A Generic Framework for Analyzing Model Co-Evolution

Sinem Getir[1], Michaela Rindt[2] and Timo Kehrer[2]

[1]Reliable Software Systems, University of Stuttgart, Germany
sinem.getir@informatik.uni-stuttart.de
[2]Software Engineering Group, University of Siegen, Germany
{mrindt,kehrer}@informatik.uni-siegen.de

Abstract. Iterative development and changing requirements lead to continuously changing models. In particular, this leads to the problem of consistently co-evolving different views of a model-based system. Whenever one model undergoes changes, related models should evolve with respect to this change. Domain engineers are faced with the huge challenge to find proper co-evolution rules which can be finally used to assist developers in the co-evolution process. In this paper, we propose an approach to learn about co-evolution steps from a given co-evolution history using an extensive analysis framework. We describe our methodology and provide the results of a case study on the developed tool support.

Keywords: Model-driven engineering, model evolution, multi-view modeling, model co-evolution, model synchronization, model differencing

1 Introduction

The multi-view paradigm is a well-established methodology to manage complexity in the construction of large-scale software systems. In Model-driven Engineering (MDE), this paradigm leads to the concept of multi-view modeling; different modeling notations are used to describe different aspects such as structure, behavior, performance, reliability etc. of a system.

Iterative development and changing requirements lead to continuously changing models. Consequently, this entails the special challenge to consistently co-evolve different views of a system [12]. In practice, this challenge usually appears as a synchronization problem; different (sub-)models, each of them representing a dedicated view on the system, are usually edited independently of each other. This occurs if they are assigned to different developers or due to the fact that a developer concentrates on a single aspect at a specific point of time [13]. Thus, changes to one model must be propagated to all related models in order to keep the views synchronized and to avoid inconsistencies.

We assume a setting as shown by the bottom-left part of Figure 1, the terminology is partly adopted from related work on model synchronization and model co-evolution [5,6]: A source model $M_{src,n}$ is related to a target model $M_{tgt,n}$ via traces. A source model is the model that undergoes changes and a target
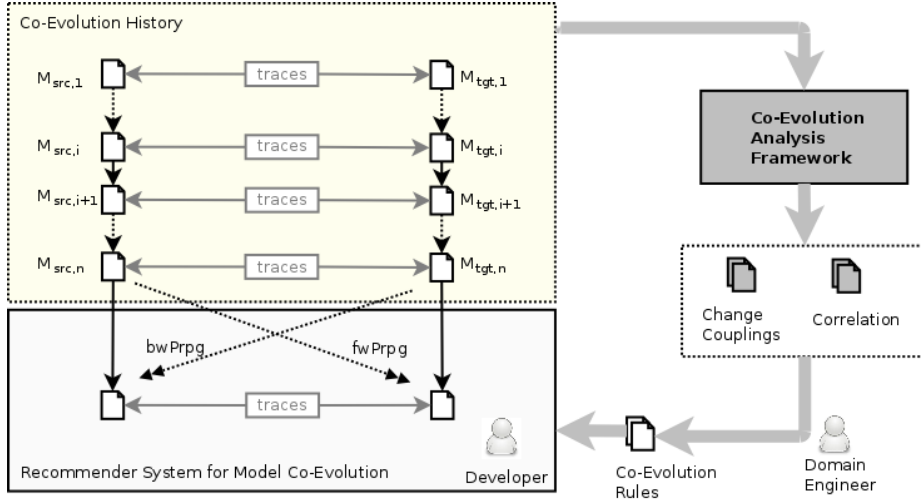
Fig. 1. Overview of the overall co-evolution process

model is the model to which these changes have to be propagated. Finally, a trace is a relationship between elements in these two different models. Forward propagation (*fwPrpg*) denotes the migration of the target model in response to changes occurring in the source model. Backward propagation (*bwPrpg*) denotes the migration of the source model in response to changes occurring in the target model. We refer to both kinds of propagations as *co-evolution steps*. From a technical point of view, co-evolution steps can be (semi-)automated via bidirectional model transformations. We call the transformation rules from which propagation rules can be derived as *co-evolution rules*.

However, due to the multitude of different modeling notations, the manual specification of co-evolution rules is a tedious and challenging task. Domain engineers, who have to find proper co-evolution rules, are faced with two essential questions: (1) Do certain changes on a source model correlate with changes on the target model? (2) If so, how are the changes coupled with each other? There are several domains for which no simple and straightforward co-evolution exists. The only viable solution is to pre-define possible co-evolution rules which can be offered to developers as possible options. For instance, this is the case for software architecture and quality of service models [4]. In Section 2, we introduce software architecture models and state charts as another example of co-evolving models which demonstrates the aforementioned research questions. We use the same example to serve as a running example throughout the paper.

