Technische Universität Berlin

Forschungsberichte
der Fakultät IV – Elektrotechnik und Informatik

Concurrent Model Synchronization with Conflict Resolution Based on Triple Graph Grammars - Extended Version

Frank Hermann, Hartmut Ehrig, Claudia Ermel,
and Fernando Orejas

Bericht-Nr. 2011-14
ISSN 1436-9915

Concurrent Model Synchronization with Conflict Resolution Based on Triple Graph Grammars - Extended Version

Frank Hermann [1,2], Hartmut Ehrig [1], Claudia Ermel [1], and Fernando Orejas [3]
1) Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany,
{hartmut.ehrig, claudia.ermel, frank.hermann}@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(at)lsi.upc.edu

Abstract

Triple graph grammars (TGGs) have been used successfully to analyse correctness of bidirectional model transformations. Most recently, also a corresponding formal approach to model synchronization has been presented, where a forward propagation operation updates a source model modification from source to target, and symmetrically, a backward propagation operation takes care of updates from target to source models. 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 main result, we show correctness of concurrent model synchronization with respect to the TGG. This means that 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 main result, we show compatibility of concurrent with basic model synchronization. In other words, concurrent model synchronization can be realized either to coincide with forward or with backward propagation. The main 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 [21]). Triple graph grammars (TGGs) have been successfully applied in several case studies for bidirectional model transformation, model integration and synchronization [19, 24, 13] and for the implementation of QVT [14]. Based on the work of Schürr et al. [23, 24], we developed a formal theory of TGGs [9, 15], which allows handling correctness, completeness, termination and functional behaviour of model transformations. Inspired by existing synchronization tools [13] and the replica synchronization framework in [4], we proposed an approach for basic model synchronization in [16], showing its correctness. More precisely, in that paper

we studied the problem of how updates on a given domain can be correctly propagated to another model.

The main 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 the conflicts between the given updates. In particular, these conflicts may be not obvious to detect, since they may be caused by concurrent updates on apparently unrelated elements of the given models. On the other hand, 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 [16] and tentative merge constructions solving conflicts [10]. 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 correct solutions to the given conflicts. The differences between the possible solutions can be visualized by 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 App. A-C.

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]

Example 1 (Concurrent model synchronization problem)**.** *The example in Fig. 1 starts with two models in correspondence. Each of them stores information about employees of a company, but they concern different aspects. The source model contains information only about the employees of the marketing department, but shows more detailed salary information. Two model updates have to be*

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

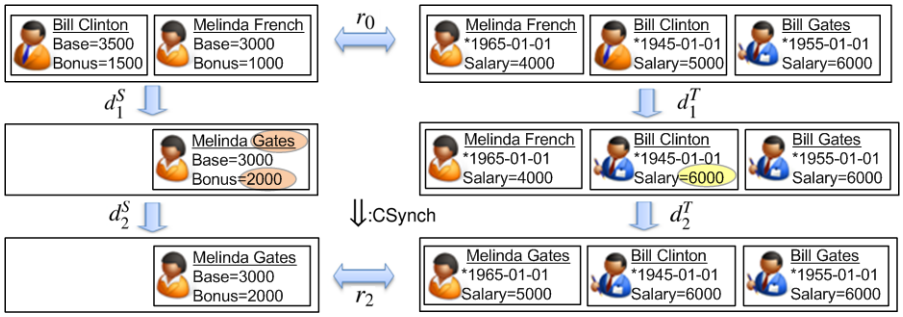


Figure 1: Concurrent model synchronization: compact example

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 to Bill Gates; 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 department to the technical department (in the visualization in Fig. 1 this is indicated by the role icon of Bill Clinton which now shows a man carrying a pencil). His department change is combined with a raise of his salary 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 [23, 9]. TGGs have been applied successfully for bidirectional model transformations [24, 15] and basic model synchronization [13, 16], 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 ($\leftrightarrow$) as an equivalent shorter notation for triple graphs, whenever the explicit correspondence graph can be omitted.

$$\begin{array}{ccccc} G = (G^S & \xleftarrow{s_G} & G^C & \xrightarrow{t_G} & G^T) \\ \downarrow{\scriptstyle m} & & \downarrow{\scriptstyle m^S} & & \downarrow{\scriptstyle m^C} & & \downarrow{\scriptstyle m^T} \\ H = (H^S & \xleftarrow[s_H]{} & H^C & \xrightarrow[t_H]{} & H^T) \end{array}$$

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.

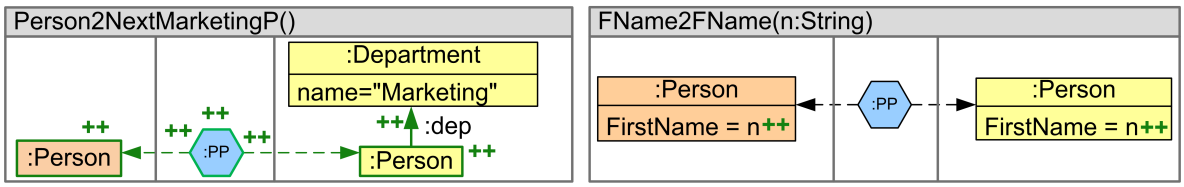
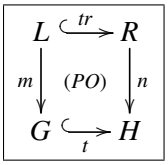


Figure 2: Two triple rules of the TGG



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 [15].

Example 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 "++". The first 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 similar rule (not depicted) for initially creating the marketing department together with one person, where an additional NAC ensures that none of the existing departments is called "Marketing". The second rule in Fig. 2 extends two corresponding persons by their first names. There are further similar rules for the handling of the remaining attributes. In particular, the rule for the 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 $VL_S = \{G^S \mid (G^S \leftarrow G^C \rightarrow G^T) \in VL(TGG)\}$ and $VL_T = \{G^T \mid (G^S \leftarrow G^C \rightarrow G^T) \in VL(TGG)\}$ 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* $d = (G \overset{i_1}{\leftarrow} I \overset{i_2}{\rightarrow} G')$. The relating 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 $d_1^S = (G_0^S \rightarrow G_1^S)$ and $d_1^T = (G_0^T \rightarrow G_1^T)$, the required result consists of updates $d_2^S = (G_1^S \rightarrow G_2^S)$ and $d_2^T = (G_1^T \rightarrow G_2^T)$ 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 Def. 1 below. The set of inputs is

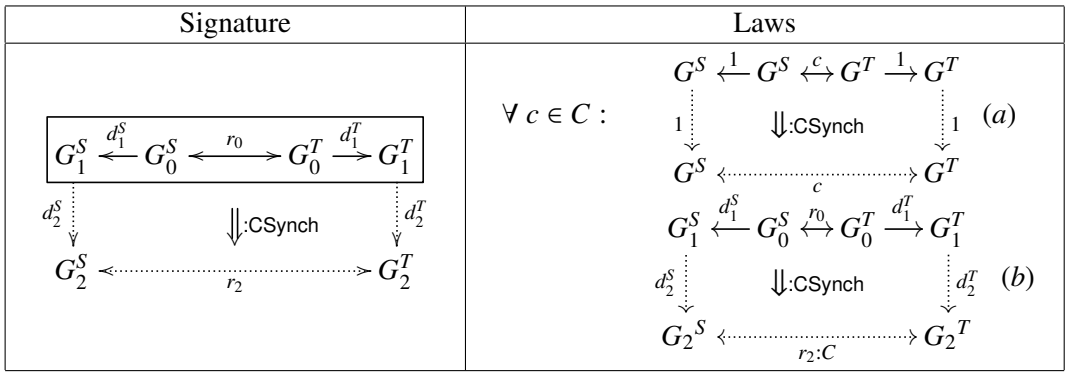


Figure 3: Signature and laws for correct concurrent synchronization frameworks

given by $(R \otimes \Delta_S \otimes \Delta_T) = \{(r, d^S, d^T) \in R \times \Delta_S \times \Delta_T \mid r\colon G_0^S \leftrightarrow G_0^T, d^S : G_0^S \to G_2{}^S, d^T : G_0^T \to G_2{}^T\}$, i.e., r coincides with d^S on G_0^S and with d^T on G_0^T.

Definition 1 (Concurrent Synchronization Problem and Framework)**.** *Given TGG, the* concurrent synchronization problem *is to construct a left total and nondeterministic operation* 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* $(prem, sol) \in$ 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* $sol \in$ 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* $= (TGG,$ 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

This Section briefly describes the basic synchronization problem and its solution according to [16], which is the basis for the solution for the concurrent synchronization problem in Sec. 5.

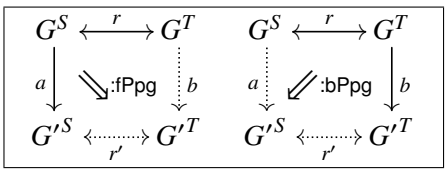


Figure 4: Propagation operations

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, [11, 18, 25, 3, 27, 17, 5, 6, 16]). Many of these approaches [11, 18, 25, 27] 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, 17, 6, 16]) are δ-based, meaning that the synchronization operations take modifications as parameters and returns modifications as results. In particular, in [16], we describe a framework based on TGGs, where we

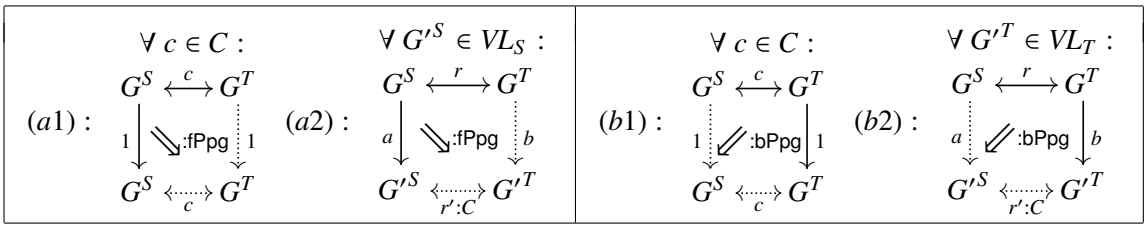


Figure 5: Laws for correct basic synchronization frameworks

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].

