Concurrent Model Synchronization with Conflict Resolution Based on Triple Graph Grammars

Frank Hermann[1,2,⋆], Hartmut Ehrig[1], Claudia Ermel[1], and Fernando Orejas[3]

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
{frank.hermann,hartmut.ehrig,claudia.ermel}@tu-berlin.de
[2] Interdisciplinary Center for Security, Reliability and Trust, Université du Luxembourg
[3] Departament de Llenguatges i Sistemes Informàtics,
Universitat Politècnica de Catalunya, Barcelona, Spain
orejas@lsi.upc.edu

Abstract. Triple graph grammars (TGGs) have been used successfully to analyse correctness of bidirectional model transformations. Recently, also a corresponding formal approach to model synchronization has been presented, where updates on a given domain (either source or target) can be correctly (forward or backward) propagated to the other model. However, a corresponding formal approach of *concurrent* model synchronization, where a source and a target modification have to be synchronized simultaneously, has not yet been presented and analysed. This paper closes this gap taking into account that the given and propagated source or target model modifications are in conflict with each other. Our conflict resolution strategy is semi-automatic, where a formal resolution strategy – known from previous work – can be combined with a user-specific strategy.

As first result, we show *correctness* of concurrent model synchronization, that is, each result of our nondeterministic concurrent update leads to a consistent correspondence between source and target models, where consistency is defined by the TGG. As second result, we show *compatibility* of concurrent with basic model synchronization: concurrent model synchronization can realize both forward and backward propagation. The results are illustrated by a running example on updating organizational models.

Keywords: model synchronization, conflict resolution, model versioning, correctness, bidirectional model transformation, triple graph grammars.

1 Introduction

Bidirectional model transformations form a key concept for model generation and synchronization within model driven engineering (MDE, see [22]). Triple graph grammars (TGGs) have been successfully applied in several case studies for bidirectional model transformation, model integration and synchronization [20,25,14] and for the implementation of QVT [15]. Based on the work of Schürr et al. [24,25], we developed a

⋆ Supported by the National Research Fund, Luxembourg (AM2a).

J. de Lara and A. Zisman (Eds.): FASE 2012, LNCS 7212, pp. 178–193, 2012.

formal theory of TGGs [9,16], which allows handling correctness, completeness, termination and functional behaviour of model transformations. Inspired by existing synchronization tools [14] and frameworks [4], we proposed an approach for basic model synchronization in [17], showing its correctness. In that paper we studied the problem of how updates on a given domain can be correctly propagated to another model.

The aim of this paper is to provide, on this basis, also a correct TGG framework for *concurrent* model synchronization, where concurrent model updates in different domains have to be merged to a consistent solution. In this case, we have the additional problem of detecting and solving conflicts between given updates. Such conflicts may be hard to detect, since they may be caused by concurrent updates on apparently unrelated elements of the given models. Furthermore, there may be apparently contradictory updates on related elements of the given domains which may not be real conflicts.

The main idea and results of our approach are the following:

1. Model synchronization is performed by propagating the changes from one model of one domain to a corresponding model in another domain using forward and backward propagation operations. The propagated changes are compared with the given local update. Possible conflicts are resolved in a semi-automated way.
2. The operations are realized by model transformations based on TGGs [17] and tentative merge constructions solving conflicts [11]. The specified TGG also defines consistency of source and target models.
3. In general, the operation of model synchronization is nondeterministic, since there may be several conflict resolutions. The different possible solutions can be visualized to the modelers, who then decide which modifications to accept or discard.
4. The main result shows that the concurrent TGG synchronization framework is correct and compatible with the basic synchronization framework, where only single updates are considered at the same time.

Based on TGGs we present the general concurrent model synchronization framework in Sec. 2, the basic model framework in Sec. 3, and conflict resolution in Sec. 4. In Sec. 5 we combine these operations with additional auxiliary ones and present the construction of the concurrent synchronization operation, for which we show its correctness and its compatibility with the basic synchronization case in Sec. 6. All constructions and results are motivated and explained by a small case study. Finally, Secs. 7 and 8 discuss related work, conclusions and future work. Full proofs and technical details on efficiency issues and the case study are presented in a technical report [10].

2 Concurrent Model Synchronization Framework

Concurrent model synchronization aims to provide a consistent merging solution for a pair of concurrent updates that are performed on two interrelated models. This section provides a formal specification of the concurrent synchronization problem and the corresponding notion of correctness. At first, we motivate the general problem with a compact example.[1]

[1] More complex case studies are also tractable by our approach, e.g. relating class diagrams to data base models [9].

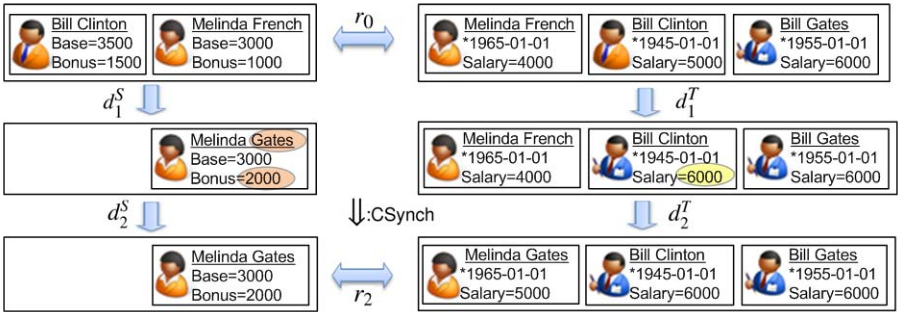

Fig. 1. Concurrent model synchronization: compact example

Example 2.1 (Concurrent model synchronization problem). Fig. 1 shows two models in correspondence that cover different aspects about employees of a company. The source model contains information about employees of the marketing department only, but shows more detailed salary information. Two model updates have to be synchronized concurrently: on the source side (model update d_1^S), Bill Clinton's node is deleted and Melinda Gates' family name changes due to her marriage; moreover, being married, her bonus is raised from 1000 to 2000. On the target side (model update d_1^T), Bill Clinton is switching from the marketing to the technical department (in the visualization in Fig. 1 this is indicated by a different role icon for Bill Clinton). His department change is combined with a salary raise from 5000 to 6000. After performing updates d_2^S and d_2^T, a "consistently integrated model" (see below) is derived that reflects as many changes as possible from the original updates in both domains and resolves inconsistencies, e.g. by computing the new Salary of Melinda Gates in the target domain as sum of the updated source attributes Base and Bonus. Note that Bill Clinton is not deleted in the target domain by the concurrent synchronization because in this case, the changes required by d_1^T could not have been realized. This conflict can be considered an apparent one. If a person leaves the marketing department, but not the company, its node should remain in the target model. Thus, a concurrent model synchronization technique has to include an adequate conflict resolution strategy.

