of correctness, conciseness, understandability, extensions, and appropriateness in transformation tool contest 2010 [Rose et al., 2012]. Of course, the manual model migration approaches can be augmented with a history of meta-model changes in order to improve the correctness and usefulness of migration instructions. The history can correctly identify deleted, updated, created and moved model elements, and guide users in writing correct model migration instructions.

4.2 Communication Management

4.2.1 Member Organization

Modeling social structures and interactions in collaborative work is crucial to analyze users' collaboration, besides, it helps to classify, organize, and represent users based on their roles [Penichet et al., 2007]. According to Montes et al. [Montes et al., 2006], a social structure is defined as *"a collection of actors responsible for carrying out group tasks and set of social dependencies among them"*. User activities and interactions can be described using activity theory. The basic unit of activity theory is a human activity [Georg, 2011; Kuutti and Arvonen, 1992], which provides enough contextual information to analyze interactions in the social structure (See Figure 4.5 [9]). Human activities are mediated by tool and rules.

[9] modified version of basic structure of activity [Kuutti and Arvonen, 1992]

The collaborative modeling environment provides tool support, which mediates interactions between a user and a model. The user transforms the model into a product using the tool. In addition, the interactions of users in community are mediated by rules and communication protocols. The communication protocol can be specified using collaboration patterns for software process development [Vo et al., 2015]. The interaction between the community and the shared modeling artifacts is mediated by the division of labor. Roles can be used to divide activities among members collaborative work.

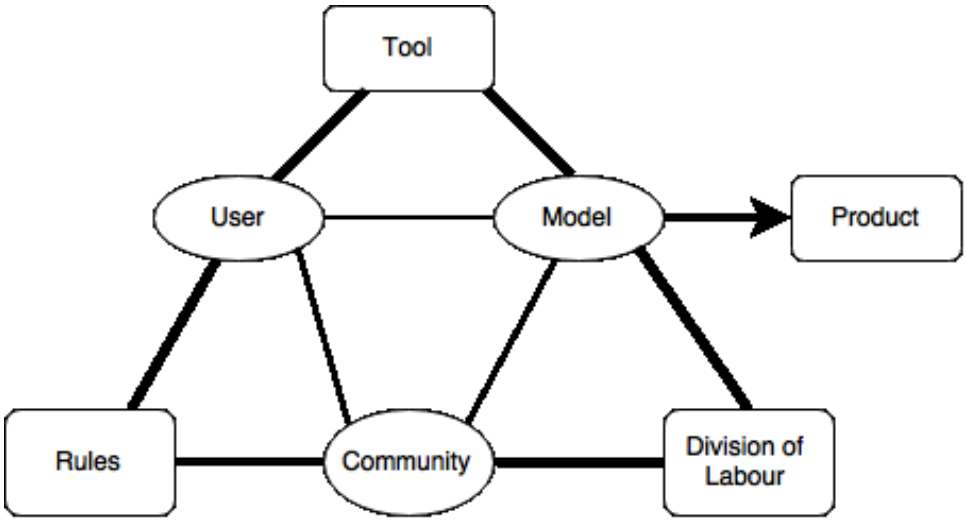

Figure 4.5 Structure of an activity

Penichet et al. [Penichet et al., 2007] model the organizational structure of users using concepts such as actor, user, role, instantiation relationships, group, and aggregation relationships. An actor is an instance of a role that can perform a task. It can be a person (a user) or some other things (not users). A role is a group of actors that perform the same tasks based on shared characteristics. It describes the relationship between actors and a shared work object as well as their interactions with the community. It is specified in terms of responsibilities and rights [Hamadache and Lancieri, 2009]. Indeed, it is assigned to an actor such that the actor must perform actions to fulfill its responsibilities. An actor can perform other actions provided that his/her rights allow him/her to do so. For example, an *editor* role lets a user to modify a model, whereas an *observer* gives read only access to a user. Instantiation relationships describes an instance of relationship between a role and an

actor, which plays such role. Aggregation relationships represents an association between the whole and its parts.

According to [Dourish and Bellotti, 1992],*"there is a further problem for role restrictive CSCW systems, This is that, although explicit roles may allow for easier social organisation of collaborative activity in conventional interactions and collaborations, one often observes roles being negotiated and reassigned dynamically. This phenomenon has been identified in other computer supported meeting situations where participants are released from the tyranny of restricted access to shared work spaces"*. Therefore, the roles should be dynamic in the collaborative modeling environments and a role-switching process should not be complex and time consuming, which could hampers the negotiated process [Hamadache and Lancieri, 2009]. A group consists of a number of users, who interact together to produce an outcome. A group refers a community in the basic structure of an activity.

Lessons Learned

Modeling structures of collaborative modeling provides facilities to analyze the interactions and organize users based on their roles. Roles also improve awareness of the character of the activity in the collaborative group. Roles are usually re-negotiated throughout the development process, hence, the collaborative environment should support dynamic role assignment. Furthermore, rules or communication protocols should be specified a priori in order to manage communication among members of a collaborative group. Of course, the environment should be flexible to modify existing rules or to add new ones.

4.2.2 Repository and Mode of Communication

As discussed in Chapter 3, Boukhebouze et al. classify collaborative modeling into a centralized approach with/without modification management and a distributed approach with/without modification management [Boukhebouze et al., 2010]. The centralized collaborative modeling approach uses a central repository as the one source of truth, where all models, histories, and files are stored. For example, EMFStore [Koegel and Helming, 2010], ModelBus [Sriplakich et al., 2008], and MetaEdit+ [Kelly, 1998] are centralized version control

systems. In this approach, members of a collaborative work group communicate their work either synchronously or asynchronously by adapting models.

The synchronous mode of communication allows members to concurrently edit a same model in real-time. Approaches that employ synchronous mode of communication use locking mechanisms in order to avoid conflicting modifications and ensure consistency of models. However, the locking technique is not scalable [Altmanninger et al., 2009; Barteit et al., 2009; Mens, 2002]. A fine-grained locking technique could mitigate some of the limitations of locking techniques by locking only the minimal set of model elements. For instance, two users could edit the same class diagram at same time, while one of the user edits the name of the method, the other user could edit the parameters of the method at the same time.

Asynchronous communication employs "checkout" $\rightarrow$ "modify" $\rightarrow$ "merge" paradigm, where users "checkout" models from the primary central repository into a local workspace. Afterwards, they modify local models in isolation, and periodically synchronize their activities with other group members. The synchronization is done by committing modifications to the central repository. In asynchronous communication, users are not interrupted by modifications from other members, since they can decide when to update their local workspace [Ignat et al., 2007b]. Besides, they can chose either to communicate completed or partial work with other members of the group. They can also work locally without having access to any network connection. For example, EMFStore [Koegel and Helming, 2010] and ModelBus [Sriplakich et al., 2008] support asynchronous mode of communication. In general, tools and frameworks adopt locking and merging ("checkout" $\rightarrow$ "modify" $\rightarrow$ "merge") techniques to ensure concurrent development.

In the centralized approach with modification management, the modification management role is assigned to a user who manages the evolution of models [Boukhebouze et al., 2010]. This user plays a role of software configuration manager [Estublier, 2000]. In contrast, a centralized approach without modification management does not have any software configuration manager. Hence, all users modify and reconcile their modifications by themselves. As described above in Section 4.1.3.2, a modification management role is crucial to reconcile conflicts. The main limitation of a centralized approach is that it forces all

members to be dependent on a central repository (a single point of failure). In addition, it might introduce unnecessary access right bureaucracy that could lead to dissatisfaction among members [Boukhebouze et al., 2010; Koshima and Englebert, 2014; Koshima et al., 2011, 2013]. Members do not have the entire history of model evolution that could help them to improve understanding about a project. Furthermore, dynamic allocation of modification management roles could be cumbersome and time consuming, since it might require movements of all centrally stored modeling artifacts to a newly installed central server.

In the distributed approach, clones of a master copy of modeling artifacts are distributed and stored at local repositories of each member of a group. This approach uses an asynchronous mode of communication such that users work in isolation and synchronize their activities later with other members. It provides users a better control of modeling artifacts and mitigates the problem of being dependent on a central server. Besides, it is suitable to ensure loosely coupled cooperative work, where users are free to join or leave the cooperative ensemble. In contrast, it might be difficult for a central change manager to leave a group in centralized approach. However, it is challenging to keep all copies of models consistent, since these shared models might be edited in parallel. A modification management role is crucial to ensure convergence of concurrently edited models at some point of the development process. For example, D-Praxis is a distributed peer-to-peer collaborative model editing framework [Mougenot et al., 2009].

Federation of model repositories is another way of distributed collaboration, where each organization have private repositories that link with publicly available repositories [Di Rocco et al., 2015]. The collaborative modeling environment with federation mechanisms should seamlessly aggregate modeling artifacts from different repositories to produce a product. Modelio [10] is an open source modeling environment that supports federation mechanisms. In [Iqbal et al., 2009], Iqbal et al. present a linked data driven software development approach, where the relationship between different federated software artifacts are made explicit and presented as a linked data. *"Linked Data is simply about using the Web to create typed links between data from different sources"* [Bizer et al., 2009].

[10] https://www.modelio.org/

There is also a hierarchical mode of collaboration, where companies work together to standardize models through international standards consortium like OMG [11], while developing their own models locally. Suppose SoftMDA and MDAGlobal are fictitious companies that have different branches in Europe and North America. These two companies develop DSMLs for financial systems and push their contribution to the fictitious standard agent, Financial DSML Standard (FDSMLS). The two companies take clones of the standard model and edit it locally, afterwards, they send local modifications (patches) to FDSMLS. FDSMLS modifies the standard model based on the modification requests and communicates the most recent version of the model with all members of the consortium. SoftMDA and MDAGlobal have different cooperative groups internally (in Europe and North America) that contribute to the development of the standard model. Indeed, the internal development groups of SoftMDA are transparent from both MDAGlobal and FDSMLS, the same is true for internal development groups of MDAGlobal. The collaborative modeling tool should support hierarchical mode of collaboration among FDSMLS, SoftMDA, and MDAGlobal as well as internal collaborative groups inside SoftMDA and MDAGlobal.

In recent years, cloud-based collaborative software development environment gains more attention. Cloud computing abstracts the location of the repository [Armbrust et al., 2010; Elzeiny et al., 2013; Hilley, 2009; Ju et al., 2011]. It stores data on multiple third-party servers to ensure high availability. Modeling as a Service (MaaS) could be used to create collaborative and distributed modeling tools, and to facilitate management of distributed global models [Bruneliere et al., 2010]. Morse [Holmes et al., 2012], MDEForge [Basciani et al., 2014], AToMPM [Syriani et al., 2013], and WebGME [Maróti et al., 2014] are cloud-based modeling environments. Cloud computing addresses a single point of failure limitation of centralized approach. However, it does not provide a local clone of master copy to each user site, which gives more power for users to manage their local data. This might still introduce access right bureaucracy among the group.

[11] http://www.omg.org/

Lessons Learned

Modeling artifacts can be stored on a central repository or distributed repositories. Cloud computing approach hides the location of the repositories, which could be local or remote. In centralized approach, users rely on a central repository, where all modeling artifacts are stored. This approach uses a pessimistic locking mechanism or → "modify" → "merge" paradigms to ensure consistency of modeling artifacts. The locking technique might not be salable, besides, it has a point-failure problem. It might also introduce access right bureaucracy among members of a group. A role-switching process could be cumbersome and consumes more time. Cloud computing based environments address limitations like a single point of failure and role-switching problems by using multiple servers to provide high availability. Cloud storage adopts a BASE (Basically Available, Soft state, Eventually consistent) mechanism to ensure data consistency and integrity [Elzeiny et al., 2013]. A distributed approach replicates clones of the master copy at each member site such that each user has a better control of his/her local repository.

4.2.3 Awareness

According to Dourish et al. [Dourish and Bellotti, 1992], *"awareness is an understanding of the activities of others, which provides a context for your own activity."* Awareness is crucial to successful collaboration among members of the collaborative group. There are different types of awareness such as workspace awareness, informal awareness, group-structure awareness, social awareness, and context awareness [Dirix, 2013; Omoronyia, 2008; Tacla and Enembreck, 2006]. Workspace awareness refers to up-to-the-moment knowledge about interactions of other members with the shared workspace. This knowledge helps members to improve their individual contributions with respect to group goals and progress. Group-structure awareness relates to knowledge about roles and status of users in a group. Informal awareness relates to passive information that does not disrupt the current activity of the user. For instance, information about who is around, where are they, and what are they doing.

Social awareness refers knowledge about attention, interest, emotional state, and activities of other users in shared collaborative environment. Users improve their social awareness through conversation or activities in a shared environment. Context awareness relates to knowledge about circumstances or facts that characterize particular activities and resources in a shared environment. Users can use the context information to negotiate and adapt their activities in the collaborative work. In addition, the context information is also useful to renegotiate roles in the collaborative modeling environment.

Lessons Learned

Awareness is crucial to ensure successful collaboration among members of the collaborative work group. Workspace awareness provides information about the current state of the shared workspace. Besides, additional information that specifies interactions among the shared workspace and users. Workspace awareness information like which modeling artifact is modified, when, what are modifications, and who performs modifications are crucial are critical information for model versioning and to ensure collaboration. Moreover, workspace awareness could help members to avoid conflicts and reduce a potential time that might be lost due to double work. Therefore, collaborative modeling environment must share such knowledge among members of a group. Group-structure awareness facilitates collaboration by providing information about roles and organizational structure of the group. Collaborative environments could share group-structure awareness among members in order to improve their efficiency. Informal awareness disseminates passive information among members of the group, for example, information about available members of a group. Group awareness simplifies interactions among members and improves coordination of activities [Gutwin et al., 2004]. Context awareness is crucial for a user to execute an activity in shared collaborative environment. Otherwise, s/he might perform inconsistent activity, which could hamper collaboration. Hence, collaborative modeling environments must share context information among members of a group.

4.3 Related Work of Collaborative Modeling Environments

In the past, much work has been done to support collaborative software development. However, most of this work focuses on collaborative merging of software codes, for instance, CVS [Vesperman, 2006], SVN [Pilato et al., 2008], and Git [Chacon, 2009]. Ignat et al. did a comparative analysis of different approaches that support collaborative editing of text documents [Ignat et al., 2007a]. Dewan et al. [Dewan and Hegde, 2007] propose a semi-synchronous distributed collaboration model that lets users create source codes asynchronously. Additionally, the framework provides facilities to detect and resolve conflicts synchronously.

In the context of collaborative modeling, Saeki [Saeki, 2006] introduces the use of a versioning system to control and manage models and metamodels, which evolve independently. The author did not consider collaboration in his work. In [Alanen and Porres, 2003; Oliveira et al., 2005], authors propose versioning of UML models, but they did not provide collaborative support. Constantin et al. [Constantin et al., 2009] propose a theoretical reconciliation framework for collaborative modeling, but they did not provide a solution. There are a few frameworks available that support collaboration among DSML tools. We summarize state-of-the-art tools and frameworks based on model management and user management. Specifically, we use criteria like repository mode, concurrency management, comparison mechanism, managed artifacts, role-based modification management, flexibility of roles, and hierarchical support.

The repository mode classifies frameworks into a centralized collaborative modeling approach (which uses a central repository) or a distributed model development approach (where each member has his/her own local copy) as discussed in Section 4.2.2. The concurrency management identifies the technique adopted by the framework to ensure concurrent development, which is either locking technique or merging ("checkout" $\rightarrow$ "modify" $\rightarrow$ "merge" paradigm). The comparison mechanism describes the type of model comparison approach that is used by the framework (see Section 4.1.1), such as a state-based model comparison and an operation-based model comparison. As discussed in Section 4.2.1, a role-based modification management is important to facilitate reconciliation process. A

controller manages the evolution of models, and s/he resolves conflicts in consultation with a user who proposed modifications. The flexibility of roles indicates whether the role is flexible or fixed. The flexibility of roles guarantees roles can be easily modified to cope with the dynamicity of the cooperative group. Fixed roles cannot be (easily) changed. The hierarchical support describes whether the framework supports a hierarchical mode of collaboration or not (see Section 4.2.2).

EMFStore

EMFStore[12] is an operation-based and a centralized model editing framework for EMF/Ecore models [Koegel and Helming, 2010]. It uses a merging technique to ensure concurrent development. This framework manages models. In the previous version of EMFStore, the tool that captures edit operations of model evolution is highly coupled with EMFStore framework. Besides, it does not support meta-model adaptation. For instance, a movement of an EAttribute from one EClass to another EClass generates an error in the old version of EMFStore. The most recent version of EMFStore, which is part of the Eclipse distribution, provides support for collaborative modeling of models and meta-models. The EClass and the EAttribute are concepts of EMF/Ecore meta-model which will be discussed in the next chapter. However, EMFStore does not support a role-based reconciliation mechanism, the user who lastly commits modifications is responsible to resolve conflicts. Besides, it does not provide a hierarchical collaborative modeling framework.

MetaEdit+

Like EMFStore, MetaEdit+[13] [Kelly, 1998] is a centralized modeling framework, but it implements Smart Mode Access Restricting Technology (Smart Locks ©) to support concurrent access of shared modeling artifacts that are stored centrally. As discussed in section 4.2.2, the limitations of centralized approaches are that locking technique is not scalable [Mens, 2002]. MetaEdit+ [Kelly, 1998] employs a fine-grained locking technique to miti-

[12] http://www.eclipse.org/emfstore/index.html
[13] http://www.metacase.com/products.html

gate some of the problems related to locking technique. MetaEdit+ handles collaborative editing of both models and meta-models. However, it does not support role-based reconciliation mechanism. Even though locking techniques are used to avoid concurrent modifications of the same (meta-)model elements by different users, it cannot avoid changes that might cause conflicts in users minds. Hence, the role-based reconciliation mechanism could help to review modifications and to resolve conflicts. Like EMSStore, it does not provide a hierarchical modeling environment.

ModelBus

ModelBus [Sriplakich et al., 2008] is a state-based and a centralized modeling framework. It uses locking techniques and merging to manage concurrency. It manages versioning of models and meta-models. Like EMFStore, it does not support a role-based reconciliation mechanism and a hierarchical mode of collaboration. The last user who commits his/her modifications is responsible to reconcile conflicts and merge conflicting versions.

Modelio

Modelio teamwork [14] is a state-based and a centralized modeling tool. This tool uses both locking and merging techniques to ensure concurrent editing of models. But, it does not manage versioning of meta-models. Unlike the above frameworks, Modelio provides a federated collaborative modeling environment, where interconnected models are stored on different repositories and each group contributes to their project without affecting the work of the other project participants. Each group adopts a centralized mode of collaboration, which relies on a central repository. Hence, we consider Modelio as a centralized approach. Like other frameworks, it does not provide facilities such as a role-based reconciliation and a hierarchical collaborative modeling environment. The last user who commits his/her modifications is responsible to reconcile conflicts and merge conflicting versions.

[14] https://www.modeliosoft.com/en/modules/teamwork-manager.html

MagicDraw

Like ModelBus, MagicDraw teamwork [15] is a state-based and a centralized modeling tool. Like Modelio, this tool uses both locking and merging techniques to manage concurrency. It only manages versioning of models, not meta-models. Like other frameworks, it does not provide facilities such as a role-based reconciliation and a hierarchical collaborative modeling environment. The last user who commits his/her modifications is responsible to reconcile conflicts and merge conflicting versions.

Visual Paradigm

Like ModelBus, Visual Paradigm teamwork [16] is a state-based and a centralized modeling tool. Like Modelio, this tool uses both locking and merging techniques to manage concurrency. It only manages versioning of models, not meta-models. Like other frameworks, it does not provide facilities such as a role-based reconciliation and a hierarchical collaborative modeling environment. The last user who commits his/her modifications is responsible to reconcile conflicts and merge conflicting versions.

D-Praxis

D-Praxis [Mougenot et al., 2009] is an operation-based and a peer-to-peer (distributed) collaborative modeling framework. This framework relies on merging technique to ensure concurrency. It uses Lamport clock [Lamport, 1978] and delete semantics to automatically solve conflicts. It supports concurrent editing of meta-models. This framework has a 'lost-update' problem and we argue that the final results of an automatic reconciliation process could not reflect the intention of users. Like other frameworks, it does not provide a hierarchical collaborative modeling environment.

Table 4.1 summarizes the state-of-the-art modeling tools and frameworks based on the aforementioned criterion such as repository mode, concurrency management, comparison

[15] http://www.nomagic.com/products/magicdraw.html
[16] http://www.visual-paradigm.com/

mechanism, managed artifacts, role-based modification management, flexibility of roles, and hierarchical support.

(R1) Repository mode : it describes whether tools and frameworks adopt a centralized repository or a distributed repository.

(R2) Concurrency management : it describes the mechanism of concurrency management

(R3) Comparison mechanism : it describes techniques adopted by modeling tools and frameworks to compare (meta-)models.

(R4) Managed artifacts : it specify the type of modeling artifacts (i.e., model, meta-model) that are managed by modeling tools and frameworks.

(R5) Role-based modification management : it specified the management of (meta-)model evolution based roles.

(R6) Flexibility of roles : it specifies the dynamicity of roles. Can someone easily change roles of a user?

(R7) Hierarchical support : it specifies whether modeling tools and frameworks support a hierarchical mode of collaboration.

Table 4.1 Summary of state-of-the-art collaborative modeling tools and frameworks

Tools and Frameworks	R1	R2	R3	R4	R5	R6	R7
MetaEdit+	Centralized	Locking	—	Model/Meta-model	—	—	—
EMFStore	Centralized	Merging	Operation-based	Model/Meta-model	—	—	—
D-Praxis	Distributed	Merging	Operation-based	Meta-model	—	—	—
Modelio	Centralized	Locking/Merging	State-based	Model	—	—	—
MagicDraw	Centralized	Locking/Merging	State-based	Model	—	—	—
Visual Paradigm	Centralized	Locking/Merging	State-based	Model	—	—	—
ModelBus	Centralized	Locking/Merging	State-based	Model/Meta-model	—	—	—
DiCoMEF	Distributed	Merging	Operation-based	Model/Meta-model	Yes	Yes	Yes

In the next chapter, we present a distributed collaborative modeling framework called DiCoMEF [Koshima and Englebert, 2014; Koshima et al., 2011; Koshima and Englebert, 2015b; Koshima et al., 2013]. It supports versioning of both models and meta-models. Besides, modifications are controlled by human agents (not automatic), and the framework can also support hierarchical collaborative modeling.

Chapter 5

Distributed Collaborative Model Editing Framework (DiCoMEF)

In this chapter, we present a distributed collaborative modeling framework called DiCoMEF. The chapter compiles materials published in different peer-reviewed journals, book chapters, conferences, and workshops [Koshima and Englebert, 2015a, 2014; Koshima et al., 2011; Koshima and Englebert, 2015b; Koshima et al., 2013].

5.1 DiCoMEF Architecture

DiCoMEF [Koshima and Englebert, 2014; Koshima et al., 2011; Koshima and Englebert, 2015b; Koshima et al., 2013] is an operation-based and a distributed collaborative modeling framework for EMF/Ecore models and meta-models (see Figure 5.1), where each member of a group has his/her own local copy of a (meta-)model (see Figure 5.2). The main concepts used in DiCoMEF are *group, user, role, role type, model, meta-model, copy model and master model* (see Figure 5.3). A master (meta-)model is the main (meta-)model which has one or more copy (meta-)models that are distributed among editors and observers. A master (meta-)model designates a (meta-)model that is stored on the main-line of a controller site, whereas, copy models are stored on branches (editors and controller sites) and main-lines of editors sites as well as observers sites. The main-line and branch concepts are discussed

later in this section. DiCoMEF uses a universal unique identifier (UUID) to differentiate (meta-)model elements (i.e. classes, attributes, references) uniquely. Two (meta-)model elements are considered as identical only if they have the same UUID.

A group contains one or more users who involve in collaborative modeling and it has one controller (i.e. a user with a controller role). A user has a role, which is typed as a *controller, editor*, or *observer*. A user who has an editor role can read and write his/her local copy (meta-)models stored on the branch, but s/he has only a read access to copy (meta-)models on the main-line. A controller is a special kind of editor who can modify master (meta-)models. An observer role only gives a read access to a local copy (meta-)models. In fact, there are two controller role types which are implemented in DiCoMEF such as a model controller or a meta-model controller. Model (resp. meta-model) controllers are software configuration managers who manage evolution of a master (resp. meta-)model. A controller role type is dynamic meaning that it can be assigned (delegated) to other members of a group as long as there is one unique coordinator per group.

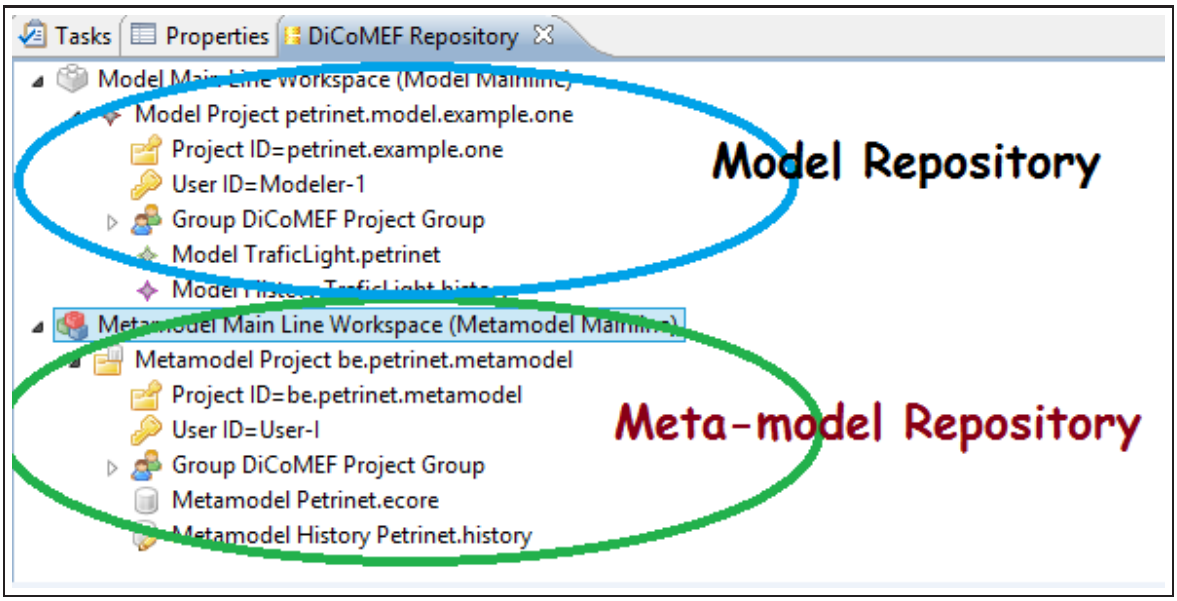

Figure 5.1 **DiCoMEF repository.** DiCoMEF repository manages both models and meta-models

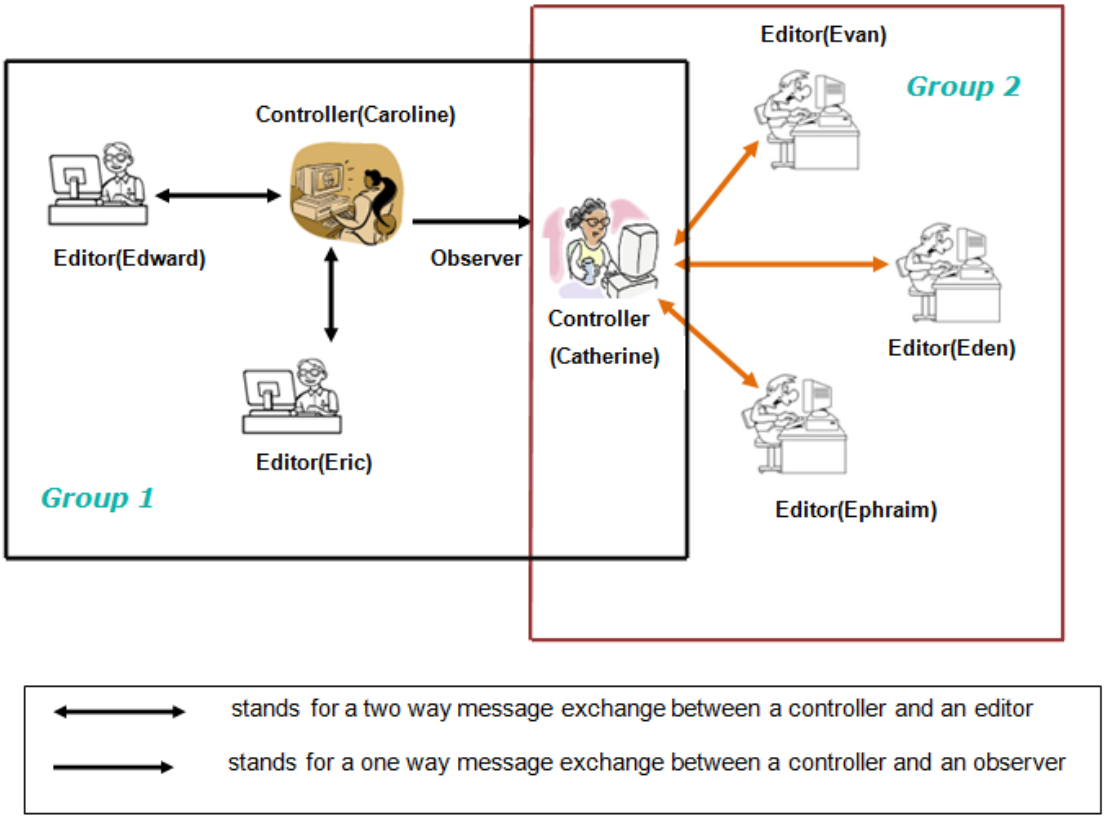

Figure 5.2 Architecture of DiCoMEF

DiCoMEF relies on two concepts such as *main-line* and *branches* in order to store models and meta-models. Besides, it uses these two concepts to facilitate communications among members of a group (see Figure 5.4)[1]. The main-line stores different versions of a copy (meta-)model locally at each editors site. An editor does not have a write access to modify a copy (meta-)model stored on the main-line. Rather s/he first creates a branch from the main-line and modifies the (meta-)model there. In order to communicate local modifications with other members, s/he sends her/his local modifications to a controller as a change request. The controller propagates accepted changes to all members of the group and changes propagated from the controller are applied on the main-line. For example, Figure 5.4 shows an evolution of a copy (meta-)model from version V_0 to version V_1 on the main-line based on changes propagated from a controller. Besides, it indicates a local

[1] Although these terms are also used by SCM programs, our framework does not rely on a central SCM.