To be precise, according to [16], 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) $a : G^S \to G'^S$, and the output is a target update $b : G^T \to G'^T$ 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 $b : G^T \to G'^T$ and it returns a source update $a : G^S \to G'^S$ 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 [16], 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 $r\colon G^S \leftrightarrow G^T$ is not required to be consistent. Laws $(b1)$ and $(b2)$ are the dual versions concerning bPpg.

In [16], we also present specific propagation operations. More precisely, 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 $G'^S \leftrightarrow G^T$. It must be said that we do not build this model from scratch, but what we really do is to 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 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 *[16]. 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 [26] we refer to [16], 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 [16].*

4 Semi-Automated Conflict Detection and Resolution

We now review the main constructions and results for conflict resolution in one domain according to [10]. 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 $(G \leftarrow D_i \rightarrow H_i)$, $(i = 1, 2)$ 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 $(G \leftarrow D \rightarrow H)$ 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 [10], 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. We summarize the main effects of the conflict resolution strategy by Fact 1 below (see also Thm. 3 in [10] for the construction).

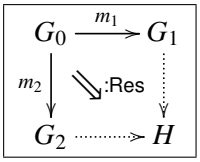
Fact 1 (Conflict resolution by tentative merge construction)**.** *Given two conflicting graph modifications $m_i = G \overset{D_i}{\Longrightarrow} H_i$ $(i = 1, 2)$ (i.e., they are not conflict-free). The tentative merge construction yields the merged graph modification $m = (G \leftarrow \overline{D}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



Example 3 (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*

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

Clinton and updates the attribute values for Surname *and* Salary *of Melinda French. The second modification* $d^T_{1,F}$ *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 the previous section cannot be applied directly to detect and solve conflicts in concurrent model synchronization. The problem here is that source and target updates occur at different graphs and not the same one. To solve this problem we use forward and backward propagation operations (Sec. 3) that allow us to see the effects of each source or target update on the other domain, such that we can apply the merge construction of Sec. 4. In addition, we use two further operations CCS and CCT for reducing a given domain model to a maximal consistent submodel according to the TGG.

Given a source update $d^S_1 : G^S_0 \to G^S_1$, the consistency creating operation CCS (left part of Fig. 6) computes a maximal consistent subgraph $G^S_{1,C} \in VL_S$ of the given source model G^S_1. The resulting update from G^S_0 to G^S_1 is derived by update composition $d^S_{1,C} \circ d^S_1$. The dual operation CCT (right part of Fig. 6) works analogously on the target component.

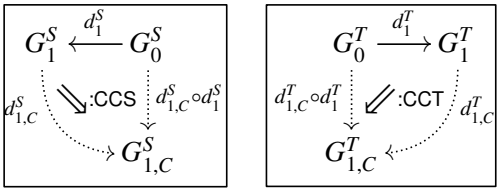


Figure 6: Consistency creating operations

Remark 2 (Execution of Consistency Creating Operation CCS)**.** *Given a source model* G^S_1*, the consistency creating operation* CCS *is executed by computing terminated forward sequences* $(H_0 \overset{tr^*_F}{\Longrightarrow} H_n)$ *with* $H_0 = (G^S_1 \leftarrow \varnothing \rightarrow \varnothing)$*. If the sets of operational rules of the TGG are deterministic (see Rem. 1), then backtracking is not necessary. If* G^S_1 *is already consistent, then* $G^S_{1,C} = G^S_1$*, which can be checked via operation* CCS*. Otherwise, operation* CCS *is creating a maximal consistent subgraph* $G^S_{1,C}$ *of* G^S_1*.* $G^S_{1,C}$ *is maximal in the sense that there is no larger consistent submodel* H^S *of* G^S_1*, i.e. with* $G^S_{1,C} \subseteq H^S \subseteq G^S_1$ *and* $H^S \in VL_S$*. From the practical point of view, operation* CCS *is performed using forward translation rules [15], 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 Construction 2).*

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

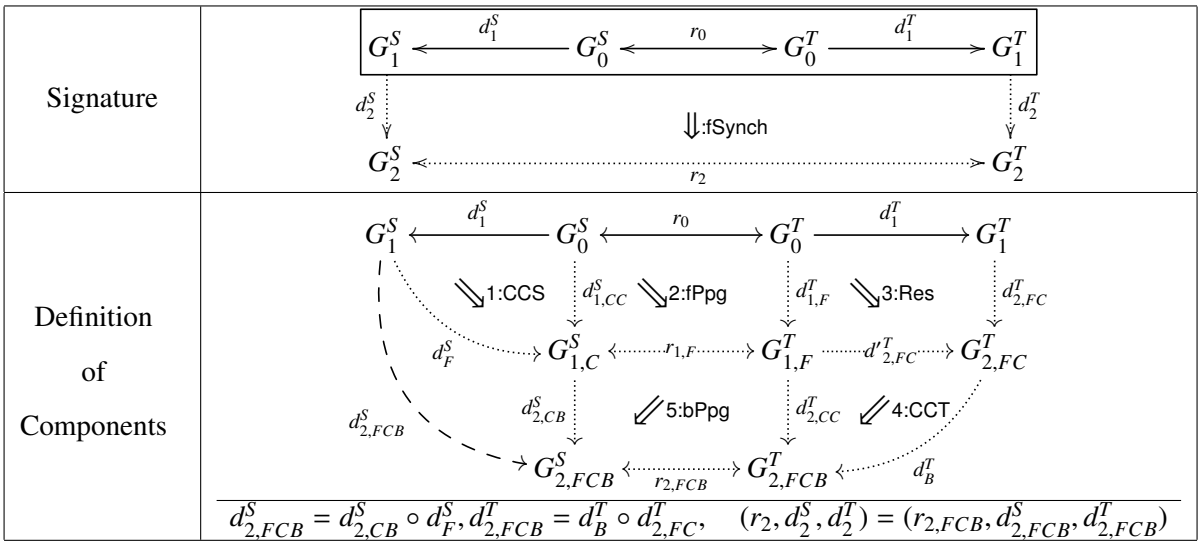


Figure 7: Concurrent model synchronization with conflict resolution (forward case: fSynch)

we can take the derived propagation operations according to [16]. The steps of operation fSynch are depicted in Fig. 7 and Construction 1 describes the steps for both operations.

Construction 1 (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 Rem. 2). 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 Rem. 2) for the target domain and derive target model $G_{2,FCB}^T \in VL_T$ in step 4. Finally, the derived target update $d_{2,CC}^T$ is backward propagated to the source domain via operation* bPpg *leading to the source model $G_{2,FCB}^S$ and source update $d_{2,CB}^S$. Altogether, we have constructed a nondeterministic solution (r_2, d_2^S, d_2^T) of operation* fSynch *for the premise (r_0, d_1^S, d_1^T) with $(r_2, d_2^S, d_2^T) = (r_{2,FCB}, d_{2,FCB}^S, d_{2,FCB}^T)$ (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 4 (Concurrent Model Synchronization with Conflict Resolution)**.** *The steps in Fig. 8 specify the execution of the concurrent synchronization in Ex. 1. Since the given model G_0^S is consistent, step 1 (*1:CCS*) can be omitted, i.e. $G_{1,C}^S = G_1^S$ and $d_{1,CC}^S = d_1^S$. 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 Ex. 3). 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*

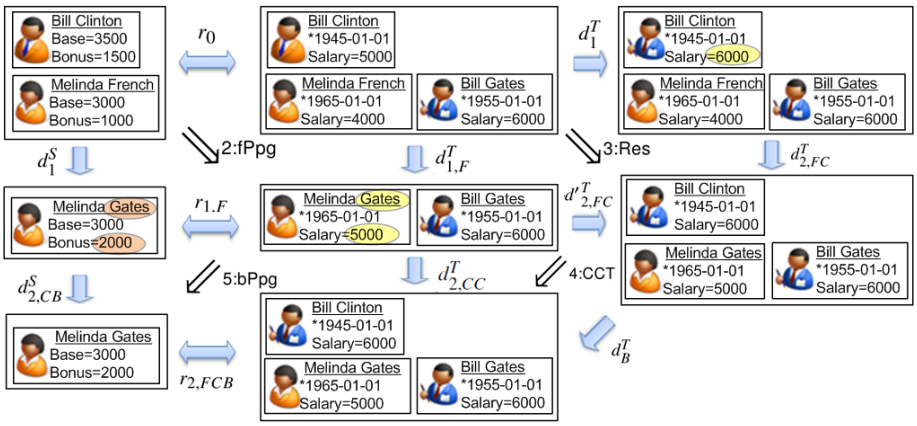


Figure 8: Concurrent model synchronization with conflict resolution applied to organizational model

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 3 (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 Rem. 5 in Appendix B. In [16], 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 2 (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 Construction 1. 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 (Def. 2) and its compatibility with the derived basic TGG synchronization framework (Sec. 3). For the proofs and technical details see A and B. Correctness of a concurrent model synchronization framework requires that the non-deterministic synchronization operation CSynch ensures laws (*a*) and (*b*) in Def. 1. 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*)).