A general way of specifying consistency between interrelated models of a source and a target domain is to provide a consistency relation that defines the consistent pairs (M^S, M^T) of source and target models. Triple graph grammars (TGGs) are a formal approach for the definition of a language of consistently integrated models [24,9]. TGGs have been applied successfully for bidirectional model transformations [25,16] and basic model synchronization [14,17], where no concurrent model updates occur.

In the framework of TGGs, an integrated model is represented by a triple graph consisting of three graphs G^S, G^C, and G^T, called source, correspondence, and target graphs, respectively, together with two mappings (graph morphisms) $s_G : G^C \to G^S$ and $t_G : G^C \to G^T$. Further concepts like attribution and inheritance can be used according to [9,8]. The two mappings in G specify a *correspondence* $r : G^S \leftrightarrow G^T$, which relates the elements of G^S with their corresponding elements of G^T and vice versa. However, it is usually sufficient to have explicit correspondences between nodes only. For simplicity, we use double arrows (↔) as an equivalent shorter notation for triple graphs, whenever the explicit correspondence graph can be omitted.

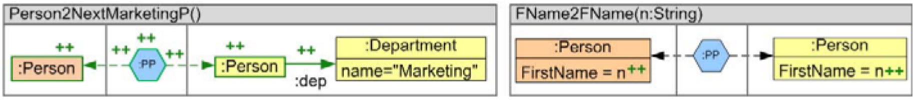

Fig. 2. Two triple rules of the TGG

$$\begin{tikzcd}
G = (\arrow[d, "m"'] & G^S \arrow[d, "m^S"] & G^C \arrow[l, hook', "s_G"'] \arrow[r, "t_G"] \arrow[d, "m^C"] & G^T) \arrow[d, "m^T"] \\
H = (& H^S & H^C \arrow[l, hook', "s_H"] \arrow[r, "t_H"'] & H^T)
\end{tikzcd}$$

Triple graphs are related by triple graph morphisms $m : G \to H$ consisting of three graph morphisms that preserve the associated correspondences (i.e., the diagrams on the right commute). Our triple graphs are typed. This means that a type triple graph TG is given (playing the role of a metamodel) and, moreover, every triple graph G is typed by a triple graph morphism $type_G : G \to TG$. It is required that morphisms between typed triple graphs preserve the typing. For $TG = (TG^S \leftarrow TG^C \to TG^T)$, we use $VL(TG)$, $VL(TG^S)$, and $VL(TG^T)$ to denote the classes of all graphs typed over TG, TG^S, and TG^T, respectively.

$$\begin{tikzcd}
L \arrow[r, hook, "tr"] \arrow[d, "m"'] \arrow[dr, phantom, "(PO)"] & R \arrow[d, "n"] \\
G \arrow[r, hook, "t"'] & H
\end{tikzcd}$$

A triple rule $tr = (tr^S, tr^C, tr^T)$ is an inclusion of triple graphs, represented $L \hookrightarrow R$. Notice that one or more of the rule components tr^S, tr^C, and tr^T may be empty, i.e. some elements in one domain may have no correspondence to elements in the other domain. In the example, this is the case for employees of the technical department within the target model. A triple rule is applied to a triple graph G by matching L to some subtriple graph of G via a match morphism $m : L \to G$. The result of this application is the triple graph H, where L is replaced by R in G. Technically, the result of the transformation is defined by a pushout diagram, as depicted above. This triple graph transformation (TGT) step is denoted by $G \overset{tr,m}{\Longrightarrow} H$. Moreover, triple rules can be extended by negative application conditions (NACs) for restricting their application to specific matches [16].

Example 2.2 (Triple Rules). Fig. 2 shows two triple rules of our running example using short notation, i.e., left- and right-hand side of a rule are depicted in one triple graph and the elements to be created have the label "++". Rule Person2NextMarketingP requires an existing marketing department. It creates a new person in the target component together with its corresponding person in the source component and the explicit correspondence structure. (The TGG contains a further rule (not depicted) for initially creating the marketing department.) Rule FName2FName extends two corresponding persons by their first names. There are further rules for handling the remaining attributes. In particular, the rule for attribute birth is the empty rule on the source component.

A triple graph grammar $TGG = (TG, S, TR)$ consists of a triple type graph TG, a triple start graph S and a set TR of triple rules, and generates the triple graph language $VL(TGG) \subseteq VL(TG)$. A TGG is, simultaneously, the specification of the classes of consistent source and target languages $VL_S = \{G^S \mid (G^S \leftarrow G^C \to G^T) \in VL(TGG)\}$ and $VL_T = \{G^T \mid (G^S \leftarrow G^C \to G^T) \in VL(TGG)\}$ and also of the class $C = VL(TGG) \subseteq VL(TG) = Rel$ of consistent correspondences which define the consistently integrated models. The possible model updates Δ_S and Δ_T are given by the sets of all graph modifications for the source and target domains. In our context, a model update $d : G \to G'$ is specified as a *graph modification* $d = (G \xleftarrow{i_1} I \xrightarrow{i_2} G')$. The relating

<table>
<tr><th>Signature</th><th>Laws</th></tr>
<tr><td>

$$\begin{tikzcd}
G_1^S \arrow[d, dashed, "d_2^S"'] & G_0^S \arrow[l, "d_1^S"'] \arrow[r, leftrightarrow, "r_0"] \arrow[r, phantom, shift right=4ex, "\Downarrow\text{:CSynch}"] & G_0^T \arrow[r, "d_1^T"] & G_1^T \arrow[d, dashed, "d_2^T"] \\
G_2^S \arrow[rrr, dashed, leftrightarrow, "r_2"'] & & & G_2^T
\end{tikzcd}$$

</td><td>

$\forall\, c \in C:$

$$\begin{tikzcd}
G^S \arrow[d, dashed, "1"'] & G^S \arrow[l, "1"'] \arrow[r, leftrightarrow, "c"] \arrow[r, phantom, shift right=4ex, "\Downarrow\text{:CSynch}"] & G^T \arrow[r, "1"] & G^T \arrow[d, dashed, "1"] & (a) \\
G^S \arrow[rrr, dashed, leftrightarrow, "c"'] & & & G^T
\end{tikzcd}$$

$$\begin{tikzcd}
G_1^S \arrow[d, dashed, "d_2^S"'] & G_0^S \arrow[l, "d_1^S"'] \arrow[r, leftrightarrow, "r_0"] \arrow[r, phantom, shift right=4ex, "\Downarrow\text{:CSynch}"] & G_0^T \arrow[r, "d_1^T"] & G_1^T \arrow[d, dashed, "d_2^T"] & (b) \\
G_2{}^S \arrow[rrr, dashed, leftrightarrow, "r_2{:}C"'] & & & G_2{}^T
\end{tikzcd}$$

</td></tr>
</table>

Fig. 3. Signature and laws for correct concurrent synchronization frameworks

morphisms $i_1 : I \hookrightarrow G$ and $i_2 : I \hookrightarrow G'$ are inclusions and specify which elements are deleted from G (all the elements in $G \setminus I$) and which elements are added by d (all the elements in $G' \setminus I$). While graph modifications can also be seen as triple graphs, it is conceptually important to distinguish between correspondences and updates δ.