This paper reports on our ongoing work on the semi-automated co-evolution of models of arbitrary source and target domains. The general process is illustrated by Figure 1. We propose to observe the co-evolution history in order to learn about developer decisions and to finally predict the co-evolution steps with a certain degree of probability. The more evolution steps are analyzed, the more accurate prediction results are expected. The contribution of this paper is the co-

evolution analysis framework which serves as a foundation for this co-evolution process. The analysis results can be used to generate co-evolution rules for a recommender system to interactively support model co-evolution. We describe our approach in Section 3. Tool support and early evaluation results which demonstrate the feasibility of our approach are briefly discussed in Section 4. Related work is analyzed in Section 5. We draw some conclusions and give an outlook on future work in Section 6.

2   Co-Evolution of Multi-View Models

Component diagrams and state charts are widely used notations to model structure and behavior in component-based software engineering. Intuitively, there are several relations between model elements of both views. For example, every state usually has a relation to a component, not necessarily the other way round. Transitions between states somehow reflect the interfaces and connections of the corresponding components in the component diagram. The hierarchy of composite states is expected to correspond to the hierarchical structure of components and their respective sub-components. Despite those rather intuitive relationships, consistently co-evolving both views is not a straight forward process, which is illustrated by the following example.
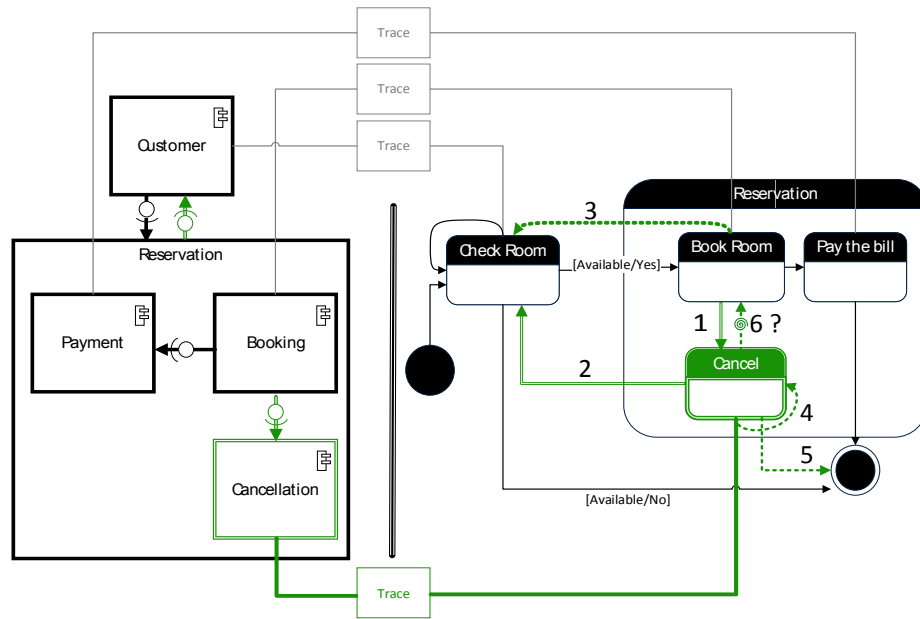

Fig. 2. Sample hotel reservation system modeled from two different viewpoints

Figure 2 shows a simple hotel reservation system modeled from two different viewpoints. The initial version of the system architecture consists of three

components, namely *Customer*, *Booking* and *Payment*. Relations between corresponding states and components are explicitly given by trace links. The system evolves at some point of time because it requires a new function to cancel a reservation process. In general, we assume that models are edited by means of a set of language-specific *edit operations*. An *edit step* invokes an edit operation and supplies appropriate actual parameters, which are also referred to as arguments. In our example, the revised version of the component model is obtained in three edit steps, namely the creation of the component *Cancellation* and two connectors. The new component and its connections to other components are highlighted in Figure 2 by doubled lines.

State chart elements printed in doubled lines indicate the developer's intention of co-evolution steps in response to the changes in the component diagram (1,2). We discuss several additional co-evolution steps which are possible on the state chart (M_{tgt}) in response to the changes in the component diagram (M_{src}). Note that these co-evolution steps are only assumptions which are based on domain knowledge, they are not meant to be a result of an empirical analysis.

