Distributed collaborative model editing framework for domain specic modeling tools

Koshima, Amanuel; Englebert, Vincent; Thiran, Philippe

Published in:
Proceedings of the 2011 6th IEEE International Conference on Global Software Engineering

Publication date:
2011

Document Version
Early version, also known as pre-print

Distributed Collaborative Model Editing Framework for Domain Specific Modeling Tools

Amanuel Koshima, Vincent Englebert, Philippe Thiran
PReCISE Research Center
University of Namur
Belgium
Email: {amanuel.koshima, vincent.englebert, philippe.thiran}@fundp.ac.be

***Abstract*—Domain Specific Modeling (DSM) tools have matured and became powerful over the past few years and are now used more frequently to model complex systems. Consequently, the demand for model management and collaboration among DSM tools becomes more important. In collaborative modeling, domain specific models are mostly edited and elaborated concurrently by different semi-autonomous users. Hence, there is a need for reconciliating these parallely evolved models so as to seamlessly work together. CSCW community proposes tools or techniques to ensure collaboration among general purpose modeling languages, but they do not give functionalities to support reconciliation and merging for asynchronous modification. In addition, management of communications among members of collaborative group could also help to facilitate collaboration in the group. In this paper, we propose a communication framework to manage exchanges of concurrently edited DSM models among users. Besides, we present a reconciliation framework to merge concurrently evolved DSM models.**

***Keywords*—global software development, coordination, collaboration, DSM**

I. Introduction

Model Driven Engineering (MDE) is a Software Engineering technique that aims to raise the level of abstraction of software development from code to model [1]. Specifically, MDE uses Domain Specific Modeling Languages (DSML) to specify the structure, behavior and requirements of applications within specific domains [2]. The main idea of Domain Specific Modeling (DSM) is to describe a solution directly using domain concepts rather than generic modeling languages, the benefits of this approach have been described in [3]. DSML uses model, meta-model and meta-meta-model to describe concepts at different abstraction levels. A model is an abstraction of a software system. A meta-model is a DSL oriented towards the representation of software development methodologies and endeavors [4]. Likewise, as models are described by meta-models, meta-models are also described by meta-meta-model (i.e. EMF/ECore [5]). Meta-meta-model is a minimum set of concepts which define the languages (including itself).

DSMs are tailored to a specific application domain so that it has to evolve to meet the new requirements of stakeholders [6], [7]. DSMs evolve by modifying their meta-model in order to satisfy new requirements. Indeed, like other software artifacts, meta-models could also evolve throughout software development life cycles (i.e. analysis, design, testing, and maintenance) as a result of a better understanding of the problem domain or error correction [8], [9]. Because of meta-model evolution, the existing models might no longer be conformed to the new version of meta-model. Therefore, these models need to be co-evolved in order to conform to the adapted meta-model.

Most of the DSM tools developed in the past consider the modeling process as a single user task [10], however, this hypothesis is too restricting with regards to how projects are managed. Modeling of software systems usually requires collaboration among members of a group with different scope and skills. Hence, there is a need for group members to share modeling artifacts (i.e. model and meta-models) and synchronize their activities. 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.

Inconsistency is one of the main challenges that hinder collaborative model editing, therefore, conflicts that cause inconsistencies need to be identified and resolved. Industry commonly uses a central repository with merge mechanisms and locks in order to handle inconsistency problems and ensure collaboration [11]. Unfortunately, locking technique is inadequate for a large number of users who work in parallel [12], [13]. Besides, in practice, this technique takes much time for users to resolve conflicts [13], [14]. In addition, this approach restricts users to be dependent on one repository. Other modes of collaboration could also be possible, where each member of a group has his/her own partial copy of the global specification model and communicate his/her activities by sending messages [11]. This type of collaboration gives users control over their data and let them work in isolation. It also addresses the problem of being dependent on a single repository. But, it is challenging to keep all copies of modeling artifacts consistent; because they could be modified parallely by users.

We believe that answering the following questions ensures collaboration among DSM tools: 1) how to manage communication among members of a group? and 2) how to detect conflicts and reconcile conflicting modifications? We propose a collaborative framework called DiCoMEF to ensure

collaboration among members. In DiCoMEF, every member of collaborative model editing group has his/her own local copy. Members exchange messages to communicate their activities with other members. Specifically, they exchange sequences of elementary change operations (i.e. create, delete, update) that are used to adapt models and meta-models. Modifications of models and meta-models are controlled by human actors rather than software agents. DiCoMEF uses EMF/ECore [5] as its meta-meta-model definition.