The concurrent synchronization problem is visualized in Fig. 3, where we use solid lines for the inputs and dashed lines for the outputs. Given an integrated model $G_0 = (G_0^S \leftrightarrow G_0^T)$ and two model updates $d_1^S = (G_0^S \to G_1^S)$ and $d_1^T = (G_0^T \to G_1^T)$, the required result consists of updates $d_2^S = (G_1^S \to G_2^S)$ and $d_2^T = (G_1^T \to G_2^T)$ and a consistently integrated model $G_2 = (G_2^S \leftrightarrow G_2^T)$. The solution for this problem is a concurrent synchronization operation CSynch, which is left total but in general non-deterministic, which we indicate by a wiggly arrow "$\rightsquigarrow$" in Thm. 2.3 below. The set of inputs is given by $(Rel \otimes \Delta_S \otimes \Delta_T) = \{(r, d^S, d^T) \in Rel \times \Delta_S \times \Delta_T \mid r\colon G_0^S \leftrightarrow G_0^T, d^S : G_0^S \to {G_2}^S, d^T : G_0^T \to {G_2}^T\}$, i.e., r coincides with d^S on G_0^S and with d^T on G_0^T.

Definition 2.3 (Concurrent Synchronization Problem and Framework). *Given TGG, the* concurrent synchronization problem *is to construct a left total and nondeterministic operation* $\mathsf{CSynch} : (Rel \otimes \Delta_S \otimes \Delta_T) \rightsquigarrow (Rel \times \Delta_S \times \Delta_T)$ *leading to the signature diagram in Fig. 3, called concurrent synchronization tile with concurrent synchronization operation* CSynch. *Given a pair* $(prem, sol) \in \mathsf{CSynch}$ *the triple* $prem = (r_0, d_1^S, d_1^T) \in Rel \otimes \Delta_S \otimes \Delta_T$ *is called premise and* $sol = (r_2, d_2^S, d_2^T) \in Rel \times \Delta_S \times \Delta_T$ *is called a solution of the synchronization problem, written* $sol \in \mathsf{CSynch}(prem)$. *The operation* CSynch *is called* correct *with respect to consistency relation C, if laws* (a) *and* (b) *in Fig. 3 are satisfied for all solutions. Given a concurrent synchronization operation* CSynch, *the* concurrent synchronization framework *CSynch is given by* $CSynch = (TGG, \mathsf{CSynch})$. *It is called* correct, *if operation* CSynch *is correct.*

Correctness of a concurrent synchronization operation CSynch ensures that any resulting integrated model $G_2 = (G_2^S \leftrightarrow G_2^T)$ is consistent (law (b)) and, the synchronization of an unchanged and already consistently integrated model always yields the identity of the input as output (law (a)).

3 Basic Model Synchronization Framework

We now briefly describe the basic synchronization problem and its solution [17], which is the basis for the solution for the concurrent synchronization problem in Sec. 5.

Given an integrated model $G^S \leftrightarrow G^T$ and an update on one domain, either G^S or G^T, the basic synchronization problem is to propagate the given changes to the other domain. This problem has been studied at a formal level by several authors (see, for instance, [12,19,26,3,28,18,5,6,17]). Many of these approaches [12,19,26,28] are state-based, meaning that they consider that the synchronization operations take as parameter the states of the models before and after the modification and yields new states of models. However, in [3,5] it is shown that state-based approaches are not adequate in general for solving the problem. Instead a number of other approaches (see, for instance, [3,18,6,17]) are δ-based, meaning that the synchronization operations take modifications as parameters and returns modifications as results. In particular, in [17], we describe a framework based on TGGs, where we include specific procedures for forward and backward propagation of modifications, proving its correctness in terms of the satisfaction of a number of laws. These results can be seen as an instantiation, in terms of TGGs, of the abstract algebraic approach presented in [6].

$$\begin{tikzcd}
G^S \arrow[r, leftrightarrow, "r"] \arrow[d, "a"'] \arrow[dr, Rightarrow, ":\mathsf{fPpg}"] & G^T \arrow[d, dotted, "b"] \\
G'^S \arrow[r, leftrightarrow, dotted, "r'"'] & G'^T
\end{tikzcd}
\qquad
\begin{tikzcd}
G^S \arrow[r, leftrightarrow, "r"] \arrow[d, dotted, "a"'] & G^T \arrow[d, "b"] \arrow[dl, Rightarrow, ":\mathsf{bPpg}"] \\
G'^S \arrow[r, leftrightarrow, dotted, "r'"'] & G'^T
\end{tikzcd}$$

Fig. 4. Propagation operations

To be precise, according to [17], a basic synchronization framework must provide suitable left total and deterministic forward and backward propagation operations fPpg and bPpg solving this problem for any input (see Fig. 4). The input for fPpg is an integrated model $G^S \leftrightarrow G^T$ together with a source model update (graph modification) $a : G^S \rightarrow G'^S$, and the output is a target update $b : G^T \rightarrow G'^T$ together with a consistently integrated model $G'^S \leftrightarrow G'^T$. The operation bPpg behaves symmetrically to fPpg. It takes as input $G^S \leftrightarrow G^T$ and a target modification $b : G^T \rightarrow G'^T$ and it returns a source update $a : G^S \rightarrow G'^S$ together with a consistently integrated model $G'^S \leftrightarrow G'^T$. Note that determinism of these operations means that their results are uniquely determined. Note also that we require that the resulting model after a propagation operation must be consistent according to the given TGG.

We may notice that in a common tool environment, the inputs for these operations are either available directly or can be obtained. For example, the graph modification of a model update can be derived via standard difference computation.