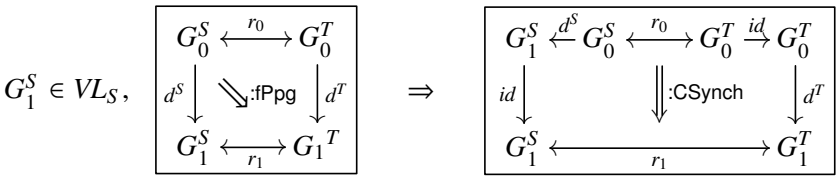


Figure 9: Compatibility with synchronization of single updates (forward case)

According to Thm. 1 below, correctness of the given forward and backward propagation operations already ensures correctness of the concurrent model synchronization framework.

Example 5 (Correctness and Compatibility)**.** *In [16], we presented a suitable realization of a correct propagation operations derived from the given TGG (see Rem. 1). This allows us to apply the following main results Thm. 1 and 2 to our case study used as running example in Sec. 2-6.*

Theorem 1 (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 Def. 2) is correct (see Def. 1).*

The second main result (Thm. 2 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. 2.

Definition 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:*

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

Theorem 2 (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 Construction 1. 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 [19, 24, 13] and for the implementation of QVT [14]. Several formal results are available concerning correctness, completeness, termination, functional behavior [15, 12] and optimization wrt. the efficiency of their execution [15, 20]. 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 [25], 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 [16], which are used for the constructions in the present paper. Concurrent updates are a central challenge in multi domain modeling as discussed in [27], 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 [22]. 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 [22], we consider an automatic conflict resolution strategy according to [10] 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 [16], 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 [10]. 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 [23, 9, 15] and the results in [16, 10].

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. 2. 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: Proc. MoDELS'10. LNCS, vol. 6394, pp. 121–135. Springer (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'10). pp. 193–204. ACM (2010)

[4] Diskin, Z.: Model Synchronization: Mappings, Tiles, and Categories. In: Generative and Transformational Techniques in Software Engineering III, LNCS, vol. 6491, pp. 92–165. Springer (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: 1–25 (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: Proc. MoDELS'11. LNCS, vol. 6981. Springer (2011)

[7] Egyed, A., Demuth, A., Ghabi, A., Lopez-Herrejon, R.E., Mäder, P., Nöhrer, A., Reder, A.: Fine-tuning model transformation: Change propagation in context of consistency, completeness, and human guidance. In: ICMT'11. LNCS, vol. 6707, pp. 1–14. Springer (2011)

[8] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theor. Comp. Science, Springer (2006)

[9] Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information preserving bidirectional model transformations. In: Proc. FASE'07. LNCS, vol. 4422, pp. 72–86. Springer (2007)

[10] Ehrig, H., Ermel, C., Taentzer, G.: A formal resolution strategy for operation-based conflicts in model versioning using graph modifications. In: Proc. FASE'11. LNCS, vol. 6603, pp. 202–216. Springer (2011)

[11] 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)

[12] 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)

[13] Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Software and Systems Modeling 8, 21–43 (2009)

[14] 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)

[15] 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'10). pp. 22–31. MDI '10, ACM (2010)

[16] Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of model synchronization based on triple graph grammars. In: Proc. Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS'11). LNCS, vol. 6981. Springer (2011)

[17] Hofmann, M., Pierce, B.C., Wagner, D.: Symmetric lenses. In: Proc. ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'11). pp. 371–384. ACM (2011)

[18] 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)

[19] 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)

[20] Klar, F., Lauder, M., Königs, A., Schürr, A.: Extended Triple Graph Grammars with Efficient and Compatible Graph Translators. In: Graph Transformations and Model Driven Enginering, LNCS, vol. 5765, pp. 141–174. Springer Verlag (2010)

[21] 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/ (2008)

[22] Rutle, A., Rossini, A., Lamo, Y., Wolter, U.: A Category-Theoretical Approach to the Formalisation of Version Control in MDE. In: Proc. of Int. Conf. on Fundamental Approaches to Software Engineering (FASE'09). LNCS, vol. 5503, pp. 64–78. Springer (2009)

[23] Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Int. Workshop on Graph-Theoretic Concepts in Computer Science. LNCS, vol. 903, pp. 151–163. Springer (1994)

[24] Schürr, A., Klar, F.: 15 years of triple graph grammars. In: Intern. Conf. on Graph Transformation (ICGT 2008). pp. 411–425 (2008)

[25] Stevens, P.: Bidirectional model transformations in qvt: semantic issues and open questions. Software and System Modeling 9(1), 7–20 (2010)

[26] TFS-Group, TU Berlin: AGG (2011), http://tfs.cs.tu-berlin.de/agg

[27] Xiong, Y., Song, H., Hu, Z., Takeichi, M.: Synchronizing concurrent model updates based on bidirectional transformation. Software and Systems Modeling p. 116 (2011)

A Proofs of Main Results

This appendix provides the proofs for the main theorems of this paper. For this purpose, we first review the correctness result for the basic TGG synchronization framework presented in [16].

As shown by Thm. 1 in [16] and stated below, the basic TGG synchronization framework derived from a TGG with deterministic sets of operational rules is correct. This means that we can use the derived forward and backward propagation operations for the derived TGG concurrent synchronization framework. The sets of operational rules TR_{OP} of a TGG are deterministic, if the corresponding forward and backward translation operations via TR_{FT} and TR_{BT} as well as the consistency creating operation via TR_{CC} are total functions. According to [15], we can check functional behaviour of these sets with the automated analysis engine for critical pairs of the tool AGG[3].

Fact 2 (Correctness of Basic TGG Model Synchronization Framework)**.** *Let TGG be a triple graph grammar with deterministic sets of operational rules. Then, the derived basic TGG synchronization framework is correct, i.e., it satisfies laws* $(a1)$ *-* $(b2)$ *in Fig. 5.*

Based on Fact 2 above shown in [16], we now show correctness of concurrent model synchronization by Thm. 1 below.

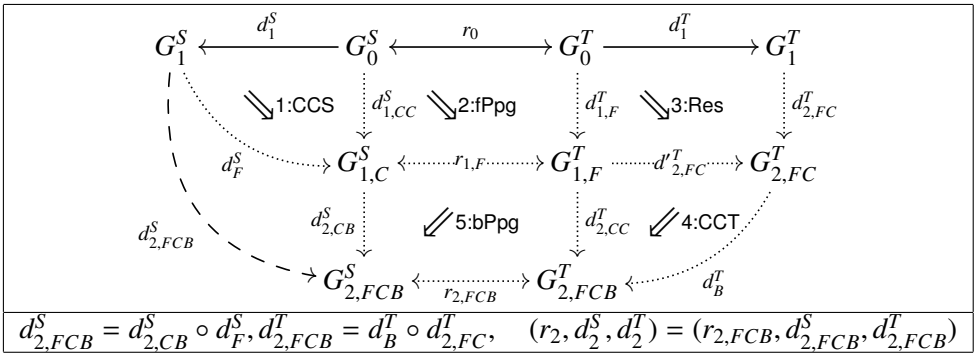
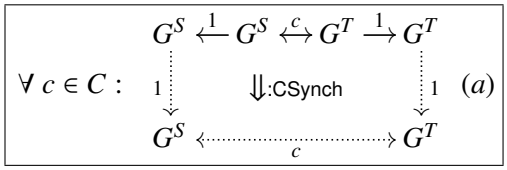


Figure 10: Concurrent model synchronization: operation fSynch (see Fig. 7)

Theorem 1 (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$)$ *is correct (see Def. 1).*

Proof.

Law (a) **in Fig. 3:** Let $(r, d_1^S, d_1^T) \in (R \otimes \Delta_S \otimes \Delta_T)$ with $r = c \in C = VL(TGG)$ and identities $d_1^S = id^S : G_0^S \to G_0^S$, $d_1^T = id^T : G_0^T \to G_0^T$, such that $G = (G^S \leftrightarrow G^T) = (G_0^S \leftrightarrow G_0^T) = G_0$. We have to show that operation CSynch yields (c, id^S, id^T) as result, i.e. no further result is possible.



We apply operation fSynch according to Fig. 7 and Fig. 10. Since $r = G_0 \in VL(TGG)$ and $G_1^S = G_0^S \in VL_S$ we know that there is a model transformation sequence $s_F = (G_1^S, G_0' \overset{tr_F^*}{\Longrightarrow} G_n', G_1^T)$ based on forward rules using the completeness result for model transformations based on forward rules

[3] See: TFS-Group, TU Berlin: AGG (2011), http://tfs.cs.tu-berlin.de/agg.