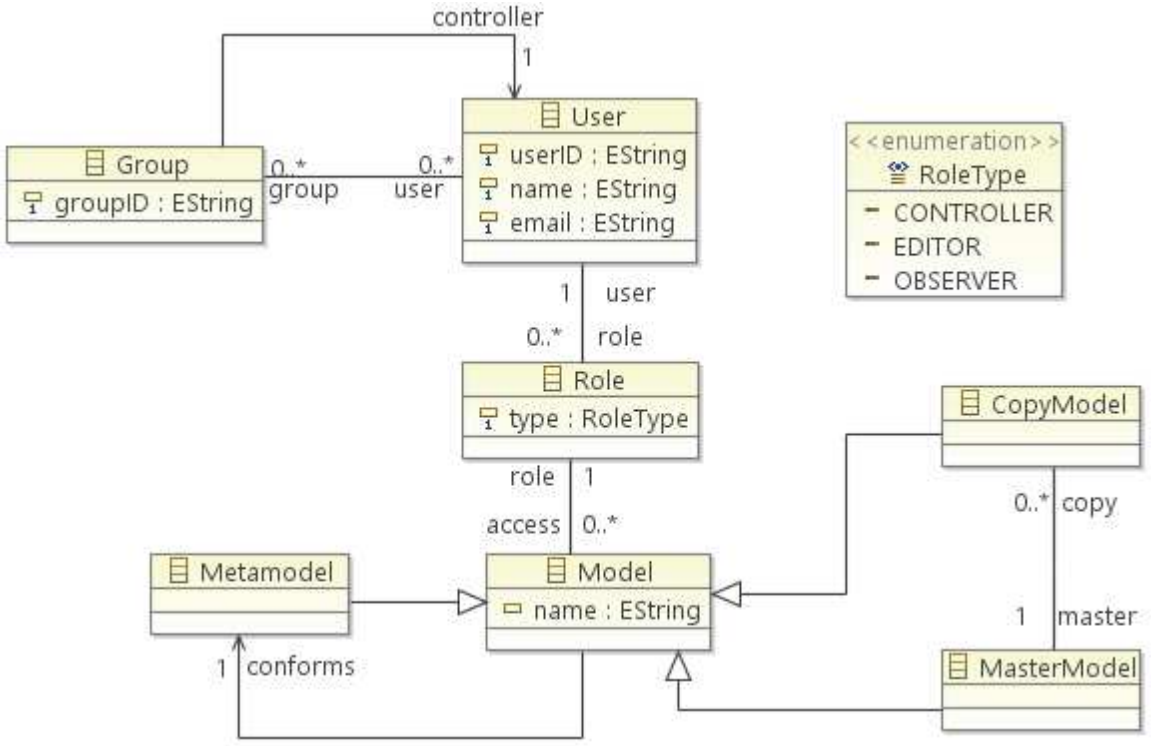

Figure 5.3 DiCoMEF meta-model

modification performed by an editor on the branch that evolves a copy (meta-)model from version V_0 to version $V_{0.1}$; a branch was created before a copy (meta-)model on the main-line evolves from version V_0 to version V_1.

The communication framework of DiCoMEF is organized around the controller that acts as a central hub w.r.t. his/her (meta-)model for which he/she is responsible. This could be a limitation of DiCoMEF, but at the same time it might be considered as its strength as well. Indeed, DiCoMEF provides a technical framework over which different communication strategies can be employed using method engineering techniques (e.g., delegation mechanisms, pooling). For example, a token can be used and whoever has a token is a controller who can modify a (meta-)model and propagates changes.

In DiCoMEF, when members of a group modify (meta-)models locally, elementary change operations (*create, delete and updates*) are stored locally in a local repository. These elementary operations constitute a history that is used to propagate local modifications to the

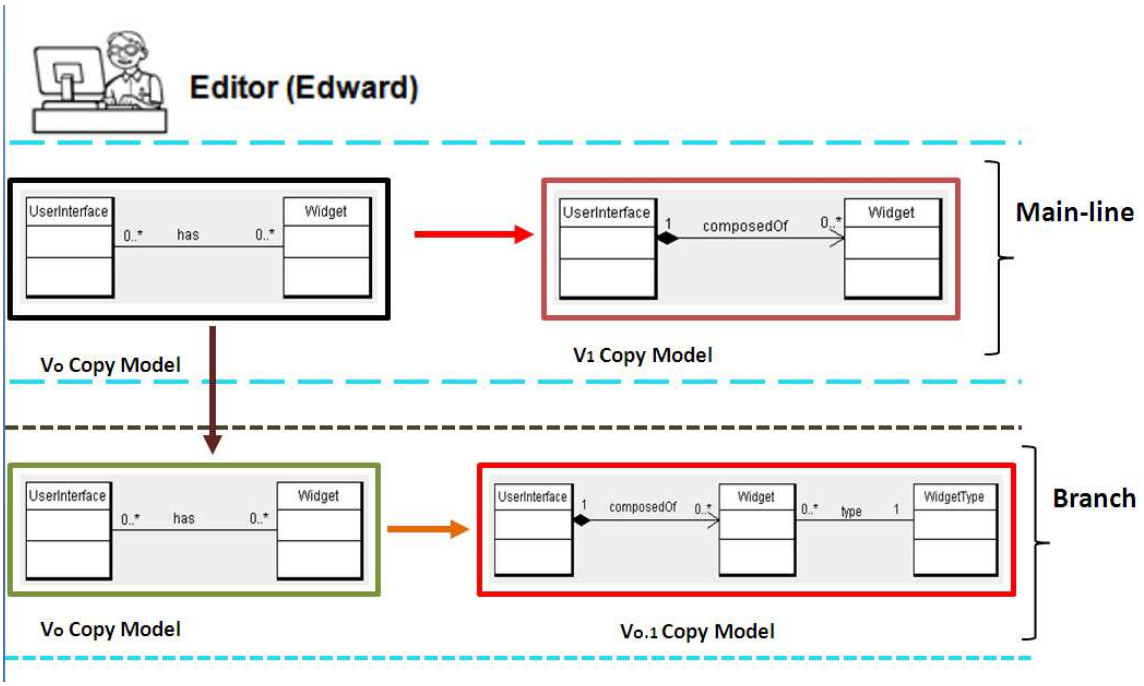

Figure 5.4 Main-line and Branch

controller and secondarily to other members. Histories are defined by a history meta-model. The history meta-model, conflict detection, and reconciliation are discussed later.

A *change request* is a set of local modifications that are performed by an editor and sent to a controller in order to share local modifications with other members (commit changes). A change request could be either accepted, rejected, or modified by a controller before being committed to the main-line (and then shared with other members). A controller works by consulting a rationale of modification or an editor who proposed the change request in case of conflicts. Afterwards, if the change request is accepted, the controller sends a *change propagation* to all members so as to evolve (meta-)models. (Meta-)models on the main-lines evolve automatically, whereas (meta-)models on the branches evolve when users update these (meta-)models based on the change propagation. For example, in Figure 5.4, a copy (meta-)model evolves from version V_0 to version V_1 on the main-line based on changes propagated from a controller. It also shows a branch that is created by an editor to modify a copy (meta-)model locally from version V_0 to version $V_{0.1}$; a branch was created before a copy (meta-)model evolves from version V_0 to version V_1.

We have implemented DiCoMEF as an Eclipse plug-in that captures the history of (meta-)model adaptation when a user edits (meta-)models using the EMF treeview editor [Steinberg et al., 2009] or the GMF editor[2]. The communication framework of DiCoMEF was implemented using Java Message Service (JMS) [Richards et al., 2009] such that users exchange modifications via e-mail. DiCoMEF has a MessageListenerImp that implements an IMessageListner interface and checks whether there is a new email message or not. If it receives a new email message, it downloads the file and updates the DiCoMEF repository automatically. Besides, it displays a pop-up window so as to inform users about the message type (i.e., change request or change propagation message)

5.2 Model Management in DiCoMEF

The DiCoMEF framework supports collaborative editing of both models and meta-models. In the following subsections, we present the formal specification of models and meta-models using Set theory [Jech, 2013]. Besides, model comparison, conflict detection, reconciliation, merging, versioning of (meta-)models are presented as well.

5.2.1 Formalization of Models

Some research work has been done in the past to formally specify an EMF meta-model using graph theory [Taentzer et al., 2012]. In [Monperrus et al., 2009], the authors used set theory to define an abstraction level of MOF [OMG, 2002]. This work used set theory to formalize an EMF Ecore model [Steinberg et al., 2009], because we believe that most people are familiar with the set theory, as a result, it is easy for them to understand and reason about models.

5.2.1.1 Notation

In this work, we will use several ad-hoc notations that are defined in this preliminary section. In a binary Cartesian product, identifying components are underlined: if $R \subseteq \underline{A} \times B$ then

[2] https://www.eclipse.org/gmf-tooling/

$\forall a \in A, \forall b_1, b_2 \in B : (a,b_1),(a,b_2) \in R \implies b_1 = b_2$ and we define $R(a) = b \triangleq (a,b) \in R$. The presence of partial orders can be indicated with the $<$ superscript: $R \subseteq A \times B^{<}$ means that tuples with a common element in first position are ordered $(a,b_1) < (a,b_2) < (a,b_3)$ w.r.t. a and this information can be abbreviated as $R(a) = [b_1, b_2, b_3]$. The position (index) of an element in the list is represented with *pos* function as follows $pos(b_2, R(a)) = 1$ and $[e_1, \ldots, e_2] - i \triangleq [e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_n]$. If R is a binary relation $\subseteq A \times B$, then we denote R^- the inverse relation $\subseteq B \times A$: $(a,b) \in R \Leftrightarrow (b,a) \in R^-$ and we denote $R^\star$ its transitive closure, i.e. $\{(a,b) \mid (a,b) \in R \vee \exists c : (a,c) \in R \wedge (c,b) \in R^\star\}$. 2^S denotes the powerset of a set S and $A \mapsto B$ a mapping function from set A to set B.

5.2.1.2 The Ecore Meta-meta-model

The Eclipse Modeling Framework(EMF) is widely used to build tools and applications. It generates code (i.e. classes for the meta-model and adapter classes for viewing and editing models) based on the structured data model [Steinberg et al., 2009]. A model can be expressed using annotated Java interfaces, XML Schema, or UML modeling tools. EMF provides a facility to generate one form of representation from the other (using the EMF framework). EMF uses Ecore as a meta-meta-model to define different DSL languages and itself. Figure 5.5 shows the UML class diagram of the Ecore meta-model. The associations depicted with blue color are derived associations where as the black lines are non-derived associations.

The root element of an Ecore meta-meta-model is an EPackage. An EPackage contains zero or more sub-packages and EClassifiers (i.e. EClass, EDataType, EEnum). A model class is represented by using an EClass, which is identified by a name and has zero or more attributes and references. A class can have zero or more super types. It can have zero or more operations. Properties (attributes) of a class are modelled using an EAttribute, which has a name and a type. Associations are modelled by EReference(s). An EReference models an end of an association between two classes; it has a name and a type (the EClass at the

[3] http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/doc-files/EcoreRelations.gif

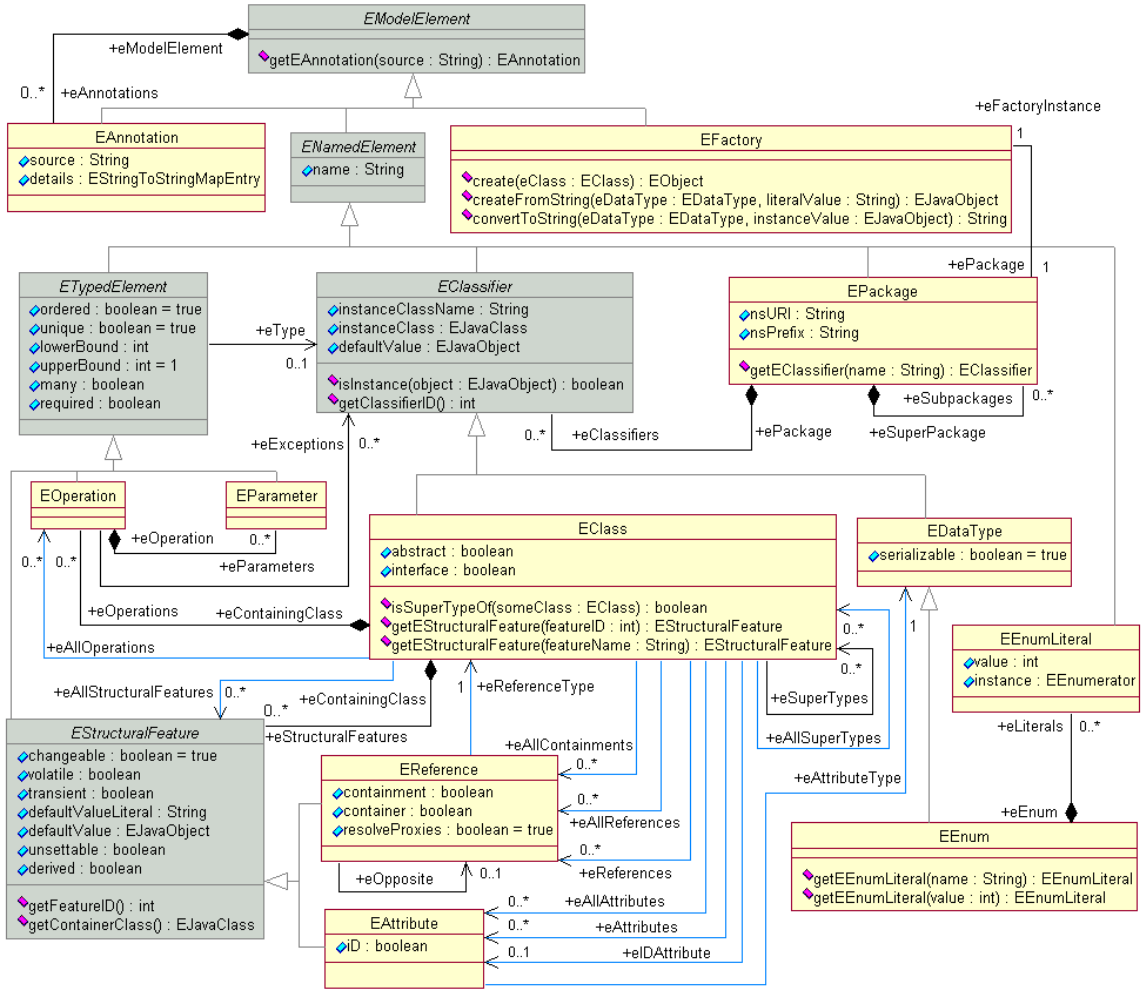

Figure 5.5 EMF/Ecore Meta-model [3]

opposite end of the association). A bi-directional navigable association is modelled using two references that are related to each other by an eOpposite link. Besides, a composition association is represented by setting a containment boolean property of an EReference to `true`. The cardinality of a reference is modeled by setting lowerBound and upperBound values. Like references, an attribute's cardinality could be specified using lowerBound and upperBound features. The Ecore meta-meta-model is attached in the appendix and we also invite interested readers to refer to [Steinberg et al., 2009].

Semantics The semantics of the Ecore meta-meta-model is formally defined by a systematic mapping of its structural elements onto mathematical constructs.

We define a set Σ that encompasses a set of constraints. For each class C in Ecore, we define a set E_C. For each association r between classes A and B in Ecore, we define a set $\rho_r \subseteq E_A \times E_B$. Let's observe that in all the Ecore meta-meta-model diagrams published so far, the relations denote accessor methods and not sets of tuples as specified in the UML standard [OMG, 2011]. For this reason, multiplicities in our mapping may not match the cardinality of the accessor links in the diagrams published so far. The product denoting this association is annotated with the $\underline{\ldots}$ and $^{<}$ symbols depending on its semantics in Ecore: *is the association ordered? is it one-to-many, or many-to-many?*. For each attribute a of type T in class C, a set $\alpha_a \subseteq E_C \times T$ is defined where $T \in E_D$.

Inheritance between classes is mapped to inclusion constraints between the corresponding sets, hence, if A isa B, then the constraint $E_A \subseteq E_B$ is added to Σ. When the superclass is abstract, the inclusion is replaced with the equality operator. We bootstrap first the process by defining some sets:

$$
\begin{aligned}
E_D &= \{EString, EInteger, \ldots\} \\
EString &= \{\text{`'}, \text{`}a\text{'}, \text{`}aa\text{'}, \text{`}ab\text{'}, \ldots\} \\
EInteger &= \{EInteger.min, \ldots, -1, 0, 1, \ldots, EInteger.max\} \cdots
\end{aligned}
$$

E_D elements are data types. In EMF, a data type denotes simple data types in Java, classes, interfaces, and arrays that are not modeled by using with E_C elements [Steinberg et al., 2009]. We define Val as the union of all data type values: $Val = \cup_{T \in E_D} T$.

Ecore classes in the meta-meta-model are mapped to sets: E_C (aka EClass), E_D (aka EDataType), E_P (aka EPackage), E_R (aka EReference), E_A (aka EAttribute), E_E (aka EEnum), E_L (aka EEnumLiteral), E_O (aka EOperation), E_{PA} (aka EParameter), E_{AN} (aka EAnnotation), E_{ne} (aka ENamedElements), E_{me} (aka EModelElement), E_c (aka EClassifier), E_{te} (aka ETypedElement), E_{sf} (aka EStructuralFeature) and E_{OB} (aka EObject). Lower case subscripts denote abstract classes. For the sake of simplicity, EFactory and EStringToStringMapEntry are not considered in this work.

The inheritance relationship between non-abstract class and its subtypes are modeled using set inclusion constraints and the equality constraints (if the supertype is abstract). The base class of all Ecore model elements is an EObject.

$$E_{me} \cup E_{OB} \qquad E_{AN} \cup E_{ne} = E_{me}$$
$$E_{te} \cup E_c \cup E_L \cup E_P = E_{ne} \qquad E_O \cup E_{PA} \cup E_{sf} = E_{te}$$
$$E_C \cup E_D = E_c \qquad E_A \cup E_R = E_{sf}$$
$$E_E \subseteq E_D$$

In this formalization, we don't consider associations that denote derived associations or facilities to access objects neither opposite associations. Each relevant association is translated as a relation between its ends.

$$\rho_{eClassifiers} \subseteq E_P \times E_c{}^{<}$$
$$\rho_{eSubPackages} \subseteq E_P \times E_P{}^{<}$$
$$\rho_{eSuperTypes} \subseteq E_C \times E_C{}^{<}$$
$$\rho_{eSF} \subseteq E_C \times E_{sf}{}^{<} \quad (eSF \text{ is a shortcut for } eStructuralFeatures)$$
$$\rho_{eIDAttribute} \subseteq \underline{E_C} \times E_A$$
$$\rho_{eType} \subseteq \underline{E_{te}} \times E_c$$
$$\rho_{eOpposite} \subseteq \underline{E_R} \times E_R$$
$$\rho_{eKeys} \subseteq E_R \times E_A{}^{<}$$
$$\rho_{eOperations} \subseteq E_C \times E_O{}^{<}$$
$$\rho_{eParameters} \subseteq E_O \times [E_{PA}]^{<}$$
$$\rho_{eExceptions} \subseteq E_O \times E_c$$
$$\rho_{eLiterals} \subseteq E_E \times E_L{}^{<}$$
$$\rho_{eAnnotations} \subseteq E_{me} \times E_{AN}{}^{<}$$
$$\rho_{details} \subseteq E_{AN} \times E_M{}^{<}$$

For sake of simplicity, we define $\mathsf{owner}(sf) \triangleq \rho^{-}_{eSF}(sf)$ and $\mathsf{type}(te) \triangleq \rho_{eType}(te)$ as respectively the owner of a structural feature and its type. Σ is completed with all the integrity constraints defined for Ecore such as "EPackage must have unique names" or "the values of the lowerbound attribute must less or equal than the value of the upperbound attribute for a same class", ...

An EMF/Ecore meta-model MM is thus defined as a tuple of sets:

$$(E_C, E_A, \ldots, \rho_{eClassifiers}, \rho_{eSubPackages}, \ldots, \alpha_{name}, \ldots, \Sigma)$$

Example: A simple Petri net meta-model (Figure 5.6 depicts its class diagram) could be defined as:

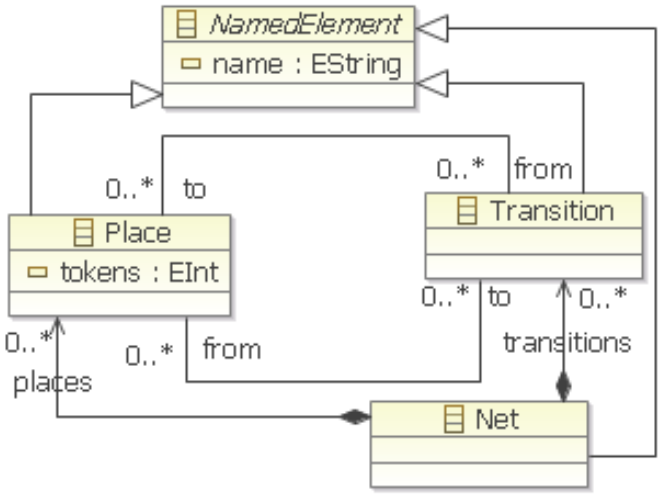

Figure 5.6 Petri net meta-model

$$E_C = \{Pl, Tr, Nt, Ne\}$$
$$E_A = \{Ne.name, Pl.tokens\}$$
$$E_R = \{Pl.to, Pl.from, Tr.from, Tr.to, Nt.places, Nt.transitions\}$$
$$E_P = \{PN\}$$

$$\rho_{eSF}(Ne) = [Ne.name]$$
$$\rho_{eSF}(Pl) = [Pl.tokens, Pl.to, Pl.from]$$
$$\rho_{eSF}(Tr) = [Tr.to, Tr.from]$$
$$\rho_{eSF}(Nt) = [Nt.places, Nt.transitions]$$

$$\alpha_{name} = \{(Pl, \text{`}Place\text{'}), (Tr, \text{`}Transiton\text{'}), (Nt, \text{`}Net\text{'}),$$
$$(Ne, \text{`}NamedElement\text{'}), (Ne.name, \text{`}name\text{'}), (Pl.tokens, \text{`}tokens\text{'}),$$
$$(Pl.to, \text{`}to\text{'}), (Pl.from, \text{`}from\text{'}), (Tr.to, \text{`}to\text{'}), (Tr.from, \text{`}from\text{'}),$$
$$(Nt.places, \text{`}places\text{'}), (Nt.transitions, \text{`}transitions\text{'})\}$$
$$\ldots$$

5.2.1.3 Instantiation

If MM is a meta-model, a model M compliant with MM (noted M/MM) is defined as a tuple $(\llbracket . \rrbracket_C, \llbracket . \rrbracket_A, \llbracket . \rrbracket_R, \Sigma_M)$ where each component is defined hereafter (E_{OB} denotes an infinite set of objects):

- $\llbracket . \rrbracket_C : E_C \mapsto 2^{E_{OB}}$ A class is modeled as a set of objects.
- $\llbracket . \rrbracket_A : E_A \mapsto 2^{(E_{OB} \times T^<)}$ where T is the type of the attribute ($T \in E_D$). An attribute associates an object with values of type T.

- $[\![.]\!]_R : E_R \mapsto 2^{(E_{OB} \times E_{OB}{}^{<})}$ Same for references, but with objects.

We define $\tau(o) : E_{OB} \mapsto E_C$ the function that maps an object o to its class $c \in E_C$ such that $o \in [\![c]\!]_C \wedge \neg \exists d \in E_C : \rho_{eSuperTypes}(d,c) \wedge o \in [\![d]\!]_C$.

Example: A Petri net instance model (depicted as an object diagram in Figure 5.7) can be formalized as:

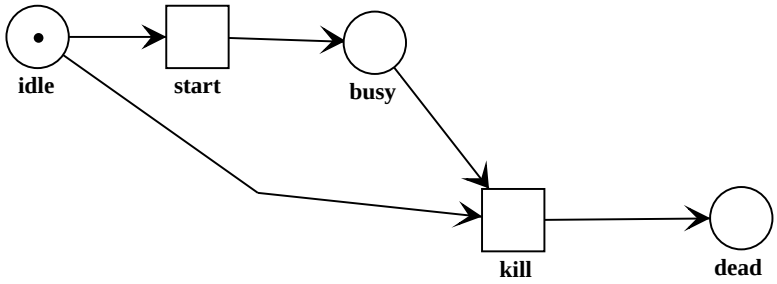

Figure 5.7 Petri net instance model

$$[\![Nt]\!]_C = \{net\}$$

$$[\![Pl]\!]_C = \{idle, busy, dead\}$$

$$[\![Tr]\!]_C = \{start, kill\}$$

$$[\![Pl.name]\!]_A = \{(idle, \text{`}idle\text{'}), (busy, \text{`}busy\text{'}), (dead, \text{`}dead\text{'})\}$$

$$[\![Tr.name]\!]_A = \{(start, \text{`}start\text{'}), (kill, \text{`}kill\text{'})\}$$

$$[\![Pl.tokens]\!]_A = \{(idle, 1), (busy, 0), (dead, 0)\}$$

$$[\![Pl.to]\!]_R(idle) = [start, kill]$$

$$[\![Pl.from]\!]_R = \{(dead, kill), (busy, start)\}$$

$$[\![Tr.to]\!]_R = \{(start, busy), (kill, dead)\}$$

$$[\![Tr.from]\!]_R = \{(start, idle), (kill, idle)\}$$

$$[\![Nt.place]\!]_R(net) = [idle, busy, dead]$$

$$[\![Nt.transition]\!]_R(Nt.transition) = [start, kill]$$

As explained in Section 5.2.1.1, we used the notation $R(a) = [b_1, b_2, b_3]$ to summarize $(a, b_1) < (a, b_2) < (a, b_3)$ for the $[\![]\!]_R$ construct.

5.2.1.4 Reflexivity

Since the base class of all Ecore model elements is EObject, this implies that the Ecore meta-meta-model could specify itself (reflexive definition): the Ecore meta-meta-model can then be modeled as an Ecore meta-model (e.g. MM_{Ecore}). We could expect to observe the same property in our semantics framework of Ecore. We only present a partial definition of MM_{Ecore} for brevity and clarity:

$$
\begin{aligned}
E_C &= \{E_{ne}, E_P, E_c, E_R, \dots\} \\
E_A &= \{E_{ne}.name, E_P.nsURI, E_P.nsPrefix, E_c.instanceClassName, \\
&\qquad E_c.instanceTypeName, \dots\} \\
E_R &= \{E_P.eSubPackages, E_P.eClassifiers, E_c.eStructuralFeatures, \dots\} \\
E_P &= \{Ecore\} \\
\rho_{eSF} &= \{(E_{ne}, E_{ne}.name), (E_P, E_P.nsURI), (E_P, E_P.nsPrefix), (E_P, E_P.eClassifiers), \\
&\qquad (E_P, E_P.eSubPackages), (E_c, E_c.eStructuralFeatures), \\
&\qquad (E_c, E_c.instanceClassName), \dots\}
\end{aligned}
$$

$$
\begin{aligned}
\alpha_{ne.name} = \{&(E_{ne}, \text{`}ENamedElement\text{'}), (E_P, \text{`}EPackage\text{'}), \\
&(E_c, \text{`}EClassifier\text{'}), (E_{ne}.name, \text{`}name\text{'}), \\
&(E_P.nsURI, \text{`}nsURI\text{'}), (E_P.nsPrefix, \text{`}nsPrefix\text{'}), \\
&(E_c.instanceClassName, \text{`}instanceClassName\text{'}), \\
&(E_c.instanceTypeName, \text{`}instanceTypeName\text{'}), \dots\} \\
&\dots
\end{aligned}
$$

$$\rho_{eType}(E_{ne}.name) = EString$$
$$\rho_{eType}(E_c.instanceClassName) = EString$$
$$\rho_{eType}(E_c.instanceTypeName) = EString$$
$$\rho_{eType}(E_c.eStructuralFeatures) = E_{sf}$$
$$\rho_{eType}(E_P.nsPrefix) = EString$$
$$\rho_{eType}(E_P.nsURI) = EString$$
$$\rho_{eType}(E_P.nsURI) = EString$$
$$\rho_{eType}(E_P.eSubPackages) = E_P$$
$$\rho_{eType}(E_P.eClassifiers) = E_c$$
$$\rho_{eSuperTypes}(E_c) = E_{ne}$$
$$\rho_{eSuperTypes}(E_P) = E_{ne}$$
$$\rho_{eClassifiers}(E_P) = E_c$$

This process could be continued with the other constructs of the Ecore meta-model and it shows that we can seamlessly define an Ecore meta-model by using its own self, meaning that our formalization supports the reflexive nature of Ecore. Moreover, the constructs used to define the semantics of a meta-model *MM* at the meta-model level or at the model level when it is reified and consistent. Indeed, for each element of their domain, semantics functions send them to an element that is compliant with the meta-model level: for every attribute a of the Ecore meta-meta-model, we have $[\![a]\!]_A \subseteq E_{OB} \times T^{<}$, and since $E_C \subseteq E_{OB}$, $[\![a]\!]_A$ matches well the type of α_a, i.e., $E_C \times T$. Particularly for $Ne.name$, assertion $[\![Ne.name]\!]_A = \alpha_{Ne.name}$ is verified. For every reference r, we have $[\![r]\!]_R \subseteq E_{OB} \times E_{OB}^{<}$ that matches the definition of ρ_r since every $E_X \subseteq E_{OB}, \forall X$. For every class k, we have $[\![k]\!]_C \subseteq E_{OB}$ that also matches the definition E_k. Moreover $[\![k]\!]_C = k$, e.g. $[\![E_C]\!]_C = E_C$. Hence, isa relationships that have been translated in Σ with set inclusions are still preserved.

5.2.2 Definition of History Meta-model

The operation-based framework uses sequences of elementary (meta-)model change operations to express models [Blanc et al., 2008; Koshima and Englebert, 2014; Koshima et al., 2011, 2013]. It captures atomic edit operations while a user adapts (meta-)models. Change operations are also used to exchange (meta-)models modifications between users [Blanc et al., 2009]. Besides, they are also used to detect conflicts and help the reconciliation process. Hence, it is important to specify the change operations unambiguously and formally.

A history meta-model has been defined to capture the information denoted by the change operations (*create, delete and updates*[4]) of models: a model element can be created (or deleted), a value of a single valued attribute or reference might be set. Besides, a new value can be added (or removed) to a multi-valued attribute or reference. Once this information is captured locally by this meta-model, an history can later be exchanged with other members.

Some work in the past has already used history meta-models. In [Falleri et al., 2014; Fluri et al., 2007; Gall et al., 2009], the authors present different approaches that compare two abstract syntax trees of source code revisions and represents deltas in terms of basic tree edit operations such as insert, delete, move, or update of tree nodes. These approaches do not capture edit operations whenever they occur so that exact time sequences of edit operations cannot be preserved. Monticello [Black et al., 2010], Torch [Gómez et al., 2010], Ring [Gómez et al., 2012], and Hismo [Gîrba et al., 2005] transform a snapshot of a program into a version history rather than recording modifications as they happen. They lack preserving the exact time sequences of modifications that are performed by a user.