This paper is organized as follows: Section 2 describes collaborative model editing. Section 3 describes DiCoMEF's architecture, communication and reconciliation. Section 4 mentions related works. Section 5 describes benefits and finally Section 6 presents future work and conclusion.

A. Collaborative Modeling

As stated in [15], *cooperative work* is attributed to mutual interdependence of tasks among multiple users to produce specific products or services. *Computer Supported Cooperative Work (CSCW)* is a type of cooperative work in which computer systems are implemented to support these mutually interdependent works. In fact, CSCW is a broad discipline that deals with how people are working in groups and the underlaying technological support. *Groupware* is a software system that is designed to support cooperative work [15]. *Collaborative model editing*is a groupware in which computer system are employed to ease users communication and reconciliation work.

Since multiple users with different goals, strategies and experience levels are involved in collaborative modeling, their communication needs to be controlled so as to have effective collaborations. In fact, members of a group communicate one another by modifying common field of works such as models and meta-models and use message exchanges [16] to propagate local modifications. Hence, there is a need to have a protocol or a guideline for controlling change propagations. Uncontrolled change propagation lets every member of a group to propagate his/her local modifications to other members directly. This could cause a continues discussion among members to solve conflicting proposals and it may even hamper collaborative work. On the contrary, a controlled change propagation manages modifications. Specifically, a controller is assigned to supervise changes. Management of changes could be more efficient and effective if a controller has a knowledge of business domain and has a good modeling experience. Besides, s/he has authority to accept or reject modifications.

Reconciliation is a process of merging conflicting versions of (meta)models into a new version. It constitutes activities like detection of inconsistencies and meshing. Specifically, differences between two versions of (meta)models need to be identified, conflicts among two versions should be detected and resolved so as to merge the two versions into a new version. Differences between two versions of (meta)models are derived using either *State-based comparison* or *change-based comparison* [13], [12]. *State-based comparison* takes states of two versions of (meta)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 and change post-mortem [17]. *Change-based comparison* keeps track of changes whenever they occur, and then it stores them into a repository. So that, there is no need to calculate deltas later. *Operation-based comparison* is a special type of *change-based comparison* where deltas are represented as a sequence of change-operations [18]. *Operation-based comparison* captures the exact time sequences of changes that could help to understand changes and detect conflicts [12]. Besides, it can also express sets of operations that occurred in a common context as composite operations. According to Koegel et. al. [17], time sequences of changes and composite operations help users to easily understand changes in *operation-based comparison* than in *state-based comparison*.

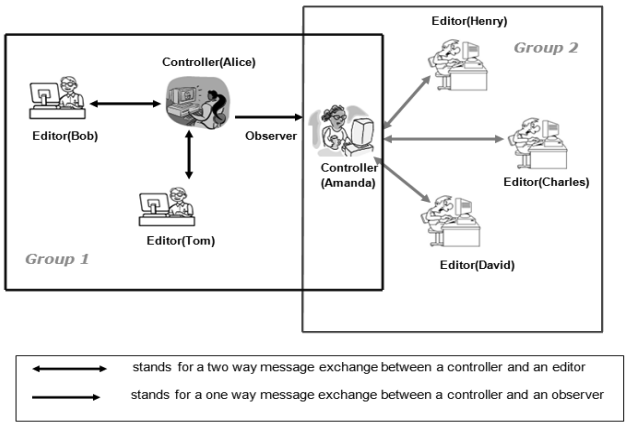

Figure 1. DiCoMEF's architecture

II. DiCoMEF

A. Architecture

DiCoMEF is a distributed collaborative model editing framework, which lets each member of a group to have his/her own local copy. As it is shown in *Figure 1*, editors of *Group 1* (Bob and Tom) and editors of *Group 2* (David, Charles and Henry) modify their local copies and send their modifications as a change request to a controller of *Group 1* (Alice) and a controller of *Group 2* (Amanda) respectively so as to propagate their modifications to other members. In fact, the collaboration scenario could becomes more complex with a member of one group may be involved in many other groups. For example, an observer of one group might be a controller in another group. But, we consider the simplified form of collaboration scenario (the one in a rectangle as shown in *Figure 1*) in this work. We assume that (meta)models owned by one controller is independent from (meta)models owned by other controllers. DiCoMEF allows a member to exchange his/her local modification directly without supervision of a controller, but this type of communication has a risk to become out of synchrony with other members.