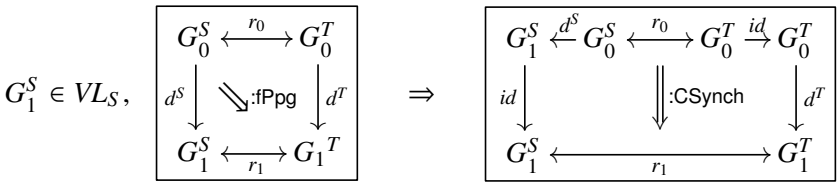
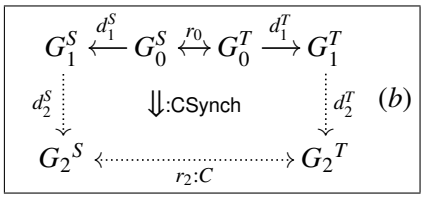


Figure 11: Compatibility with synchronization of single updates (forward case)

(Thm. 1 in [12]). Therefore, operation CCS yields the maximal consistent subgraph $G^S_{1,C} = G^S_0$ and update $d^S_{1,CC} = id^S$. By correctness of operation fPpg (law $(a1)$ in Fig. 5) we derive that the second step yields target model $G^T_{1,F} = G^T_0$, correspondence $r_{1,F} = r$ and target update $d^T_{1,F} = id^T$. Therefore, the resolution in step 3 concerns the updates $d^T_1 = id^T$ and $d^T_{1,F} = id^T$, which are parallel independent by definition leading to the merging result $G^T_{2,FC} = G^T_0$ and updates $d^T_{2,FC} = id^T$ and $d'^T_{2,FC} = id^T$. This means that manual modification is not necessary and therefore not executed. Again, since $G^T_0 \in VL_T$ we know that there is a model transformation sequence $s_B = (G^T_0, H'_0 \overset{tr^*_B}{\Longrightarrow} H'_n, G^S_1)$ based on backward translation rules using the dual version of the completeness result for model transformations based on forward rules (Thm. 1 in [12]). Therefore, operation CCT yields the maximal consistent subgraph $G^T_{2,FCB} = G^T_0$ and update $d^T_B = id^T$. By correctness of operation bPpg (law $(b1)$ in Fig. 5) we derive that the fifth step yields source model $G^S_{2,FCB} = G^S_0$, correspondence $r_{2,FCB} = r_{1,F} = r = c$ and source update $d^S_{2,CB} = id^S$. All together, operation fSynch yields the result (c, id^S, id^T) as required by law (a). The same result holds for operation bSynch using the symmetry of the precondition and the symmetric definition of TGGs and the derived operations. Therefore, the result holds for the concurrent synchronization operation CSynch.

Law (b) **in Fig. 3:** Let $(r_0, d^S_1, d^T_1) \in (R \otimes \Delta_S \otimes \Delta_T)$ as depicted on the right and in Fig. 3. We have to show that the concurrent synchronization operation yields a consistent correspondence $r_2 \in VL(TGG)$.



(b)

We apply the concurrent synchronization operation fSynch according to Fig. 7 and Fig. 10. All steps are well defined according to Rem. 3. Step 4 (operation CCT) provides a maximal consistent subgraph $G^T_{2,FCB} \in VL_T$. Therefore, we can apply law $(b2)$ in Fig. 5 and derive that operation bPpg in step 5 yields a consistent correspondence r_2 (see Fact 2). The proof for the concurrent synchronization operation bSynch is analogous using the symmetric definition of TGGs and the dual definitions of the steps according to Rem. 3. Therefore, the concurrent synchronization operation CSynch always yields a consistent correspondence $r_2 \in VL(TGG)$. □

Finally, we show the compatibility of concurrent with basic model synchronization. This means that the non-deterministic concurrent synchronization operation is a generalization of the propagation operations of the basic model synchronization framework.

Theorem 2 (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 according to Rem. 1. Then, the* derived concurrent TGG synchronization operation CSynch *is compatible with propagation operations* fPpg, bPpg.

Proof. Let CSynch be obtained from the derived forward and backward synchronization operations

fSynch and bSynch, i.e. CSynch = (fSynch ∪ bSynch). According to Def. 3, we have to show for the forward case that $\forall\,(d^S, r_0) \in \Delta_S \otimes R$, with $d^S : G_0^S \to G_1^S \wedge G_1^S \in VL_S : (id, \mathsf{fPpg}(d^S, r_0)) \in \mathsf{CSynch}(d^S, r_0, id)$. The result for the backward case holds by dualization due to the symmetric definition of TGGs and the derived operations.

Let $r_0 = (G_0^S \leftrightarrow G_0^T)$ with $G_0^S \in VL_S$ and let $d^S : G_0^S \to G_1^S$ be a source model update. Let further (r_1, d^T) be the result of applying fPpg to (d^S, r_0) with $r_1 = (G_1^S \leftrightarrow G_1^T)$ and $d^T : G_0^T \to G_1^T$. We show that $(id, r_1, d^T) \in \mathsf{fSynch}(d^S, r_0, id)$ according to Fig. 7 and Fig. 10. Since $G_1^S \in VL_S$ we have that step 1 (CCS) yields the maximal consistent subgraph $G_{1,C}^S = G_1^S$ (see Rem. 2) and source update $d_F^S = id : G_1^S \to G_1^S$. Thus, step 2 applies operation fPpg to the same input as in the precondition, such that we derive correspondence $r_{1,F} = r_1$ and target model update $d_{1,F}^T = d^T$. By correctness of fPpg (law $a2$ in Fig. 5) we know that $r_1 = r_{1,F} \in VL(TGG)$ and therefore, $G_1^T \in VL_T$. In step 3 (operation Res), the merge construction is applied to d^T and the target update id, which does not delete nor create anything (the corresponding minimal rule is the empty rule). This means that the updates are conflict-free and the merge construction yields the target updates $d'^T_{2,FC} = id$ and $d_{2,FC}^T = d^T$. Since $G_1^T \in VL_T$ (see step 2 above), step 4 yields the maximal consistent subgraph $G_{2,FCB}^T = G_1^T$ (see Rem. 2) and the target update $d_B^T = id : G_1^T \to G_1^T$. Therefore, the target update $d_{2,CC}^T = d_B^T \circ d'^T_{2,FC}$ is given by $d_{2,CC}^T = id$. By $r_1 = r_{1,F} \in VL(TGG)$ (see step 2 above) and correctness of operation bPpg (law $(b1)$ in Fig. 5) we know that step 5 yields source model update $d_{2,CB}^S = id$ and correspondence $r_{2,FCB} = r_1 \in VL(TGG)$. This leads to source model update $d_{2,FCB}^S = d_{2,CB}^S \circ d_F^S = id$. All together, we have that $s = (id, r_1, d^T) \in \mathsf{CSynch}(d^S, r_0, id)$, i.e. s is a valid solution for the concurrent synchronization problem of $\mathsf{CSynch}(d^S, r_0, id)$. □

B Efficient Execution

This section provides further details for the execution of the concurrent synchronization operation in practice and discusses possibilities for an efficient execution.

At first, we formalize the notion of maximal consistent graphs, which is used for consistency creating operations CCS and CCT.

Definition 4 (Maximal Consistent Sub Graph)**.** *Let $TGG = (TG, TR)$ be a triple graph grammar and G^S be a graph typed over TG^S, i.e., $G^S \in VLTG^S$. A graph $H^S \in VL(TG^S)$ is a maximal sub graph of G^S, if $H^S \subseteq G^S$ and there is no graph $H'^S \in VL(TG^S)$ with $H^S \subseteq H'^S \subseteq G^S$ and $H'^S \neq H^S$. The dual notion for maximal subgraphs concerning the target domain VL_T is derived by replacing above supscript S with T.*

Construction 2 (Consistency Creating Sequences (Operation CCS))**.** *Let $TGG = (TG, TR)$ be a triple graph grammar and $G^S \in VL(TG^S)$. A maximal consistent subgraph H^S of G^S can be obtained by applying forward translation rules [15] in the following way.*

By Def. 4 we derive the requirement that $H^S \in VL_S$ and $H^S \subseteq G^S$. According to Thm. 1 in [15] (completeness of model transformations via forward translation rules) there is a model transformation sequence $(H^S, H'_0 \overset{tr^*_{FT}}{\Longrightarrow} H'_n, H^T)$ via forward translation rules. The initial graph H'_0 is given by $H'_0 = (H'^S_0) \leftarrow \varnothing \rightarrow \varnothing)$, where H'^S_0 is derived from H^S by adding translation attributes for each element and assigning them to the Boolean value **F**. *This means that initially, no element is translated. The terminal graph H'_n is given by $H'_n = (H'^S_n \leftarrow H^C \rightarrow H^T)$, where H'^S_n is derived from H^S by adding translation attributes for each element and assigning them to the Boolean value* **T**, *i.e., it differs from H'^S_0 only on the value of the translation attributes. Using $H^S \subseteq G^S$ we can extend*

TGT-*sequence* $s_1 = (H'_0 \overset{tr^*_{FT}}{\Longrightarrow} H'_n)$ *to* $s_2 = (G'_0 \overset{tr^*_{FT}}{\Longrightarrow} G'_n)$*, where we only add the additional context elements of* $G^S \setminus H^S$ *and assign the corresponding translation attributes with the Boolean value* **F***, i.e. these context elements are effectively not involved in the transformation steps.* [4] *This means that the translation attributes in* G'^S_n *are set to true for the elements of* $H^S \subseteq G^S$ *and set to false for the elements of* $G^S \setminus H^S$*. Summing up, for each possible maximal consistent subgraph* H^S *we can construct a* TGT-*sequence via forward translation rules yielding* H^S *as result. This means that the construction of possible forward translation sequences will potentially produce all possible maximal subgraphs* H^S*. For termination of this process, see Rem. 4.*

Remark 4 (Termination of Operation CCS)**.** *Operation* CCS *is based on the execution of model transformations based on forward translation rules. In order to ensure termination, we can check statically that each forward translation rule is modifying at least one translation attribute, which is a sufficient condition as shown by Thm. 1 in [15]. As a common requirement we assume that the set of triple rules TR is finite and the given source model is finite on the structural part, i.e. only the data part may be infinite. This ensures that there are only finitely many possible forward translation sequences. Similarly, operation* CCT *is terminating if each backward translation rule is modifying at least one translation attribute.*

In order to improve the efficiency of the execution of the concurrent model synchronization operation CSynch, we can combine some of the steps according to Rem. 5 below using the available equivalence results for TGT-sequences based on the different kinds of operational rules.