The propagation operations are considered *correct* in [17], if they satisfy the four laws depicted in Fig. 5. Law $(a1)$ means that if the given update is the identity and the given correspondence is consistent, then fPpg changes nothing. Law $(a2)$ means that fPpg always produces consistent correspondences from consistent updated source models G'^S, where the given correspondence r: $G^S \leftrightarrow G^T$ is not required to be consistent. Laws $(b1)$ and $(b2)$ are the dual versions concerning bPpg.

$$\forall\, c \in C : \quad (a1): \begin{tikzcd}
G^S \arrow[r, leftrightarrow, "c"] \arrow[d, "1"'] \arrow[dr, Rightarrow, ":\mathsf{fPpg}"] & G^T \arrow[d, dotted, "1"] \\
G^S \arrow[r, leftrightarrow, dotted, "c"'] & G^T
\end{tikzcd}
\qquad
\forall\, G'^S \in VL_S : \quad (a2): \begin{tikzcd}
G^S \arrow[r, leftrightarrow, "r"] \arrow[d, "a"'] \arrow[dr, Rightarrow, ":\mathsf{fPpg}"] & G^T \arrow[d, dotted, "b"] \\
G'^S \arrow[r, leftrightarrow, dotted, "r':C"'] & G'^T
\end{tikzcd}$$

$$\forall\, c \in C : \quad (b1): \begin{tikzcd}
G^S \arrow[r, leftrightarrow, "c"] \arrow[d, dotted, "1"'] & G^T \arrow[d, "1"] \arrow[dl, Rightarrow, ":\mathsf{bPpg}"] \\
G^S \arrow[r, leftrightarrow, dotted, "c"'] & G^T
\end{tikzcd}
\qquad
\forall\, G'^T \in VL_T : \quad (b2): \begin{tikzcd}
G^S \arrow[r, leftrightarrow, "r"] \arrow[d, dotted, "a"'] & G^T \arrow[d, "b"] \arrow[dl, Rightarrow, ":\mathsf{bPpg}"] \\
G'^S \arrow[r, leftrightarrow, dotted, "r':C"'] & G'^T
\end{tikzcd}$$

Fig. 5. Laws for correct basic synchronization frameworks

In [17], we also present specific propagation operations: Given $G^S \leftrightarrow G^T$ and the modification $a : G^S \to G'^S$, the forward propagation operation consists of three steps. In the first step, we compute an integrated model $G'^S \leftrightarrow G^T$ by deleting from the correspondence graph all the elements that were related to the elements deleted by the modification a. In the second step, we compute the largest consistently integrated model $G_0^S \leftrightarrow G_0^T$ that is included in $G'^S \leftrightarrow G^T$. Note that we do not build this model from scratch, but mark the corresponding elements in $G'^S \leftrightarrow G^T$. Moreover, we delete from G^T all the unmarked elements. Finally, using the TGG, we build the missing part of the target model that corresponds to $G'^S \setminus G_0^S$ yielding the consistently integrated model $G'^S \leftrightarrow G'^T$. Backward propagation works dually.

Remark 3.1 (Correctness of Derived Basic TGG Synchronization Framework). Correctness of the derived propagation operations fPpg, bPpg is ensured if the given TGG is equipped with *deterministic sets of operational rules* [17]. This essentially means that the forward and backward translation rules ensure functional behaviour for consistent inputs. For the technical details and automated analysis of this property using the tool AGG [27] we refer to [17], where we have shown this property for the TGG of our example and discussed the required conditions of a TGG in more detail. Note that the concurrent synchronization procedure in Sec. 5 only requires correctness of the given propagation operations and does not rely on the specific definition in [17].

4 Semi-automated Conflict Detection and Resolution

We now review the main constructions and results for conflict resolution in one domain according to [11]. Note that we apply conflict resolution either to two conflicting target model updates (one of them induced by a forward propagation operation fPpg) or to two conflicting source model updates (one of them induced by backward propagation). Hence, we here consider updates over *standard graphs* and not over triple graphs.

Two graph modifications $(G \leftarrow D_i \to H_i)$, $(i = 1, 2)$ are called *conflict-free* if they do not interfere with each other, i.e., if one modification does not delete a graph element the other one needs to perform its changes. Conflict-free graph modifications can be merged to one graph modification $(G \leftarrow D \to H)$ that realizes both original graph modifications simultaneously.

If two graph modifications are not conflict-free, then at least one conflict occurs which can be of the following kinds: (1) *delete-delete conflict*: both modifications delete the same graph element, or (2) *delete-insert conflict*: m_1 deletes a node which shall be source or target of a new edge inserted by m_2 (or vice versa). Of course, several of such conflicts may occur simultaneously. In [11], we propose a *merge construction* that resolves conflicts by giving *insertion* priority over *deletion* in case of delete-insert conflicts. The result is a merged graph modification where the changes of both original graph modifications are realized as far as possible[2] We call this construction *tentative* merge because usually the modeler is asked to finish the conflict resolution manually, e.g. by opting for deletion instead of insertion of certain conflicting elements. The resolution strategy to prioritize insertion over deletion preserves all model elements that are

[2] Note that the conflict-free case is a special case of the tentative merge construction.

parts of conflicts and allows to highlight these elements to the user to support manual conflict resolution. We summarize the main effects of the conflict resolution strategy by Thm. 4.1 below (see also Thm. 3 in [11] for the construction).

Fact 4.1 (Conflict Resolution by Tentative Merge Construction). *Given two conflicting graph modifications* $m_i = G \overset{D_i}{\Longrightarrow} H_i$ $(i = 1,2)$ *(i.e., they are not conflict-free). The tentative merge construction yields the merged graph modification* $m = (G \leftarrow \overline{D} \rightarrow H)$ *and resolves conflicts as follows:*

1. *If* (m_1, m_2) *are in* delete-delete conflict, *with both* m_1 *and* m_2 *deleting* $x \in G$*, then* x *is deleted by* m*.*
2. *If* (m_1, m_2) *are in* delete-insert conflict, *there is an edge* e_2 *created by* m_2 *with* $x = s(e_2)$ *or* $x = t(e_2)$ *preserved by* m_2*, but deleted by* m_1*. Then* x *is preserved by* m *(and vice versa for* (m_2, m_1) *being in delete-insert conflict).*