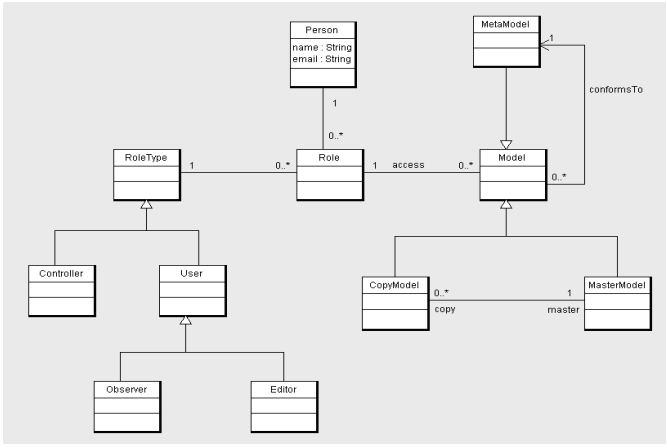

Figure 2. DiCoMEF's Meta-Model

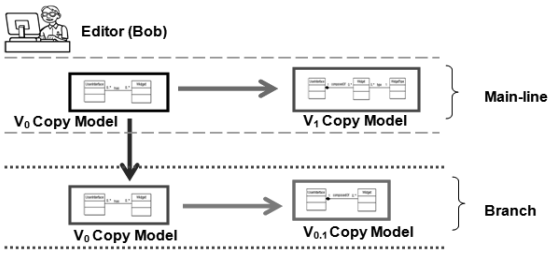

Figure 3. Main-line and Branch

Basic concepts used in DiCoMEF are expressed in the DiCoMEF meta-model (see *Figure 2*). These concepts are person, role, role type, model, meta-model, copy model and master model. A master (meta)model has one or more copy (meta)models which are distributed among members. DiCoMEF implements universal unique identifier (UUID) to differentiate (meta)model elements uniquely. Each newly created (meta)model elements has a unique identifier. That means that, two (meta)model elements are considered as equal (same) if and only if they have a same UUID. A person involved in collaborative modeling has a role(s) with respect to a (meta)model, which is typed as a controller, an editor or an observer. In fact, there are two controller role types which are implemented in DiCoMEF such as a model controller or a meta-model controller. A meta-model controller (and a model-controller) manages evolution of a master meta-model (respectively a master model). A controller role type is flexible meaning that it can be assigned to other members of a group. A person can thus be both editor, controller, or observer for distinct (meta)models. A person with an editor role type has a write and read access to copy models s/he owns locally, whereas an observer has only a read access.

DiCoMEF implements two additional main concepts such as main-line and branches to store models and meta-models locally. The main line stores different versions of a copy (meta)model. Editors cannot modify copy (meta)models stored

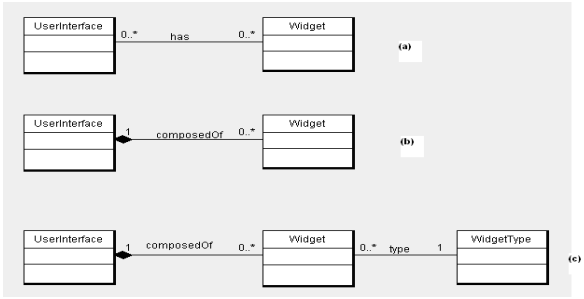

Figure 4. User Interface Model

on the main-line. Whenever they want to modify copy (meta)models locally, they first create branches from the main line and do modifications there. For example, in *Figure 3*, a copy 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, Bob, to modify the copy model locally from version V_0 to version $V_{0.1}$; a branch was created before a copy model evolves from version V_0 to version V_1.

We used the following running example to demonstrate our framework. Let Alice, Bob and Tom are involving in collaborative model editing, where Alice is a controller and Bob and Tom are editors. Alice firstly hands out the initial version of a meta-model, V_0, to all members; a user interface has a widget(s) *(see Figure 4(a))*. Later, Bob creates a branch from his local copy meta-model and changes the association link into a composition relationship and he renames the association into "composedOf" as shown in *Figure 4(b)*. And he also incorporates his rationale of changes into multimedia files as "a user interface is composed of widgets". Afterwards, the controller, Alice, and Bob modify the change requests together as shown in *Figure 4(c)*.