Ebraert et al. present a change model in ChEOPS (Change-and-Evolution-Oriented Programming Support) [Ebraert et al., 2007]. This change model is defined based on the FAMIX meta-model [Demeyer et al., 2001], and it captures edit operations that are performed on object-oriented programs as they happen. SpyWare [Robbes and Lanza, 2007, 2008] and Epicea [Dias et al., 2013] also define a history model to tracks changes of a program whenever they occur. D-Praxis [Mougenot et al., 2009] also provides a history

[4]Let's note that read operations are not taken into consideration.

model to captures atomic edit operations (i.e., add, delete, create, and set operations) on EMF/Ecore models.

Edapt [5], previously called COPE [Herrmannsdoerfer, 2009], is a tool based on the EMF/Ecore met-meta-model that captures edit operations of meta-model adaptations whenever they occur. EMFStore [Koegel and Helming, 2010] uses a history meta-model, which captures adaptation of EMF/Ecore based models and meta-models. By the time this research was conducted, the history meta-model of EMFStore was tightly coupled with other components of the EMFStore implementation. As a result, EMFStore cannot be used/installed as an autonomous component for capturing history of meta-model adaptation. In addition, the history meta-model of EMFStore [6] does not record meta-model adaptation well, for instance, moving an EAttribute from one parent EClass to a new parent EClass generates a run time error. As described in Section 5.2.1, the Ecore meta-model is reflexive, hence, constructs used to define the Ecore meta-model can be reused to define an Ecore model and its instances. Hence, we have extended Edapt to capture both the adaptations of model and meta-model as part of the DiCoMEF implementation.

The history meta-model should fulfill the following requirements in order to be efficiently used in distributed collaborative (meta-)model editing framework.

(R1) Self contained: it must not have links (references) to model elements (surrogate technique should be used to reference model elements).

(R2) Universal Unique Identifier (UUI): it should have unique identifiers that identify change operations (create, set, delete, ...). Besides, it should also have UUIs for identifying (meta-)model elements uniquely.

(R3) Composition: it allows users to create composite of changes from other changes or composite changes.

(R4) Meta-model adaptation: it has to capture meta-model adaptation operations.

[5] https://www.eclipse.org/edapt/

[6] Recently, EMFStore has had refactoring to reduce a coupling between parts of the implementation that captures history with rest of implementation. The history meta-model of EMFStore records both models and meta-models adaptations

	R1	R2	R3	R4	R5	R6	R7	R8	R9
Edapt				√		√		√	
EMFStore	√	√	√	√	√				√
DiCoMEF	√	√	√	√	√	√	√	√	√

Table 5.1 Comparison of EMFStore, Edapt, DiCoMEF.

(R5) Model adaptation: it has to capture model adaptation operations.

(R6) Understandability: users intention should be easy to understand. For example, Edapt represents a changing of a parent element of a model element with a Move operation, which is easier to understand than EMFStore, which models the same modification with a composite operation that is composed of remove and add operations.

(R7) Multimedia Annotation: it has to give a user the facility to annotate his rationale with multimedia files.

(R8) Cascade operations: a delete operation should capture cascade operations that are caused by it. For example, when an EClass is deleted from an EPackage, all the references that point to the deleted EClass should be set to null. The delete operation should contain reference operations (copy of them) that set null value (or remove the deleted EClass from a collection). This could help only to roll back conflicting operations during merging process (to reconstruct references that are set to null or deleted due to the deletion of a model element). Roll back is different from undo operations that store operations in the stack. Roll back could be applied when an editor is closed and re-opened again.

(R9) Who performs changes and when: it has to provide facilities to identify an actor who performs changes and when the changes are made.

Based on these requirements, we compare EMFStore, Edapt, and DiCoMEF in Table 5.1. Indeed, Edapt provides a facility to create a composite change from a set of primitive changes, but it does not support creating a composite change from other composite change(s).

For the rest, we define an operation trace ω as the complete documentation of a transformation step $M'/MM = M/MM \gg \omega$ where M'/MM is the new model obtained after application of operation trace ω—*M* denotes a model or a meta-model, that doesn't matter anymore. And a history could then be defined as a sequence $M/MM \gg \omega_1 \gg \omega_2 \gg \omega_3 \gg \omega_{...}$. A trace provides both the information about the precondition and the post-condition of operations.

Figure 5.8 shows the history meta-model of DiCoMEF. We did not show a user model element in Figure 5.8 for the sake of simplicity. The *Create* operation creates a model element in the context of a container element. *Delete* operation deletes an existing model element from its parent element. *Move* operation changes the container of an element. *Add* operation adds a model element (data values) to a list of elements. *Remove* operation removes a model element from the collection. *MoveIndex* operation changes the index of an element in a collection. *Set* operation updates a value of a single-valued attribute or reference. Each operation step has been formally defined as a transition between a state before and a state after (denoted by the $'$ superscript). The formal definitions of these operations are provided below using the formalization defined in section 5.2.1.

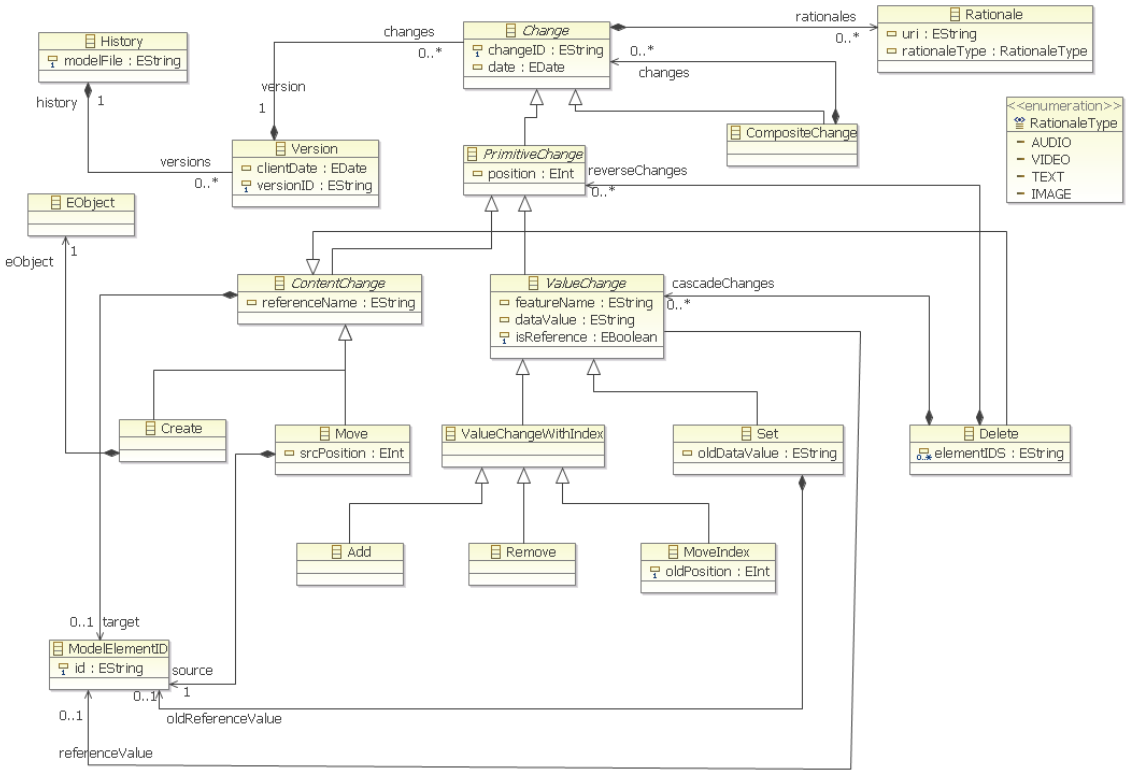

Figure 5.8 History Meta-model

Create Operation: Create operation creates objects in the context of a container.

$$\underline{\mathrm{M}/\mathrm{MM} \gg \mathrm{Create}(e_1, r, e_2, i) \gg \mathrm{M}'/\mathrm{MM}}$$

$$r \in E_R \wedge e_2 \in E'_{OB}$$

$$\wedge\, [\![\mathsf{type}(r)]\!]'_C = [\![\mathsf{type}(r)]\!]_C \cup \{e_2\}$$

$$\wedge \left(\tau(e_1) = \mathsf{owner}(r) \vee (\tau(e_1), \mathsf{owner}(r)) \in \rho_{eSuperTypes}{}^{\star}\right)$$

$$\wedge\, [\![r]\!]'_R = [\![r]\!]_R \cup \{(e_1, e_2)\}$$

$$\wedge\, pos(e_2, [\![r]\!]'_R) = i \wedge [\![r]\!]'_R - i = [\![r]\!]_R$$

$$\wedge\, [\![E_R.containment]\!]_A(r) = \mathbf{true}^{\dagger}$$

† Since $E_R \in E_C$ (see 5.2.1.4) and $type(r) = E_R$ and $E_R.containement \in E_A$ and $\mathsf{owner}(E_R.containment) = E_R$, expression $[\![E_R.containment]\!]_A(r)$ denotes if the reference r must be

considered as a containment or not. EMF attaches importance to the organization of the information in a strict containment relationship and many operations provided by the Ecore API depend on this hierarchy. For sake of simplicity, we define $\kappa(r) \triangleq [\![E_R.containment]\!]_A(r)$.

Example: `create(`$net, Nt.place, start, 1$`)`

Delete Operation : Delete operation deletes an existing model element along with its contents (child elements) from its parent element.

$$\underline{\mathrm{M}/\mathrm{MM} \gg \mathrm{Delete}(e_1, r, e_2) \gg \mathrm{M}'/\mathrm{MM}}$$

$$r \in E_R \wedge e_2 \notin E_{OB} \wedge \kappa(r) = \mathbf{true}$$

$$\wedge\, [\![\mathsf{type}(r)]\!]'_C = [\![\mathsf{type}(r)]\!]_C \backslash \{e_2\}$$

$$\wedge \left(\tau(e_1) = \mathsf{owner}(r) \vee (\tau(e_1), \mathsf{owner}(r)) \in \rho_{eSuperTypes}{}^{\star}\right)$$

$$\wedge\, [\![r]\!]'_R = [\![r]\!]_R \backslash \{(e_1, e_2)\}$$

Example: `delete(`$net, Nt.place, start$`)`

Add Operation : Add operation adds a value to a multi-valued attribute or reference of an an existing model element.

$$\underline{\mathrm{M}/\mathrm{MM} \gg \mathrm{Add}(e, a, \nu, i) \gg \mathrm{M}'/\mathrm{MM}\ \textit{(for an EAttribute)}}$$

$$a \in E_A \wedge \nu \in Val$$

$$\wedge\, [\![\mathsf{type}(a)]\!]'_D = [\![\mathsf{type}(a)]\!]_D \cup \{\nu\}$$

$$\wedge \left(\tau(e) = \mathsf{owner}(a) \vee (\tau(e), \mathsf{owner}(a)) \in \rho_{eSuperTypes}{}^{\star}\right)$$

$$\wedge\, [\![a]\!]'_A = [\![a]\!]_A \cup \{(e, \nu)\}$$

$$\wedge\, pos(\nu, [\![a]\!]'_A) = i \wedge [\![a]\!]'_A - i = [\![a]\!]_A$$

$$\underline{\mathrm{M}/\mathrm{MM} \gg \mathrm{Add}(e_1, r, e_2, i) \gg \mathrm{M}'/\mathrm{MM} \textit{ (for an EReference)}}$$

$$r \in E_R \wedge e_2 \in E_C$$

$$\wedge \, [\![\mathsf{type}(r)]\!]'_C = [\![\mathsf{type}(r)]\!]_C \cup \{e_2\}$$

$$\wedge \left(\tau(e_1) = \mathsf{owner}(r) \vee (\tau(e_1), \mathsf{owner}(r)) \in \rho_{eSuperTypes}{}^{\star}\right)$$

$$\wedge \, [\![r]\!]'_R = [\![r]\!]_R \cup \{(e_1, e_2)\}$$

$$\wedge \, pos(e_2, [\![r]\!]'_R) = i \wedge [\![r]\!]'_R - i = [\![r]\!]_R$$

Remove Operation : Remove operation removes a value from a multi-valued attribute or reference of an an existing model element.

$$\underline{\mathrm{M}/\mathrm{MM} \gg \mathrm{Remove}(e, a, \nu) \gg \mathrm{M}'/\mathrm{MM} \textit{ (for an EAttribute)}}$$

$$a \in E_A \wedge \nu \in Val$$

$$\wedge \, [\![\mathsf{type}(a)]\!]'_D = [\![\mathsf{type}(a)]\!]_D \backslash \{\nu\}$$

$$\wedge \left(\tau(e) = \mathsf{owner}(a) \vee (\tau(e), \mathsf{owner}(a)) \in \rho_{eSuperTypes}{}^{\star}\right)$$

$$\wedge \, [\![a]\!]'_A = [\![a]\!]_A \backslash \{(e, \nu)\}$$

$$\underline{\mathrm{M}/\mathrm{MM} \gg \mathrm{Remove}(e_1, r, e_2) \gg \mathrm{M}'/\mathrm{MM} \textit{ (for an EReference)}}$$

$$r \in E_R \wedge e_2 \in E_C$$

$$\wedge \, [\![\mathsf{type}(r)]\!]'_C = [\![\mathsf{type}(r)]\!]_C \backslash \{e_2\}$$

$$\wedge \left(\tau(e_1) = \mathsf{owner}(r) \vee (\tau(e_1), \mathsf{owner}(r)) \in \rho_{eSuperTypes}{}^{\star}\right)$$

$$\wedge \, [\![r]\!]'_R = [\![r]\!]_R \backslash \{(e_1, e_2)\}$$

Set Operation : Set operation updates a single-valued attribute or reference of an an existing model element.

$$\underline{\mathrm{M}/\mathrm{MM} \gg \mathrm{Set}(e,a,\nu_n,\nu_o) \gg \mathrm{M}'/\mathrm{MM}\ \textit{(for an EAttribute)}}$$

$$a \in E_A \wedge \nu_n, \nu_o \in Val$$
$$\wedge\, [\![\mathsf{type}(a)]\!]'_D = [\![\mathsf{type}(a)]\!]_D \cup \{\nu_n\}$$
$$\wedge\, [\![\mathsf{type}(a)]\!]'_D = [\![\mathsf{type}(a)]\!]_D \backslash \{\nu_o\}$$
$$\wedge\, \big(\tau(e) = \mathsf{owner}(a) \vee (\tau(e), \mathsf{owner}(a)) \in \rho_{eSuperTypes}{}^{\star}\big)$$
$$\wedge\, [\![a]\!]'_A = [\![a]\!]_A \cup \{(e,\nu_n)\}$$
$$\wedge\, [\![a]\!]'_A = [\![a]\!]_A \backslash \{(e,\nu_o)\}$$

$$\underline{\mathrm{M}/\mathrm{MM} \gg \mathrm{Set}(e_1,r,e_2,e_3) \gg \mathrm{M}'/\mathrm{MM}\ \textit{(for an EReference)}}$$

$$r \in E_R \wedge e_2, e_3 \in E_C$$
$$\wedge\, [\![\mathsf{type}(r)]\!]'_C = [\![\mathsf{type}(r)]\!]_C \cup \{e_2\}$$
$$\wedge\, [\![\mathsf{type}(r)]\!]'_C = [\![\mathsf{type}(r)]\!]_C \backslash \{e_3\}$$
$$\wedge\, \big(\tau(e_1) = \mathsf{owner}(r) \vee (\tau(r), \mathsf{owner}(r)) \in \rho_{eSuperTypes}{}^{\star}\big)$$
$$\wedge\, [\![r]\!]'_R = [\![r]\!]_R \cup \{(e_1,e_2\}$$
$$\wedge\, [\![r]\!]'_R = [\![r]\!]_R \backslash \{(e_1,e_3)\}$$

Move Operation : Move operation changes a parent of a model element this means that it moves an element from one containment reference to another containment relation. Indeed, a move operation removes an EObject with its content from its parent and adds it and its

contents to another parent.

$$\underline{\mathrm{M}/\mathrm{MM} \gg \mathrm{Move}(e_1, r_2, r_3, e_2, e_3, i) \gg \mathrm{M}'/\mathrm{MM}}$$
$$r_2, r_3 \in E_R \wedge \kappa(r_2) = \textbf{true} \wedge \kappa(r_3) = \textbf{true} \wedge e_2, e_3 \in E_{OB}$$
$$\wedge [\![\mathsf{type}(r_3)]\!]'_C = [\![\mathsf{type}(r_3)]\!]_C \cup \{e_1\}$$
$$\wedge [\![\mathsf{type}(r_2)]\!]'_C = [\![\mathsf{type}(r_2)]\!]_C \backslash \{e_1\}$$
$$\wedge \left(\tau(e_2) = \mathsf{owner}(r_2) \vee (\tau(r_2), \mathsf{owner}(r_2)) \in \rho_{eSuperTypes}{}^{\star}\right)$$
$$\wedge \left(\tau(e_3) = \mathsf{owner}(r_3) \vee (\tau(r_3), \mathsf{owner}(r_3)) \in \rho_{eSuperTypes}{}^{\star}\right)$$
$$\wedge [\![r_3]\!]'_R = [\![r_3]\!]_R \cup \{(e_3, e_1\}$$
$$\wedge [\![r_2]\!]'_R = [\![r_2]\!]_R \backslash \{(e_2, e_1)\}$$
$$\wedge pos(e_1, [\![r_3]\!]'_R) = i \wedge [\![r_3]\!]'_R - i = [\![r_3]\!]_R$$

MoveIndex Operation : MoveIndex operation moves an element of multi-valued attribute or references from old position to new position within the list (change position).

$$\underline{\mathrm{M}/\mathrm{MM} \gg \mathrm{MoveIndex}(e, a, \nu, i, j) \gg \mathrm{M}'/\mathrm{MM}\ \textit{(for an EAttribute)}}$$
$$a \in E_A \wedge \nu \in Val$$
$$\wedge \left(\tau(e) = \mathsf{owner}(a) \vee (\tau(e), \mathsf{owner}(a)) \in \rho_{eSuperTypes}{}^{\star}\right)$$
$$\wedge pos(\nu, [\![a]\!]'_A) = i \wedge [\![a]\!]'_A - i = [\![a]\!]_A = j$$

$$\underline{\mathrm{M}/\mathrm{MM} \gg \mathrm{MoveIndex}(e_1, r, e_2, i, j) \gg \mathrm{M}'/\mathrm{MM}\ \textit{(for an EReference)}}$$
$$r \in E_R \wedge e_2 \in E_{OB}$$
$$\wedge \left(\tau(e_1) = \mathsf{owner}(r) \vee (\tau(e_1), \mathsf{owner}(r)) \in \rho_{eSuperTypes}{}^{\star}\right)$$
$$\wedge pos(e_2, [\![r]\!]'_R) = i \wedge [\![r]\!]'_R - i = [\![r]\!]_R = j$$

Figure 5.9 demonstrates the history of model adaptation using DiCoMEF, whereas Figure 5.10 illustrates meta-model evolution and the history of edit operations.

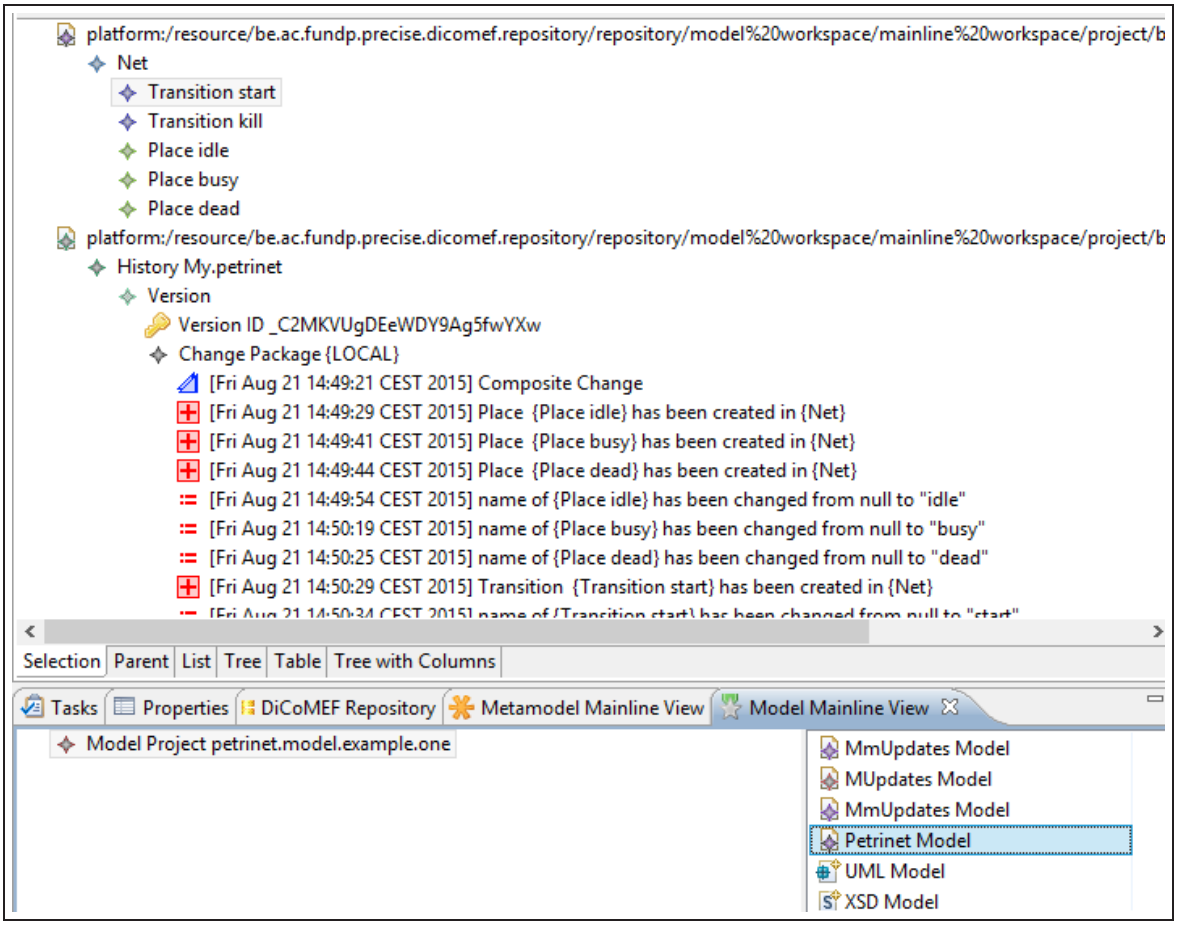

Figure 5.9 History of model adaptation in DiCoMEF

5.2.3 Change Management

DiCoMEF modification management system organizes changes into different groups such as *"local modification"*, *"requested modification"*, *"propagated modification"*, and *"committed modifications"*. Local modifications are changes performed by editors locally, but they are not requested for a commit. These modifications are stored in a change package with a **local** label. Requested modifications are local modifications that have been sent to a controller for a commit, and are stored in a change package with a **requested** label. The changes that are reviewed and propagated by the controller to all members of a group are

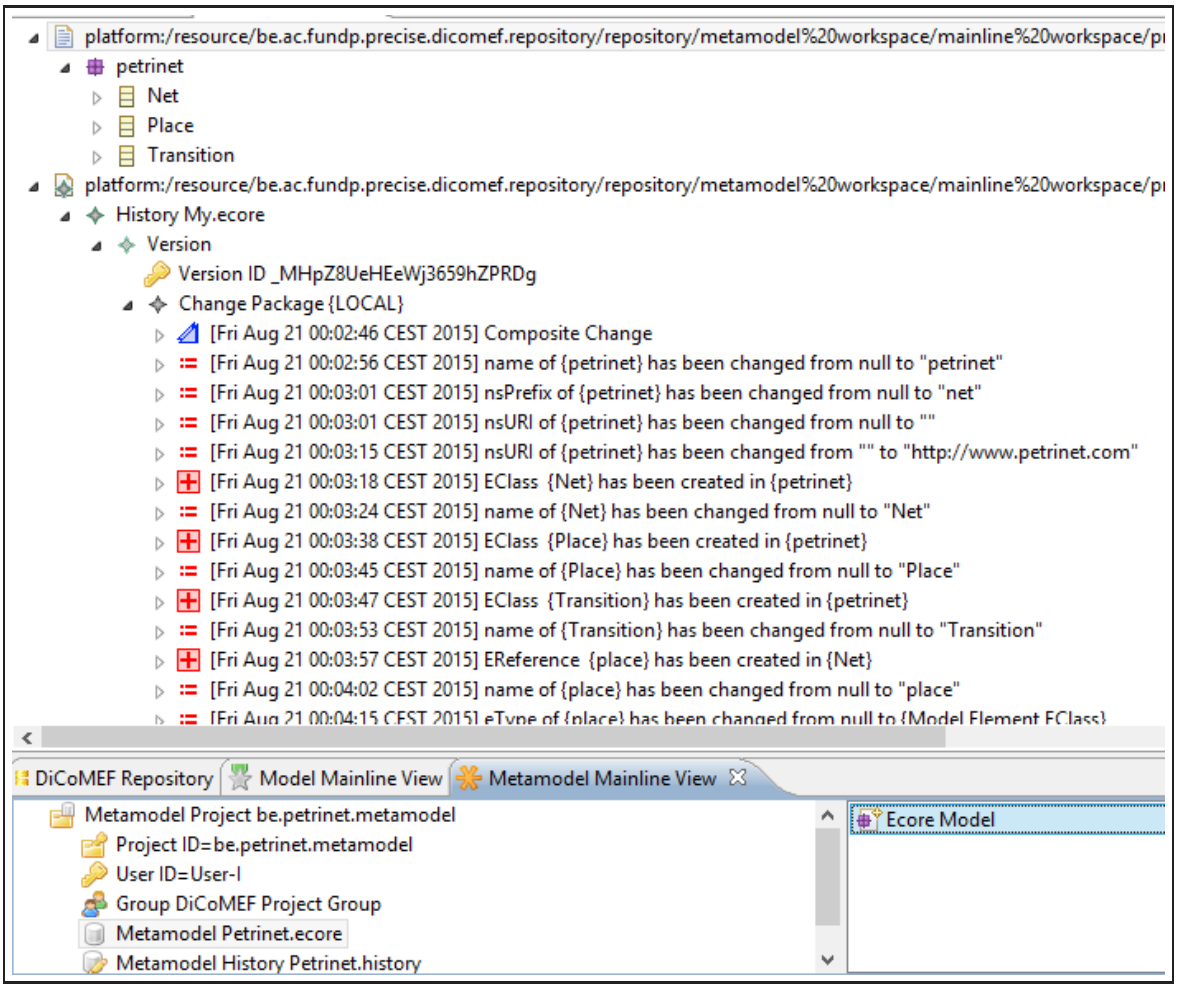

Figure 5.10 History of meta-model adaptation in DiCoMEF

stored in change package, which is marked by **propagated**. Committed modifications are those changes that are applied on the main-line and the branch, which are stored in a package with a **committed** label. The controller can aggregate two or more requested modifications into one propagated modification. The propagated modification contains UUIDs of those aggregated requested modifications. When an editor commits a propagated modification, the label of a requested modification package whose identifier is contained by the propagated change package is converted to **committed**.

The easiest way to manage changes during merging is to temporarily rollback local modifications and applies propagated modifications, and to re-apply local modifications again. Local modifications are always the most recent modifications, so that an editor can easily identify these changes. The downside of this approach is that it requires to rollback local modifications for every minor changes propagated from the controller. In order to address this problem, the DiCoMEF modification management system classifies modifications into different groups and stores them in change packages labeled with **local**, **requested**, **propagated**, and **committed**. Whenever an editor tries to generate a requested modifications, the DiCoMEF modification management system canonizes all changes stored in change packages starting from the first change package with a **local** label to the most recent change package. If the canonization result is empty, then there is no change request to generate. If not, the canonized changes that are a member of change packages with **local** labels are used to generate change requests. Likewise, the controller generates propagated modifications from the canonized changes that are a member of change packages with **local** labels.

5.2.4 Model Comparison

DiCoMEF uses a universal unique identifier (UUID) to uniquely identify each model and meta-model element. Two (meta-)model elements are considered as equal if and only if they do have the same UUID. It also uses a UUID to identify edit operations and versions. As discussed in Chapter 4 Section 4.1.1, model comparison techniques are classified as state-based model comparison and operation-based model comparison [Altmanninger et al., 2009; Lippe and van Oosterom, 1992a; Mens, 2002]. DiCoMEF employs operation-based

model comparison approach, where sequences of edit operations that adapt (meta-)models are considered as deltas between two versions of (meta-)models.

5.2.5 Conflict Detection

In collaborative modeling, (meta-)models are concurrently edited by different members of a group. Later, these concurrently edited (meta-)models need to be integrated (merged), but most of the time they might not seamlessly work together as a result of inconsistent modifications (conflict). Due to conflicts, a user might not be able to execute all operations propagated from a controller on his local copy. For instance, this could be the case when the user deletes a model element and the controller propagates a change which modifies the same model element (i.e., a delete operation and a set operation on the same model element). Hence, the deleted element needs to be re-created (with the same UUID). DiCoMEF only rolls back the delete operation (re-create) and the dependent operations (a copy of these operations is stored in the delete operation). For instance, suppose that an editor and a controller work on the same model instance described in section 5.7. The Editor deletes the *start* model element (instance of a *Transition* class) and sets the name of the *busy* model element (instance of a *Place* class) to *"active"*. A controller sends a change propagation to rename a *start* model element to *"begin"*. In order to apply the change propagation, a deleted element (*start*) must be re-created. During rolling back, DiCoMEF firstly creates the *start* model element and afterwards it re-establishes the relationships between the *start* and the *busy*, and the *idle* and the *start* model elements (see Figure 5.12). Rolling back only the delete operation is important, especially, if there are many changes performed by a user after deleting a model element. Rolling back all changes to re-create a deleted model element could be time consuming. Finally, it renames *start* to *"begin"*.