Elements printed in dotted lines represent expected co-evolution steps which are, however, not intended by the user (3,4,5). Finally, a dotted line with spiral indicates an unexpected co-evolution step which is nonetheless intended by the user (6). We do not claim the set of possible options (1)-(6) to be complete. Nevertheless, it demonstrates the huge challenge of predicting the proper co-evolution steps:

– As the component *Cancellation* is added as a sub-component of reservation, a new state called *Cancel* is expected to be created as a sub-state of the corresponding composite state *Reservation*.
– The creation of transition (1) is expected due to the creation of port and interface relations of the corresponding components in the component model.
– Although there is no explicit relation between the components *Cancellation* and *Customer*, the creation of transition (2) is expected. Because the newly created relation between the composite component *Reservation* and the top-level component *Customer*, as a result of the creation of *Cancellation*. However, the new component may lead to an interaction between the components *Booking* and *Customer* indirectly via interfaces as well, therefore we should also consider the transition (3) with a small expectation.
– The required information for the proposed transitions (4) and (5) cannot be gathered from the component diagram. However, taking general state chart semantics into account, they can be presented to the developer as a possible option.
– Finally, we point out transition (6). The developer wants to create a loop between the states *Book Room* and *Cancel* which cannot be clearly anticipated from the component diagram since we observe only one direction for communication. Nonetheless, this option can be offered to the developer with a low probability.

We can conclude that each edit step on the component model may lead to many arbitrary co-evolution steps on the state chart. Some forward propagations

can be expected with a high probability based on the changes in the component diagram, others can only be offered as a set of possible choices.

3 Co-Evolution Analysis Framework

In Section 2, we have demonstrated a running example as a motivation of our analysis framework. We have presented possible co-evolution steps and observed that there are highly expected, less expected and unexpected changes for state charts when the component diagram evolves.

To study such changes and their relations, our co-evolution analysis framework takes a co-evolution history as illustrated by Figure 1 as input. Each pair of successive versions $i \rightarrow i+1$ from the given history is referred to as *evolution scenario $ev_{i \rightarrow i+1}$*. We assume that the co-evolution history includes consistent views for every evolution scenario. We further assume a model differencing engine to be available, which, given a set of possible edit operations for instances of a meta-model MM and successive model versions M_i and M_{i+1}, calculates a difference diff(M_i, M_{i+1}). A difference diff(M_i, M_{i+1}) is defined to be a partially ordered set of edit steps $s_1 \ldots s_k$. We finally offer two kinds of analysis functions; the *correlation analysis* is described in Section 3.1, the additional *coupling analysis* is presented in Section 3.2.

3.1 Correlation Analysis

We use the well-known Pearson correlation coefficient to assess the dependency between edit operations which are applicable to the source and target models. The basic processing steps of our correlation analysis are shown by Figure 3. For each evolution scenario $ev_{i \rightarrow i+1}$ of the co-evolution history, we first compute the differences diff($M_{src,i}$, $M_{src,i+1}$) and diff($M_{tgt,i}$, $M_{tgt,i+1}$). Subsequently, we count the edit steps contained by each of the obtained differences and group them by evolution scenarios and edit operations invoked by the respective edit steps. The sets of edit operations, which are available for instances of MM_{src} and MM_{tgt}, are given as additional input parameters of the correlation analysis.

Based on the calculated differences, we basically construct two matrices. For source model changes, we construct an e-$\times$-s matrix where e denotes the number of evolution scenarios in the history (i.e., $e = n-1$), s denotes the number of edit operations available for instances of MM_{src}. A variable $a_{i,j}$ ($i \in \{1, ..., e\}$, $j \in \{1, ..., s\}$) represents the number of edit steps of type j (i.e. edit steps invoking edit operations represented by j, e.g. *createComponent* in our running example) in evolution scenario i. Analogously, an e-$\times$-t matrix is being constructed for target model changes, where t denotes the number of edit operations available for instances of MM_{tgt}.