Note that attributed nodes which shall be deleted on the one hand and change their values on the other hand would cause delete/insert-conflicts and therefore, would not be deleted by the tentative merge construction. Attributes which are differently changed by both modifications would lead (tentatively) to attributes with two values which would cause conflicts to be solved by the modeller, since an attribute is not allowed to have more than one value at a particular time. Throughout the paper, we depict conflict resolution based on the tentative merge construction and manual modifications as shown to the right, where m_1 and m_2 are conflicting graph modifications, and H is their merge after conflict resolution. The dashed lines correspond to derived graph modifications $(G_1 \leftarrow D_3 \rightarrow H)$ and $(G_2 \leftarrow D_4 \rightarrow H)$ with interfaces D_3 and D_4.

$$\begin{tikzcd} G_0 \arrow[r, "m_1"] \arrow[d, "m_2"'] \arrow[dr, Rightarrow, ":\mathsf{Res}"] & G_1 \arrow[d, dotted] \\ G_2 \arrow[r, dotted] & H \end{tikzcd}$$

Example 4.2 (Conflict resolution by tentative merge construction). Consider the conflict resolution square 3:Res in the upper right part of Fig. 8. The first modification $d^T_{1,F}$ deletes the node for Bill Clinton and updates the attribute values for Surname and Salary of Melinda French. The second modification d^T_1 relinks Bill Clinton's node from the marketing department to the technical department and updates his Salary attribute. The result of the tentative merge construction keeps the Bill Clinton node, due to the policy that nodes that are needed as source or target for newly inserted edges or attributes will be preserved. Technically, the attribute values are not preserved automatically. This means that the tentative merge construction only yields the structure node of "Bill Clinton" (and the updated attribute), and the modeller should confirm that the remaining attribute values should be preserved (this is necessary for the attribute values for FirstName, LastName and Birth of the "Bill Clinton" node).

Variant: As a slight variant to the above example, let us consider the case that modification d^T_1 also modifies Melinda's surname from "French" to "Smith". Since the same attribute is updated differently by both modifications, we now have two tentative attribute values for this attribute (we would indicate this by <Gates|French> as attribute value for Melinda's Surname attribute). This can be solved by the modeller, as well, who should select one attribute value.

5 Concurrent Model Synchronization with Conflict Resolution

The merge construction described in Sec. 4 cannot be applied directly to detect and solve conflicts in concurrent model synchronization. The problem here is that source and target updates occur in different graphs and not the same one. To solve this problem we use forward and backward propagation operations (Sec. 3) allowing us to see the effects of each source or target update on the other domain, so that we can apply the merge construction. In addition, we use two further operations CCS and CCT to reduce a given domain model to a maximal consistent submodel according to the TGG.

Given a source update $d_1^S : G_0^S \to G_1^S$, the consistency creating operation CCS (left part of Fig. 6) computes a maximal consistent subgraph $G_{1,C}^S \in VL_S$ of the given source model G_1^S. The resulting update from G_0^S to G_1^S is derived by update composition $d_{1,C}^S \circ d_1^S$. The dual operation CCT (right part of Fig. 6) works analogously on the target component.

$$\begin{tikzcd}
G_1^S \arrow[dr, dotted, bend right, "d_{1,C}^S"'] \arrow[dr, phantom, shift left=2, "\Downarrow{:}\mathsf{CCS}"] & G_0^S \arrow[l, "d_1^S"'] \arrow[d, dashed, "d_{1,C}^S \circ d_1^S"] \\
 & G_{1,C}^S
\end{tikzcd}
\qquad
\begin{tikzcd}
G_0^T \arrow[r, "d_1^T"] \arrow[d, dashed, "d_{1,C}^T \circ d_1^T"'] & G_1^T \arrow[dl, dotted, bend left, "d_{1,C}^T"] \arrow[dl, phantom, shift right=2, "\Downarrow{:}\mathsf{CCT}"] \\
G_{1,C}^T &
\end{tikzcd}$$

Fig. 6. Consistency creating operations

Remark 5.1 (Execution of Consistency Creating Operation CCS*).* Given a source model G_1^S, the consistency creating operation CCS is executed by computing terminated forward sequences $(H_0 \overset{tr_F^*}{\Longrightarrow} H_n)$ with $H_0 = (G_1^S \leftarrow \varnothing \rightarrow \varnothing)$. If the sets of operational rules of the TGG are deterministic (see Thm. 3.1), then backtracking is not necessary. If G_1^S is already consistent, then $G_{1,C}^S = G_1^S$, which can be checked via operation CCS. Otherwise, operation CCS is creating a maximal consistent subgraph $G_{1,C}^S$ of G_1^S. $G_{1,C}^S$ is maximal in the sense that there is no larger consistent submodel H^S of G_1^S, i.e. with $G_{1,C}^S \subseteq H^S \subseteq G_1^S$ and $H^S \in VL_S$. From the practical point of view, operation CCS is performed using forward translation rules [16], which mark in each step the elements of a given source model that have been translated so far. This construction is well defined due to the equivalence with the corresponding triple sequence $(\varnothing \overset{tr^*}{\Longrightarrow} H_n)$ via the triple rules TR of the TGG (see App. B in [10]).

The concurrent model synchronization operation CSynch derived from the given TGG is executed in five steps. Moreover, it combines operations fSynch and bSynch depending on the order in which the steps are performed. The used propagation operations fPpg, bPpg are required to be correct and we can take the derived propagation operations according to [17]. The steps of operation fSynch are depicted in Fig. 7 and Thm. 5.2 describes the steps for both operations.

Construction 5.2 (Operation fSynch and CSynch). *In the first step (operation* CCS*), a maximal consistent subgraph $G_{1,C}^S \in VL_S$ of G_1^S is computed (see Thm. 5.1). In step 2, the update $d_{1,CC}^S$ is forward propagated to the target domain via operation* fPpg*. This leads to the pair $(r_{1,F}, d_{1,F}^T)$ and thus, to the pair $(d_{1,F}^T, d_1^T)$ of target updates, which may show conflicts. Step 3 applies the conflict resolution operation* Res *including optional manual modifications (see Sec. 4). In order to ensure consistency of the resulting target model $G_{2,FC}^T$ we apply the consistency creating operation* CCT *(see Thm. 5.1) for the*