Remark 5 (Efficient Execution of Concurrent Model Synchronization)**.** *Steps 1 and 2 as well as 4 and 5 in Fig. 7 and Fig. 10 can be executed in a combined way in order to improve the efficiency of the execution. Essentially, when executing operation* fPpg *according to the presented construction in [16], then the derived forward translation sequence can be used as well for operation* CCS*. The only difference is that* CCS *allows for the additional inconsistent context of the given source model. In more detail, given source model* G^S_1*, then operation* CCS *computes the maximal consistent subgraph* $G^S_{1,C}$ *of* G^S_1 *by computing terminating forward translation sequences* $(H'_0 \overset{tr^*_{FT}}{\Longrightarrow} H'_n)$ *via forward translation rules according to Construction 2 above. Thus,* $H'^C_0 = H'^T_0 = \emptyset$*, where* H'^S_0 *extends* G^S_1 *with translation attributes that are assigned to the Boolean value* **F***, and* H'^S_n *differs from* H'^S_0 *on the values of the translation attributes for the subgraph* $G^S_{1,C} \subseteq G^S_1$*. The construction of the basic synchronization operation* fPpg *as presented in [16] consists of three steps, where step 2 yields a consistency creating sequence* $s_{1,CC}$ *via* TR_{CC} *and step 3 yields a forward translation sequence* $s_{2,FT}$*. The marking sequence* $s_{1,CC}$ *is equivalent to a corresponding triple sequence via TR* [5] *and thus, to a forward translation sequence* $s_{1,FT}$ *by completeness of model transformations based on forward translation rules (Thm. 1 in [15]). Sequence* $s_{1,FT}$ *can be composed with* $s_{2,FT}$ *leading to one terminated forward translation sequence* s_{FT}*. This means that the sequence does not need to be computed via operation* CCS *before, but can be derived during the execution of* fPpg*. This means that the restriction of* G^S_1 *to* $G^S_{1,C}$ *can be performed on-the-fly during the computation of possible forward translation sequences via the construction of* fPpg*. Similarly, the computation of backward translation sequences*

[4] For the detailed construction of this extension see Fact 11 in Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of Model Synchronization Based on Triple Graph Grammars - Extended Version. Tech. Rep. TR 2011-07, TU Berlin, Fak. IV (2011), to appear, online available at http://tfs.cs.tu-berlin.de/publikationen/Papers11/HEOCDX11b.pdf

[5] This equivalence is shown by Fact 11 in Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of Model Synchronization Based on Triple Graph Grammars - Extended Version. Tech. Rep. TR 2011-07, TU Berlin, Fak. IV (2011), to appear, online available at http://tfs.cs.tu-berlin.de/publikationen/Papers11/HEOCDX11b.pdf

via operations CCT *and* bPpg *can be joined and performed within* bPpg*. Effectively, this means that operations* fPpg *and* bPpg *are extended to possibly inconsistent input models and thus, they become non-deterministic operations due to the possible different maximal consistent sub graphs.*

Finally, as described in [16], the performed TGT*-steps in operations* fPpg *and* bPpg *can be reduced by reusing the steps that were executed in a previous synchronization and are still valid.*

C Details of the Case Study

In this section, we provide the details of our case study. At first we present the complete TGG in Sec. C.1 for generating consistent organizational models. Thereafter, in Sec. C.2 – Sec. C.5, we explain the steps 2–5 of fSynch for concurrent model synchronization as sketched in Fig. 8: In Sec. C.2, we illustrate how operation 2:fPpg is constructed as composition of auxiliary operations ⟨fAln, Del, fAdd⟩ according to [16]. In Sec. C.3, we show how operation 3:Res resolves the conflict between the two model updates on the target domain part. Operation 4:CCT is described in Sec. C.4 for the variant of Ex. 4 that the resolution yields two "married"-edges between the same two nodes. Finally, in Sec. C.5, operation 5:bPpg is explained, which is constructed symmetrically to operation 2:fPpg).

C.1 Components of the TGG

The triple graph grammar $TGG = (TR, \emptyset, TR)$ of our case study is given by the triple type graph TG in Fig. 12, the empty start graph and the triple rules in the subsequent figures.

Example 6 (Type Graph)**.** *According to the triple type graph TG in Fig. 12, the models of the source domain contain persons including their detailed salary information (bonus and base salary) and their names. Models of the target domain additionally contain the departments to which a person is assigned to, information about his marriage status (if married to a person in the same company), the birth date of a person, and a single value for the complete salary of a person, while the details about bonus and base salary are not provided.*

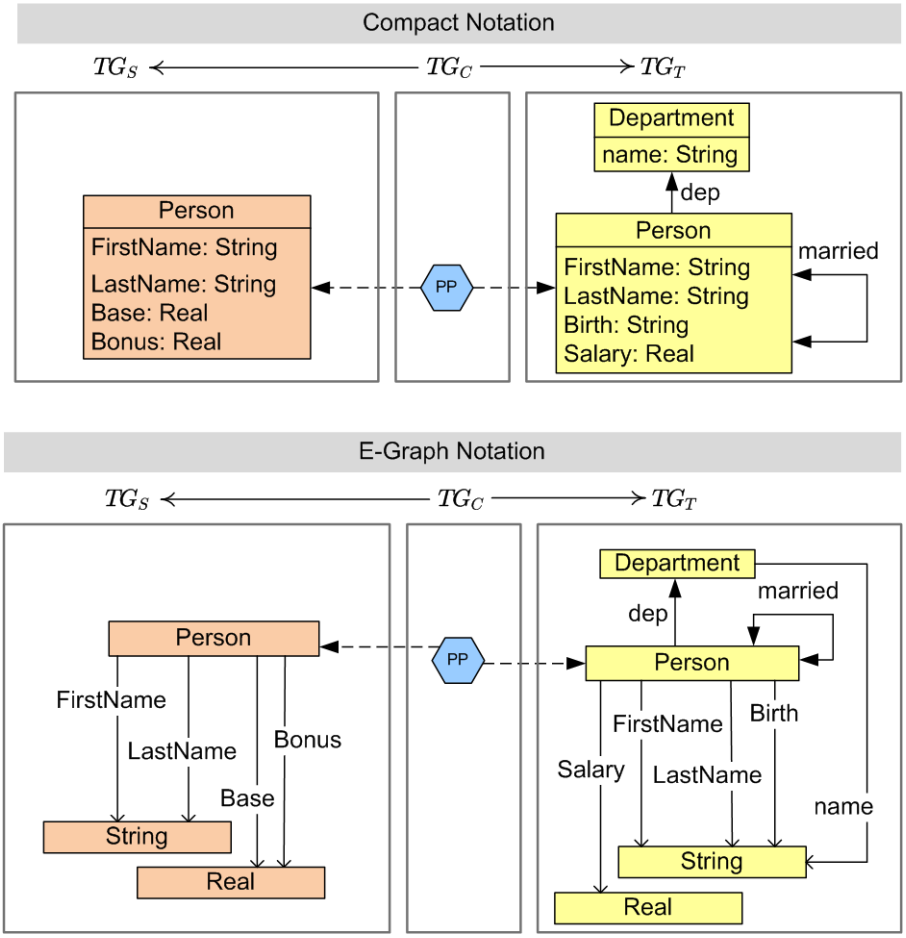


Figure 12: Triple type graph *TG*

Example 7 (Triple Rules (Parts 1 and 2))**.** *The triple rules of the TGG are depicted in short notation in Figs. 13 to 16 and in we first explain Figs. 13 and 14. The first rule (Empty2OtherDepartment) is depicted additionally with explicit left and right hand side. It creates a new department in the target model, but does not change the source model. The negative application condition (NAC) ensures that this rule cannot be applied for creating a department with name "Marketing". For this purpose, rule 2 (Person2FirstMarketing) is used, where the NAC ensures that the given target model does not contain already a department with name "Marketing". This rule additionally creates a person of the new department in the target model and a corresponding person in the source model. Rule "Person2NextMarketingP" is applied in order to extend both models with further persons in the marketing department. Note that the attributes of the created persons are not set. This is possible in our formal framework of attributed graph transformation based on the notion of E-graphs*[6]*. The main advantage is that we can propagate changes of attribute values without the need for deleting and recreating the owning structural nodes. This is important from the efficiency and application point of view.*

[6] for details see: Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theor. Comp. Science, Springer (2006).