Let $X = \langle x_1, x_2, ..., x_e \rangle$ be a column vector of the e-$\times$-s matrix, and $Y = \langle y_1, y_2, ..., y_e \rangle$ be a column vector of the e-$\times$-t matrix. Then we can compute the Pearson correlation coefficient $r_{X,Y}$ for each combination of column vectors X and Y in order to quantify the linear relationship between edit operations that have been applied to the source and target models.
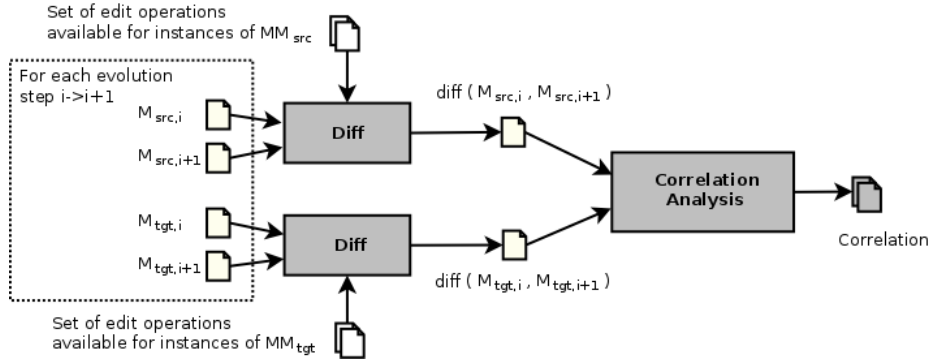
Fig. 3. Correlation analysis: basic proceeding, input and configuration parameter

3.2 Coupling Analysis

The correlation analysis has the advantage that it only requires the source and target models of each evolution scenario $ev_{i\rightarrow i+1}$. Thus, this approach can also be applied to study the co-evolution history in cases where no explicit trace links between the observed source and target model exist. However, a correlation between edit operations does not imply that the respective edit steps are actually coupled. In other words, they can have a dependency by coincidence such that none of the involved arguments are actually related by a trace. Hence, we also provide a second analysis function which is capable of identifying *coupled changes*. Such an analysis can provide knowledge about user's modeling intentions enhancing correlation analysis results, for example learning of the loop intention by the user, as provided in Figure 2 with transition (6).
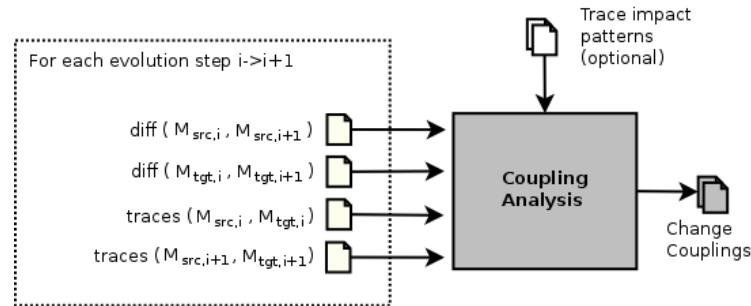

Fig. 4. Coupling analysis: basic proceeding, input and configuration parameter

In general, a coupled change identifies a pair of edit steps which have happened in the same evolution scenario. It also identifies the changed model elements which are connected (either directly or indirectly) to each other and were not just coincidentally changed in the same evolution scenario. We assume here that trace links identify related model elements of the source and target model.

17

These are, together with the model differences for each evolution scenario, provided as additional input parameters of the coupling analysis (see Figure 4).

Let $args(s)$ be the set of arguments of an edit step s. Basically, a pair of edit steps (s_{src}, s_{tgt}) is considered to be a coupled change, if we can find a pair of arguments (a_{src}, a_{tgt}), with $a_{src} \in args(s_{src})$ and $a_{tgt} \in args(s_{tgt})$, which are connected via a trace link.

Additionally, domain-specific *trace impact patterns* can be specified as optional inputs of the coupling analysis. These patterns allow to extend the search for coupled edit steps to the "neighborhood" of elements which are directly connected by a trace link. Consider for instance our running example shown in Figure 2. Here, trace links are only provided for related states and components. However, component connectors and state transitions

Fig. 5. Example of a trace impact pattern

are also to be considered as related if the connected components/states are related. This can be specified by a trace impact pattern as shown in Figure 5, i.e. the component connector labelled as cc and the state transition labelled as st are implicitly related. Consequently, a pair of edit steps modifying occurrences of cc and st, respectively, are to be considered as coupled.

Coupled changes are summarized over all evolution scenarios of the history as follows: We construct a s-$\times$-t matrix where s denotes the number of edit operations available for instances of MM_{src} and t denotes the number of edit operations available for instances of MM_{tgt}. A variable $a_{i,j}$ ($i \in \{1, ..., s\}$, $j \in \{1, ..., t\}$) is computed as the fraction of coupled edit steps of types i and j (i.e. edit steps invoking edit operations represented by i and j, respectively) with respect to all edit steps of type i being observed in the source model history.