Signature	$$\begin{tikzcd} G^S_1 \arrow[d, dotted, "d^S_2"'] & G^S_0 \arrow[l, "d^S_1"'] \arrow[r, leftrightarrow, "r_0"] \arrow[dr, phantom, "\Downarrow:\mathsf{fSynch}"] & G^T_0 \arrow[r, "d^T_1"] & G^T_1 \arrow[d, dotted, "d^T_2"] \\ G^S_2 \arrow[rrr, dotted, leftrightarrow, "r_2"'] & & & G^T_2 \end{tikzcd}$$
Definition of Components	$$\begin{tikzcd} G^S_1 \arrow[dr, dotted, "d^S_F"'] \arrow[ddr, dashed, "d^S_{2,FCB}"'] & G^S_0 \arrow[l, "d^S_1"'] \arrow[r, leftrightarrow, "r_0"] \arrow[d, dotted, "d^S_{1,CC}"] \arrow[dl, phantom, "\searrow 1{:}\mathsf{CCS}"] \arrow[dr, phantom, "\searrow 2{:}\mathsf{fPpg}"] & G^T_0 \arrow[r, "d^T_1"] \arrow[d, dotted, "d^T_{1,F}"] \arrow[dr, phantom, "\searrow 3{:}\mathsf{Res}"] & G^T_1 \arrow[d, dotted, "d^T_{2,FC}"] \\ & G^S_{1,C} \arrow[r, dotted, leftrightarrow, "r_{1,F}"] \arrow[d, dotted, "d^S_{2,CB}"'] \arrow[dr, phantom, "\swarrow 5{:}\mathsf{bPpg}"] & G^T_{1,F} \arrow[r, dotted, "d'^T_{2,FC}"] \arrow[d, dotted, "d^T_{2,CC}"] \arrow[dr, phantom, "\swarrow 4{:}\mathsf{CCT}"] & G^T_{2,FC} \arrow[dl, dotted, "d^T_B"] \\ & G^S_{2,FCB} \arrow[r, dotted, leftrightarrow, "r_{2,FCB}"'] & G^T_{2,FCB} & \end{tikzcd}$$ $d^S_{2,FCB} = d^S_{2,CB} \circ d^S_F, d^T_{2,FCB} = d^T_B \circ d^T_{2,FC}, \quad (r_2, d^S_2, d^T_2) = (r_{2,FCB}, d^S_{2,FCB}, d^T_{2,FCB})$

Fig. 7. Concurrent model synchronization with conflict resolution (forward case: fSynch)

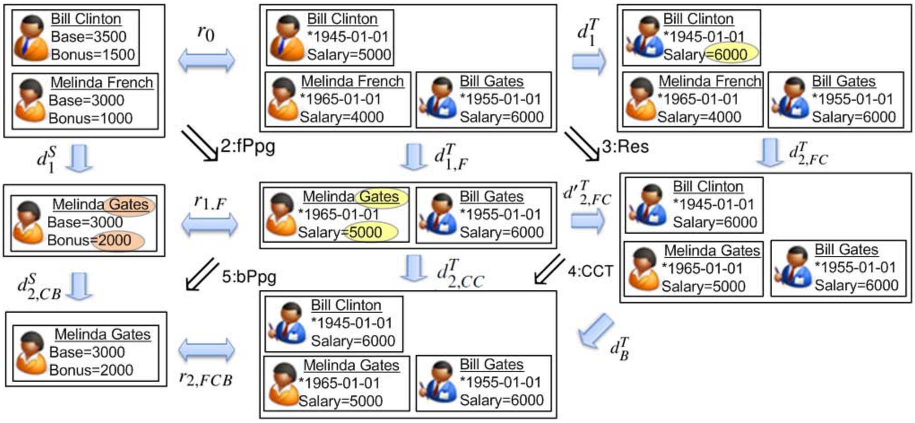

Fig. 8. Concurrent model synchronization with conflict resolution applied to organizational model

target domain and derive target model $G^T_{2,FCB} \in VL_T$ in step 4. Finally, the derived target update $d^T_{2,CC}$ is backward propagated to the source domain via operation bPpg *leading to the source model $G^S_{2,FCB}$ and source update $d^S_{2,CB}$. Altogether, we have constructed a nondeterministic solution (r_2, d^S_2, d^T_2) of operation* fSynch *for the premise (r_0, d^S_1, d^T_1) with $(r_2, d^S_2, d^T_2) = (r_{2,FCB}, d^S_{2,FCB}, d^T_{2,FCB})$ (see Fig. 7). The concurrent synchronization operation* bSynch *is executed analogously via the dual constructions. Starting with* CCT *in step 1, it continues via* bPpg *in step 2,* Res *in step 3,* CCS *in step 4, and finishes with* fPpg *in step 5. The non-deterministic operation* CSynch = (fSynch ∪ bSynch) *is obtained by joining the two concurrent synchronizations operations* fSynch bSynch.

Example 5.3 (Concurrent Model Synchronization with Conflict Resolution). The steps in Fig. 8 specify the execution of the concurrent synchronization in Thm. 2.1. Since the given model G^S_0 is consistent, step 1 (1:CCS) can be omitted, i.e. $G^S_{1,C} = G^S_1$ and $d^S_{1,CC} = d^S_1$. Step 2:fPpg propagates the source update to the target domain: Melinda Gates' attributes are updated and the node representing Bill Clinton is deleted. The resolution 3:Res resolves the conflict between the target model update d_1^T and the propagated source model update on the target side $d_{1,F}^T$ (see Thm. 4.2). We assume that the modeler selected the old attribute value for Bill Clinton's birthday. Step 4:CCT does not change anything, since the model is consistent already. Finally, all elements that were introduced during the conflict resolution and concern the source domain are propagated to the source model via (5:bPpg). This concerns only the Bill Clinton node, which now is assigned to the technical department. According to the TGG, such persons are not reflected in the source model, such that the backward propagation does not change anything in the source model. The result of the concurrent model synchronization with conflict resolution is $r_{2,FCB}$, where as many as possible of both proposed update changes have been kept and insertion got priority over deletion.

Variant: Let us consider the case that both modifications d_1^T $d_{1,F}^T$ insert additionally an edge of type married between the nodes of Melinda French and Bill Gates. The conflict resolution operation 3:Res would yield two married edges between the two nodes. But the subsequent consistency creating operation 4:CCT would detect that this is an inconsistent state and would delete one of the two married edges.

Remark 5.4 (Execution and Termination of Concurrent Model Synchronization). Note that the efficiency of the execution of the concurrent synchronization operations can be significantly improved by reusing parts of previously computed transformation sequences as described in App. B in [10]. In [17], we provided sufficient static conditions that ensure termination for the propagation operations and they can be applied similarly for the consistency creating operations. Update cycles cannot occur, because the second propagation step does not lead to a new conflict.

Note that operation CSynch is nondeterministic for several reasons: the choice between fSynch and bSynch, the reduction of domain models to maximal consistent sub graphs, and the semi automated conflict resolution strategy.

Definition 5.5 (Derived Concurrent TGG Synchronization Framework). *Let* fPpg *and* bPpg *be correct basic synchronization operations for a triple graph grammar TGG and let operation* CSynch *be derived from* fPpg *and* bPpg *according to Thm. 5.2. Then, the* derived concurrent TGG synchronization framework *is given by CSynch =* (TGG, CSynch).