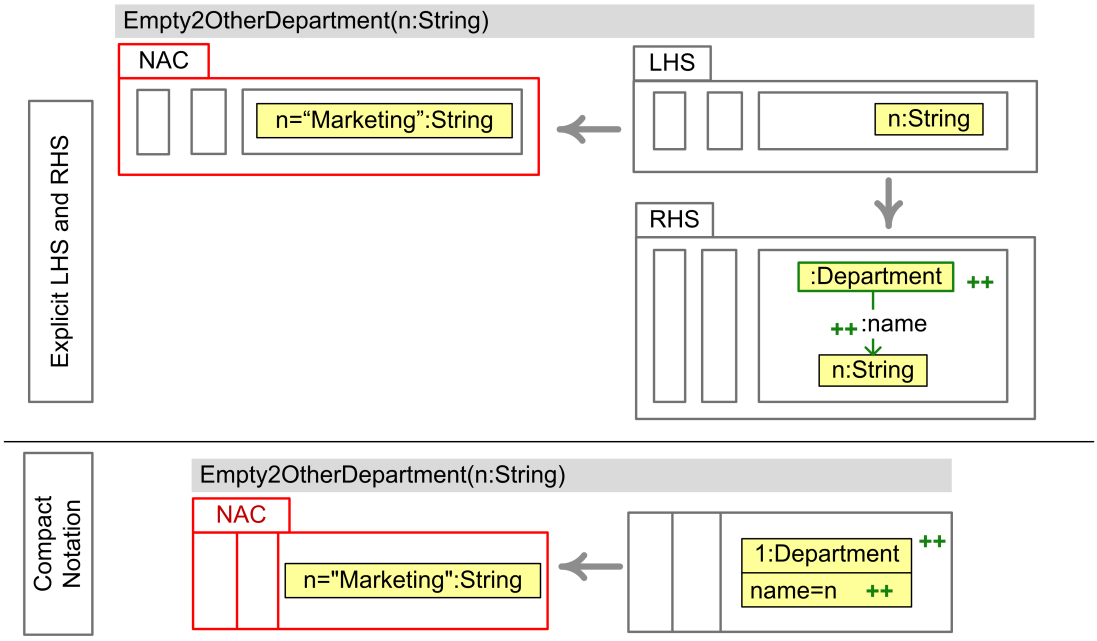


Figure 13: Triple rules - part 1

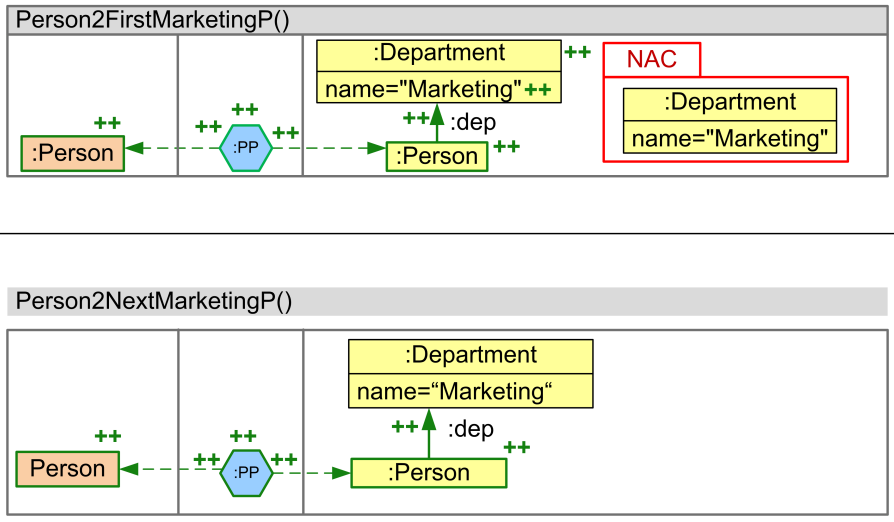


Figure 14: Triple rules - part 2

Example 8 (Triple Rules (Parts 3 and 4))**.** *The further triple rules of the TGG are depicted in Fig. 15 to Fig. 16. The four rules in Fig. 15 concern the creation of attribute values only. Rules "FName2FName" and "LName2LName" create new corresponding values for first and last names, respectively. The next rule "Empty2Birth" assigns the of a birth date of a person in the target component and does not change the source component. Rule "DetailedSalary2Salary" assigns the detailed salary values (bonus and base) in the source component and the sum of them in the target component. Rule "Empty2OtherPerson" of the TGG is presented in Fig. 16 and creates a new person of a department that is not the marketing department. Therefore, there is no correspondence to the source model and the rule directly creates the person including all attribute values. Similarly, the last rule*

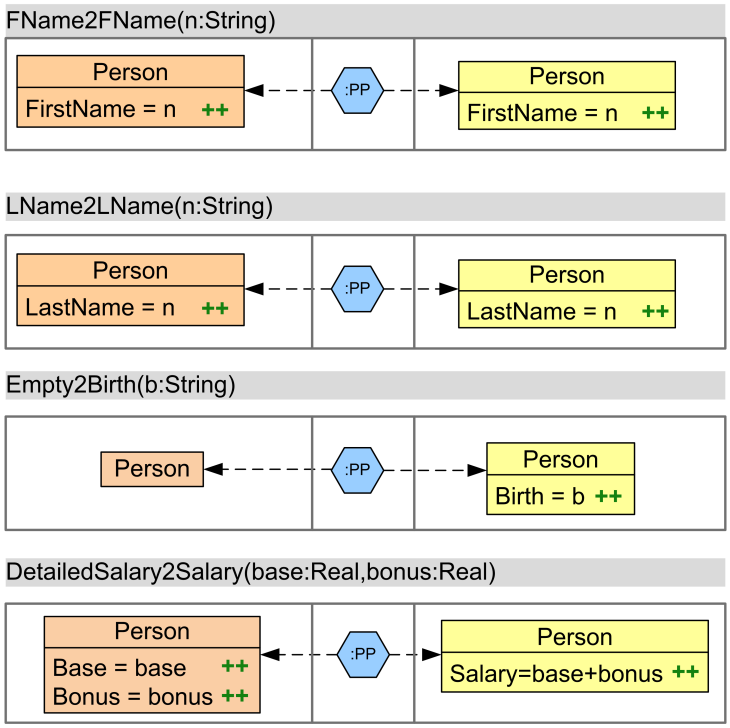


Figure 15: Triple rules - part 3

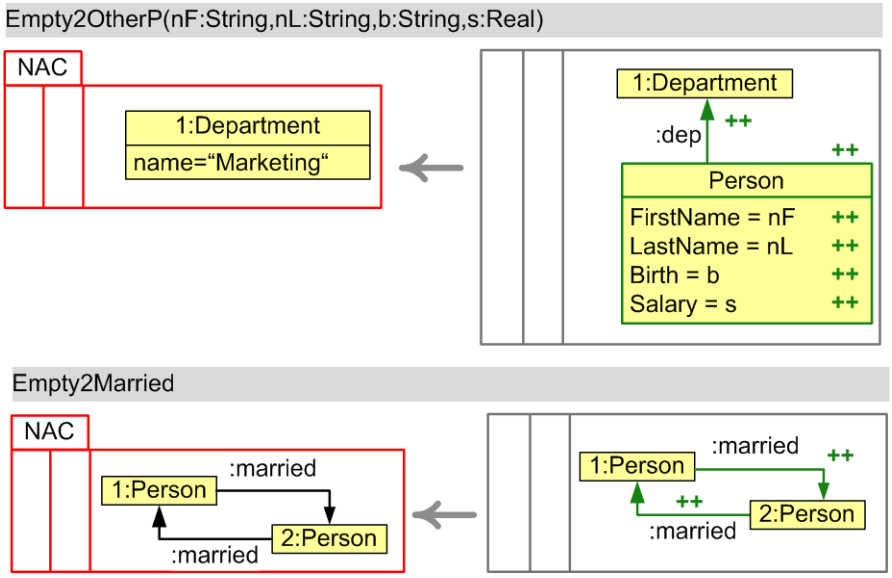


Figure 16: Triple rules - part 4

"Empty2Married" in Fig. 16 inserts a "Married"-edge between two person nodes when these persons get married[7].

[7]Note that we model a bidirectional edge by two edges, one for each direction.

C.2 Operation 2:fPpg: Forward Propagation

Given two corresponding models G^S and G^T and an update of G^S via the graph modification $a = (G^S \xleftarrow{a_1} D^S \xrightarrow{a_2} G'^S)$ with $G'^S \in VL_S$, then, according to [16], the forward propagation fPpg of δ_S is performed in three steps via auxiliary operations fAln, Del, and fAdd. First of all, the deletion performed in a is reflected to the correspondence relation between G^S and G^T by calculating the forward alignment remainder via operation fAln. Intuitively, operation fAln constructs the new correspondence graph D^C from the given G^C by deleting all correspondence elements in G^C whose associated elements in G^S are deleted via update a and for this reason, do not occur in D^S (see upper row in Fig. 17) for the construction of the new correspondence). This step deletes all correspondence elements whose elements in G^S have been deleted. In the second step (see center row in Fig. 17), performed via operation Del, the maximal subgraphs $G_k^S \subseteq G^S$ and $G_k^T \subseteq G^T$ are computed such that they form a consistent integrated model in $VL(TGG)$ according to the triple graph grammar. All elements that are in G^T but not in G_k^T are deleted, i.e. the new target model is given by G_k^T. Finally, in the last step (operation fAdd, defined in the last row in Fig. 17), the elements in G'^S that extend G_k^S are transformed to corresponding structures in G'^T, i.e. G_k^T is extended by these new structures. The result of fAdd, and hence also fPpg, is a consistent integrated model.