4   Tool Support

We have prototypically implemented the analysis framework proposed in Section 3 on the widely used Eclipse Modeling Framework (EMF) and the model differencing engine SiLift [8, 9]. It is made available to the general public at the SiLift website[1] in order to enable other researchers to study the co-evolution of any EMF-based models.

Adaption of the generic framework. In order to adapt the generic framework to new modeling languages, i.e., to adapt it to a given source and target domain, one has to configure the SiLift differencing tool chain. Primarily, suitable edit operations for the source and target domain have to be provided. In SiLift, we use the model transformation language and system Henshin [1] to implement edit operations as declarative transformation rules, to which we refer to as *edit*
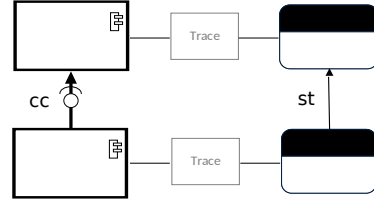
---

[1] http://pi.informatik.uni-siegen.de/Projekte/SiLift/coevolution.php

rules. Domain engineers can make use of the EMF-based meta-tool SERGe (SiDiff Edit Rule Generator) [10] in order to generate basic edit rules, which can be derived from MM_{src} and MM_{tgt}, respectively. Basic edit rules can be complemented by semantically rich complex edit rules such as refactoring operations. Typically, many complex edit rules can be composed of basic edit rules generated by SERGe.

Optionally, a set of trace impact patterns can be specified as additional input for the coupling analysis. Trace impact patterns are also specified in Henshin. We refer to these pattern specifications as *trace impact rules*. Trace impact rules do not implement in-place transformations, but serve as specifications of graph patterns which are to be found by the Henshin matching engine. Obviously, trace impact rules have to be specified manually by a domain engineer.

PPU Case Study. In order to demonstrate the feasibility of our approach, we have adapted the analysis framework to be used in the PPU(Pick and Place Unit) case study [11], which provides several evolution scenarios of a laboratory plant. In our previous work [4], we modeled each of the scenarios from two different viewpoints using two types of modeling languages: A simple architecture description language (SA) was used to model the system architecture, fault trees (FT) were used to model undesired system states and their possible causes.

All configuration artifacts which are needed to adapt the analysis framework to SA and FT models are available at the EnSure website[2]. In summary, we identified 82 edit rules available for FT models, 69 of them could be generated with SERGe. For SA models, we identified 42 suitable edit rules of which only one had to be specified manually, all other 41 edit rules could be generated with SERGe. In addition, we specified 6 trace impact rules serving as additional input of the coupling analysis. Consequently, we were able to automatically generate the results that have been produced by a manual analysis in our previous work [4].

5   Related work

Most approaches to model co-evolution address the migration of different types of MDE artifacts in response to meta-model adaptions. MDE artifacts which have to be migrated are, for example, instance models [7], model transformations [14], or syntactic and semantic constraints [3].

Only a few approaches address the evolution of multi-view models, which is most often considered as a model synchronization problem. Solutions are often based on the principle of bidirectional model transformations which are used to derive incremental change propagation rules, e.g., [5,16,6]. Among them, the approaches of Giese et al. [5] and Hermann et al. [6] are based on Triple Graph Grammars (TGGs). TGG rules describe correspondences between elements of source and target models together with the according forward and backward

---

[2] http://www.iste.uni-stuttgart.de/rss/projects/ensure/co-evolution

editing behavior. Bergmann et al. [2] present a novel type of model transformation to which they refer to as change-driven transformations. Change-driven transformations are directly triggered by complex model changes and thus can be utilized to specify sophisticated co-evolution patterns. A similar approach is presented by Wimmer et al. [15].

In contrast to our approach, TGG rules and change-driven transformations must be specified manually, whereas we intend to generate our co-evolution rules. In fact, we believe that existing approaches based on TGGs, change-driven transformations or similar techniques, can be also supported by our co-evolution analysis framework. Up to the best of our knowledge, we are not aware of any approach providing a framework to empirically study co-evolution by analyzing the history of co-evolving models.

6   Conclusion and Future Work

Many approaches for consistently co-evolving models and other related MDE artifacts have been proposed recently. Some are tailored to fixed source and target domains while others are more generic and adaptable.