6 Correctness and Compatibility

Our main results show correctness of the derived concurrent TGG synchronization framework (Thm. 5.5) and its compatibility with the derived basic TGG synchronization framework (Sec. 3). For the proofs and technical details see App. A and B in [10]. Correctness of a concurrent model synchronization framework requires that the nondeterministic synchronization operation CSynch ensures laws (a) and (b) in Thm. 2.3. In other words, CSynch guarantees consistency of the resulting integrated model and, moreover, the synchronization of an unchanged and already consistently integrated model always yields the identity of the input as output (law (a)).

$$G_1^S \in VL_S,\ \begin{tikzcd} G_0^S \arrow[r, leftrightarrow, "r_0"] \arrow[d, "d^S"'] \arrow[dr, Rightarrow, ":\mathsf{fPpg}"] & G_0^T \arrow[d, "d^T"] \\ G_1^S \arrow[r, leftrightarrow, "r_1"'] & G_1{}^T \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} G_1^S \arrow[d, "id"'] & G_0^S \arrow[l, "d^S"'] \arrow[r, leftrightarrow, "r_0"] \arrow[d, phantom, "\Downarrow:\mathsf{CSynch}"] & G_0^T \arrow[r, "id"] & G_0^T \arrow[d, "d^T"] \\ G_1^S \arrow[rrr, leftrightarrow, "r_1"'] & & & G_1^T \end{tikzcd}$$

Fig. 9. Compatibility with synchronization of single updates (forward case)

According to Thm. 6.2 below, correctness of given forward and backward propagation operations ensures correctness of the concurrent model synchronization framework.

Example 6.1 (Correctness and Compatibility). In [17], we presented a suitable realization of a correct propagation operations derived from the given TGG (see Thm. 3.1). This allows us to apply the following main results Thm. 6.2 and 6.4 to our case study used as running example in Sec. 2-6.

Theorem 6.2 (Correctness of Concurrent Model Synchronization). *Let* fPpg *and* bPpg *be correct basic synchronization operations for a triple graph grammar TGG. Then, the derived concurrent TGG synchronization framework CSynch* = (TGG, CSynch) *(see Thm. 5.5) is correct (see Thm. 2.3).*

The second main result (Thm. 6.4 below) shows that the concurrent TGG synchronization framework is compatible with the basic synchronization framework. This means that the propagation operations (fPpg, bPpg) (see Sec. 3) provide the same result as the concurrent synchronization operation CSynch, if one update of one domain is the identity. Fig. 9 visualizes the case for the forward propagation operation fPpg. Given a forward propagation (depicted left) with solution (r_1, d^T), then a specific solution of the corresponding concurrent synchronization problem (depicted right) is given by $sol = (r_1, id, d^T)$, i.e. the resulting integrated model and the resulting updates are the same. Due to the symmetric definition of TGGs, we can show the same result concerning the backward propagation operation leading to the general result of compatibility in Thm. 6.4.

Definition 6.3 (Compatibility of Concurrent with Basic Model Synchronization). *Let* fPpg, bPpg *be basic TGG synchronization operations and let* CSynch *be a concurrent TGG synchronization operation for a given TGG. The non-deterministic synchronization operation* CSynch *is* compatible *with the propagation operations* fPpg *and* bPpg, *if the following condition holds for the forward case (see Fig. 9) and a similar one for the backward case:*

$$\forall\, (d^S, r_0) \in \Delta_S \otimes Rel,\ with\ (d^S : G_0^S \to G_1^S) \wedge (G_1^S \in VL_S):$$
$$(id, \mathsf{fPpg}(d^S, r_0)) \in \mathsf{CSynch}(d^S, r_0, id)$$

Theorem 6.4 (Compatibility of Concurrent with Basic Model Synchronization). *Let* fPpg *and* bPpg *be correct basic synchronization operations for a given TGG and let operation* CSynch *be derived from* fPpg *and* bPpg *according to Thm. 5.2. Then, the* derived concurrent TGG synchronization operation CSynch *is compatible with propagation operations* fPpg, bPpg.

7 Related Work

Triple graph grammars have been successfully applied in several case studies for bidirectional model transformation, model integration and synchronization [20,25,14] and for the implementation of QVT [15]. Several formal results are available concerning correctness, completeness, termination, functional behavior [16,13] and optimization wrt. the efficiency of their execution [16,21]. The presented approach to concurrent model synchronization is based on these results and concerns model synchronization of concurrent updates including the resolution of possible merging conflicts.

Egyed et. al [7] discuss challenges and opportunities for change propagation in multiple view systems based on model transformations concerning consistency (correctness and completeness), partiality, and the need for bidirectional change propagation and user interaction. Our presented approach based on TGGs reflects these issues. In particular, TGGs automatically ensure consistency for those consistency constraints that can be specified with a triple rule. This means that the effort for consistency checking with respect to domain language constraints is substantially reduced.

Stevens developed an abstract state-based view on symmetric model synchronization based on the concept of constraint maintainers [26], and Diskin described a more general delta-based view within the *tile algebra* framework [4,6]. These tile operations inspired the constructions for the basic synchronization operations [17], which are used for the constructions in the present paper. Concurrent updates are a central challenge in multi domain modeling as discussed in [28], where the general idea of combining propagation operations with conflict resolution is used as well. However, the paper does not focus on concrete propagation and resolution operations and requires that model updates are computed as model differences. The latter can lead to unintended results by hiding the insertion of new model elements that are similar to deleted ones.

Merging of model modifications usually means that non-conflicting parts are merged automatically, while conflicts have to be resolved manually. A survey on model versioning approaches and on (semi-automatic) conflict resolution strategies is given in [1]. A category-theoretical approach formalizing model versioning is given in [23]. Similar to our approach, modifications are considered as spans of morphisms to describe a partial mapping of models, and merging of model changes is based on pushout constructions. In contrast to [23], we consider an automatic conflict resolution strategy according to [11] that is formally defined.

8 Conclusion and Future Work

This paper combines two main concepts and results recently studied in the literature. On the one hand, basic model synchronization based on triple graph grammars (TGGs) has been studied in [17], where source model modifications can be updated to target model modifications and vice versa. On the other hand, a formal resolution strategy for conflicting model modifications has been presented in [11]. The main new contribution of this paper is the formal concept of concurrent model synchronization together with a correct procedure to implement it, where source and target modifications have to be synchronized simultaneously, which includes conflict resolution of different source or target modifications. The main results concerning correctness and compatibility of basic and concurrent model synchronization are based on the formalization of bidirectional model transformations in the framework of TGGs [24,9,16] and the results in [17,11].