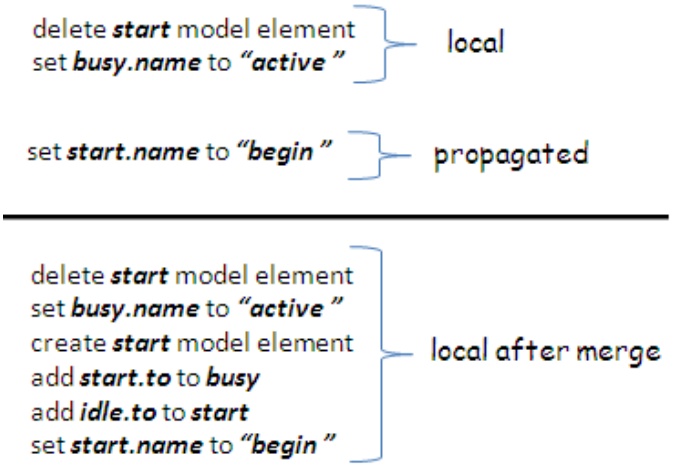

Figure 5.11 Change propagation and local operations

We employ the *Conflict* (Table 5.2 and 5.3) and *Require* (Table 5.4) relations to detect structural conflicts between H^C and H^L [Koegel et al., 2009b]. Where H^C is a history of changes propagated from a controller and H^L is a history of local changes performed by an editor. An operation ω_i^C is conflicting with another operation ω_j^L if the order of serialization of these operations affects the final state of the (meta-)model (e.g., two *set* operations that rename an EObject differently) [Koegel et al., 2009b]. Besides, the execution of one of the operations could invalidate a precondition of another one. The preconditions of operations are implicitly specified in the formalization of operations (see section 5.2.2). For instance, a target model element must exist in a model ($e_3 \in \mathrm{M}/\mathrm{MM}$) so as to create a non-root model element (i.e., $\mathrm{M}/\mathrm{MM} \gg \mathrm{Create}(e_3, r_1, e_1, i) \gg \mathrm{M}'/\mathrm{MM}$) . Hence, $e_3 \in \mathrm{M}/\mathrm{MM}$ is a precondition and must be satisfied so as to execute $\mathrm{Create}(e_3, r_1, e_1, i)$ operation. In fact, the semantics of conflicts sometimes depend on whether multi-valued features need to be ordered or not. In ordered multi-valued features r_2, two create operations such as $Create(e_1, r_2, e_2, i)$ and $Create(e_1, r_2, e_3, j)$ could be conflicting. The order of execution of these operations could leave an element in different positions in a list ($pos(e_2, [\![r_2]\!]''_R) \neq pos(e_2, [\![r_2]\!]''_R)$. On the other hand, these two operations are not conflicting in non-ordered multi-valued features.

Conflict relation calculates a set of conflicting operations. The level of severity of conflicts could vary based on the type of conflicts meaning that some conflicts need user inter-

actions whereas other conflicts could be solved automatically [Koegel et al., 2009b]. *Hard conflicts* require a user interaction. For instance, a delete-update conflict is a hard conflict. *Soft conflicts* can be resolved automatically by employing some conflict reconciliation strategies. For example, conflicting positions in ordered multi-valued features are soft conflicts. Tables 5.2 and 5.3 show the conflicting relation where ***h*** (resp ***s***) denotes a hard (resp soft) conflict.

The $\succ_H$ operator shows that one operation is succeeded (directly or indirectly) by another operation in history H. For instance, Create$(e_1, r_1, e_2, i) \succ_H$ Delete(e_3, r_2, e_1) and Delete$(e_3, r_2, e_1) \succ_H$ Create(e_1, r_1, e_2, i) give different result. In the first case, both operations execute and the model element e_1 along with its child are deleted from the model. But, in the second case, the create operation cannot execute because the delete operation deletes the target model element e_1 and makes the precondition of the create operation invalid. Therefore, $Create(e_1, r_1, e_2, i)$ and $Delete(e_3, r_2, e_1)$ operations are conflicting operations: $Create(e_1, r_1, e_2, i) \otimes Delete(e_3, r_2, e_1)$. $\otimes$ symbol is used to show conflicting relations: $Create(e_1, r_2, e_2, i) \otimes Delete(e_3, r_1, e_1)$. A conflicting relation is symmetric: $Create(e_1, r_2, e_2, i) \otimes Delete(e_3, r_1, e_1) \Rightarrow Delete(e_3, r_1, e_1) \otimes Create\ (e_1, r_2, e_2, i)$. A *Delete* operation conflicts with a *Move* operation if it deletes a same model element, its source model element, or a new target model element. A delete-update conflict is a hard conflict that needs user intervention. In addition, a *Delete* operation is conflicting with *ValueChange* operations if it deletes either a same model element or a reference value of *ValueChange* operations.

A *Move* operation is conflicting with another *Move* operation if they move a same model element to different target elements (hard conflict). Besides, they could also be in conflict with one another if they moved different model elements to a same multi-valued features of a target model element (soft conflict). A *Move* operation might also raise soft conflicts with *Create* operations and *ValueChange* operations such as *Add, Remove, and MoveIndex*. The index position of elements in a list could be different depending on their serialization of operations. A composite operation ω_c is in conflict with another operation ω_1 if at least

one of its member operation is conflicting with ω_1. The semantics of conflicts for other operations could easily be expressed as well.

A *Set* operation conflicts with another *Set* operation that assigns different value for a same feature of model element. An *Add* and a *Remove* operations are also conflicting with each other if the *Add* operation adds a model element and the *Remove* operation removes the same model element. Two *Add* or two *Remove* operations could also raise conflicts when they are applied on ordered multi-valued features. *MoveIndex* operation raises disagreement with another *MoveIndex* operation if an application of these operations give different position to a same model element. *MoveIndex* operation also conflicts with *Move* operation, if a *Move* operation changes a parent of an element that is concerned by *MoveIndex* operation. A composite operation ω_c is in conflict with another operation ω_1 if at least one of its member operation is conflicting with ω_1.

An operation, ω_j, requires another operation, ω_i, if and only if ω_i must be executed before ω_j so that the precondition of ω_j is entailed by the post-condition of ω_i. The *require* binary relation is transitive, but it is not symmetric. As it was discussed above in section 5.2.2(see Figure 5.8), *ContentChange* operations (i.e., *Create*, *Delete*, and *Move*) require *Create* operations that created their target model elements (containers). For instance, a given create operation ($\mathrm{M}/\mathrm{MM} \gg \mathrm{Create}(e_3, r_1, e_1, i) \gg \mathrm{M}'/\mathrm{MM}$) needs a creation of its target model element ($e_3 \in \mathrm{M}/\mathrm{MM}$) so as to execute successfully. For example, an EClass *Place* should be created before an EAttribute *tokens* (see Figure 5.6). After successful execution of the create operation, a new model element is added to the model, $e_1 \in \mathrm{M}'/\mathrm{MM}$ (i.e, an EAttribute *tokens* is added to the model). The require relationship between *Create* operations has one exception: a creation of root model element does not have any target element. *ContentChange* operations also require creation of model elements (EObjects) on which the operations are performed. *Move* operation depends on a creation of a source model element, EObject. The require relationships of *ContentChange* operations are represented as follows:

$$\begin{aligned}
&\text{Create}(e_1, r_2, e_2, j) \in H \Rightarrow \\
&\qquad \exists e_3, r_1, i : \text{Create}(e_3, r_1, e_1, i) \succ_H \text{Create}(e_1, r_2, e_2, j) \\
&\text{Delete}(e_1, r_2, e_2) \in H \Rightarrow \\
&\qquad \exists i : \text{Create}(e_1, r_2, e_2, i) \succ_H \text{Delete}(e_1, r_2, e_2) \\
&\text{Move}(e_2, r_2, r_4, e_1, e_3, i) \in H \Rightarrow \\
&\qquad\quad \exists r_1, l : \text{Create}(e_3, r_1, e_1, l) \succ_H \text{Move}(e_2, r_2, r_4, e_1, e_3, i) \\
&\qquad \wedge\ \exists j : \text{Create}(e_1, r_2, e_2, j) \succ_H \text{Move}(e_2, r_2, r_4, e_1, e_3, i) \\
&\qquad \wedge\ \exists e_4, r_3, m : \text{Create}(e_4, r_3, e_3, m) \succ_H \text{Move}(e_2, r_2, r_4, e_1, e_3, i)
\end{aligned}$$

ValueChange operations such as *Add*, *Remove*, *Set*, and *MoveIndex* could be reference value change operations or data value change operations. A *ValueChange* operation requires a creation of an element (EObject) on which the operation is performed. Besides, a reference *ValueChange* operation requires a creation of a reference value, EObject. For instance, a given *Set* operation (i.e., $\text{M}/\text{MM} \gg \text{Set}(e_1, r_4, e_2, e_3) \gg \text{M}'/\text{MM}$) requires create operations that add e_1 and e_2 to the model, $e_1, e_2 \in \text{M}/\text{MM}$. In fact, a *Set* could assign a *NULL* value to a single-feature of a model element. Remove operation needs an addition of a model element to multi-valued feature. Hence, these relationships are depicted as:

$$\begin{aligned}
&\text{Set}(e_1, r_4, e_2, e_3) \in H \Rightarrow \\
&\qquad\quad \exists r_1, i : \text{Create}(e_3, r_1, e_1, i) \succ_H \text{Set}(e_1, r_4, e_2, e_3) \\
&\qquad \wedge\ \exists e_4, r_2, j : \text{Create}(e_4, r_2, e_2, j) \succ_H \text{Set}(e_1, r_4, e_2, e_3) \\
&\text{Add}(e_1, r_4, e_2, i) \in H \Rightarrow \\
&\qquad\quad \exists e_3, r_1, j : \text{Create}(e_3, r_1, e_1, j) \succ_H \text{Add}(e_1, r_4, e_2, i) \\
&\qquad \wedge\ \exists e_4, r_2, l : \text{Create}(e_4, r_2, e_2, l) \succ_H \text{Add}(e_1, r_4, e_2, i)
\end{aligned}$$

$$\begin{aligned}
&\text{Remove}(e_1, r_4, e_2) \in H \Rightarrow \\
&\qquad \exists e_3, r_1, j : \text{Create}(e_3, r_1, e_1, j) \succ_H \text{Remove}(e_1, r_4, e_2) \\
&\quad \wedge\ \exists e_4, r_2, l : \text{Create}(e_4, r_2, e_2, l) \succ_H \text{Remove}(e_1, r_4, e_2) \\
&\quad \wedge\ \exists i : \text{Add}(e_1, r_4, e_2, i) \succ_H \text{Remove}(e_1, r_4, e_2) \\
&\text{MoveIndex}(e_1, r_4, e_2, l, m) \in H \Rightarrow \\
&\qquad \exists e_3, r_1, i : \text{Create}(e_3, r_1, e_1, i) \succ_H \text{MoveIndex}(e_1, r_4, e_2, l, m) \\
&\quad \wedge\ \exists r_2 : \text{Create}(e_1, r_2, e_2, l) \succ_H \text{MoveIndex}(e_1, r_4, e_2, l, m)
\end{aligned}$$

A Composite operation requires all operations that should be performed firstly before executing its contents. The require relation could be extended with the following pattern:

$$(\text{create}(e_1, r_1, e_2, i), \text{create}(e_3, r_2, e_1, j))$$

This relation is resumed in Table 5.4. There is a correlation between the *require* and *conflict* relationships: if operation ω_1 requires ω_2 and ω_2 conflicts with ω_3, then ω_1 also conflicts with ω_3.

Meta-model adaptation could also lead to a precondition violation, for instance, a reference feature of a meta-model element could be deleted in a new version of meta-model that results in violation of precondition for Create, Set, Add, ... operations. In this case, both the instance model and its respective history model needs to co-evolve with meta-model. But model co-evolution and history migration are not in the scope this work. It will be incorporated on the top of DiCoMEF as part of future work.

When hard conflicts occur, the DiCoMEF framework shows the conflicting operations to the user with all the required information and the rationale about them (see Figure 5.12). A user can select some conflicting changes so as to apply them locally and reject the rest

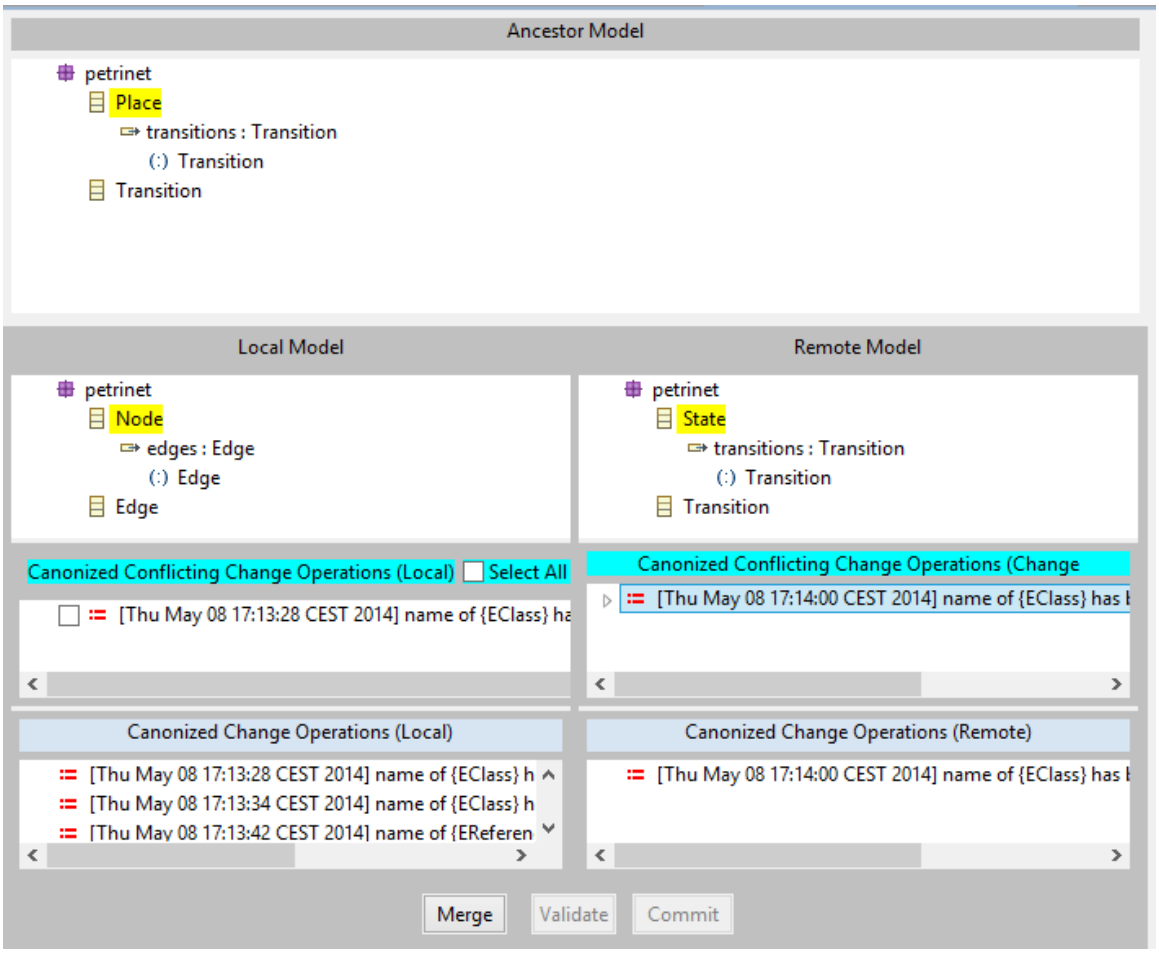

Figure 5.12 DiCoMEF merge tool

of conflicting changes (not selected conflicting changes). In the next phase, DiCoMEF uses EMF validation framework [Steinberg et al., 2009] to detect semantic conflicts. In fact, it checks whether the merging of two histories results in any OCL constraints violation. DiCoMEF provides a tree view editor with different icons and information to facilitate resolving of semantic conflicts. Once the user has solved conflicts, the merging process can be continued.

Table 5.2 Conflicting relation (ordered-multivalued).

	Create	Delete	Move	MoveIndex	Add	Remove	Set
Create	s	h	s	s	s	s	
Delete	h		h	h	h		h
Move	s	h	h	h	s	s	
MoveIndex	s	h	h	s	s	s	
Add	s	h	s	s	s	s	
Remove	s		s	s	s		
Set		h					s

Table 5.3 Conflicting relation (unordered-multivalued).

	Create	Delete	Move	Add	Remove	Set
Create		h				
Delete	h		h	h		h
Move		h	h			
Add		h			s	
Remove				s		
Set		h				s

Table 5.4 Requires relation.

	Create	Delete	Move	MoveIndex	Add	Remove	Set
Create	√						
Delete	√						
Move	√						
MoveIndex	√						
Add	√						
Remove	√				√		
Set	√						

5.2.6 Conflict Resolution and Merging

DiCoMEF uses a human controller to manage the evolution of (meta-)models. S/He is assumed to be a business domain expert with good modeling experience. Besides, s/he has a right to accept or reject change requests received from users. Once a new release is available, changes are propagated to all users who must take them into consideration before their own operations. In DiCoMEF, the controller role can be assigned or delegated to other members. This could help to facilitate collaboration among users with different expertise (database, user interface design, business domain, ...). In case of conflict with his/her local change, DiCoMEF supports a semi-automatic conflict reconciliation strategy. Later, a user can send his/her local modifications merged with the last release of the model as a change request to the model controller. DiCoMEF provides users a facility to compose changes so as to put them in a same context (i.e., refactoring changes). This could later help users to understand changes during reconciliation process. It also lets users to annotate rationale of changes with multimedia files (i.e., audio, video, image, or text). During the reconciliation process, users can consult them to better understand rationale of changes and resolve conflicts.

Operation-based merging is a process of fusion of the ω^L and ω^C histories in such a way that conflicts are avoided. Where ω^C is a sequence of changes propagated from a controller and ω^L is a sequence of local changes performed by an editor. Some order of execution of operations could be imposed to facilitate merging. For instance, if ω^C must be executed before ω^L, then this would force ω^L to be rolled-back, next ω^C would be applied and finally ω^L could be re-applied. But this process is time consuming (e.g., roll-back a one day work for a propagated change that renames an EObject). Another option could preserve ω^L and next apply ω^C while there is no conflict. When a conflict is detected, ω^L is rolled-back and the scenario is reversed. A user can keep or drop some changes from ω^L when it is re-applied. But this ordering is also a time consuming process . The optimal option is to consider M/MM$\gg \omega^L \gg \omega^C$. In this strategy, if a conflict occurs while applying ω^C, changes in ω^L that caused the conflict are rolled back. This last strategy has been chosen in DiCoMEF and relies on an in-depth analysis conflicts and of the causal relationship between

the rolled back operations and other traces in the histories. This is discussed in the next section.

5.2.7 Model Versioning

DiCoMEF framework provides operation-based versioning support for both models and meta-models. It specifically manages revisions of (meta-)models, not variants. The framework stores direct deltas as differences between two successive versions. Indeed, DiCoMEF employs an extensional versioning technique, where each version is explicitly defined and has been sequentially checked-in to the version control system. It identifies each version by using an UUID. The versions are stored in a forward mode, DiCoMEF applies the reverse of direct deltas in order to generate previous versions. DiCoMEF does not remove deltas and versions from a history, as a result, a version with same UUID can occur more than once in the history. For example, if the history H contains three versions, $H = \{\upsilon_1, \upsilon_2, \upsilon_3\}$, then each version contains a direct delta and υ_3 is the most recent version. Later, if a user rollbacks the last version, DiCoMEF will add $\upsilon_2^{'}$ to the history to finally get $H = \{\upsilon_1, \upsilon_2, \upsilon_3, \upsilon_2^{'}\}$. Both $\upsilon_2^{'}$ and υ_2 refer to a logical version with the same identifier '2', but their contents (deltas) are different. $\upsilon_2^{'}$ contains the inverse of the deltas that evolves (meta-)models from version '2' to version '3', whereas υ_2 contains deltas between υ_1 and υ_2. After rolling-back, the history contains two logical versions (i.e., υ_1 and υ_2). DiCoMEF version management identifies and manages logical versions and other equivalent versions created due to reverse of direct delta operations.

5.2.8 Composite Operation Recovering and Detection Framework

Operations constituted in deltas improve users understanding about the evolution of modeling artifacts [Langer et al., 2013]. These operations can be classified as atomic, refactoring, or composite change operations [Langer et al., 2013]. *Atomic operations* (i.e., creation, addition, deletion, update) are low level operations and it is a cognitively challenging task for users to understand and re-construct high level changes (which might reflect the intention of

modifications) from primitive changes. Hence, atomic operations do not scale. A *composite change operation* denotes any type of in-place model transformation that executes all its children operations within one transaction [Langer et al., 2013].

Refactoring operations are composite operations that modify internal structures of software artifacts without changing their external behaviour [Mens and Taentzer, 2007]. Model refactoring is a type of model transformation that does not alter the semantics of the model. It does not add new functionality or remove existing once from the model, rather it simplifies the design model without changing its external behaviour [Mohamed and Romdhani, 2009; Van Der Straeten et al., 2007]. Model refactoring operations help to improve the understanding of modifications and intentions of a user who performed changes. The recovering of model refactoring operations is an important activity in model management [Langer et al., 2013] and is even a crucial task to ensure collaborative modeling.

Meta-model adaptation might cause instance models inconsistent. Instance models might no longer satisfy the set of rules and constraints specified by the meta-model. Therefore, models need to be co-evolved with their respective meta-models in order to preserve the conformance between models and meta-models [Herrmannsdoerfer, 2009]. Indeed, complex operations, which adapt a meta-model, can be coupled with model adaptation instructions to migrate instance models [Herrmannsdoerfer, 2009]. Detecting such complex operations helps to identify a set of model migration instructions [Vermolen et al., 2012] that transform instance models.

Figure 5.13 depicts a sample Petri net meta-model along with a journal of atomic operations. When cooperative frameworks use a change-based approach to merge and reconcile concurrent versions, users are confronted with this information that may prove very difficult to understand. Indeed, it is not related to users intents, but to a technical API. Users mostly reason about modifications in terms of high level operations (i.e., refactoring or composite operations). For example, one refactoring operation like “refine a class into subtypes” may concern many atomic operations that wouldn’t have any meaning for the users. When they have to solve a conflict, they would prefer to accept or refuse the “refinement” operation and certainly not each atomic operation that constitutes it. There is thus a cognitive gap between

the primitive operations and the mental setup of the modelers about changes.

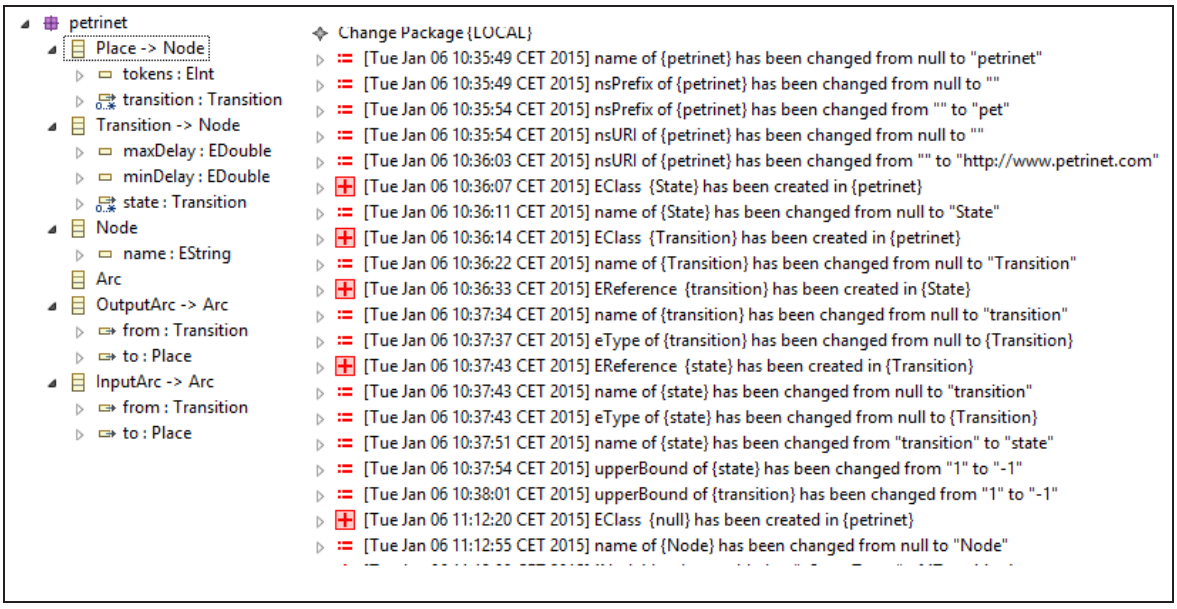

Figure 5.13 A Petri net meta-model and atomic meta-model adaptation operations

The objective of this work is to recover and detect composite operations (including refactoring operations) from a journal composed of primitive operations. This information is required to guide users when they have to reconcile concurrent (meta-)model versions produced by cooperative editing tasks. Indeed, the reconciliation process may oblige users to choose between concurrent and conflicting operations (e.g., two modifications on the same resource). Confronted with atomic operations, this decision may be impossible to take, since operations have no pertinent meaning from the user's viewpoint. In addition, the detected complex operations can be used to specify model migration instructions that co-evolve instance models along with meta-model adaptation [Vermolen et al., 2012].

Much research work has already be done to detect refactoring and complex change operations in the context of object oriented programming. Demeyer et al. propose a method to detect refactoring operations from successive versions based on change metrics [Demeyer et al., 2000]. Besides, Dig et al. present a RefactoringCrawler framework that uses a combination of syntactic analysis and semantic analysis to detect refactoring operations [Dig et al., 2006].

Composite operations can be included as part of the development environment and they will be tracked whenever they are executed [Herrmannsdoerfer, 2009]. In operation-based collaborative modeling, change operations are usually canonized (normalized) to speed up the transfer of data and to facilitate the merging process [Mens, 1999b]. As a result, operations that are superseded by new ones can thus be cleaned from the history (i.e. the record of change operations). Hence, canonization of composite operations could produce a different type of composite operations that might not be defined by the modeling environment. Solving this problem requires to re-group primitive operations into other composite operations manually, that is a tedious and difficult task. Moreover, removing atomic changes from a composite operation might invalidate its constraints [Koshima et al., 2013]. Hence, there should be a tool support to specify composite operations.

Prete et al. use refactoring template rules to recover refactoring operations between two program versions [Prete et al., 2010]. In another work, Xing et al. [Xing and Stroulia, 2006] propose an approach to detect refactoring operations based on UMLDiff. This is a domain specific algorithm (UML-aware) that compares the structural changes between two successive versions of class models and drives their differences [Xing and Stroulia, 2005]. Queries are applied on deltas so as to detect different complex change operations. However, this approach is limited to a specific modeling language [Langer et al., 2013]. Vermolen et al. also demonstrate a modeling language specific approach that reconstructs complex meta-model adaptation operations [Vermolen et al., 2012]. In [Langer et al., 2013], Philip et al. present an approach that automatically detects composite operations between two successive models, which are defined in any Ecore [Steinberg et al., 2009] based modeling languages. However, their approach does not provide a facility to aggregate composite changes from another composite change(s). Nevertheless, this raises interesting challenges that are described below.

An automatic composite operation detection mechanism might find results that don not reflect users intentions. It would neither allow user interaction to identify the correct composite operations that reflect his/her intent. For instance, a user may wish to remove some atomic change operations from a composite operation while keeping the constraints of the

composite operation. Indeed, an automatic process necessarily tries to compute the largest set of operations while recovering a composite operation. Nevertheless, some operations could be irrelevant because they were executed with a different intent or in distinct stages. Besides, s/he might also add or remove rationale of changes attached to specific changes based on a new context of composite change operation. Hence, we argue that interactive composite change detection approach could improve the final result. However, the approaches presented in [Langer et al., 2013; Prete et al., 2010; Vermolen et al., 2012; Xing and Stroulia, 2006] do not support user interactions to iteratively identify composite change operations.

In this PhD thesis, we apply a rule-based system in order to detect and recover composite and refactoring operations. The rule-based system uses rules to draw conclusions from premises [Hill, 2003]. A rule is a kind of instruction that has a left hand side and a ride hand side. The left hand side of the rule is a premise that contains the domain of the rule (facts), and must be true in order for the rule to potentially fire and derives conclusions (the right hand side). The inference engine determines the order of execution of rules. Rule-based systems employ either a *forward-chaining* or a *backward-chaining* inference method [Gilman et al., 2010; Hill, 2003]. *"Forward-chaining is a bottom-up computational model. It starts with a set of known facts and applies rules to generate new facts whose premises match the known facts, and continues this process until it reaches a predetermined goal, or until no further facts can be derived whose premises match the known facts"* [Al-Ajlan, 2015]. As its name suggests, backward chaining inference method works backward from the goal and tries to attempts to find evidence to support these conditions (hypotheses).

Prolog [Wielemaker, 2004] is a rule-based language that employs a backward chaining inference method, and it consumes less memory compared with Java Expert System Shell (Jess) [Hill, 2003], which adopts a forward chaining inference method. Jess implements the RETE algorithm that computes things once and reuses them such that it explores medium-sized numbers of possibilities repeatedly. As a result, Jess is faster than Prolog [Gilman et al., 2010]. Besides, Jess also supports backward-chaining inferences. Jess language can be easily integrated with Java applications either by using Java to extend Jess or the Jess

library can be used from Java [Hill, 2003]. In this work, we use the Jess language so as to recover and detect composite and refactoring operations from a set of change operations, which are recorded during the (meta-)model editing phase. Nevertheless, the work can also be used to find composite operations from *diffs* computed using state-based comparison.

Figure 5.14 describes the proposed framework. Whenever a user adapts a (meta-)model, edit operations are recorded in a history formally defined by the history meta-model of DiCoMEF (see Figure 5.8). Later, these operations are canonized and transformed into Jess facts by using a model-to-text transformation engine. A Create change operation is transformed into a Create fact in the Jess fact base. Likewise, Set, Add, Remove, Move, MoveIndex, and Delete change operations are transformed into Set, Add, Remove, Move, MoveIndex, and Delete facts in the Jess fact base (see Figure 5.15). The types of Jess facts are defined using Jess templates (see Figure 5.17). In the Figure 5.14, the rectangle shape represents an automatic process, the trapezoid shape indicates a manual work, and the arrows represents data flows.