B. Communication

In DiCoMEF, members of a group communicate by modifying (meta)models locally and send their modifications to others members through a controller. While an editor modifies (meta)models, elementary change operations (*create, delete and updates*) that adapt (meta)models are stored locally on a branch (i.e repository). DiCoMEF uses a history meta-model to define structures of these elementary operations, but discussion about the history meta-model is out of scope of this work. Editors could annotate change operations with multimedia files (i.e. audio, text, video) to describe their rationale behind modifications.

Editors send (annotated) sequences of change operations (change request) to a controller. They could also send their local modifications directly to other editors, but this type of collaboration has a risk of having inconsistent (meta)models with other members of a group. In this work, we assume that a controller is a senior staff with good knowledge of business domain and modeling experience. Besides, s/he has given an authority to accept or reject modifications proposed

by editors. The controller applies a change request on a master (meta)model in order to evolve it from version V_n to version V_{n+1}. In fact, s/he examines proposed changes and solves conflicts by consulting multimedia files that describe rationale of modifications. S/he could also contact an editor who has proposed changes to better understand them and solve conflicts together. Eventually, a controller propagates accepted changes to all members of a group so as to automatically evolve copy (meta)models (i.e. stored on main-lines) from version V_n to version V_{n+1}. These modifications could cause a new version of a copy (meta)model inconsistent with local modifications. Hence, these conflicts should be identified and reconciled locally by the editor.

For example, Bob sends sequences of annotated elementary operations (i.e. create, delete and updates) that are used to adapt the association link (i.e. renaming and change it into composition relation) to Alice in order to propagate his modifications to other member (Tom), see *Figure 4(b)*. Alice examines changes proposed by Bob and they solve conflicts together. Afterwards, she propagates changes to all members so as to evolve copy meta-models from version V_n to version V_{n+1}, see *Figure 4(c)*. She also sends models migration strategy for a model controller to evolve models to be conformed with new definitions of meta-model. The model migration instructions could be generated automatically and(or) some of them might also be incorporated manually by an editor or a controller.

C. Reconciliation

To ensure collaboration, conflicting changes need to be detected and resolved. Indeed, DiCoMEF adopts *operation-based comparison* approach to derive differences between two (meta)models. Since models and meta-model live at different abstraction levels, we adopted two different conflict detection strategies to detect conflicts between models and conflicts between meta-models. Conflicts are detected by inspecting sets of concurrent operations that change model and meta-model elements, respectively [13], [12]. This type of conflict detection approach could be regarded as *operation-based* conflict detection [19]. In cases of conflicts, DiCoMEF provides facilities for editors and controller(s) to consult multimedia files that describe rationale of modifications so as to solve conflicts in interactive way. Editors could also contact (i.e via video conferencing, email, chat) a controller or an editor who proposed changes to solve conflicts together. As a general rule of reconciliation, we propose that every editor is forced to choose modifications proposed by a controller whenever conflicts occur. But s/he can propose her/his local modifications (i.e. the one in conflict with propagated changes) as a change request later.

DiCoMEF lets editors to decide when to synchronize their local modifications with changes propagated from a controller. Editors synchronize their local modifications with propagated changes first and then continue with their work. The other option is that, editors pend synchronization until they finish their work and merge their local modifications with propagated changes later. In both cases, a copy (meta)model firstly evolve from version V_n to version V_{n+1}. Afterwards, operations that adapt a copy (meta)model V_n locally are re-played on a new version, V_{n+1}. Indeed, DiCoMEF presents sets of local modifications that are conflicting with propagated changes to an editor. The editor selects those changes that s/he wants to keep and sends as a change request later. This way DiCoMEF keeps and re-apply local modifications after adapting (meta)models using change propagations

Reconciliation process is done both by a controller and (meta)model editors so as to merge their modifications with proposed changes. For example, when Bob sends his modifications to Alice *(see Figure 44(b))*, she examines proposed changes. But, she could also suggest to add a new class called a WidgetType because a same widget type (i.e. Button, TextField, Label) might be used by different user interfaces. Afterwards, Alice and Bob work together to modify the proposed changes by adding a class (WidgetType) and annotate change operations with multimedia files to describe their rationale *(see Figure 4(c))*. Then, Alice sends changes that she and Bob agreed up on as a change propagation to other members.

Reconciliation is done for pairs of changes that are in conflict with each other. Some pairs of changes are not conflicting, for instance, a pair of changes that modify a state of a (meta)model element in a same way. For example, if both Bob and Tom rename an association link from “has” to “composedOf”, these two modifications are not conflicting with each other. On the other hand, some changes may cause conflicts, for example, if Tom deletes a class widget in his local modifications. This modification is conflicting with changes propagated from Alice (controller), hence, Tom firstly needs to adopt changes propagated by Alice and sends his local modifications as change request later.