In future work, we plan to develop extended characterizations of the correctness and maximality criteria of a concurrent synchronization procedure. In this paper, correctness is defined explicitly in terms of the two laws formulated in Sec. 3 and, implicitly, in terms of the properties of compatibility with basic model synchronization proven in Thm. 6.4. We think that this can be strengthened by relating correctness of a synchronization procedure with the total or partial *realization* of the given source and target updates, for a suitable notion of realization. At a different level, we also believe that studying in detail, both from theoretical and practical viewpoints, the combination of fSynch and bSynch operations, discussed in Sec. 5, should also be a relevant matter. Finally, we also consider the possibility of taking a quite different approach for defining concurrent synchronization. In the current paper, our solution is based on implementing synchronization in terms of conflict resolution and the operations of forward and backward propagation. A completely different approach would be to obtain synchronization by the application of transformation rules, derived from the given TGG, that simultaneously implement changes associated to the source and target modifications. In particular, it would be interesting to know if the two approaches would be equally powerful, and which of them could give rise to a better implementation, on which we are working on the basis of the EMF transformation tool Henshin [2].

References

1. Altmanninger, K., Seidl, M., Wimmer, M.: A survey on model versioning approaches. IJWIS 5(3), 271–304 (2009)
2. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010. LNCS, vol. 6394, pp. 121–135. Springer, Heidelberg (2010)
3. Barbosa, D.M.J., Cretin, J., Foster, N., Greenberg, M., Pierce, B.C.: Matching lenses: alignment and view update. In: Proc. Int. Conf. on Functional Programming (ICFP 2010), pp. 193–204. ACM (2010)
4. Diskin, Z.: Model Synchronization: Mappings, Tiles, and Categories. In: Fernandes, J.M., Lämmel, R., Visser, J., Saraiva, J. (eds.) GTTSE 2009. LNCS, vol. 6491, pp. 92–165. Springer, Heidelberg (2011)
5. Diskin, Z., Xiong, Y., Czarnecki, K.: From state- to delta-based bidirectional model transformations: the asymmetric case. Journal of Object Technology 10, 6:1–6:25 (2011)
6. Diskin, Z., Xiong, Y., Czarnecki, K., Ehrig, H., Hermann, F., Orejas, F.: From State- to Delta-Based Bidirectional Model Transformations: The Symmetric Case. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 304–318. Springer, Heidelberg (2011)
7. Egyed, A., Demuth, A., Ghabi, A., Lopez-Herrejon, R., Mäder, P., Nöhrer, A., Reder, A.: Fine-Tuning Model Transformation: Change Propagation in Context of Consistency, Completeness, and Human Guidance. In: Cabot, J., Visser, E. (eds.) ICMT 2011. LNCS, vol. 6707, pp. 1–14. Springer, Heidelberg (2011)

8. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theor. Comp. Science. Springer, Heidelberg (2006)
9. Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information Preserving Bidirectional Model Transformations. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 72–86. Springer, Heidelberg (2007)
10. Ehrig, H., Ermel, C., Hermann, F., Orejas, F.: Concurrent model synchronization with conflict resolution based on triple graph grammars - extended version. Tech. Rep. TR 2011-14, TU Berlin, Fak. IV (2011)
11. Ehrig, H., Ermel, C., Taentzer, G.: A Formal Resolution Strategy for Operation-Based Conflicts in Model Versioning Using Graph Modifications. In: Giannakopoulou, D., Orejas, F. (eds.) FASE 2011. LNCS, vol. 6603, pp. 202–216. Springer, Heidelberg (2011)
12. Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C., Schmitt, A.: Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. ACM Trans. Program. Lang. Syst. 29(3) (2007)
13. Giese, H., Hildebrandt, S., Lambers, L.: Toward Bridging the Gap Between Formal Semantics and Implementation of Triple Graph Grammars. Tech. Rep. 37, Hasso Plattner Institute at the University of Potsdam (2010)
14. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Software and Systems Modeling 8, 21–43 (2009)
15. Greenyer, J., Kindler, E.: Comparing relational model transformation technologies: implementing query/view/transformation with triple graph grammars. Software and Systems Modeling (SoSyM) 9(1), 21–46 (2010)
16. Hermann, F., Ehrig, H., Golas, U., Orejas, F.: Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars. In: Proc. Int. Workshop on Model Driven Interoperability (MDI 2010), pp. 22–31. ACM (2010)
17. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of Model Synchronization Based on Triple Graph Grammars. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 668–682. Springer, Heidelberg (2011)
18. Hofmann, M., Pierce, B.C., Wagner, D.: Symmetric lenses. In: Proc. ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2011), pp. 371–384. ACM (2011)
19. Hu, Z., Mu, S.C., Takeichi, M.: A programmable editor for developing structured documents based on bidirectional transformations. Higher-Order and Symbolic Computation 21(1-2), 89–118 (2008)
20. Kindler, E., Wagner, R.: Triple graph grammars: Concepts, extensions, implementations, and application scenarios. Tech. Rep. TR-ri-07-284, Dept. of Comp. Science, Univ. Paderborn, Germany (2007)
21. Klar, F., Lauder, M., Königs, A., Schürr, A.: Extended Triple Graph Grammars with Efficient and Compatible Graph Translators. In: Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., Westfechtel, B. (eds.) Nagl Festschrift. LNCS, vol. 5765, pp. 141–174. Springer, Heidelberg (2010)
22. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Version 1.0 formal/08-04-03, `http://www.omg.org/spec/QVT/1.0/`
23. Rutle, A., Rossini, A., Lamo, Y., Wolter, U.: A Category-Theoretical Approach to the Formalisation of Version Control in MDE. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 64–78. Springer, Heidelberg (2009)
24. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 151–163. Springer, Heidelberg (1995)

25. Schürr, A., Klar, F.: 15 Years of Triple Graph Grammars Research Challenges, New Contributions, Open Problems. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 411–425. Springer, Heidelberg (2008)
26. Stevens, P.: Bidirectional model transformations in qvt: semantic issues and open questions. Software and System Modeling 9(1), 7–20 (2010)
27. TFS-Group, TU Berlin: AGG (2011), http://tfs.cs.tu-berlin.de/agg
28. Xiong, Y., Song, H., Hu, Z., Takeichi, M.: Synchronizing concurrent model updates based on bidirectional transformation. Software and Systems Modeling, 1–16 (2011)