Jess templates are used to define structures of Jess facts that correspond with composite operations (see Figure 5.16), and Jess rules are specified to derive Jess facts. As a proof of concept, we manually define Jess templates and Jess rules for some of composite operations presented in the the Edapt [7] framework. For instance, we encode composite operations such as extract subclass, extract super class, pull up feature, and push down feature into Jess templates and Jess rules. During the execution, when a rule's left hand side is satisfied, a new fact, which represents a composite operation, is asserted into the fact base. Composite operation facts can also be aggregated into larger facts. The result of this inference is a hierarchical representation of composite change operations. Jess facts are Java objects, such that it is possible to interrogate a list of slots of a fact and their values. Later, we translate the detected and recovered composite Jess facts into composite change operations. As discussed in 5.2.2 and Table 5.1, Edapt does not provide facilities to aggregate composite operations from other composite operations. In addition, it does not support instance model evolution. Therefore, the composite change operation is defined using the DiCoMEF history

[7] https://www.eclipse.org/edapt/operations.php

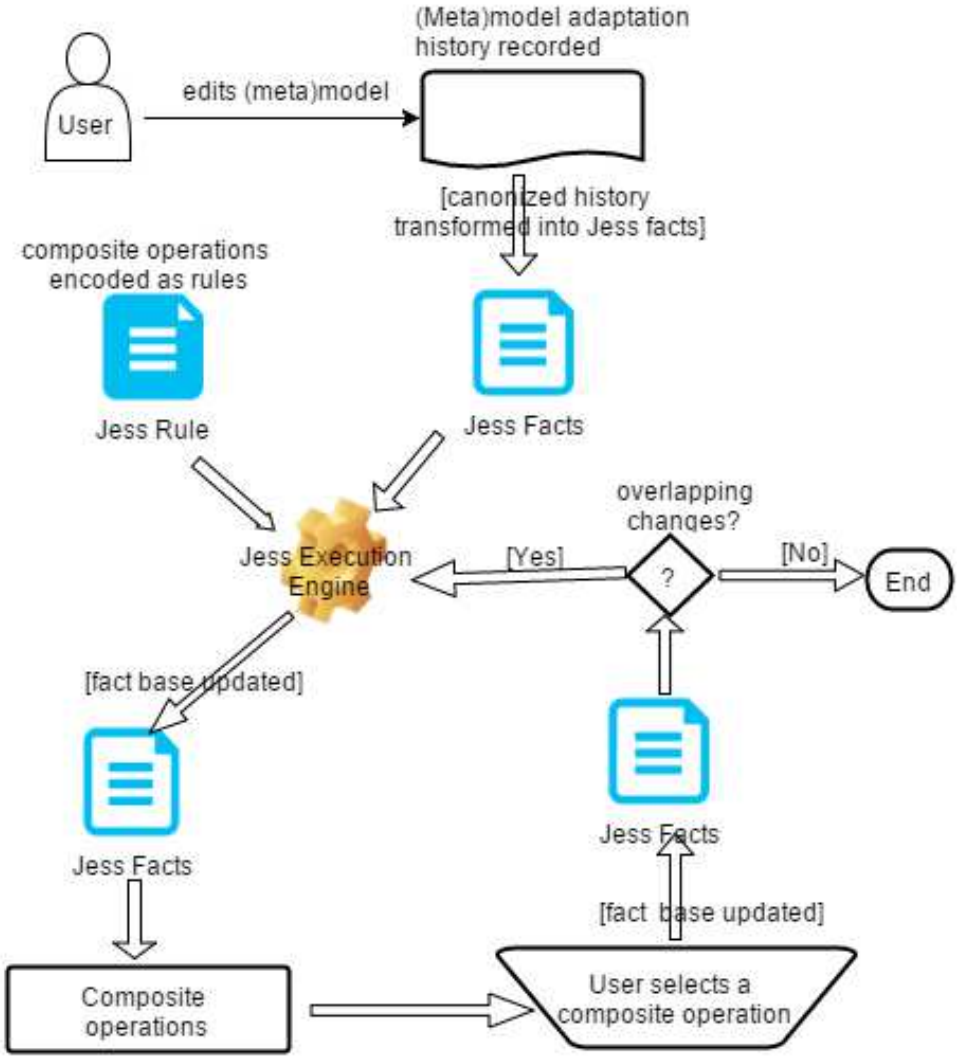

Figure 5.14 A rule-based composite operation detection and recovery steps

meta-model as presented in Section 5.2.2 (see Figure 5.8). Like the approach presented in [Langer et al., 2013], the composite operation detection and recovering tool can work with any modeling language based on Ecore.

The inference process of recovering and detecting composite operations may be non-deterministic. The analysis engine identifies overlapping composite operations and displays the result to the user, so that he/she can select the "best" composite operation that reflects his/her intentions. Besides, if rationale of modifications are attached to change operations, the engine asks the user to verify if the rationale is still valid for the new composite operation.

```
(deffacts primitiveChanges
    (create (id _T4sCkuNwEeS327Bhro-9Xw) (element _attNewField)
        (referenceName eStructuralFeatures)
        (target _classB) (eObject "ecore:EAttribute"))
    (set (id _T4sCluNwEeS327Bhro-9Xw) (element _attField2)
        (featureName eType) (dataValue nil)
        (oldDataValue nil) (referenceValue Ecore_EDataType:EByte)
        (oldReferenceValue nil))
    (set (id _T4spgeNwEeS327Bhro-9Xw) (element _refReferTo)
        (featureName eType) (dataValue nil)
        (oldDataValue nil) (referenceValue nil)
        (oldReferenceValue _classC))
    (create (id _T4tQkONwEeS327Bhro-9Xw) (element _attField2)
        (referenceName eStructuralFeatures)
        (target _classD) (eObject "ecore:EAttribute"))
    (move (id _T4tQlONwEeS327Bhro-9Xw) (element _attField)
        (referenceName eStructuralFeatures)
        (source _classC) (target _classD))
    (delete (id _T4tQmeNwEeS327Bhro-9Xw) (element _refReferTo)
        (referenceName eStructuralFeatures)
        (target _classD))
    (delete (id _T4tQneNwEeS327Bhro-9Xw) (element _classC)
        (referenceName eClassifiers)
        (target _4uatkEvVEd-NDfg2t2Dz8Q)))
```

Figure 5.15 Primitive operations represented as Jess facts

Indeed, a user can remove atomic operations from the proposed composite operation as long as constraints of the composite operations are satisfied.

The analysis engine examines dependencies between change operations and ordered composite operations. Of course, it uses the "*require*" relationship specified in Section 5.2.5 to identify the pre-condition of atomic and composite operations so as to order them. Hence, a user can sequentially execute an ordered list of composite operations on a base version of a (meta-)model and studies their effect. Moreover, the analysis engine updates the fact-base based on the user's decisions and the Jess engine re-executes the rules until there is no more overlapping operations (composite operations that belong to different paths of a tree can't share the same operation(s)). Figure 5.15, Figure 5.17, and Figure 5.16 illustrate snippet

codes of Jess templates (which defines types of Jess facts), Jess facts that represents a useful pieces of information, and Jess rules, which represent composite operations.

```
;;(defmodule COMPOSITE)
(deftemplate extractSuperClass
         "Extract SuperClass Refactoring Operation"
         (slot subClasses)
         (multislot  toExtract)
         (slot superClassName)
         (multislot changes))

(deftemplate createEClass
         "Create an EClass Operation"
         (slot element)
         (multislot changes))

(deftemplate createEAttribute
         "Create an EAttribute Operation"
         (slot element)
         (multislot changes))

(deftemplate renameModelElement
         (slot element)
         (slot change))

(defrule createEAttribute


(defrule renameModelElement
        ?s <- (set (element ?e))
        (not (create (element ?e)))
       =>
         (assert (renameModelElement (element ?e) (change ?s))))
```

Figure 5.16 Composite operations expressed in Jess rules

```
(deftemplate set "The Primitive Operation Set"
          (slot id (type STRING))
          (slot element (type STRING))
          (slot featureName (type STRING))
          (slot dataValue (type STRING))
          (slot oldDataValue (type STRING))
          (slot referenceValue (type STRING))
          (slot oldReferenceValue (type STRING)))

(deftemplate add "The Primitive Operation Add"
          (slot id (type STRING))
          (slot element (type STRING))
          (slot featureName (type STRING))
          (slot dataValue (type STRING))
          (slot referenceValue (type STRING)))

(deftemplate remove "The Primitive Operation Remove"
          (slot id (type STRING))
          (slot element (type STRING))
          (slot featureName (type STRING))
          (slot dataValue (type STRING))
          (slot referenceValue (type STRING)))

(deftemplate create "The Primitive Operation Create"
          (slot id (type STRING))
          (slot element (type STRING))
          (slot referenceName (type STRING))
          (slot target (type STRING))
          (slot eObject (type STRING)))

(deftemplate delete "The Primitive Operation Create"
          (slot id (type STRING))
          (slot element (type STRING))
          (slot referenceName (type STRING))
          (slot target (type STRING)))

(deftemplate move "The Primitive Operation Move"
          (slot id (type STRING))
          (slot element (type STRING))
          (slot referenceName (type STRING))
          (slot source (type STRING))
          (slot target (type STRING)))
```

Figure 5.17 DiCoMEF history representation using Jess templates

The prototype of composite operation detection and recovery tool is implemented as an Eclipse plugin. However, the prototype has limitations, for instance, it asks users to encode Jess templates and Jess rules manually in order to recover and detect composite operations. This requires a high level of Jess programming knowledge from the user. The prototype also displays the recovered and detected composite operations in the console window, and it is not also fully integrated with the DiCoMEF framework. In the future work, we will investigate how a user can specify composite operations by-examples [Brosch et al., 2009a,b]. For instance, a user could use the Edapt framework to adapt a meta-model, subsequently, the tool might (semi-)automatically generate Jess templates and Jess rules related to these operations. We will also fully integrate the composite operation detection and recovery tool with the DiCoMEF framework, and improve the visualization of the tool.

5.3 Communication Management in DiCoMEF

As discussed in Section 5.1, DiCoMEF is a distributed collaborative modeling framework, where each member of the group has his/her local copy. In DiCoMEF, members communicate their work by exchanging change operations that adapt (meta-)models. As described in Section 5.2.5, these operations are also used to detect conflicts and to help the reconciliation process. Besides, the history of (meta-)model adaptations could be mined to guide software changes [Zimmermann et al., 2004a], to identify architectural violations, and to find common error patterns [Livshits and Zimmermann, 2005]. Moreover, it can be used to detect methodological inconsistencies [Blanc et al., 2008], and serve as the source of information for group members to study the evolution of the software project. Hence, we argue that keeping the exact sequences of histories at every local repository is important. DiCoMEF uses concepts such as a controller, a main-line, and a branch to ensure exactly the same history and (meta-)models at each member site (see Section 5.1).

Figure 5.3 presents concepts that are used to model the social organization of the DiCoMEF framework. A user with a controller role manages the evolution of (meta-)models. As shown in figure 5.2, members communicate their modifications via the controller. In

order to edit (meta-)models, an editor should create a branch from the main-line (see Figure 5.4). Afterwards, s/he modifies the (meta-)model locally and sends her/his modifications to the controller as a change request later. The controller inspects modifications and sends accepted modifications to all members. Only these accepted changes are applied on the main-line. Indeed, this guarantees that the evolution of (meta-)models on the main-line contains the same sequence of histories and (meta-)model in all members' local repositories. The controller role is flexible, even a new member can be assigned as a controller, since s/he has exactly the same copy of (meta-)models and histories. The observer role is assigned to someone who passively contributes to the project, and s/he cannot directly edit (meta-)models.

The DiCoMEF framework could be extended to support a large community of users as shown in Figure 5.18, where an editor acts as a *virtual controller* for other editors (*side editors*) working under her/his supervision. These new roles (virtual controller and side editor) are transparent for the DiCoMEF controller. Side editors could also modify (meta-)models concurrently (e.g. by using the *Cloud*) but these modifications would be out of the scope of DiCoMEF. This type of communication might be needed among a group of competitor companies that work together to standardize the (meta-)model. The central standardization agent plays a role of a controller and companies play roles of editors. Inside these big companies, there might be different (meta-)modelers distributed globally. Hence, these companies can be considered as virtual controllers and (meta-)modelers of these companies play side editor roles. DiCoMEF considers the branch of virtual controllers as virtual main-lines. These virtual main-lines are synchronized with the main-lines of side editors. In other words, the relationship between main-lines and branches are changed when we move vertically on hierarchical collaborative modeling. At the top level of the hierarchy, the main-line is the root of the communication, it stores different versions of the global (meta-)models. When editors add one level of collaborative modeling under their supervision, DiCoMEF considers their branches as virtual main-lines.

The DiCoMEF history captures user identifiers, date, and the time during (meta-)model adaptation. These improves the awareness of the group about who contributes these mod-

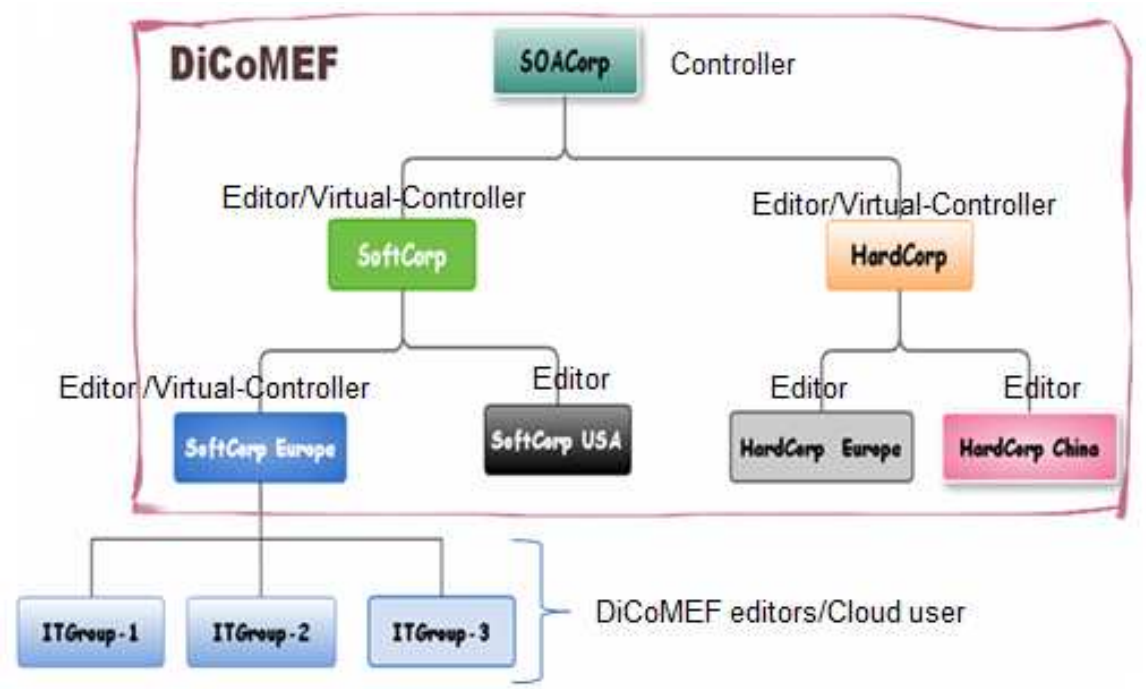

Figure 5.18 Extended DiCoMEF Architecture

ifications and when. These is important for users to communicate the responsible person during merging of conflicting modifications in their local branches. Of course, users can always communicate the controller to resolve conflicts. Moreover, the history meta-model of DiCoMEF provides facilities to annotate rationales of modifications with multimedia files (i.e., audio, video, image, and text). Users can also compose changes from another changes to put modifications into context (i.e., refactoring operations). Besides, RuCORD could improve awareness of users about intentions behind modifications of other members by detecting and recovering composite and refactoring operations. DiCoMEF also provides group information and their e-mail contact addresses. It is possible to implement on top of DiCoMEF to support online-chat rooms and to notify available users using Eclipse Communication Framework (ECF)[8].

[8]https://eclipse.org/ecf/

Chapter 6

Evaluation

This chapter presents the preliminary evaluation of DiCoMEF framework conducted with masters students at the university of Namur, Belgium. The result was also presented in [Koshima and Englebert, 2015b].

We have conducted a preliminary evaluation of the DiCoMEF framework with graduate students (masters in computer science) at the University of Namur. Two second year and four first year students participated in this evaluation during the "Advanced questions in information systems engineering" (INFOM435) course. They had also followed the course "Software architecture engineering: Advanced topics" (INFOM434) that introduced them to advanced modelling theories, domain specific modeling languages in software engineering, and the Eclipse development environment. Besides, we provided them ten hours of training and exercises on Eclipse Modeling Framework (EMF) and DiCoMEF framework. Moreover, screen-casts (i.e. video tutorials) of the DiCoMEF framework and EMF framework were available for students two weeks before the evaluation to let them practice at home.

The evaluation was mandatory and it accounted as 40% of the course and students were encouraged to give their honest opinions about the framework. We explicitly stated that the goal of the evaluation was to improve the framework, hence, their feedback was valuable whatever were their answers. The evaluation was conducted for two hours, afterwards, students were asked to fill a questionnaire. In addition, each student was invited to submit a report about the strengths and drawbacks of DiCoMEF framework. The evaluation was

done anonymously, but the small size of the population is of course a possible bias. Unfortunately we had only six students who had attended the course and participated in this preliminary evaluation. However, the result could give an indication of the potential use of DiCoMEF framework for collaborative modeling.

The objective, design, result, and discussion of the case study are presented in the following sections.

6.1 Objectives

The evaluation consisted in guiding several teams with a cooperative scenario during the elaboration of a car DSML and a Petri net DSML using the DiCoMEF framework. Our objectives were to evaluate the model management functionality of the DiCoMEF framework such as model comparison, conflict detection, conflict reconciliation, model merging, and versioning. Besides, we were also interested to evaluate a role-based member management, workflow of the DiCoMEF framework, and its usability. Our objectives are described hereafter.

1. Evaluating the effectiveness of DiCoMEF to support the cooperative design of meta-models for DSML, more specifically:
   (a) its versioning: *does DiCoMEF support versioning of meta-models?*
   (b) its workflow: *are the proposed communication and the meta-model evolution management of DiCoMEF framework effective?*
   (c) its usability: *is the framework easy to use and to learn?*
   (d) its role management: *which efforts are required by a user to join/leave a group? Which efforts are required to change a controller?*
2. DiCoMEF detects both structural and static semantic conflicts specified with the Object Constraint Language (OCL): *is the detection mechanism accurate and complete?*

3. Evaluating the benefits of the reconciliation and merging processes of DiCoMEF, more particularly:

   (a) the usability of merge tool: *is the merge tool easier to use and to learn?*

   (b) the benefits of multimedia files which are attached to change operations: *do rationales effectively help users to understand modifications performed by other members? Is the rationale of modifications helpful to facilitate reconciliation and merging?*

   (c) the benefits of the merge tool: *does the DiCoMEF framework provide enough information about the conflict (i.e., conflicting operations along with concerned model elements and rationale of modifications)? Is it easy to identify conflicts? Does it merge conflicting versions? Is it easier and faster to detect conflicts and merge conflicting versions using DiCoMEF framework than manual work?*

6.2 Experimental Design

As discussed above, the case study was conducted with graduate students that we distributed into three groups of two. Students of each group were asked to cooperate together to design a new DSML for the automotive domain. Besides, we also provided them two conflicting versions of the Petri net meta-model and they had to merge conflicting meta-models.

In this case study, we have used the "benchmark for conflict detection of model versioning systems" [1] and more specifically the "class diagram versioning" test case described in [Langer and Wimmer, 2013]. It lists a series of common operations that may cause troubles during concurrent modifications. They were used to design a cooperative scenario that students had to follow in each group. Nevertheless, some cases that were not supported by the DiCoMEF merge tool were discarded. For instance, "contradiction in hierarchy" (CH) and "semantics in associations" (SA) conflicts listed in the benchmark can not be detected by DiCoMEF. The CH conflict is raised when a user applies a refactoring operation to extract an

[1] http://www.modelversioning.org/index.php?option=com_content&view=article&id=62&Itemid=91

abstract super class by pulling up a method(s), while another user does the same operation but extracts an interface. SA conflicts occur when modelers express the same information in different ways. The following test cases were used in our scenario: “Add Different Model Element”, “Rename Model Element and Unit of Consistency”, “Delete/Update Model Element”, and “Delete/Delete Model Element and Add Model Elements with Same Name”. The details of these modifications are provided below in this section. Some terms used in the benchmark were also renamed in order to suit the EMF peculiarities. For example “Add Different Model Elements” stands for an addition of different classes, references, and attributes.

The case study consisted in two phases. In the first phase, students were asked to develop a DSML specific to automobiles. Each group started with a simple meta-model that contained only one class and one attribute (see Fig. 6.1). Afterwards, we asked the students to develop the language iteratively by introducing different modifications sequentially. They are explained hereafter:

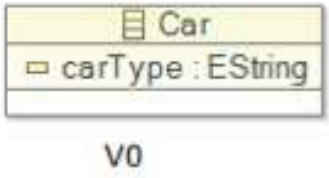

Figure 6.1 Automobile Meta-model Version$_0$

1. **Add different Model Elements:** Two EClasses, two EReferences, and one EAttribute are added with different identities and names (see Fig. 6.2).

   - Original Version (V_0): a model containing an EClass `Car` and an EAttribute `carType`.
   - Working Copy 1 (V_1'): an EClass `Body` and an EReference `body` are added.
   - Working Copy 2 (V_1"): an EClass `Engine`, an EReference `engine`, and an EAttribute `engineType` are added.
   - Expected Conflicts: none

- Expected Merge Result: The model that includes the `Body` and `Engine` EClasses, the `body` and `engine` EReferences, and the `engineType` and `carType` EAttributes.

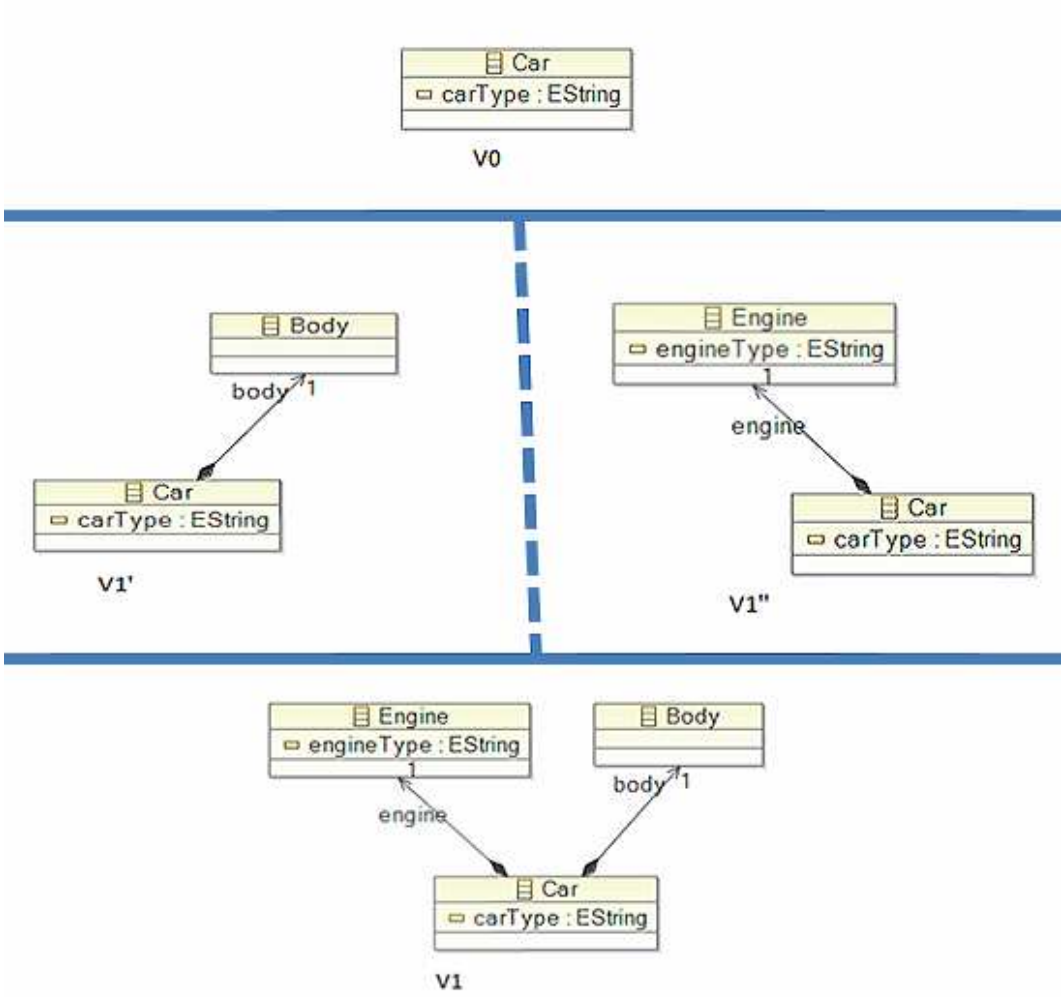

Figure 6.2 Automobile Meta-model Version$_1$

2. **Rename Model Element and Unit of Consistency:** A "unit of consistency" validates the granularity of conflict detection mechanism. For instance, two different properties of a model element might be changed independently (e.g., the name of an EReference and its cardinality value are changed independently). "Rename model element" describes the scenario where the name of a same model element is changed either in a different or a same way. If they are changed the same way, there is no

conflict. Otherwise, there is a conflict and either of changes should be applied (see Figure 6.3).

- Original Version (V_1): the model includes EClasses `Car`, `Body` and `Engine`, EReferences `body` and `engine`, and EAttributes `engineType` and `carType`.
- Working Copy 1 (V_2'): an EClass `Body` is renamed to `Component` and an EReference `body` is renamed to `component`.
- Working Copy 2 (V_2"): an EClass `Body` is renamed to `Component` and an EReference `body` is renamed to `components`. Besides, the upper bound of a reference (`body`) is changed to unlimited (*).
- Expected Conflicts: contradicting and overlapping modifications. A reference `body` is renamed differently by two users like `component` and `components`.
- Expected Merge Result: user decision is required.

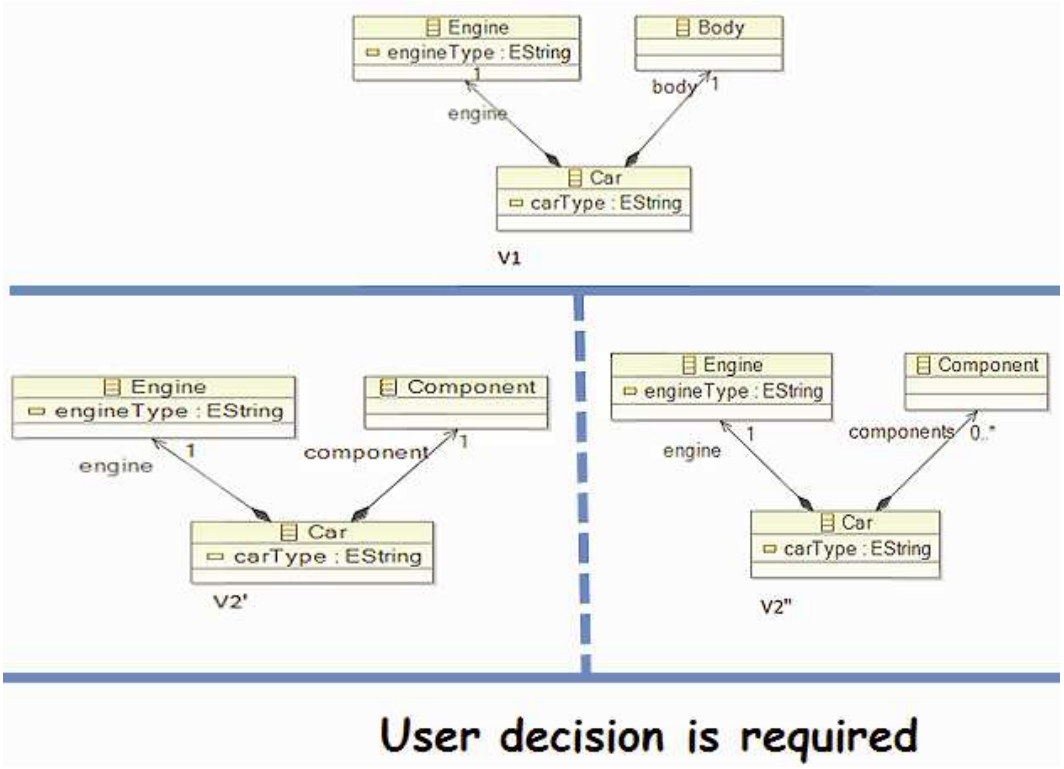

Figure 6.3 Automobile Meta-model Version$_2$

3. **Delete/Update Model Element:** One user modifies a property of a model element while another one deletes the same element concurrently (see Figure 6.4).

   - Original Version (V_2): the model includes EClasses `Car`, `Engine` and `Component`, EReferences `components` and `engine`, and EAttributes `engineType` and `carType`.
   - Working Copy 1 (V_3'): an EClass `Engine` is renamed to `Motor`. In addition, an EClass `MotorType` is created and an EAttribute `engineType` is deleted. A new EReference `type` is created in the `Motor` and its EType is set to `MotorType`.
   - Working Copy 2 (V_3''): an EEnum `EngineType` is created and an EAttribute `engineType` is renamed to `type`, besides, its EType is changed from EString to `EngineType`.
   - Expected Conflicts: contradicting and overlapping modifications. An EAttribute `engineType` is deleted in the first working copy, whereas it is renamed in the second working copy.
   - Expected Merge Result: user decision is required.

4. **Delete/Delete Model Element and Add Model Elements with Same Name:** "Delete/Delete" denotes the deletion of a same model element on both sides. "Add new model elements with same name" creates a new model element with the same name in both working copies (see Fig. 6.5).

   - Original Version (V_3): the model includes EClasses `Car`, `Motor` and `Component`, EReferences `components` and `motor`, EAttributes `type`, `carType`, and EEnum `MotorType`.
   - Working Copy 1 (V_4'): a user deletes an EAttribute `carType`, and creates an EClass `CarType`. Besides, s/he adds an EAttribute `name` in the `CarType` and creates an EReference `type` in the `Car`. The EType of the `type` reference is `CarType`. S/he also creates new classes: `Brake`, `Wheel`, `Seat`, `Door`, `GearBox`, and `Roof` that are inherited from EClass `Component`.

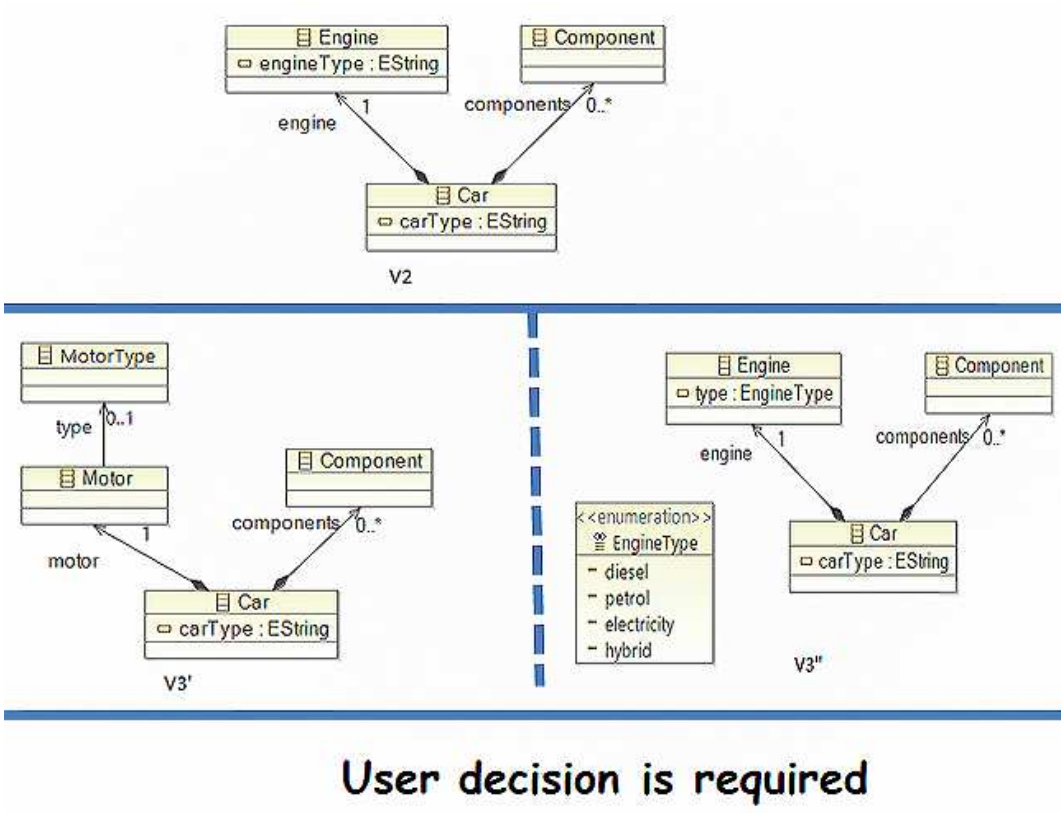

Figure 6.4 Automobile Meta-model Version$_3$

- Working Copy 2 (V_4"): a user deletes EAttribute `carType` and creates EClass `CarType`. Besides, s/he adds EAttribute `name` in `CarType` and EReference `type` in `Car`. The `type` reference has EType `CarType`. S/he also creates EClasses `Option` and EEnum `OptionType`. S/He also established an association between the `Option` and `OptionType`.
- Expected Conflicts: contradicting and overlapping modifications. It violates an OCL constraint: two model elements of a same parent container must have distinct names.
- Expected Merge Result: user decision is required.