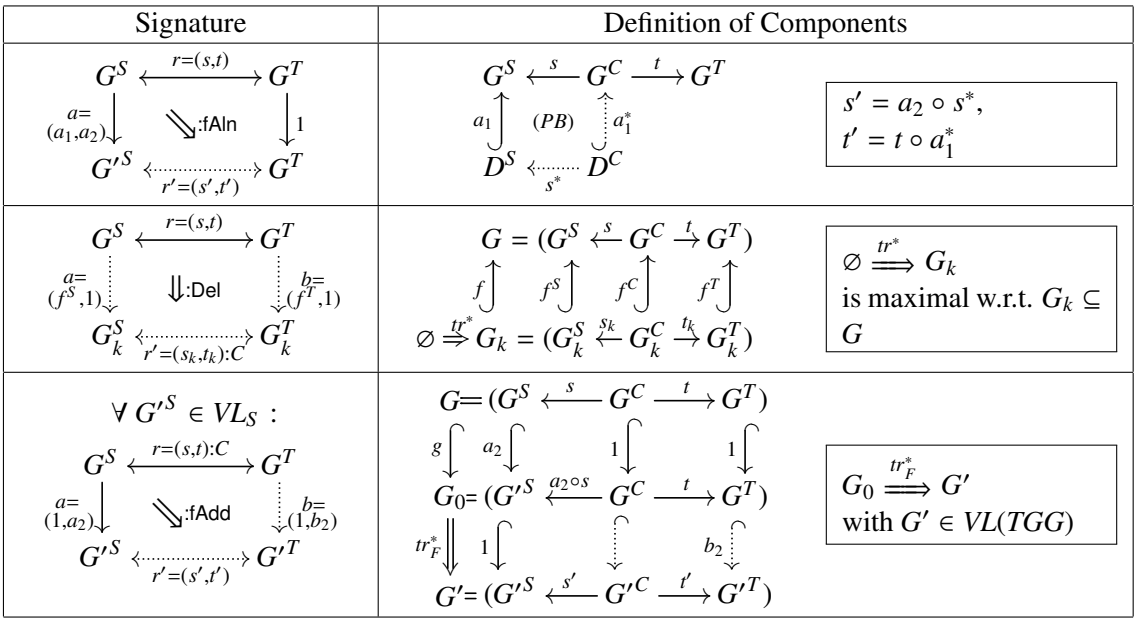


Figure 17: Auxiliary operations fAln, Del and fAdd

Example 9 (Forward Propagation via Operation fPpg)**.** *Fig. 18 shows the application of the three steps of synchronization operation* fPpg *to the visual models of our running example.*

*After removing the dangling correspondence node of the alignment in the first step (*fAln*), the maximal consistent subgraph of the integrated model is computed (*Del*) by stepwise marking the consistent parts: consistent parts are indicated by grey boxes with checkmarks in visual notation and by bold font faces in graph representation. Notice that node "Bill Gates" is part of the target graph in this maximal consistent subgraph, even if it is not in correspondence with any element of the source graph. In the final step (*fAdd*), the inconsistent elements in the target model are removed and the remaining new elements of the update are propagated towards the target model by model transformation, such that all elements are finally marked as consistent.*

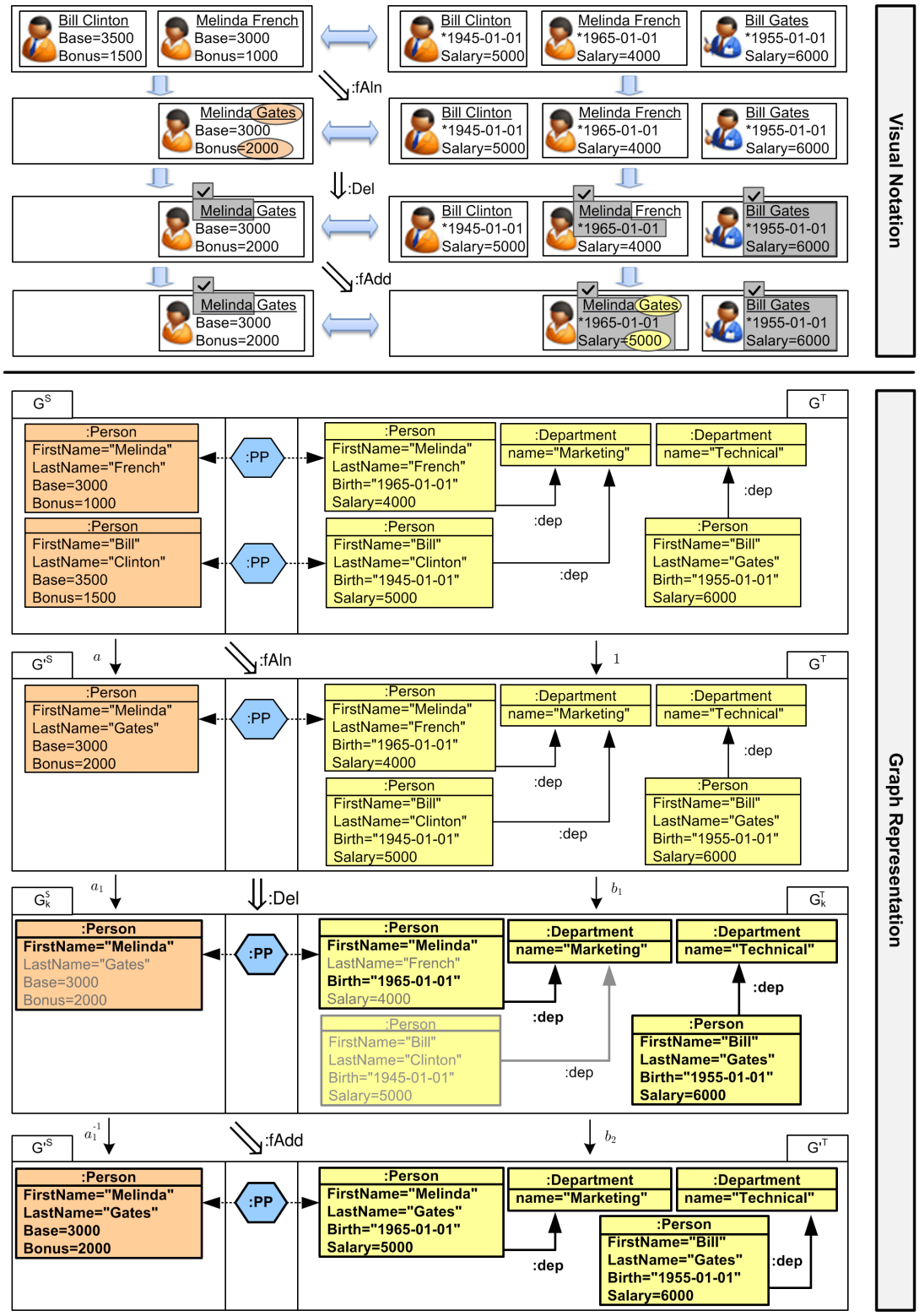


Figure 18: Forward propagation in detail - top: visual notation, bottom: graph representation

C.3 Operation 3:Res: Conflict Resolution

In this section, we show how operation 3:Res resolves a possible conflict between the two model updates $m_1 = (G \leftarrow D_1 \rightarrow H_1)$ and $m_2 = (G \leftarrow D_2 \rightarrow H_2)$ on one domain based on the tentative merge construction for two conflicting graph modifications [10].

The tentative merge construction is performed in three steps via auxiliary operations rDel, rIns, and rUnify: At first (using operation rDel in the upper row in Fig. 19), all deletion actions are merged by computing the intersection of intermediate graphs yielding graph D. In case of delete-insert conflicts, D is too small and has to be extended again (to $\overline{D}$, formally constricted in [10]). I.e. those node deletion actions which conflict with edge insertions are taken back and the intermediate graphs of the original modifications are extended such that all insertions can now take place. Thereafter (applying operation rIns in the center row in Fig. 19 to each modification), insertions are performed on both sides, yielding graphs $\overline{X_1}$ and $\overline{X_2}$ (a construction that is shown to be well-defined in [10]). Finally (using operations rUnify in the last row in Fig. 19), the results from both sides are merged. Afterwards, $G \leftarrow \overline{D} \rightarrow H$ forms the tentatively merged graph modification of m_1 and m_2. Note that this tentative merge construction does not always lead to desired results, since the standard resolution of delete-insert conflicts is not always adequate. In the second part of conflict resolution, we need to identify non-performed deletions or even further, we have to identify not or just partially performed operations which shall be resolved differently. Possible solutions are to take back the (potentially partial) execution of an operation, to complete a partial execution, to perform a different operation or a combination of those.

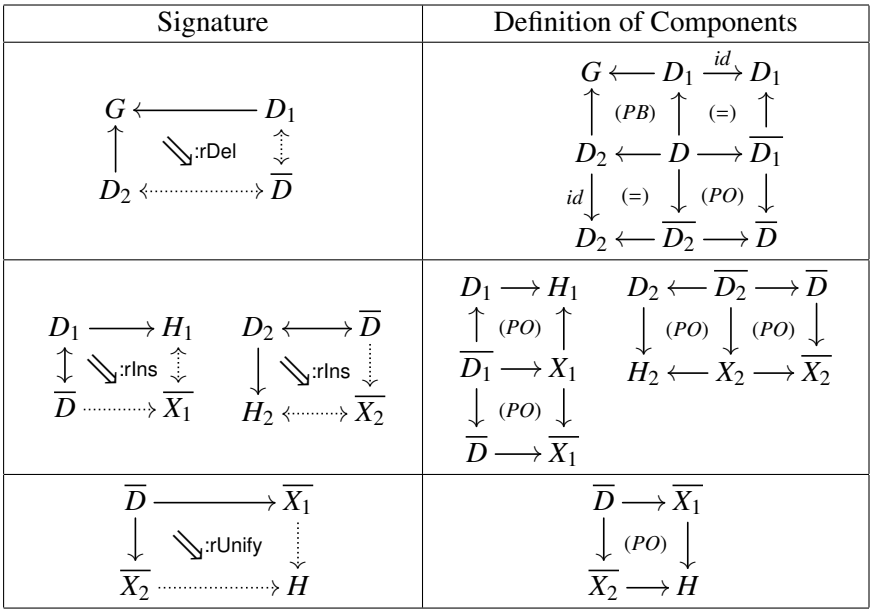


Figure 19: Auxiliary operations rDel, rIns and rUnify for conflict resolution

Example 10 (Conflict resolution based on tentative merge construction)**.** *We describe the conflict resolution based on the tentatively merged graph modification for graph modifications* $m_1 = (G \leftarrow D_1 \rightarrow H_1)$ *and* $m_2 = (G \leftarrow D_2 \rightarrow H_2)$ *and manual modifications in step* 3:Res *in Fig. 8.*