As it was said above, DiCoMEF uses UUID to differentiate (meta)model elements uniquely. Hence, newly created (meta)model elements are considered as different. For example, suppose Tom changes the first version of model *(Figure 4(a))* by modifying a multiplicity of an association link and rename it to *composedOf*. He also adds a new class called a *WidgetType* as it is shown in *Figure 4(c)* before Alice propagates accepted changes to all members. In this case, new classes created by Tom and change propagation *(WidgetType)* are considered as different classes. Therefore, Tom has to consult the rationales (i.e. annotations, multimedia records, ...) which annotate changes so as to understand them and merge redundant model elements. He can also contact Alice and/or Bob for further explanation. As a general rule, if a same model element is created locally and in a change propagation, then a model element that is created locally should be removed. It is also possible for a model element to be created only either in Tom’s local modification or in change propagation. If it is in the change propagation, a new model element is added to Tom’s repository based on the synchronization strategy he choses. Newly created model elements by Tom, will be sent as a change proposal to Alice

later.

It is also worth mentioning that models needs to co-evolve with an adapted meta-model so as to conform with new specifications. Let us suppose that Alice (controller) was editing a meta-model, then she should send model migration instructions to a model controller(s) in order to evolve models with new definitions of a meta-model. Since Alice (meta-model controller) does not have access to instance models, the migration strategies generated by her may not be complete to migrate all instance models. As a result, there is a need to incorporate migration strategies manually to migrate those models that are still not valid with the new specifications of meta-model. DiCoMEF provides facilities relaying on COPE [20] to help model controller to incorporate migration strategies manually in order to migrate those inconsistent models.

DiCoMEF does not ensure that models are consistent with their meta-models after model integration/migration activities are performed. This could help to facilitate the exchange of "partial" models that do not fulfill constraints of their respective meta-models among modelers.

III. Related Works

Many research has been done in the past to address challenges of collaborative software development. In [21], Ignat et. al. compared different approaches for collaboratively editing a text or tree based documents. Dewan and Hedge [22] also proposed a collaboration model that lets users to handle conflicts and merge their intentions collaboratively. However, most of previous works deal with collaborative merging of software codes.

In the context of collaborative modeling, there are few frameworks available that support collaboration among DSML tools. These frameworks commonly adopt approaches like using central repository with merge mechanisms and locks [23] in order to ensure collaboration and handle inconsistency problems. EMFStore is an operation-based collaborative model editing framework for Eclipse Modeling Framework(EMF) based models [24]. EMFStore uses a central repository with copy-merge techniques to ensure collaboration. MetaEdit+ [25] implements *Smart Mode Access Restricting Technology* (Smart Locks ©) to support concurrent access of shared modeling artifacts that are stored centrally. Even though locking technique assumes strict consistency model, it becomes inadequate when a number of users who edit (meta)models in parallel reach a very low threshold [12]. These approaches constrain all members to be dependent on a central repository. D-Paxis[11] is an operation based peer-to-peer collaborative framework for model editing. It avoids the problem of being dependent on one central repository. But, it uses an automatic conflict resolution approach that is based on the Lamport clock and the delete semantics of Praxis. But, we argue that final results of automatic reconciliation process could not reflect the intention of users. So, we propose a distributed collaborative framework called DiCoMEF which free users being dependent on a central repository. Besides, modifications are controlled by human agents (not automatic).

IV. Benefits

DiCoMEF allows a group of modelers to work in isolation and exchange their local modifications with other members later. It implements two main concepts such as main-line and branch that help to manage communication among members and ensure collaboration. Besides, changes are managed by a controller (human agent). More importantly, a controller role is flexible meaning that it could be easily assigned to another member. This dynamic roles assignment could let people to implement more elaborated strategies on top of DiCoMEF, i.e. a user can delegate his/her role to another person. DiCoMEF also provides facilities for editors to annotate their rationale of changes with multimedia files.

V. Future Work and Conclusion

To fully benefit from DSM tools, it is important to improve cooperation among them. In this work, we have presented a theoretical framework to ensure collaboration among DSM tools. Specifically, managing communications among members of a collaborative work and re conciliating concurrently evolved DSMs. The proposed framework, DiCoMEF, is at the early stage of implementation to validate theoretical concepts presented in this paper.