In the second phase of the case study, we grouped all students into one group that was composed of six students and a controller. The controller role was played by A. Koshima, but his task was limited to add students in groups to allow them to send later the base version

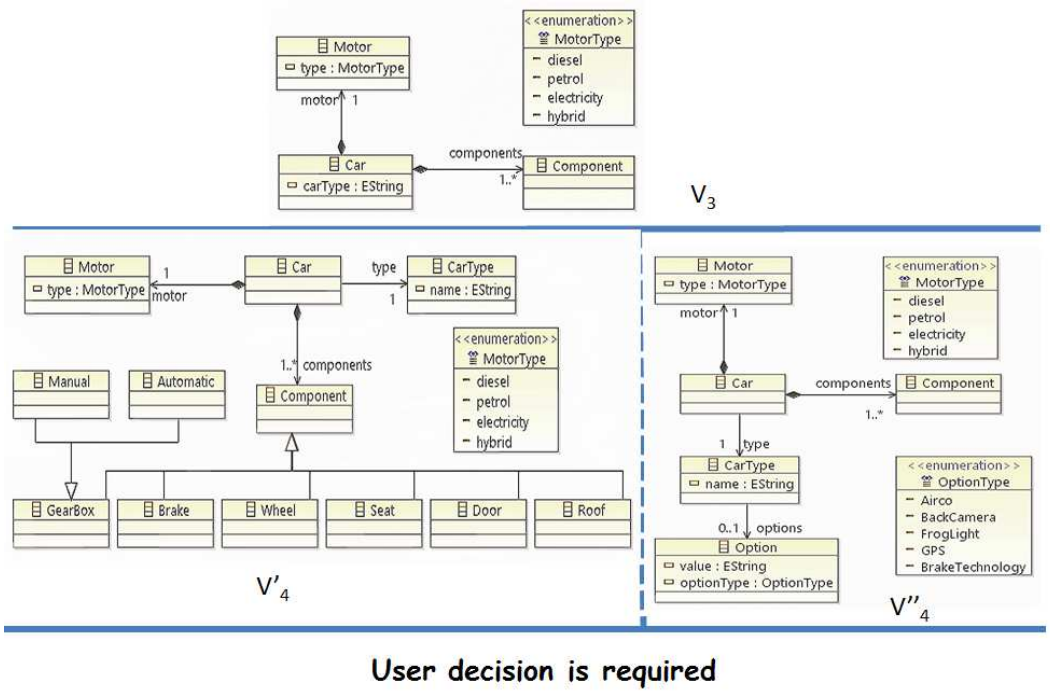

Figure 6.5 Automobile Meta-model Version$_4$

of Petri net meta-model along with the history (see Figure 6.6) to each new member. He also modified the Petri net meta-model locally and propagated his modifications to other members (see Figure 6.7). He did not participated in solving conflicts with students. We gave an instruction to each student about how to evolve the base version of the Petri net meta-model locally (see Figure 6.8). The students only used an EMF treeview editor to visualize and to edit the Petri net meta-model. Afterwards, each member were firstly asked to integrate their local modifications with propagated changes using DiCoMEF framework. They were also requested to merge modifications manually and produce the same result with the previous one. The objective of the second part of the case study is to validate the conflict detection, reconciliation, and merging process of DiCoMEF. After finishing the second phase, each student filled a questionnaire and submitted a report that summarized the drawbacks and strengths of DiCoMEF. We will present the findings in Section 6.3.

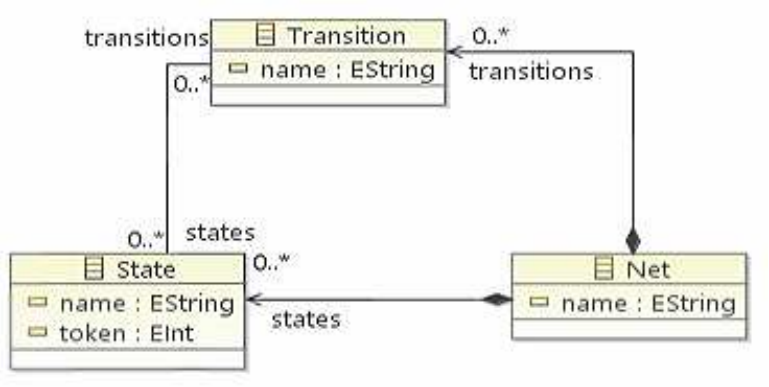

Figure 6.6 Petri net meta-model base version

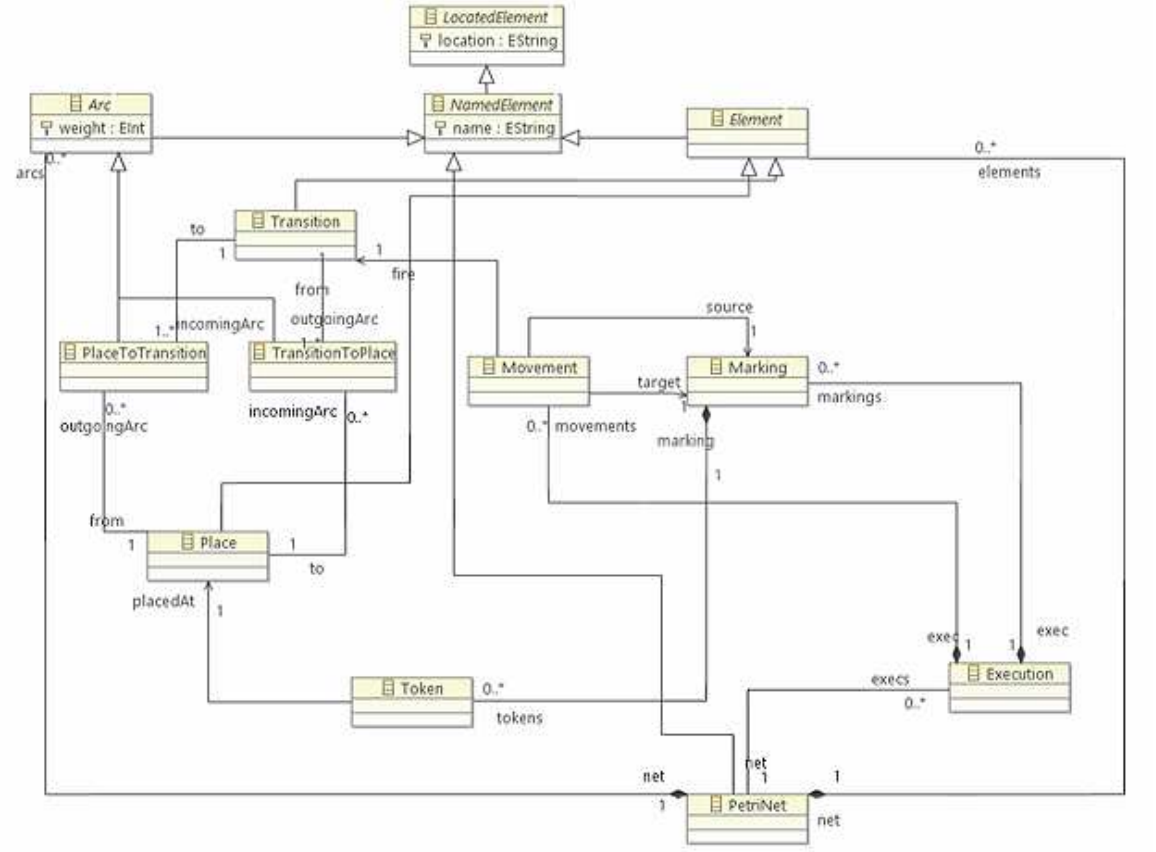

Figure 6.7 Propagated Petri net net meta-model

6.3 Results and Discussion

We grouped the results of the questionnaire based on the objectives. The questionnaire had closed questions with responses either in scale range from 1 (very poor) to 5 (very good) or

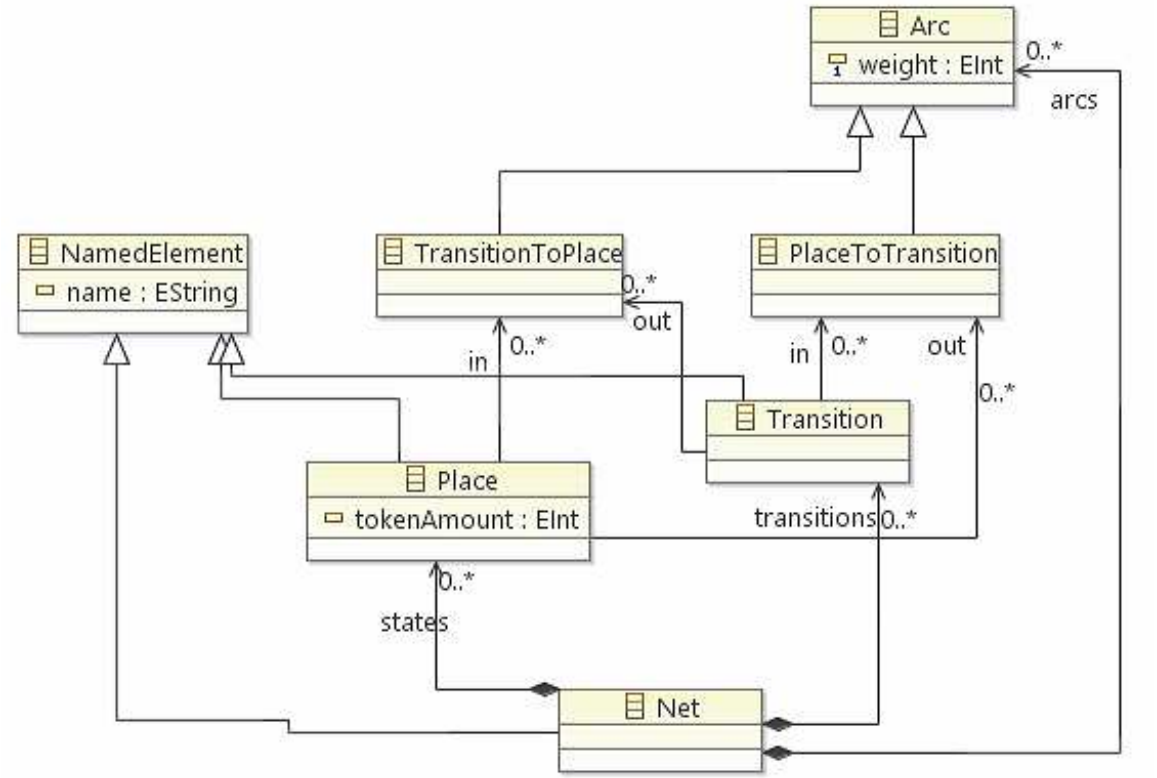

Figure 6.8 Local Petri net meta-model

Yes/No. It also contained open questions to collect students opinions on some topics. The results are summarized as follows:

Table 6.1 Objective 1: evaluating the effectiveness of DiCoMEF to support the cooperative design of meta-models for DSML

How do you evaluate the model versioning facility of DiCoMEF?							
3	4	4	4	4	5	objective: 1.a	average = 4
How do you evaluate the workflow of DiCoMEF?							
2	2	3	4	4	4	objective: 1.b	average = 3.2
How do you evaluate the usability of DiCoMEF?							
3	3	4	4	4	5	objective: 1.c	average = 3.4
How do you evaluate group management of DiCoMEF?							
3	3	3	4	4	5	objective: 1.d	average = 3.7

We also asked students to submit an individual report that answered the question below. Our motivation was to make sure that what they filled in the questionnaire was consistent with the report. The questions asked in the report were a summary of the questionnaire.

Table 6.2 Objective 2: is the conflict detection mechanism accurate and complete?

Does the merge tool detect all conflicts?							
Yes	Yes	Yes	Yes	Yes	Yes	objective: 2	6 Yes ~ 0 No
Does the merge tool give false positive?							
No	No	No	No	No	No	objective: 2	0 Yes ~ 6 No

Table 6.3 Objective 3: evaluating the benefits of the reconciliation and merging processes of DiCoMEF

Is the visualization of conflicts understandable?							
2	3	4	4	5	5	objective: 3.a	average = 3.8
How do you evaluate the usability of the merge tool?							
4	4	4	4	5	5	objective: 3.a	average = 4.3
Do the rationale of modifications useful to understand your colleague's intention?							
2	3	3	4	5	5	objective: 3.b	average = 3.7
Does DiCoMEF easily identify conflicts and facilitate the merging process as compared to manual work?							
4	4	4	4	5	5	objective: 3.c	average = 4.3
How do you evaluate the overall merging tool?							
4	4	4	4	5	5	objective: 3.c	average = 4.3
Manually: Time spent for merging (case study 2)?							
10	15	18	19	20	20	objective: 3.c	average = 17
DiCoMEF framework: time spent for merging (case study 2)?							
4	4	4	7	10	15	objective: 3.c	average = 7.3

- *What is the strongest side of DiCoMEF in your opinion?* The logging of change operations and the visualization of change operations that could help users to comprehend modifications made by other members. In addition, to setup the collaborative group is easy: a new user needs to provide his email account and the framework handles configuring the repository, files, and notification. The framework provides support for a controller (a senior member of the group) to manage the evolution of (meta-)model.

- *What is the weakest side of DiCoMEF in your opinion?* The usability aspect should be improved, specifically, the workflow should incorporate default activities to reduce the number of steps required to send change requests and to propagate changes. Email based communication is not convenient for exchanging large files. It is not enough integrated with the Eclipse Graphical Model Editing framework.

- *What is the strongest side of the merge tool in your opinion?* It is easy to use and to learn. Besides, it detects conflicts and provides a visualization that shows the effect of changes. The merge tool is flexible and provide a choice to accept or reject modifications performed by other member.

- *What is the weakest side of the merge tool in your opinion?* The usability, specifically, the user interface needs to be improved. For instance, the options of the merge tool needs to be clearly visible.

- *Which difficulties have you encountered?* The workflow is not easy to understand at the first glance, but, it becomes natural after some explanations. The framework does not support default activities so that it is much work to propose change request or propagate change propagation. One student also had a problem to understand OCL constraints violation messages. DiCoMEF works with indigo version of eclipse, but it needs to be updated with a recent version of eclipse.

- *Which future improvements would you recommend for the framework?* It should have a better integration with the graphical modeling framework. Besides, its conflict detection and merging should have a graphical support. It is also important to reduce the number of steps required by the workflow by introducing default activities. The framework needs to provide shortcuts for different activities in the workflow. Last but not least, the email based communication should be replaced with other form of communication in order to transfer large data files.

This validation work was a preliminary evaluation of DiCoMEF framework and it has some threats to the validity of the result. For instance, the number of students in the group is very few (two students per group) and it did not represent a real collaborative work scenario. In addition, we had only three groups that was not statistically significant to draw a conclusion. Although these treats, the results seem very encouraging for a preliminary evaluation and they allowed us to collect interesting comments to improve the framework. Based on the result presented in tables 6.1,6.2, and 6.3, DiCoMEF could be “effective” to support collaborative work. The preliminary result indicates the DiCoMEF framework might be

easy to learn and to use. Besides, it could manage the communication of the group and their roles. The framework could support model comparison, conflict detection (i.e., structural conflicts and static semantic conflicts), conflict reconciliation, and merging of conflict version of meta-models.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The software engineering community adopts the MDE approach to deal with complexities of software solutions that arise from inherent complexities of the business domain, time-to-market pressures, changes in user requirements, and evolution of the underlying software platforms. MDE uses separation of concern principles that reduces complexity, improves reusability, and ensures simpler evolution of modeling languages [Tarr et al., 1999]. It shifts the level of software development from code-centric to model-centric [Bézivin, 2005; Bézivin, 2004; Favre, 2004; Kent, 2002].

Modeling is an act and science of creating an abstraction of parts of the system understudy. It usually requires collaboration members of a group with different scope and skills (i.e., middleware engineers, human interface designers, database experts, and business analysts). *"Any software project with more than one person is created through a process of collaborative software engineering"* [Whitehead et al., 2010]. However, despite the fact that Domain Specific Modeling tools are becoming very powerful and more frequently used, the support for their cooperation has not reached its full strength, and demand for model management is growing. In cooperative work, the decision agents are semi-autonomous and therefore a solution for reconciliating DSM after a concurrent evolution is needed. Conflict detection and reconciliation are important steps for merging of concurrently evolved (meta-

)models in order to ensure collaboration. In this PhD thesis, we presented an operation-based distributed collaborative model editing framework, DiCoMEF. The contribution of the PhD thesis is summarized as follows:

Distributed collaborative framework for models and meta-models: DiCoMEF is a distributed collaborative modeling framework for both models and meta-models. DiCoMEF framework distributes clones of (meta-)models and histories among all members of the cooperative ensemble. Besides, it ensures consistent (meta-)models and histories among all members of the collaborative groups using main-line and branches as discussed in Chapter 5 and Section 5.1. DiCoMEF relies on the controller as a central hub to facilitate collaboration among members of the collaborative group. Of course, this might be considered as a bottleneck, since the controller can be overloaded with lots of tasks. Indeed, DiCoMEF provides a technical framework on top of which different communication strategies can be employed using method engineering techniques (e.g., delegation mechanisms, pooling). For example, a token can be used and whoever has a token is a controller, who can modify a (meta-)model and propagates changes. DiCoMEF framework can also support a hierarchical collaborative modeling.

Formalization of model: This PhD thesis formalizes EMF/Ecore models, meta-models, and meta-meta-models using the Set theory. Because we believe that most people are familiar with set theory, as a result, it is easy for people to understand and reason about models. It also uses the same Set theory constructs to define the edit operations that adapt (meta-)models. Besides, this work formally defines conflicts between models produced by different tasks using the Set theory.

Uniform history meta-model for both models and meta-models: The Set theory formalization demonstrates that the same Set theory constructs, which are used to define models, are also applied to specify meta-models and meta-meta-models. Hence, the same history meta-model language can be used to describe model and meta-model adaptations. As dis-

cussed in Chapter 5 in Section 5.2.2, DiCoMEF use a single history meta-model to capture edit operations of model and meta-model evolution.

Conflict detection: In DiCoMEF, we define conflicting sets tables to detect syntactic conflicts (see Table 5.2 and Table 5.3). The conflicting table specifies which operations are possibly conflicting, and also shows the severity level of the conflicts. For example, a Set operation conflicts with another Set operation, if both operations modify a property of a same model element and assign different values. This conflict is a soft conflict that can be handled automatically. The DiCoMEF conflict detection tool can be configured to identify conflicts in ordered multi-value elements and unordered multi-value elements. Besides, the DiCoMEF conflict detection tool detects static semantic conflicts using EMF validation framework. The DiCoMEF merge tool visualizes conflicting modifications, it colors conflicts with different colors. The DiCoMEF merge tool lets users chose some/all of conflicting modifications, which are performed by other colleagues, and to apply them on their local (meta-)models for analyzing their effects during merging. Since DiCoMEF framework relies on UUIDs to identify model elements, it cannot match equivalent concepts that are modeled differently and have different UUIDs. Hence, it cannot detect semantic conflicts that could be raised due to equivalent modeling concepts. Indeed, semantic conflicts are difficult to detect, because they most of the time resides in users mind.

Conflict reconciliation: DiCoMEF uses a role-based conflict resolution mechanism, where modification of (meta-)models are managed by the controller. The controller is assumed to be a senior staff in the collaborative ensemble, who has good expertise in modeling and business domain. In DiCoMEF, the controller role is flexible, hence, it can be easily assigned to another user in the collaborative group. The DiCoMEF merge tool provides facilities either to accept or reject modifications. Besides, it provides a (meta-)model editor in order to manually resolve static semantic conflicts. Furthermore, DiCoMEF also provides facilities to annotate change operations with rationale of modifications using multimedia files. Users can consult multimedia files to understand the rationale behind modifications, DiCoMEF plays multimedia files inside the DiCoMEF merge tool.

Model merging: Modifications proposed by the controller have a high priority, as a result, those modifications are always applied to local (meta-)models. But, editors can mark local changes that they want to keep, then the DiCoMEF merge tool applies the selected local modifications after applying the propagated modifications. Users can merge their local modifications with propagates modifications whenever they want, of course, they can also use the DiCoMEF merge tool to evaluate the effects of merging of concurrent modifications. DiCoMEF merge tool creates a copy of local model and performed merging so that users do need to worry about unnecessary modifications on their local (meta-)model. It modifies local (meta-)model only if the user confirms the merge result. During merging, DiCoMEF does not rollback all modifications due to conflicts, rather it only rollbacks delete operations that cause conflicts with a high severity value. As shown in history meta-model (see Figure 5.8), the delete operation contains reverse changes that creates deleted (meta-)model elements and its children. Besides, the cascading changes reestablish all references that are removed due to the deletion of the element. However, DiCoMEF merge tool does not provide facilities to select two or more (meta-)model elements and merge them into new (meta-)model elements.

Composite operation detection and recovery: DiCoMEF framework recovers and detects refactoring and composite operations from canonized list operations or deltas. This framework is flexible enough to allow users to guide the result based on her/his preferences. Users can remove parts of composite operations detected by the analysis engine as long as the validity of the composite operations are preserved. The analysis rule can add/remove rationale of modifications to/from composite operations based on the user confirmation. It also analyzes dependencies among composite changes and orders them. Users can add their own composite operation patterns in defining their own Jess rules.

DiCoMEF framework has been implemented as an eclipse plugin (54K LOC) and fully supports collaborative metamodeling. The support of instance models is still under implementation. Besides, composite operation recovery and detection tool is also integrated with the DiCoMEF framework. Screenshots and other publications of DiCoMEF can be found

in DiCoMEF site [1]. We evaluated the DiCoMEF framework with master students with regards to the following criteria: (1) the feasibility of collaborative methods and processes with DiCoMEF, (2) the correctness of conflict detection mechanisms (recall and precision), (3) the usability of the merge tool and DiCoMEF framework, (4) measuring user efforts (time) needed to merge concurrently edited meta-models either manually or by using the DiCoMEF merge tool. This preliminary evaluation reveals overall positive results. The results indicated that the collaborative process of DiCoMEF is feasible and that the merge tool is usable (user friendly), correct, and helpful in the resolution of conflicts. Furthermore, the hierarchical support of DiCoMEF framework is underdevelopment.

7.2 Future work

The DiCoMEF framework will be improved based on the results collected from the preliminary evaluation. The workflow of the DiCoMEF framework will incorporate default activities so as to reduce the number of steps required to send change requests and to propagate changes. Besides, the Email based communication mechanism of the DiCoMEF framework will be replaced by a pertinent alternative solution that can facilitate a sharing of large files. We will also integrate the DiCoMEF framework with the recent versions of Eclipse. Besides, we will improve the usability of the merge tool by displaying high-level composite operations (i.e., refactoring operations) so that a user can easily understand intentions conflicting changes. Besides, we will study the use of ontology that could improve the merge tool by identifying equivalent model elements. The DiCoMEF framework will also provide a better integration with the graphical modeling framework. Besides, the conflict detection and merging will also have a graphical support.

Different method engineering techniques and strategies (e.g., delegation mechanisms, pooling) will be studied to improve the scalability of the DiCoMEF framework. For instance, the controller can delegate part of his/her tasks to others to speedup the collaboration

[1] https://sites.google.com/site/dicomef

process. In addition, the collaborative modeling and hierarchical (meta-)modeling will be fully implemented.

The composite operation detection and recovering tool will be fully implemented and integrated with the DiCoMEF framework. Besides, we will study how the user can specify composite operations by-examples [Brosch et al., 2009a,b], such that the system will use these examples to (semi-)automatically generate Jess rules. Moreover, the generation of model migration instructions from composite change operations will be studied. Of course, we will relay on the Edapt [2] framework in order to generate the model migration instructions.

The evaluation of the DiCoMEF framework will be conducted based on the objectives presented in Chapter 6 Section 6.1. In addition, we will also extend the evaluation to assess the scalability of the DiCoMEF framework in terms of a team size (i.e., 5, 10, or 20 members). This evaluation will carefully select participants from different representative groups such as a junior group (i.e., students) and a senior group (i.e., researchers and industrial (meta-)modelers). Moreover, big models with hundreds and thousands of model elements will also be used to study the scalability of the DiCoMEF framework. Of course, we will automate the validation process of conflict detection mechanism. Specifically, we randomly adapt (meta-)model in parallel, and evaluate the accuracy of conflict detection mechanism by the measures precision and recall [Olson and Delen, 2008]. For instance, Ecore Mutator [3] can be used to randomly mutate Ecore models.

The accuracy of composite operation detection and recovery mechanism of DiCoMEF will be evaluated by the measures precision and recall. For this experiment, we will use the benchmarks presented in [Langer et al., 2013]. In addition, the usability and usefulness of the interactive and iterative recovering and detection process will be evaluated by the participants of the experiment. Moreover, the validity of the generated model migration instructions will also be evaluated by the participants.

[2] https://www.eclipse.org/edapt/

[3] https://code.google.com/a/eclipselabs.org/p/ecore-mutator/

References

Al-Ajlan A. The comparison between forward and backward chaining. *International Journal of Machine Learning and Computing*, 5(2):106, 2015.

Alanen M. and Porres I. Difference and union of models. In Stevens P., Whittle J., and Booch G., editors, *«UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications*, volume 2863 of *Lecture Notes in Computer Science*, pages 2–17. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20243-1. doi: 10.1007/978-3-540-45221-8_2. URL http://dx.doi.org/10.1007/978-3-540-45221-8_2.

Altmanninger K., Kappel G., Kusel A., Retschitzegger W., Seidl M., Schwinger W., and Wimmer M. AMOR - Towards Adaptable Model Versioning, 2008. URL http://publik.tuwien.ac.at/files/PubDat_175517.pdf.

Altmanninger K., Seidl M., and Wimmer M. A Survey on Model Versioning Approaches. Technical report, Johannes Kepler University Linz, 2009. URL http://smover.tk.uni-linz.ac.at/docs/IJWIS09_paper_Altmanninger.pdf.

Altmanninger K., Schwinger W., and Kotsis G. Semantics for accurate conflict detection in smover: Specification, detection and presentation by example. *IJEIS*, 6(1):68–84, 2010. doi: 10.4018/jeis.2010120206. URL http://dx.doi.org/10.4018/jeis.2010120206.

Amrani M., Dingel J., Lambers L., Lúcio L., Salay R., Selim G., Syriani E., and Wimmer M. Towards a model transformation intent catalog. In *Proceedings of the First Workshop on the Analysis of Model Transformations*, pages 3–8. ACM, 2012.

Arendt T., Biermann E., Jurack S., Krause C., and Taentzer G. Henshin: Advanced concepts and tools for in-place emf model transformations. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Part I*, MODELS'10, pages 121–135, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16144-8, 978-3-642-16144-5. URL http://dl.acm.org/citation.cfm?id=1926458.1926471.

Armbrust M., Fox A., Griffith R., Joseph A. D., Katz R., Konwinski A., Lee G., Patterson D., Rabkin A., Stoica I., and Zaharia M. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010. ISSN 0001-0782. doi: 10.1145/1721654.1721672. URL http://doi.acm.org/10.1145/1721654.1721672.

Asklund U. *Identifying Conflicts During Structural Merge*. Department of Computer Science, Lund University, Lund Institute of Technology, Lund, Sweden. Lund University, Department of Computer Science, 1994a. URL http://books.google.be/books?id=emauMwAACAAJ.

Asklund U. Identifying conflicts during structural merge. In *Proceedings of the Nordic Workshop Programming Environment Research*, pages 231–242, 1994b.

Atkinson C. and Kühne T. Rearchitecting the uml infrastructure. *ACM Trans. Model. Comput. Simul.*, 12(4): 290–321, October 2002. ISSN 1049-3301. doi: 10.1145/643120.643123. URL http://doi.acm.org/10.1145/643120.643123.

Atkinson C. and Kühne T. The essence of multilevel metamodeling. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, «UML» '01, pages 19–33. Springer-Verlag, London, UK, UK, 2001. ISBN 3-540-42667-1. URL http://dl.acm.org/citation.cfm?id=647245.719475.

Badr R. D. T.-B. A., N. A conflict resolution control architecture for self-adaptive. In *Proceedings of International Workshop on Architecting Dependable Systems WADS 2002 (ICSE 2002), Orlando, Florida*, 2002.

Bani-Salameh H. A. *A Social Collaborative Distributed Software Development Environment*. PhD thesis, Moscow, ID, USA, 2011. AAI3486380.

Barrett S. C. *BLENDING STATE DIFFERENCES AND CHANGE OPERATIONS FOR METAMODEL INDEPENDENT MERGING OF SOFTWARE MODELS*. Ph.d. dissertation, Concordia University, Montréal, Québec, Canada, April 2011. URL http://spectrum.library.concordia.ca/7368/1/Barrett_PhD_S2011.pdf.

Barteit C., Molter G., and Schumann T. A model repository for collaborative modeling with the jazz development platform. In *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, pages 1–10, Jan 2009. doi: 10.1109/HICSS.2009.23.

Bartelt C. Consistence preserving model merge in collaborative development processes. In *Proceedings of the 2008 International Workshop on Comparison and Versioning of Software Models*, CVSM '08, pages 13–18, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-045-6. doi: 10.1145/1370152.1370157. URL http://doi.acm.org/10.1145/1370152.1370157.