However, correlation and coupling of changes has not been researched in-depth for many types of co-evolving (sub-)models. In order to close this research gap, we focus on establishing a co-evolution analysis framework to analyze the history of co-evolving models of arbitrary types. This will provide the foundation for synthesizing co-evolution rules in an automated way. Although the analysis framework still needs some configuration data as input, we conclude from the PPU case study that this adaption to a dedicated source and target domain can be done with moderate effort. Currently, the co-evolution rules which we finally intend to use as input of a co-evolution framework (see Figure 1) still have to be manually synthesized based on the information which is produced by the analysis framework. Larger case studies are needed to evaluate how far we can push the generation of co-evolution rules and how much training data is needed to derive appropriate co-evolution rules.

On the one hand, these co-evolution rules can be used to configure existing model synchronization frameworks in cases of domains where the co-evolution process can be fully automated. On the other hand, co-evolution rules serve as basis for a recommender system, which is able to handle semi-automated co-evolution of models. The latter one is subject to our future work.

References

1. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: advanced concepts and tools for in-place emf model transformations. In: Intl. Conf. on Model Driven Engineering Languages and Systems, pp. 121–135 (2010)
2. Bergmann, G., Ráth, I., Varró, G., Varró, D.: Change-driven model transformations - change (in) the rule to rule the change. Software and System Modeling 11(3), 431–461 (2012), http://dx.doi.org/10.1007/s10270-011-0197-9
3. Demuth, A., Lopez-Herrejon, R.E., Egyed, A.: Supporting the co-evolution of meta-models and constraints through incremental constraint management. In: Intl. Conf. on Model Driven Engineering Languages and Systems. pp. 287–303 (2013)
4. Getir, S., Van Hoorn, A., Grunske, L., Tichy, M.: Co-evolution of software architecture and fault tree models: An explorative case study on a pick and place factory automation system. In: Intl. Workshop on Non-functional Properties in Modeling: Analysis, Languages, Processes. pp. 32–40 (2013)
5. Giese, H., Wagner, R.: Incremental model synchronization with triple graph grammars. In: Int. Conf. on Model Driven Engineering Languages and Systems, pp. 543–557 (2006)
6. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y., Gottmann, S., Engel, T.: Model synchronization based on triple graph grammars: correctness, completeness and invertibility. Software & Systems Modeling pp. 1–29 (2013)
7. Herrmannsdörfer, M., Wachsmuth, G.: Coupled evolution of software metamodels and models. In: Evolving Software Systems, pp. 33–63 (2014)
8. Kehrer, T., Kelter, U., Taentzer, G.: A rule-based approach to the semantic lifting of model differences in the context of model versioning. In: Intl. Conf. on Automated Software Engineering. pp. 163–172 (2011)
9. Kehrer, T., Kelter, U., Taentzer, G.: Consistency-preserving edit scripts in model versioning. In: Intl. Conf. on Automated Software Engineering. pp. 191–201 (2013)
10. Kehrer, T., Rindt, M., Pietsch, P., Kelter, U.: Generating edit operations for profiled UML models. In: Intl. Workshop on Models and Evolution. pp. 30–39 (2013)
11. Legat, C., Folmer, J., Vogel-Heuser, B.: Evolution in industrial plant automation: A case study. In: Proc. of IECON 2013. IEEE (2013)
12. Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., Jazayeri, M.: Challenges in software evolution. In: Intl. Workshop on Principles of Software Evolution. pp. 13–22 (2005)
13. Ruhroth, T., Gärtner, S., Bürger, J., Jürjens, J., Schneider, K.: Versioning and evolution requirements for model-based system development. In: Intl. Workshop on Comparison and Versioning of Software Models (2014)
14. Taentzer, G., Mantz, F., Lamo, Y.: Co-transformation of graphs and type graphs with application to model co-evolution. In: Graph Transformations, pp. 326–340. Springer (2012)
15. Wimmer, M., Moreno, N., Vallecillo, A.: Viewpoint co-evolution through coarse-grained changes and coupled transformations. In: Objects, Models, Components, Patterns - 50th International Conference, TOOLS 2012, Prague, Czech Republic, May 29-31, 2012. Proceedings. pp. 336–352 (2012), http://dx.doi.org/10.1007/978-3-642-30561-0_23
16. Xiong, Y., Song, H., Hu, Z., Takeichi, M.: Synchronizing concurrent model updates based on bidirectional transformation. Software & Systems Modeling 12(1), 89–104 (2013)