References

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

[2] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[3] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling. Enabling full code generation*. Wiley-IEEE Computer Society Pr, 2008.

[4] C. Gonzalez-Perez and B. Henderson-Sellers, *Metamodelling for Software Engineering*. New York: John Wiley, 2008.

[5] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.

[6] J. Zhang, "Metamodel-driven model interpreter evolution," in *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ser. OOPSLA '05. New York, NY, USA: ACM, 2005, pp. 214–215. [Online]. Available: http://doi.acm.org/10.1145/1094855.1094941

[7] M. Herrmannsdoerfer, S. Benz, and E. Juergens, "Cope - automating coupled evolution of metamodels and models," in *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*, ser. Genoa. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 52–76. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03013-0_4

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

[9] G. Wachsmuth, "Metamodel adaptation and model co-adaptation," in *ECOOP*, ser. Lecture Notes in Computer Science, E. Ernst, Ed., vol. 4609. Springer, 2007, pp. 600–624.

[10] C. Constantin, V. Englebert, and P. Thiran, "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*, vol. 457, 2009.

[11] A. Mougenot, X. Blanc, and M.-P. Gervais, "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*, ser. DAIS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 16–29. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02164-0_2

[12] T. Mens, "A state-of-the-art survey on software merging," *IEEE Trans. Softw. Eng.*, vol. 28, pp. 449–462, May 2002. [Online]. Available: http://portal.acm.org/citation.cfm?id=567176.567178

[13] K. Altmanninger, M. Seidl, and M. Wimmer, "A Survey on Model Versioning Approaches," Johannes Kepler University Linz, Tech. Rep., 2009. [Online]. Available: http://smover.tk.uni-linz.ac.at/docs/IJWIS09_paper_Altmanninger.pdf

[14] C. Pilato, B. Collins-Sussman, and B. Fitzpatrick, *Version Control with Subversion*, 2nd ed. O'Reilly Media, Inc., 2008.

[15] K. Schmidt and L. Bannon, "Taking cscw seriously: Supporting articulation work," *Computer Supported Cooperative Work*, vol. 1, pp. 7–40, 1992.

[16] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Comput. Surv.*, vol. 37, pp. 42–81, March 2005. [Online]. Available: http://doi.acm.org/10.1145/1057977.1057980

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

[18] R. Conradi and B. Westfechtel, "Version models for software configuration management," *ACM Comput. Surv.*, vol. 30, pp. 232–282, June 1998. [Online]. Available: http://doi.acm.org/10.1145/280277.280280

[19] M. Koegel, M. Herrmannsdoerfer, O. von Wesendonk, and J. Helming, "Operation-based conflict detection," in *Proceedings of the 1st International Workshop on Model Comparison in Practice*, ser. IWMCP '10. New York, NY, USA: ACM, 2010, pp. 21–30. [Online]. Available: http://doi.acm.org/10.1145/1826147.1826154

[20] M. Herrmannsdoerfer, "Operation-based versioning of metamodels with cope," in *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, ser. CVSM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 49–54. [Online]. Available: http://dx.doi.org/10.1109/CVSM.2009.5071722

[21] C.-L. Ignat, G. Oster, P. Molli, M. Cart, J. Ferrie, A.-M. Kermarrec, P. Sutra, M. Shapiro, L. Benmouffok, J.-M. Busca, and R. Guerraoui, "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*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 474–483. [Online]. Available: http://portal.acm.org/citation.cfm?id=1545009.1545344

[22] P. Dewan and R. Hegde, "Semi-synchronous conflict detection and resolution in asynchronous software development," in *ECSCW*, R. Harper and C. Gutwin, Eds. Springer, 2007, pp. 159–178.

[23] P. Sriplakich, X. Blanc, and M.-P. Gervais, "Supporting collaborative development in an open mda environment," in *Proceedings of the 22nd IEEE International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 244–253. [Online]. Available: http://portal.acm.org/citation.cfm?id=1172962.1173001

[24] M. Koegel and J. Helming, "EMFStore: a model repository for emf models," in *ICSE (2)*, J. Kramer, J. Bishop, P. T. Devanbu, and S. Uchitel, Eds. ACM, 2010, pp. 307–308.

[25] S. Kelly, "Case tool support for co-operative work in information system design," in *Information Systems in the WWW Environment*, ser. IFIP Conference Proceedings, C. Rolland, Y. Chen, and M. Fang, Eds., vol. 115. Chapman & Hall, 1998, pp. 49–69.