Basciani F., Di Rocco J., Di Ruscio D., Di Salle A., Iovino L., and Pierantonio A. Mdeforge: an extensible web-based modeling platform*. *CloudMDE 2014*, page 66, 2014.

Belaunde M., Casanave C., DSouza D., Duddy K., El Kaim W., Kennedy A., Frank W., Frankel D., Hauch R., Hendryx S., et al. Mda guide version 1.0. 2003. URL http://www.omg.org/news/meetings/workshops/UML_2003_Manual/00-2_MDA_Guide_v1.0.1.pdf.

Benmouffok L., Busca J.-M., Marquès J. M., Shapiro M., Sutra P., and Tsoukalas G. Telex: A Semantic Platform for Cooperative Application Development. In *Conf. Française sur les Systémes d'Exploitation (CFSE)*, Toulouse, France, September 2009. URL papers/Telex-CFSE-2009.pdf.

Bergmann G., Horváth Á., Ráth I., and Varró D. Incremental evaluation of model queries over EMF models: A tutorial on emf-incquery. In France R. B., Küster J. M., Bordbar B., and Paige R. F., editors, *Modelling Foundations and Applications - 7th European Conference, ECMFA 2011, Birmingham, UK, June 6 - 9, 2011 Proceedings*, volume 6698 of *Lecture Notes in Computer Science*, pages 389–390. Springer, 2011. ISBN 978-3-642-21469-1. doi: 10.1007/978-3-642-21470-7_32. URL http://dx.doi.org/10.1007/978-3-642-21470-7_32.

Bézivin J. On the unification power of models. *Software and System Modeling*, 4(2):171–188, 2005.

Bézivin J. In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2): 21–24, 2004.

Bézivin J. and Gerbé O. Towards a precise definition of the omg/mda framework. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, ASE'01, Washington, DC, USA, 2001. IEEE Computer Society. URL http://dl.acm.org/citation.cfm?id=872023.872565.

Bizer C., Heath T., and Berners-Lee T. Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227, 2009.

Black A. P., Nierstrasz O., Ducasse S., and Pollet D. *Pharo by example*. Lulu. com, 2010.

Blackburn M. R. What's model driven engineering (mde) and how can it impact process, people, tools and productivity. Technical report, 2008.

Blanc X., Mounier I., Mougenot A., and Mens T. Detecting model inconsistency through operation-based model construction. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, pages 511–520. IEEE, 2008.

Blanc X., Mougenot A., Mounier I., and Mens T. Incremental detection of model inconsistencies based on model operations. In Eck P., Gordijn J., and Wieringa R., editors, *Advanced Information Systems Engineering*, volume 5565 of *Lecture Notes in Computer Science*, pages 32–46. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-02143-5.

Booch G., Brown A. W., Iyengar S., Rumbaugh J., and Selic B. An MDA Manifesto, May 2004. URL http://www.bptrends.com/publicationfiles/05%2D04%20COL%20IBM%20Manifesto%20%2D%20Frankel%20%2D3%2Epdf.

Borges M. R. and Pino J. A. Awareness mechanisms for coordination activities in asynchronous cscw. In *In Proceedings of the 9th Workshop on Information Technologies and Systems*, 1999.

Borghoff U. M. and Schlichter J. Computer-supported cooperative work: Introduction to distributed applications. 2000.

Boukhebouze M., Koshima A., Thiran P., and Englebert V. Comparative analysis of collaborative approaches for UsiXML meta-models evolution. In *1st International USer Interface eXtensible Markup Language workshop, In the frame of the EICS 2010 conference*, pages 9–14, France, 2010. Thales Research and Technology France. ISBN 978-2-9536757-0-2. URL http://itea.defimedia.be/sites/default/files/Proceedings.zip.

Brambilla M., Cabot J., and Wimmer M. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012.

Bressen T. *Consensus decision making*. Berrett-Koehler Publishers, San Francisco, CA, 2007.

Brinkkemper S., Saeki M., and Harmsen F. Meta-modelling based assembly techniques for situational method engineering. *Information Systems*, 24(3):209–228, 1999.

Brooks F. P., Jr. No silver bullet essence and accidents of software engineering. *Computer*, 20(4):10–19, April 1987. ISSN 0018-9162. doi: 10.1109/MC.1987.1663532. URL http://dx.doi.org/10.1109/MC.1987.1663532.

Brosch P. Improving conflict resolution in model versioning systems. In *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 355–358, May 2009. doi: 10.1109/ICSE-COMPANION.2009.5071020.

Brosch P., Langer P., Seidl M., Wieland K., Wimmer M., Kappel G., Retschitzegger W., and Schwinger W. An example is worth a thousand words: Composite operation modeling by-example. In *Model Driven Engineering Languages and Systems*, pages 271–285. Springer, 2009a.

Brosch P., Langer P., Seidl M., and Wimmer M. Towards end-user adaptable model versioning: The by-example operation recorder. In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, CVSM '09, pages 55–60, Washington, DC, USA, 2009b. IEEE Computer Society. ISBN 978-1-4244-3714-6. doi: 10.1109/CVSM.2009.5071723. URL http://dx.doi.org/10.1109/CVSM.2009.5071723.

Bruneliere H., Cabot J., and Jouault F. Combining model-driven engineering and cloud computing. In *Modeling, Design, and Analysis for the Service Cloud-MDA4ServiceCloud'10: Workshop's 4th edition (co-located with the 6th European Conference on Modelling Foundations and Applications-ECMFA 2010)*, 2010.

Buffenbarger J. Syntactic software merging. In *Selected papers from the ICSE SCM-4 and SCM-5 Workshops, on Software Configuration Management*, pages 153–172, London, UK, 1995. Springer-Verlag. ISBN 3-540-60578-9. URL http://portal.acm.org/citation.cfm?id=647174.716256.

Cánovas Izquierdo J. L. and Cabot J. Community-driven language development. In *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*, pages 29–35. IEEE, 2012.

Carstensen P. and Schmidt K. Computer supported cooperative work: new challenges to systems design, 1999. *Handbook of Human Factors, Kenji Itoh, Tokio*.

Chacon S. *Pro Git*. Apress, Berkely, CA, USA, 1st edition, 2009. ISBN 1430218339, 9781430218333.

Chawathe S. S. and Garcia-Molina H. Meaningful change detection in structured data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 26–37, New York, NY, USA, 1997. ACM. ISBN 0-89791-911-4. doi: 10.1145/253260.253266. URL http://doi.acm.org/10.1145/253260.253266.

Cicchetti A. *Difference Representation and Conflict Management in Model-Driven Engineering*. PhD thesis, Università di L'Aquila, 2008.

Cicchetti A., Ruscio D. D., and Pierantonio A. A metamodel independent approach to difference representation. *Journal of Object Technology*, 6(9):165–185, October 2007. ISSN 1660-1769. doi: 10.5381/jot.2007.6.9.a9. URL http://www.jot.fm/contents/issue_2007_10/paper9.html. TOOLS EUROPE 2007 — Objects, Models, Components, Patterns.

Cicchetti A., Ruscio D., and Pierantonio A. Managing model conflicts in distributed development. In *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems*, MoDELS '08, pages 311–325, Berlin, Heidelberg, 2008a. Springer-Verlag. ISBN 978-3-540-87874-2. doi: 10.1007/978-3-540-87875-9_23. URL http://dx.doi.org/10.1007/978-3-540-87875-9_23.

Cicchetti A., Ruscio D. D., Eramo R., and Pierantonio A. Automating co-evolution in model-driven engineering. In *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, EDOC '08, pages 222–231, Washington, DC, USA, 2008b. IEEE Computer Society. ISBN 978-0-7695-3373-5. doi: 10.1109/EDOC.2008.44. URL http://dx.doi.org/10.1109/EDOC.2008.44.

Conradi R. and Westfechtel B. Towards a uniform version model for software configuration management. In *Proceedings of the SCM-7 Workshop on System Configuration Management*, ICSE '97, pages 1–17, London, UK, 1997. Springer-Verlag. ISBN 3-540-63014-7. URL http://portal.acm.org/citation.cfm?id=647176.716423.

Conradi R. and Westfechtel B. Version models for software configuration management. *ACM Comput. Surv.*, 30:232–282, June 1998. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/280277.280280. URL http://doi.acm.org/10.1145/280277.280280.

Constantin C., Englebert V., and Thiran P. A reconciliation framework to support cooperative work with DSM. In *Proceedings of the First International Workshop on Domain Engineering held in conjunction with CAiSE'09 Conference, collection CEUR-WS.org*, volume 457, 2009.

Cormen T. H., Leiserson C. E., Rivest R. L., and Stein C. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844, 9780262033848.

Czarnecki K. and Helsen S. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45(3):621–645, July 2006. ISSN 0018-8670. doi: 10.1147/sj.453.0621. URL http://dx.doi.org/10.1147/sj.453.0621.

De Lucia A., Fasano F., Scanniello G., and Tortora G. Enhancing collaborative synchronous uml modelling with fine-grained versioning of software artefacts. *J. Vis. Lang. Comput.*, 18:492–503, October 2007. ISSN 1045-926X. doi: 10.1016/j.jvlc.2007.08.005. URL http://portal.acm.org/citation.cfm?id=1314708.1314891.

Demeyer S., Tichelaar S., and Ducasse S. FAMIX 2.1- the FAMOOS information exchange model, 2001.

Demeyer S., Ducasse S., and Nierstrasz O. Finding refactorings via change metrics. *SIGPLAN Not.*, 35(10):166–177, October 2000. ISSN 0362-1340. doi: 10.1145/354222.353183. URL http://doi.acm.org/10.1145/354222.353183.

Dewan P. and Hegde R. Semi-synchronous conflict detection and resolution in asynchronous software development. In Harper R. and Gutwin C., editors, *ECSCW*, pages 159–178. Springer, 2007. ISBN 978-1-84800-030-8.

Dewan P. and Riedl J. Toward computer-supported concurrent software engineering. *Computer*, 26:17–27, January 1993. ISSN 0018-9162. doi: 10.1109/2.179149. URL http://portal.acm.org/citation.cfm?id=165312.165318.

Di Rocco J., Di Ruscio D., Iovino L., and Pierantonio A. Collaborative repositories in model-driven engineering [software technology]. *Software, IEEE*, 32(3):28–34, May 2015. ISSN 0740-7459. doi: 10.1109/MS.2015.61.

Dias M., Cassou D., and Ducasse S. Representing code history with development environment events. *CoRR*, abs/1309.4334, 2013. URL http://arxiv.org/abs/1309.4334.

Dig D., Comertoglu C., Marinov D., and Johnson R. Automated detection of refactorings in evolving components. In *Proceedings of the 20th European Conference on Object-Oriented Programming*, ECOOP'06, pages 404–428, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-35726-2, 978-3-540-35726-1. doi: 10.1007/11785477_24. URL http://dx.doi.org/10.1007/11785477_24.

Dirix M. Awareness in computer-supported collaborative modelling. application to genmymodel. 2013.

Dourish P. and Bellotti V. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM Conference on Computer-supported Cooperative Work*, CSCW '92, pages 107–114, New York, NY, USA, 1992. ACM. ISBN 0-89791-542-9. doi: 10.1145/143457.143468. URL http://doi.acm.org/10.1145/143457.143468.

Ebraert P., Vallejos J., Costanza P., Van Paesschen E., and D'Hondt T. Change-oriented software engineering. In *Proceedings of the 2007 international conference on Dynamic languages: in conjunction with the 15th International Smalltalk Joint Conference 2007*, pages 3–24. ACM, 2007.

Edwards W. K. Policies and roles in collaborative applications. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, CSCW '96, pages 11–20, New York, NY, USA, 1996. ACM. ISBN 0-89791-765-0. doi: http://doi.acm.org/10.1145/240080.240175. URL http://doi.acm.org/10.1145/240080.240175.

Edwards W. K. Flexible conflict detection and management in collaborative applications. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, UIST '97, pages 139–148, New York, NY, USA, 1997. ACM. ISBN 0-89791-881-9. doi: 10.1145/263407.263533. URL http://doi.acm.org/10.1145/263407.263533.

Ellis C. and Wainer J. 10 groupware and computer supported cooperative work. *Multiagent Systems: a modern approach to distributed artificial intelligence*, page 425, 1999.

Elzeiny A., Elfetouh A. A., and Riad A. Cloud storage: A survey. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, 2, July – August 2013.

Emami H. and Narimanifar K. A comparison framework for conflict detection and resolution multi agent modeling methods in air traffic management. *The International Journal of Information Technology, Control and Automation (IJITCA)*, 2(4):51–64, October 2012. doi: 10.5121/ijitca.2012.2405. URL http://airccse.org/journal/ijitca/papers/2412ijitca05.pdf.

Endriss U. Monotonic concession protocols for multilateral negotiation. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 392–399, New York, NY, USA, 2006. ACM. ISBN 1-59593-303-4. doi: 10.1145/1160633.1160702. URL http://doi.acm.org/10.1145/1160633.1160702.

Englebert V. and Heymans P. Towards more extensible metaCASE tools. In Krogstie J., Opdhal A., and Sindre G., editors, *International Conference on Advanced Information Systems Engineering (CAiSE'07)*, number 4495 in LNCS, pages 454–468, 2007.

Estublier J. Software configuration management: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 279–289, New York, NY, USA, 2000. ACM. ISBN 1-58113-253-0. doi: 10.1145/336512.336576. URL http://doi.acm.org/10.1145/336512.336576.

Estublier J., Leblang D., Hoek A. v. d., Conradi R., Clemm G., Tichy W., and Wiborg-Weber D. Impact of software engineering research on the practice of software configuration management. *ACM Trans. Softw. Eng. Methodol.*, 14(4):383–430, October 2005. ISSN 1049-331X. doi: 10.1145/1101815.1101817. URL http://doi.acm.org/10.1145/1101815.1101817.

Falleri J.-R., Morandat F., Blanc X., Martinez M., and Montperrus M. Fine-grained and accurate source code differencing. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 313–324. ACM, 2014.

Favre J.-M. Towards a basic theory to model model driven engineering. In *In Workshop on Software Model Engineering, WISME 2004, joint event with UML2004*, 2004.

Favre J.-M. Foundations of meta-pyramids: Languages vs. metamodels – episode ii: Story of thotus the baboon1. In Bezivin J. and Heckel R., editors, *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. URL http://drops.dagstuhl.de/opus/volltexte/2005/21.

Fluri B., Wursch M., PInzger M., and Gall H. C. Change distilling: Tree differencing for fine-grained source code change extraction. *Software Engineering, IEEE Transactions on*, 33(11):725–743, 2007.

Fondement F. *Concrete syntax definition for modeling languages*. PhD thesis, IC, Lausanne, 2007.

Fowler M. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA, 1999. ISBN 0-201-48567-2.

France R. and Rumpe B. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society, 2007.

Frankel D. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, 2003.

Gall H. C., Fluri B., and Pinzger M. Change analysis with evolizer and changedistiller. *IEEE Software*, (1): 26–33, 2009.

Gallardo J., Bravo C., and Redondo M. A. A model-driven development method for collaborative modeling tools. *Journal of Network and Computer Applications*, 35(3):1086–1105, 2012.

Garcés K., Jouault F., Cointe P., and Bézivin J. Managing model adaptation by precise detection of metamodel changes. In *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications*, ECMDA-FA '09, pages 34–49, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-02673-7. doi: 10.1007/978-3-642-02674-4_4. URL http://dx.doi.org/10.1007/978-3-642-02674-4_4.

Georg G. Activity theory and its applications in software engineering and technology. Technical report, 2011.

Gilman E., Sánchez I., Saloranta T., and Riekki J. Reasoning for smart space application: comparing three reasoning engines clips, jess and win-prolog. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1340–1345. IEEE, 2010.

Gîrba T., Favre J.-M., and Ducasse S. Using meta-model transformation to model software evolution. *Electron. Notes Theor. Comput. Sci.*, 137:57–64, September 2005. ISSN 1571-0661.

Goldberg D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. ISBN 0201157675.

Gomes C., Barroca B., and Amaral V. Classification of model transformation tools: Pattern matching techniques. In Dingel J., Schulte W., Ramos I., Abrahão S., and Insfran E., editors, *Model-Driven Engineering Languages and Systems*, volume 8767 of *Lecture Notes in Computer Science*, pages 619–635. Springer International Publishing, 2014. ISBN 978-3-319-11652-5. doi: 10.1007/978-3-319-11653-2_38. URL http://dx.doi.org/10.1007/978-3-319-11653-2_38.

Gómez V. U., Ducasse S., and D'Hondt T. Visually supporting source code changes integration: the torch dashboard. In *Reverse Engineering (WCRE), 2010 17th Working Conference on*, pages 55–64. IEEE, 2010.

Gómez V. U., Ducasse S., and D'Hondt T. Ring: a unifying meta-model and infrastructure for smalltalk source code analysis tools. *Computer Languages, Systems & Structures*, 38(1):44–60, 2012.

Gonzalez-Perez C. Tools for an extended object modelling environment. In *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems*, ICECCS '05, pages 20–23, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2284-X. doi: 10.1109/ICECCS.2005.80. URL http://dx.doi.org/10.1109/ICECCS.2005.80.

Gonzalez-Perez C. and Henderson-Sellers B. A representation-theoretical analysis of the omg modelling suite. In *Proceedings of the 2005 Conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the Fourth SoMeT_W05*, pages 252–262, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press. ISBN 1-58603-556-8. URL http://dl.acm.org/citation.cfm?id=1563296.1563320.

Gonzalez-Perez C. and Henderson-Sellers B. *Metamodelling for Software Engineering*. John Wiley, New York, 2008. ISBN 9780470030363.

Grune D. Concurrent versions system, a method for independent cooperation. Technical report, 1986. URL ftp://progwww.vub.ac.be/education/EMOOSE/ReuseSlides/merge-papers/Grune198X-cvs.pdf.

Grune D. et al. Concurrent versions system (cvs). Programa de Computador, June 1989. URL http://cvshome.org.

Gruschko B., Kolovos D. S., and Paige R. F. Towards synchronizing models with evolving metamodels. In *Workshop on Model-Driven Software Evolution at CSMR 2007*, 2007.

Gupta D. and Dwivedi R. Method configuration from situational method engineering. *SIGSOFT Softw. Eng. Notes*, 37(3):1–11, May 2012. ISSN 0163-5948. doi: 10.1145/180921.2180934. URL http://doi.acm.org/10.1145/180921.2180934.

Gutwin C., Penner R., and Schneider K. Group awareness in distributed software development. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04, pages 72–81, New York, NY, USA, 2004. ACM. ISBN 1-58113-810-5. doi: 10.1145/1031607.1031621. URL http://doi.acm.org/10.1145/1031607.1031621.

Hamadache K. and Lancieri L. Role-based collaboration extended to pervasive computing. In *Intelligent Networking and Collaborative Systems, 2009. INCOS '09. International Conference on*, pages 9–15, Nov 2009. doi: 10.1109/INCOS.2009.45.

Han J. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 1558609016.

Harel D. and Rumpe B. Meaningful modeling: What's the semantics of "semantics"? *Computer*, 37(10):64–72, October 2004. ISSN 0018-9162. doi: 10.1109/MC.2004.172. URL http://dx.doi.org/10.1109/MC.2004.172.

Henderson-Sellers B. Method engineering: Theory and practice. In Karagiannis D. and Mayr H. C., editors, *Information Systems Technology and its Applications, 5th International Conference ISTA'2006, May 30-31, 2006, Klagenfurt, Austria*, volume 84 of *LNI*, pages 13–23. GI, 2006. ISBN 3-88579-178-1. URL http://subs.emis.de/LNI/Proceedings/Proceedings84/article4300.html.

Herrmannsdoerfer M. Operation-based versioning of metamodels with COPE. In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, CVSM '09, pages 49–54, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3714-6. doi: http://dx.doi.org/10.1109/CVSM.2009.5071722. URL http://dx.doi.org/10.1109/CVSM.2009.5071722.

Herrmannsdoerfer M., Benz S., and Juergens E. COPE: A Language for the Coupled Evolution of Metamodels and Models . In *Proc. of the 1st International Workshop on Model Co-Evolution and Consistency Management.* ACM, 2008.

Hill E. F. *Jess in Action: Java Rule-Based Systems*. Manning Publications Co., Greenwich, CT, USA, 2003. ISBN 1930110898.

Hilley D. Cloud computing: A taxonomy of platform and infrastructure-level offerings. *Georgia Institute of Technology, Tech. Rep*, 2009.

Hohpe G. and Woolf B. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 0321200683.

Holmes T., Zdun U., and Dustdar S. Automating the management and versioning of service models at runtime to support service monitoring. In *Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International*, pages 211–218. IEEE, 2012.

Holt R. C., Schürr A., Sim S. E., and Winter A. Gxl: A graph-based standard exchange format for reengineering, 2006.

Ignat C.-L., Oster G., Molli P., Cart M., Ferrie J., Kermarrec A.-M., Sutra P., Shapiro M., Benmouffok L., Busca J.-M., and Guerraoui R. A comparison of optimistic approaches to collaborative editing of wiki pages. In *Proceedings of the 2007 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 474–483, Washington, DC, USA, 2007a. IEEE Computer Society. ISBN 978-1-4244-1318-8. doi: 10.1109/COLCOM.2007.4553878. URL http://portal.acm.org/citation.cfm?id=1545009.1545344.

Ignat C.-L. and Norrie M. C. Flexible Collaboration over XML Documents. In *Proceedings of the International Conference on Cooperative Design, Visualization and Engineering (CDVE'06)*, pages 267–274, Mallorca, Spain, September 2006. ISBN 3-540-44494-7.

Ignat C.-L. and Norrie M. C. Operation-based versus State-based Merging in Asynchronous Graphical Collaborative Editing. *Sixth International Workshop on Collaborative Editing Systems, CSCW'04, IEEE Distributed Systems online*, November 2004. ISSN 1541-4922.

Ignat C.-L., Oster G., Molli P., and Skaf-Molli H. Gasper: A collaborative writing mode for avoiding blind modifications. Research Report RR-6204, LORIA – INRIA Lorraine, May 2007b. URL http://hal.inria.fr/inria-00150013/en/.

Iqbal A., Ureche O., Hausenblas M., and Tummarello G. Ld2sd: Linked data driven software development. In *In 21st International Conference on Software Engineering and Knowledge Engineering (SEKE 09*, 2009.

Jech T. *Set theory*. Springer Science & Business Media, 2013.

Johansen R. *Groupware: Computer support for business teams*. The Free Press, 1988.

Jouault F. and Bézivin J. Km3: A dsl for metamodel specification. In Gorrieri R. and Wehrheim H., editors, *FMOODS*, volume 4037 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 2006. ISBN 3-540-34893-X.

Jouault F. and Kurtev I. Transforming models with atl. In *Proceedings of the 2005 International Conference on Satellite Events at the MoDELS*, MoDELS'05, pages 128–138, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-31780-5, 978-3-540-31780-7. doi: 10.1007/11663430_14. URL http://dx.doi.org/10.1007/11663430_14.

Jouault F., Allilaire F., Bézivin J., and Kurtev I. Atl: A model transformation tool. *Science of computer programming*, 72(1):31–39, 2008.

Ju J., Wu J., Fu J., Lin Z., and Zhang J. A survey on cloud storage. *Journal of Computers*, 6(8):1764–1771, 2011.

Kelly S. Case tool support for co-operative work in information system design. In Rolland C., Chen Y., and Fang M., editors, *Information Systems in the WWW Environment*, volume 115 of *IFIP Conference Proceedings*, pages 49–69. Chapman & Hall, 1998. ISBN 0-412-82980-0.

Kelly S. and Tolvanen J.-P. *Domain-Specific Modeling: Enabling full code generation*. Wiley-IEEE Computer Society Pr, 2008. ISBN 978-0-470-03666-2.

Kent S. Model driven engineering. In *Proceedings of the Third International Conference on Integrated Formal Methods*, IFM '02, pages 286–298, London, UK, 2002. Springer-Verlag. ISBN 3-540-43703-7.

Kessentini M., Werda W., Langer P., and Wimmer M. Search-based model merging. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1453–1460, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1963-8. doi: 10.1145/2463372.2463553. URL http://doi.acm.org/10.1145/2463372.2463553.

Klein M. Supporting conflict resolution in cooperative design systems. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(6):1379–1390, Nov 1991. ISSN 0018-9472. doi: 10.1109/21.135683.

Klein M. and Lu S. C.-Y. Conflict resolution in cooperative design. *Artificial Intelligence in Engineering*, 4(4): 168 – 180, 1989. ISSN 0954-1810. doi: http://dx.doi.org/10.1016/0954-1810(89)90013-7. URL http://www.sciencedirect.com/science/article/pii/0954181089900137.

Kleppe A. G., Warmer J., and Bast W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 032119442X.

Koegel M. and Helming J. EMFStore: a model repository for emf models. In Kramer J., Bishop J., Devanbu P. T., and Uchitel S., editors, *ICSE (2)*, pages 307–308. ACM, 2010. ISBN 978-1-60558-719-6.

Koegel M., Helming J., and Seyboth S. Operation-based conflict detection and resolution. In *JProceedings of the Joint ModSE-MCCM Workshop on Models and Evolution*, ModSE-MCCM '09, Denver, USA, 2009a. URL http://wwwbruegge.in.tum.de/static/publications/pdf/205/Paper3.pdf.

Koegel M., Helming J., and Seyboth S. Operation-based conflict detection and resolution. In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, CVSM '09, pages 43–48, Washington, DC, USA, 2009b. IEEE Computer Society. ISBN 978-1-4244-3714-6. doi: http://dx.doi.org/10.1109/CVSM.2009.5071721. URL http://dx.doi.org/10.1109/CVSM.2009.5071721.

Koegel M., Herrmannsdoerfer M., Helming J., and Li Y. State-based vs. operation-based change tracking. In *proceedings of MODELS'09 MoDSE-MCCM Workshop*, Denver, USA, 2009, 2009c. URL http://wwwbruegge.in.tum.de/static/publications/pdf/205/Paper3.pdf.

Koegel M., Herrmannsdoerfer M., von Wesendonk O., and Helming J. Operation-based conflict detection. In *Proceedings of the 1st International Workshop on Model Comparison in Practice*, IWMCP '10, pages 21–30, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-960-2. doi: http://doi.acm.org/10.1145/1826147.1826154. URL http://doi.acm.org/10.1145/1826147.1826154.

Kolovos D. S., Paige R. F., and Polack F. A. Model comparison: A foundation for model composition and model transformation testing. In *Proceedings of the 2006 International Workshop on Global Integrated Model Management*, GaMMa '06, pages 13–20, New York, NY, USA, 2006. ACM. ISBN 1-59593-410-3. doi: 10.1145/1138304.1138308. URL http://doi.acm.org/10.1145/1138304.1138308.

Kolovos D. S., Paige R. F., and Polack F. A. The epsilon transformation language. In *Proceedings of the 1st International Conference on Theory and Practice of Model Transformations*, ICMT '08, pages 46–60, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-69926-2. doi: 10.1007/978-3-540-69927-9_4. URL http://dx.doi.org/10.1007/978-3-540-69927-9_4.

Kolovos D. S., Di Ruscio D., Pierantonio A., and Paige R. F. Different models for model matching: An analysis of approaches to support model differencing. In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, CVSM '09, pages 1–6, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3714-6. doi: 10.1109/CVSM.2009.5071714. URL http://dx.doi.org/10.1109/CVSM.2009.5071714.

Koshima A. and Englebert V. Rucord: Rule-based composite operation recovering and detection to support cooperative edition of (meta)models. In Hammoudi S., Ferreira Pires L., Desfray P., and Filipe J., editors, *MODELSWARD 2015 - Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development, ESEO, Angers, Loire Valley, France, 9-11 February, 2015.*, pages 585–591. SciTePress, 2015a. ISBN 978-989-758-083-3. doi: 10.5220/0005339305850591. URL http://dx.doi.org/10.5220/0005339305850591.

Koshima A. and Englebert V. Collaborative editing of emf/ecore metamodels and models: Conflict detection, reconciliation, and merging in dicomef. In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development – (MODELSWARD 2014)*, pages 55 – 66, Lisbon, Portugal, 2014.

Koshima A., Englebert V., and Thiran P. Distributed collaborative model editing framework for domain specific modeling tools. In *In Proceedings of the 2011 6th IEEE International Conference on Global Software Engineering*, ICGSE '11, Marina Congress Centre, Helsinki, Finland, 2011. IEEE Computer Society.

Koshima A. A. and Englebert V. Collaborative editing of emf/ecore meta-models and models: Conflict detection, reconciliation, and merging in dicomef. *Science of Computer Programming*, 113, Part 1:3 – 28, 2015b. ISSN 0167-6423. doi: http://dx.doi.org/10.1016/j.scico.2015.07.004. URL http://www.sciencedirect.com/science/article/pii/S0167642315001380. Model Driven Development (Selected & extended papers from {MODELSWARD} 2014).

Koshima A. A., Englebert V., and Thiran P. A reconciliation framework to support cooperative work with dsm. In Reinhartz-Berger I., Sturm A., Clark T., Cohen S., and Bettin J., editors, *Domain Engineering*, pages 239–259. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-36653-6. doi: 10.1007/978-3-642-36654-3_10. URL http://dx.doi.org/10.1007/978-3-642-36654-3_10.

Kühn H. and Murzek M. Interoperability issues in metamodelling platforms. In Konstantas D., Bourrières J.-P., Léonard M., and Boudjlida N., editors, *Interoperability of Enterprise Software and Applications*, pages 215–226. Springer London, 2006. ISBN 978-1-84628-151-8. doi: 10.1007/1-84628-152-0_20. URL http://dx.doi.org/10.1007/1-84628-152-0_20.

Kühne T. Matters of (meta-)modeling. *Software and System Modeling*, 5(4):369–385, 2006.

Kuutti K. and Arvonen T. Identifying potential CSCW applications by means of activity theory concepts: A case example. In Mantel M. and Baecker R., editors, *CSCW '92, Proceedings of the Conference on Computer Supported Cooperative Work, Toronto, Canada, October 31 - November 4, 1992*, pages 233–240. ACM, 1992. ISBN 0-89791-542-9. doi: 10.1145/143457.150955. URL http://doi.acm.org/10.1145/143457.150955.

Lamport L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21:558–565, July 1978. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/359545.359563. URL http://doi.acm.org/10.1145/359545.359563.

Langer P. *Adaptable Model Versioning based on Model Transformation By Demonstration*. Ph.d. dissertation, Faculty of Informatics, Vienna University of Technology, December 2011. URL http://publik.tuwien.ac.at/files/PubDat_203931.pdf.

Langer P. and Wimmer M. A benchmark for conflict detection components of model versioning systems. *Softwaretechnik-Trends*, 2013.

Langer P., Wimmer M., Brosch P., Herrmannsdörfer M., Seidl M., Wieland K., and Kappel G. A posteriori operation detection in evolving software models. *J. Syst. Softw.*, 86(2):551–566, February 2013. ISSN 0164-1212. doi: 10.1016/j.jss.2012.09.037. URL http://dx.doi.org/10.1016/j.jss.2012.09.037.

Lin Y., Gray J., and Jouault F. DSMDiff: A Differentiation Tool for Domain-Specific Models. 2007.

Lippe E. and van Oosterom N. Operation-based merging. In *Proceedings of the fifth ACM SIGSOFT symposium on Software development environments*, SDE 5, pages 78–87, New York, NY, USA, 1992a. ACM. ISBN 0-89791-554-2. doi: http://doi.acm.org/10.1145/142868.143753. URL http://doi.acm.org/10.1145/142868.143753.

Lippe E. and van Oosterom N. Operation-based merging. *SIGSOFT Softw. Eng. Notes*, 17:78–87, November 1992b. ISSN 0163-5948. doi: http://doi.acm.org/10.1145/142882.143753. URL http://doi.acm.org/10.1145/142882.143753.

Livshits B. and Zimmermann T. Dynamine: finding common error patterns by mining software revision histories. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 296–305. ACM, 2005.

Lúcio L., Amrani M., Dingel J., Lambers L., Salay R., Selim G., Syriani E., and Wimmer M. Model transformation intents and their properties. *Software & Systems Modeling*, pages 1–38, 2014. ISSN 1619-1366. doi: 10.1007/s10270-014-0429-x. URL http://dx.doi.org/10.1007/s10270-014-0429-x.

Magnusson B., Asklund U., and Minör S. Fine-grained revision control for collaborative software development. *SIGSOFT Softw. Eng. Notes*, 18(5):33–41, December 1993a. ISSN 0163-5948. doi: 10.1145/167049.167061. URL http://doi.acm.org/10.1145/167049.167061.

Magnusson B., Asklund U., and Minör S. Fine-grained revision control for collaborative software development. In *Proceedings of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT '93, pages 33–41, New York, NY, USA, 1993b. ACM. ISBN 0-89791-625-5. doi: 10.1145/256428.167061. URL http://doi.acm.org/10.1145/256428.167061.

Mansoor U., Kessentini M., Langer P., Wimmer M., Bechikh S., and Deb" K. Momm: Multi-objective model merging. *Journal of Systems and Software*, 103(0):423 – 439, 2015. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2014.11.043. URL http://www.sciencedirect.com/science/article/pii/S016412121400274X.

Maoz S., Ringert J. O., and Rumpe B. Addiff: semantic differencing for activity diagrams. In Gyimóthy T. and Zeller A., editors, *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13rd European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*, pages 179–189. ACM, 2011. ISBN 978-1-4503-0443-6. doi: 10.1145/2025113.2025140. URL http://doi.acm.org/10.1145/2025113.2025140.

Maróti M., Kecskés T., Kereskényi R., Broll B., Völgyesi P., Jurácz L., Levendoszky T., and Lédeczi Á. Next generation (meta) modeling: Web-and cloud-based collaborative tool infrastructure. *Proceedings of MPM*, page 41, 2014.

Mens T. A state-of-the-art survey on software merging. *IEEE Trans. Softw. Eng.*, 28:449–462, May 2002. ISSN 0098-5589. doi: 10.1109/TSE.2002.1000449. URL http://portal.acm.org/citation.cfm?id=567176.567178.

Mens T. Conditional graph rewriting as a domain-independent formalism for software evolution. In Nagl M., Schürr A., and Münch M., editors, *Applications of Graph Transformations with Industrial Relevance, International Workshop, AGTIVE'99, Kerkrade, The Netherlands, September 1-3, 1999, Proceedings*, volume 1779 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 1999a. ISBN 3-540-67658-9. doi: 10.1007/3-540-45104-8_10. URL http://dx.doi.org/10.1007/3-540-45104-8_10.

Mens T. *A Formal Foundation for Object-Oriented Software Evolution*. PhD thesis, Brussels, Belgium, 1999b.

Mens T. and Gorp P. V. A taxonomy of model transformation. *Electr. Notes Theor. Comput. Sci.*, 152:125–142, 2006.

Mens T. and Taentzer G. Model-driven software refactoring. In *1st Workshop on Refactoring Tools, WRT 2007, in conjunction with 21st European Conference on Object-Oriented Programming, July 30 - August 03, 2007, Berlin, Proceedings*, pages 25–27, 2007. URL http://netfiles.uiuc.edu/dig/RefactoringWorkshop/.

Mens T. and Van Der Straeten R. Incremental resolution of model inconsistencies. In *Recent Trends in Algebraic Development Techniques*, pages 111–126. Springer, 2007.

Mens T., Czarnecki K., and Gorp P. V. A taxonomy of model transformation. In *Proc. Dagstuhl Seminar on "Language Engineering for Model-Driven Software Development". Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl*. Electronic, 2005a.

Mens T., Taentzer G., and Runge O. Detecting structural refactoring conflicts using critical pair analysis. *Electr. Notes Theor. Comput. Sci.*, 127(3):113–128, 2005b. doi: 10.1016/j.entcs.2004.08.038. URL http://dx.doi.org/10.1016/j.entcs.2004.08.038.

Mens T., Van Der Straeten R., and D'Hondt M. Detecting and resolving model inconsistencies using transformation dependency analysis. In *Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems*, MoDELS'06, pages 200–214, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-45772-0, 978-3-540-45772-5. doi: 10.1007/11880240_15. URL http://dx.doi.org/10.1007/11880240_15.

Meyers B. and Vangheluwe H. A framework for evolution of modelling languages. *Sci. Comput. Program.*, 76(12):1223–1246, December 2011. ISSN 0167-6423. doi: 10.1016/j.scico.2011.01.002. URL http://dx.doi.org/10.1016/j.scico.2011.01.002.

Michigan State University. Decision making in groups, Module G, copyright 2007, michigan state university,4-h 1068 group dynamite notebook, michigan state university extension 4-h youth development (1978). http://decision2.org/d/decision-making-in-groups-michigan-state-university-w782/, 2007.

Mills K. L. Computer-supported cooperative work. In *ENCYCLOPEDIA OF LIBRARY AND INFORMATION SCIENCES (2ND EDITION*. Citeseer, 2003.

Minör S. and Magnusson B. A model for semi-(a) synchronous collaborative editing. In *Proceedings of the Third European Conference on Computer-Supported Cooperative Work 13–17 September 1993, Milan, Italy ECSCW'93*, pages 219–231. Springer, 1993.

Mirbel I. and Ralyté J. Situational method engineering: combining assembly-based and roadmap-driven approaches. *Requirements Engineering*, 11(1):58–78, 2006.

Mohamed M. and Romdhani M. Classification of model refactoring approaches. *JOURNAL OF OBJECT TECHNOLOGY*, 8(6), 2009.

Monperrus M., Beugnard A., and Champeau J. A definition of "abstraction level" for metamodels. In *Engineering of Computer Based Systems, 2009. ECBS 2009. 16th Annual IEEE International Conference and Workshop on the*, pages 315–320, 2009. doi: 10.1109/ECBS.2009.41.

Montes J. L. I., Vela F. L. G., and Megías M. G. Supporting social organization modelling in cooperative work using patterns. In *Computer Supported Cooperative Work in Design II*, pages 112–121. Springer, 2006.

Mougenot A., Blanc X., and Gervais M.-P. D-praxis: A peer-to-peer collaborative model editing framework. In *Proceedings of the 9th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, DAIS '09, pages 16–29, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-02163-3. doi: http://dx.doi.org/10.1007/978-3-642-02164-0_2. URL http://dx.doi.org/10.1007/978-3-642-02164-0_2.

Muller P.-A., Fondement F., Baudry B., and Combemale B. Modeling modeling modeling. *Software & Systems Modeling*, 11(3):347–359, 2012.

Munson J. P. and Dewan P. A flexible object merging framework. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, pages 231–242, New York, NY, USA, 1994. ACM. ISBN 0-89791-689-1. doi: 10.1145/192844.193016. URL http://doi.acm.org/10.1145/192844.193016.

Nguyen T. N., Munson E. V., Boyland J. T., and Thao C. An infrastructure for development of object-oriented, multi-level configuration management services. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 215–224, New York, NY, USA, 2005. ACM. ISBN 1-58113-963-2. doi: 10.1145/1062455.1062504. URL http://doi.acm.org/10.1145/1062455.1062504.

Oda T. and Saeki M. Generative technique of version control systems for software diagrams. In *Proceedings of the 21st IEEE International Conference on Software Maintenance*, ICSM '05, pages 515–524, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2368-4. doi: 10.1109/ICSM.2005.49. URL http://dx.doi.org/10.1109/ICSM.2005.49.

Ohst D., Welle M., and Kelter U. Differences between versions of uml diagrams. In *Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-11, pages 227–236, New York, NY, USA, 2003. ACM. ISBN 1-58113-743-5. doi: 10.1145/940071.940102. URL http://doi.acm.org/10.1145/940071.940102.

Oliveira H., Murta L., and Werner C. Odyssey-vcs: A flexible version control system for uml model elements. In *Proceedings of the 12th International Workshop on Software Configuration Management*, SCM '05, pages 1–16, New York, NY, USA, 2005. ACM. ISBN 1-59593-310-7. doi: 10.1145/1109128.1109129. URL http://doi.acm.org/10.1145/1109128.1109129.

Olson D. L. and Delen D. *Advanced data mining techniques*. Springer Science & Business Media, 2008.

OMG. *MOF QVT Final Adopted Specification*. Object Modeling Group, June 2005. URL http://fparreiras/papers/mof_qvt_final.pdf.

OMG. *OMG Unified Modeling Language (OMG UML), Superstructure*. OMG, aug 2011. formal/2011-08-06.

OMG O. M. G. OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. Technical report, November 2007a. URL http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF.

OMG O. M. G. Meta Object Facility(MOF) Specification. http://www.omg.org/spec/MOF/1.4/PDF, April 2002.

OMG O. M. G. Model Driven Architecture (MDA), document number ormsc/2001-07-01. http://www.enterprise-architecture.info/Images/MDA/MDA%20Technical.pdf, July 2001.

OMG X. OMG, XMI mapping specification, v2.1.1, formal/07-12-0, (2007), 2007b.

Omoronyia I. *Sharing awareness during distributed collaborative software development*. Ph.d. dissertation, Department of Computer and Information Sciences, University of Strathclyde, November 2008. URL https://personal.cis.strath.ac.uk/murray.wood/efocswww/papers/InahOmoronyiaThesis.pdf.

Omoronyia I., Ferguson J., Roper M., and Wood M. A 3-dimensional relevance model for collaborative software engineering spaces. In *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pages 204–216. IEEE, 2007.

Ouksel A. M. and Sheth A. Semantic interoperability in global information systems. *SIGMOD Rec.*, 28(1): 5–12, March 1999. ISSN 0163-5808. doi: 10.1145/309844.309849. URL http://doi.acm.org/10.1145/309844.309849.

Penichet V. M. R., Paternó F., Gallud J. A., and Lozano M. D. Collaborative social structures and task modelling integration. In *Proceedings of the 13th International Conference on Interactive Systems: Design, Specification, and Verification*, DSVIS'06, pages 67–80, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-69553-0. URL http://dl.acm.org/citation.cfm?id=1756428.1756438.

Peterson J. L. Petri net theory and the modeling of systems. 1981.

Pilato C., Collins-Sussman B., and Fitzpatrick B. *Version Control with Subversion*. O'Reilly Media, Inc., 2 edition, 2008. ISBN 0596510330, 9780596510336.

Prete K., Rachatasumrit N., Sudan N., and Kim M. Template-based reconstruction of complex refactorings. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, ICSM '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-1-4244-8630-4. doi: 10.1109/ICSM.2010.5609577. URL http://dx.doi.org/10.1109/ICSM.2010.5609577.

Ralyté J. and Roll C. An approach for method reengineering. In *Proceedings of the 20 th International Conference on Conceptual Modeling (ER2001), LNCS 2224*, pages 471–484. Springer Berlin / Heidelberg, 2001.

Ralyté J., Deneckère R., and Rolland C. Towards a generic model for situational method engineering. In *Proceedings of the 15th International Conference on Advanced Information Systems Engineering*, CAiSE'03, pages 95–110, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-40442-2. URL http://dl.acm.org/citation.cfm?id=1758398.1758410.

Ralyté J., Brinkkemper S., and Henderson-Sellers B. *Situational Method Engineering: Fundamentals and Experiences: Proceedings of the IFIP WG 8.1 Working Conference, 12-14 September 2007, Geneva, Switzerland*, volume 244. Springer Science & Business Media, 2007.

Reddy R., France R., Ghosh S., Fleurey F., and Baudry B. Model composition - a signature-based approach. In *Proceedings of the AOM Workshop at MODELS'05*, Montego Bay, Jamaica, October 2005.

Rho J. and Wu C. An efficient version model of software diagrams. In *Proceedings of the Fifth Asia Pacific Software Engineering Conference*, APSEC '98, pages 236–, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-9183-2. URL http://dl.acm.org/citation.cfm?id=521008.785612.

Richards M., Monson-Haefel R., and Chappell D. A. *Java message service*. " O'Reilly Media, Inc.", 2009.

Rittgen P. Coma: A tool for collaborative modeling. In *CAiSE Forum*, volume 344, pages 61–64, 2008.

Robbes R. and Lanza M. A change-based approach to software evolution. *Electronic Notes in Theoretical Computer Science*, 166:93–109, 2007.

Robbes R. and Lanza M. Spyware: A change-aware development toolset. In *Proceedings of the 30th international conference on Software engineering*, pages 847–850. ACM, 2008.

Roebuck K. *Release Management: High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. Emereo Pty Limited, 2011. ISBN 9781743048115. URL http://books.google.be/books?id=qnWyXwAACAAJ.

Rose L., Herrmannsdoerfer M., Mazanek S., Van Gorp P., Buchwald S., Horn T., Kalnina E., Koch A., Lano K., Schätz B., and Wimmer M. Graph and model transformation tools for model migration. *Software & Systems Modeling*, pages 1–37, 2012. ISSN 1619-1366. doi: 10.1007/s10270-012-0245-0. URL http://dx.doi.org/10.1007/s10270-012-0245-0.

Rose L. M. Structures and processes for managing model-metamodel co-evolution. 2011.

Rose L. M., Paige R. F., Kolovos D. S., and Polack F. A. C. An Analysis of Approaches to Model Migration. In *Proc. Models and Evolution (MoDSE-MCCM) Workshop, 12th ACM/IEEE International Conference on Model Driven Engineering, Languages and Systems*, October 2009.

Rose L. M., Kolovos D. S., Paige R. F., and Polack F. A. C. Model migration with epsilon flock. In *Proceedings of the Third international conference on Theory and practice of model transformations*, ICMT'10, pages 184–198, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-13687-7, 978-3-642-13687-0. URL http://dl.acm.org/citation.cfm?id=1875847.1875862.

Saeki M. Configuration management in a method engineering context. In Dubois E. and Pohl K., editors, *CAiSE*, volume 4001 of *Lecture Notes in Computer Science*, pages 384–398. Springer, 2006. ISBN 3-540-34652-X.

Saeki M. and Kaiya H. On relationships among models, meta models and ontologies. In *The 6th OOPSLA Workshop on Domain-Specific Modeling*, Portland, Oregon, USA, 2006. URL http://www.dsmforum.org/events/dsm06/Papers/14-saeki.pdf.

Saeki M. and Oda T. A conceptual model of version control in method engineering environment. In Belo O., Eder J., e Cunha J. F., and Pastor O., editors, *The 17th Conference on Advanced Information Systems Engineering (CAiSE '05), Porto, Portugal, 13-17 June, 2005, CAiSE Forum, Short Paper Proceedings*, volume 161 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005. URL http://www.ceur-ws.org/Vol-161/FORUM_15.pdf.

Sattler K.-U., Conrad S., and Saake G. Interactive example-driven integration and reconciliation for accessing database federations. *Inf. Syst.*, 28(5):393–414, July 2003. ISSN 0306-4379. doi: 10.1016/S0306-4379(02)00023-6. URL http://dx.doi.org/10.1016/S0306-4379(02)00023-6.

Schmidt D. C. Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2):25–31, 2006.

Schmidt K. and Bannon L. Taking cscw seriously. *Computer Supported Cooperative Work (CSCW)*, 1 (1-2):7–40, 1992. ISSN 0925-9724. doi: 10.1007/BF00752449. URL http://dx.doi.org/10.1007/BF00752449.

Schmidt K. and Simone C. Coordination mechanisms: towards a conceptual foundation of cscw systems design. *Comput. Supported Coop. Work*, 5:155–200, December 1996. ISSN 0925-9724. doi: 10.1007/BF00133655. URL http://portal.acm.org/citation.cfm?id=247460.247462.

Schneider C., Zündorf A., and Niere J. Coobra – a small step for development tools to collaborative environments. In *WORKSHOP ON DIRECTIONS IN SOFTWARE ENGINEERING ENVIRONMENTS; WORKSHOP AT ICSE 2004*, 2004.

Seidewitz E. What models mean. *IEEE Softw.*, 20:26–32, September 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231147. URL http://portal.acm.org/citation.cfm?id=942589.942706.

Seiwald C. Inter-file branching - a practical method for representing variants. In *Proceedings of the SCM-6 Workshop on System Configuration Management*, ICSE '96, pages 67–75, London, UK, UK, 1996. Springer-Verlag. ISBN 3-540-61964-X. URL http://dl.acm.org/citation.cfm?id=647175.716396.

Selic B. The pragmatics of model-driven development. *IEEE software*, (5):19–25, 2003.

Sendall S. and Kozaczynski W. Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, 20(5):42–45, September 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231150. URL http://dx.doi.org/10.1109/MS.2003.1231150.

Shen H. and Sun C. Flexible merging for asynchronous collaborative systems. In *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002*, pages 304–321, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-00106-9. URL http://dl.acm.org/citation.cfm?id=646748.701670.

Sinclair G., Hanks P., Fox G., Moon R., and et. al. Stock P. *The Collins COBUILD English Language Dictionary*. Collins, Glasgow, 1987.

Snell J., Tidwell D., and Kulchenko P. *Programming Web services with SOAP*. " O'Reilly Media, Inc.", 2001.

Sriplakich P. *ModelBus : An Open and Distributed Environment for Model Driven Engineering*. Ph.d. dissertation, University Pierre and Marie Curie, September 2007. URL http://www-src.lip6.fr/homepages/Prawee.Sriplakich.

Sriplakich P., Blanc X., and Gervais M.-P. Supporting collaborative development in an open mda environment. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance*, pages 244–253, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2354-4. doi: 10.1109/ICSM.2006.64. URL http://portal.acm.org/citation.cfm?id=1172962.1173001.

Sriplakich P., Blanc X., and Gervals M.-P. Collaborative software engineering on large-scale models: Requirements and experience in modelbus. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 674–681, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-753-7. doi: 10.1145/1363686.1363849. URL http://doi.acm.org/10.1145/1363686.1363849.

Stachowiak H. *Allgemeine Modelltheorie*. 1973. URL http://www.amazon.de/Allgemeine-Modelltheorie-Herbert-Stachowiak/dp/3211811060/ref=sr_1_2/028-1073608-4157317?ie=UTF8&s=books&qid=1190302420&sr=1-2.

Stahl T., Vöelter M., and Czarnecki K. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006. ISBN 0470025700.

Steinberg D., Budinsky F., Paternostro M., and Merks E. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009. ISBN 0321331885.

Steven Wollkind T. R. I., John Valasek. Automated conflict resolution for air traffic management using cooperative multiagent negotiation. In *In: AIAA Guidance, Navigation, and Control Conference*, pages 2004–4992, 2004.

Syriani E. and Vangheluwe H. Matters of model transformation. *School of Computer Science, McGill University*, 2009.

Syriani E., Vangheluwe H., Mannadiar R., Hansen C., Van Mierlo S., and Ergin H. Atompm: A web-based modeling environment. In Liu Y., Zschaler S., Baudry B., Ghosh S., Ruscio D. D., Jackson E. K., and Wimmer M., editors, *Joint Proceedings of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition co-located with the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013), Miami, USA, September 29 - October 4, 2013.*, volume 1115 of *CEUR Workshop Proceedings*, pages 21–25. CEUR-WS.org, 2013. URL http://ceur-ws.org/Vol-1115/demo4.pdf.

Tacla C. A. and Enembreck F. Perception of centers of interest. In *Computer Supported Cooperative Work in Design II*, pages 21–30. Springer, 2006.

Taentzer G., Ermel C., Langer P., and Wimmer M. Conflict detection for model versioning based on graph modifications. In *Proceedings of the 5th International Conference on Graph Transformations*, ICGT'10, pages 171–186, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-15927-3, 978-3-642-15927-5. URL http://dl.acm.org/citation.cfm?id=1928162.1928178.

Taentzer G., Ermel C., Langer P., and Wimmer M. A fundamental approach to model versioning based on graph modifications: from theory to implementation. *Software and Systems Modeling*, pages 1–34, 2012. ISSN 1619-1366.

Tan W., Ma L., Xu Z., and Mao J. Application mda in a collaborative modeling environment. In *Proceedings of the 6th International Conference on Entertainment Computing*, ICEC'07, pages 225–230, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-74872-5, 978-3-540-74872-4. URL http://dl.acm.org/citation.cfm?id=2394259.2394292.

Tarr P., Ossher H., Harrison W., and Sutton S. M., Jr. N degrees of separation: Multi-dimensional separation of concerns. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 107–119, New York, NY, USA, 1999. ACM. ISBN 1-58113-074-0. doi: 10.1145/302405.302457. URL http://doi.acm.org/10.1145/302405.302457.

Thiran P., Hainaut J.-L., Bodart S., Deflorenne A., and Hick J.-M. Interoperation of independent, heterogeneous and distributed databases. methodology and case support: the interdb approach. In *CoopIS*, pages 54–63. IEEE Computer Society, 1998. ISBN 0-8186-8380-5. URL http://dblp.uni-trier.de/db/conf/coopis/coopis98.html#ThiranHBDH98.

Treude C., Berlik S., Wenzel S., and Kelter U. Difference computation of large models. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE '07, pages 295–304, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-811-4. doi: 10.1145/1287624.1287665. URL http://doi.acm.org/10.1145/1287624.1287665.

Van Der Straeten R., Jonckers V., and Mens T. A formal approach to model refactoring and model refinement. *Software & Systems Modeling*, 6(2):139–162, 2007.

Vangheluwe H., Sun X., and Bodden E. Domain-specific modelling with atom3. In Filipe J., Shishkov B., and Helfert M., editors, *ICSOFT 2007, Proceedings of the Second International Conference on Software and Data Technologies, Volume PL/DPS/KE/MUSE, Barcelona, Spain, July 22-25, 2007*, pages 298–304, 2007.

Vermolen S. D., Wachsmuth G., and Visser E. Reconstructing complex metamodel evolution. In *Proceedings of the 4th International Conference on Software Language Engineering*, SLE'11, pages 201–221, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-28829-6. doi: 10.1007/978-3-642-28830-2_11. URL http://dx.doi.org/10.1007/978-3-642-28830-2_11.

Vernadat F., Percebois C., Farail P., Vingerhoeds R., Rossignol A., Talpin J. P., and Chemouil D. The TOPCASED Project - A Toolkit in OPen-source for Critical Applications and SystEm Development. In *Data Systems In Aerospace (DASIA), Berlin, Germany, 22/05/2006-25/05/2006*. European Space Agency (ESA Publications), 2006.

Vesperman J. *Essential CVS*. O'Reilly Media, Inc., 2006. ISBN 0596527039.

Vo T. T., Coulette B., Tran H. N., and Lbath R. Defining and Using Collaboration Patterns for Software Process Development (regular paper). In *International Workshop on Cooperative Model Driven Development. Co-located with MODELSWARD 2015. (CMDD), Angers, 09/02/2015-09/02/2015*, page (electronic medium), http://www.scitepress.org/, février 2015. SciTePress.

Wachsmuth G. Metamodel adaptation and model co-adaptation. In Ernst E., editor, *Proceedings of the 21st European Conference on Object-Oriented Programming (ECOOP'07)*, volume 4609 of *Lecture Notes in Computer Science*, pages 600–624. Springer-Verlag, July 2007.

Warmer J. and Kleppe A. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edition, 2003. ISBN 0321179366.

Westfechtel B. Structure-oriented merging of revisions of software documents. In *Proceedings of the 3rd International Workshop on Software Configuration Management*, SCM '91, pages 68–79, New York, NY, USA, 1991. ACM. ISBN 0-89791-429-5. doi: 10.1145/111062.111071. URL http://doi.acm.org/10.1145/111062.111071.

Westfechtel B. A formal approach to three-way merging of emf models. In *Proceedings of the 1st International Workshop on Model Comparison in Practice*, IWMCP '10, pages 31–41, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-960-2. doi: 10.1145/1826147.1826155. URL http://doi.acm.org/10.1145/1826147.1826155.

Whitehead J., Mistrík I., Grundy J., and van der Hoek A. Collaborative software engineering: Concepts and techniques. In Mistrík I., Grundy J., Hoek A., and Whitehead J., editors, *Collaborative Software Engineering*, pages 1–30. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-10293-6. doi: 10.1007/978-3-642-10294-3_1. URL http://dx.doi.org/10.1007/978-3-642-10294-3_1.

Wielemaker J. Swi-prolog 5.3 reference manual. 2004.

Woodcock J. and Davies J. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. ISBN 0-13-948472-8.

Wuu Y. How to merge program texts. *Journal of Systems and Software*, 27(2):129–135, 1994.

Xing Z. and Stroulia E. UMLDiff: An algorithm for object-oriented design differencing. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE '05, pages 54–65, New York, NY, USA, 2005. ACM. ISBN 1-58113-993-4. doi: 10.1145/1101908.1101919. URL http://doi.acm.org/10.1145/1101908.1101919.

Xing Z. and Stroulia E. Refactoring detection based on UMLDiff change-facts queries. In *Proceedings of the 13th Working Conference on Reverse Engineering*, WCRE '06, pages 263–274, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2719-1. doi: 10.1109/WCRE.2006.48. URL http://dx.doi.org/10.1109/WCRE.2006.48.

Zhang J. Metamodel-driven model interpreter evolution. In *Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 214–215, New York, NY, USA, 2005. ACM. ISBN 1-59593-193-7. doi: 10.1145/1094855.1094941. URL http://doi.acm.org/10.1145/1094855.1094941.

Zhu H., Zhou M., and Seguin P. Supporting software development with roles. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 36(6):1110–1123, 2006.

Zimmermann T. and Bird C. Collaborative software development in ten years: Diversity, tools, and remix culture. In *Proceedings of the CSCW Workshop on the Future of Collaborative Software Development*, February 2012.

Zimmermann T., Weisgerber P., Diehl S., and Zeller A. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 563–572, Washington, DC, USA, 2004a. IEEE Computer Society. ISBN 0-7695-2163-0. URL http://dl.acm.org/citation.cfm?id=998675.999460.

Zimmermann T., Weisgerber P., Diehl S., and Zeller A. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 563–572, Washington, DC, USA, 2004b. IEEE Computer Society. ISBN 0-7695-2163-0. URL http://dl.acm.org/citation.cfm?id=998675.999460.

Appendix A

Publications

Journal papers:

1. **Collaborative Editing of EMF/Ecore Metamodels and Models: Conflict Detection, Reconciliation, and Merging in DiCoMEF**: Koshima, A. , Englebert, V., Science of Computer Programming, 2015, doi:10.1016/j.scico.2015.07.004

Book Chapters

1. **A reconciliation framework to support cooperative work with DSM**: Koshima, A. A., Englebert, V., and Thiran, P. (2013). In Reinhartz-Berger, I., Sturm, A., Clark, T., Cohen, S., and Bettin, J., editors, Domain Engineering, pages 239–259. Springer Berlin Heidelberg.

Conferences and Workshops

1. **RuCORD: Rule-based Composite Operation Recovery and Detection to Support Cooperative Edition of (Meta)Models**: Koshima, A. and Englebert, V. : Special Sessions on Cooperative Model Driven Development (CMDD2015) co-located with the

3rd International Conference on Model-Driven Engineering and Software Development – MODELSWARD 2015, Angers, France, 9-11 February 2015

2. **Collaborative Editing of EMF/Ecore Metamodels and Models: Conflict Detection, Reconciliation, and Merging in DiCoMEF**: Koshima, A. , Englebert, V. 2nd International Conference on Model-Driven Engineering and Software Development – MODELSWARD 2014, Jan 07-09, 2014, Lisbon, Portugal (best Paper Award)

3. **Distributed collaborative model editing framework for domain specific modeling tools**: Koshima, A. , Englebert, V. and Thiran, P. 2011 Proceedings of the 2011 6th IEEE International Conference on Global Software Engineering. IEEE Press

4. **A framework for collaboratively editing domain specific models**: Koshima, A. , Englebert, V. and Thiran, P. 2011 Doctoral Symposium of the 25th European Conference on Object-Oriented Programming

5. **Support Tool for the Definition & Enactment of the UsiXML Methods**: Boukhebouze, M. , Pires Ferreira Neto, W. , Koshima, A. , Thiran, P. and Englebert, V. 2011 Software Support for User Interface Description Language - UIDL'2011.

6. **Comparative Analysis of Collaborative Approaches for UsiXML Meta-Models Evolution**: Boukhebouze, M. , Koshima, A. , Englebert, V. and Thiran, P. 2010 Proceedings of the UsiXML-EICS workshop

Posters

1. **A framework for collaboratively editing domain specific models**: Amanuel Koshima: poster presentation at journée des doctorants, June 26, 2015.

2. **A framework for collaboratively editing domain specific models**: Amanuel Koshima: poster presentation at the 26th International Conference on Software Engineering and Knowledge Engineering (SEKE 2014), Hyatt Regency, Vancouver, Canada. July 1 - July 3, 2014

3. **A framework for collaboratively editing domain specific models**: Amanuel Koshima: poster presentation at the DSM-TP summer school, 25-29 August 2014, Antwerp, Belgium

4. **A framework for collaboratively editing domain specific models**: Amanuel Koshima: poster presentation at the DSM-TP summer school, 2-6 September 2013, Santiago de Compostela, Spain

5. **A framework for collaboratively editing domain specific models**: Amanuel Koshima: poster presentation at the PReCISE Research Day, April 24, 2012, Namur, Belgium

6. **A framework for collaboratively editing domain specific models**: Amanuel Koshima: poster presentation at the 25th European Conference on Object-Oriented Programming (2011), 25-29 July 2011, Lancaster, United Kingdom

Student Talks

1. **A Collaborative Model Editing Framework for Eclipse Modeling Framework**: Amanuel Koshima, The seventh edition of the International Summer School on Software Engineering, July 6 - 9, 2010 - University of Salerno, Italy

Tool Demo

1. **DiCoMEF: A Distributed Collaborative Model Editing Framework**: Amanuel Koshima, Vincent Englebert, The 26th International Conference on Software Engineering and Knowledge Engineering (SEKE 2014), Hyatt Regency, Vancouver, Canada. July 1 - July 3, 2014