Step rDel (see Fig. 20): *Intersection graph D contains only those elements that are present both in* D_1 *and in* D_2*, i.e. that are preserved by both modifications. Hence, the* Bill Clinton *node and its incident edges are not in D, nor are the attribute values of attributes that are changed by any of the two*

updates. The extension of D by nodes in D_1 which shall be deleted by modification m_2 but are needed by modification m_1 leads to the extended graph $\overline{D_1}$ where now a node corresponding to Bill Clinton *has been added since this node is needed for the insertion of an edge to the* Technical Department *node and an attribute edge to the new* Salary *value. In our example, there is no need to extend D by nodes in D_2 since there is no node deletion in m_1. Hence, $\overline{D_2}$ equals D_2. $\overline{D}$ is constructed as union $\overline{D_1} \cup \overline{D_2}$. In addition to the common subgraph D of $\overline{D_1}$ and $\overline{D_2}$, the union graph $\overline{D}$ contains a* Bill Clinton *node which is in $\overline{D_1}$ but not in $\overline{D_2}$. Now, graph $\overline{D}$ contains all objects that remain after both modifications have performed their deletions* plus *those nodes that are needed for the insertion of edges by either m_1 or m_2.*

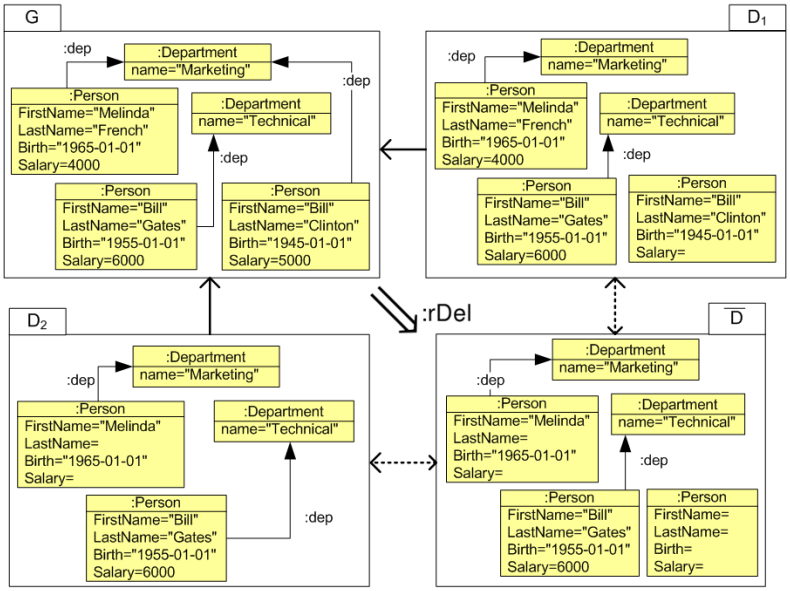


Figure 20: Step rDel of the tentative merge construction

Step rIns (see Fig. 21): *Graphs $\overline{X_i} = X_i \cup \overline{D}$ result after performing the creation of elements given by either modification on $\overline{D}$. I.e., $\overline{X_1}$ is graph $\overline{D}$ together with all elements created by m_1 (the edge from the* Bill Clinton *node to the* Technical Department *node, the new attribute value for Bill Clinton's* Salary *attribute and one new* married*-edge) , and $\overline{X_2}$ contains in addition to a copy of $\overline{D}$ one new* married*-edge and two new attribute values for* Melinda*'s attributes* LastName *and* Salary*. We here show explicitly the diagram for $\overline{X_1}$:*

Step rUnify (see Fig. 22): *In this last step, the creation parts of both modifications which have been performed independently of each other on $\overline{D}$ in step* rIns*, are now merged by a union construction, leading to the tentatively merged graph H.*

After the tentative merged graph H has been computed automatically, the user is asked to finish the conflict resolution manually. In our example, the user is asked to set values for the attributes FirstName, LastName *and* Birth *of the "Bill Clinton" node. To assist the user with this task, the previous attribute values of the corresponding deleted node may be shown to the user s.t. he only needs to confirm that these attribute values should be kept. The result after these manual modifications is graph $G^T_{2,FC}$ in Fig. 23, where now (in contrast to graph H in Fig. 22) the attribute values are set accordingly.*

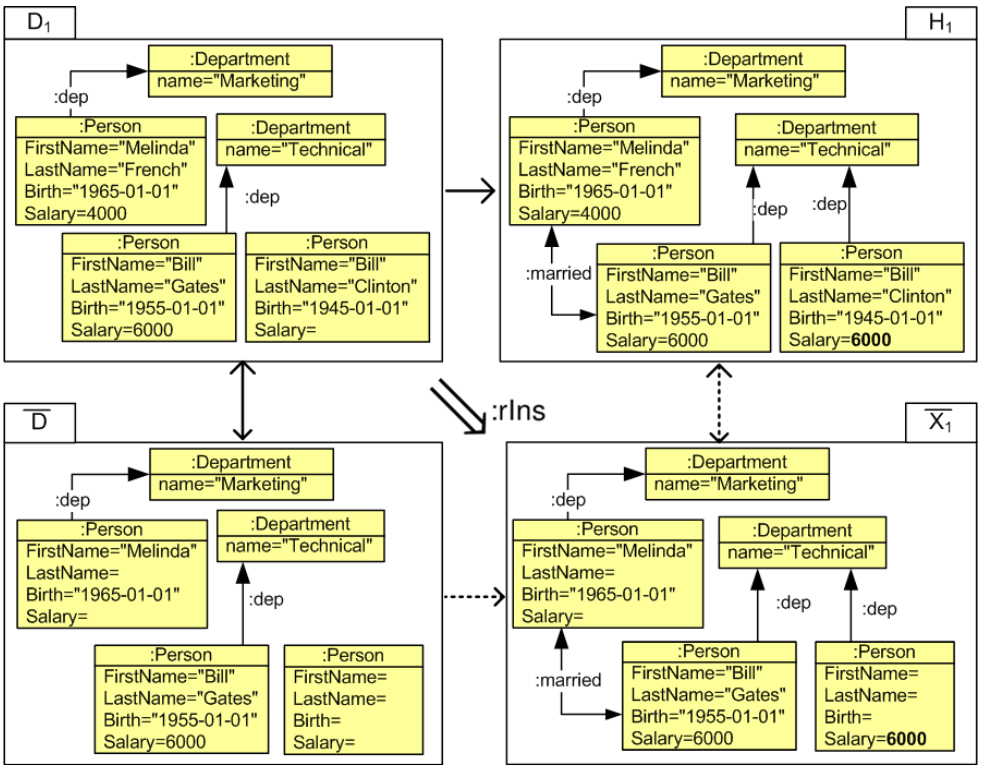


Figure 21: Step rIns of the tentative merge construction applied to m_1

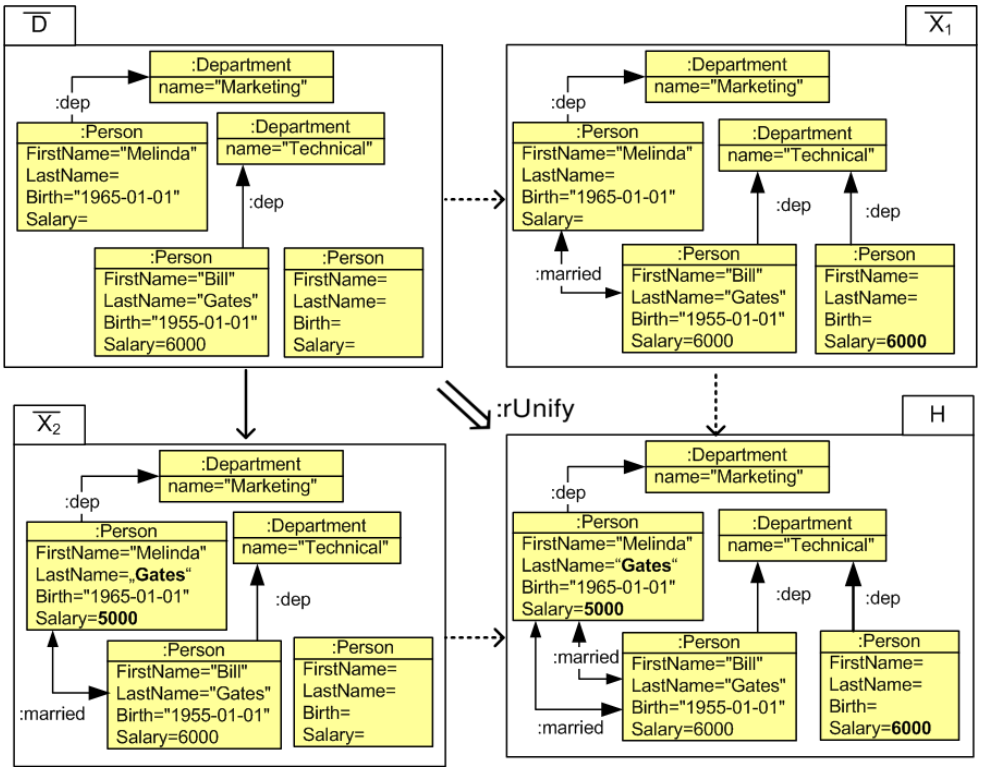


Figure 22: Step rUnify of the tentative merge construction

C.4 Operation 4:CCT: Consistency Creation of the Target Domain

Operation 4:CCT performs a consistency creation on the result of the conflict resolution on the target domain (i.e. graph $G^T_{2,FC}$ in Fig. 23). Here, basically triple rules are applied aiming to derive this graph. If this is not possible (and in our case it is not possible because we have two married-edges between the same person nodes), then the conflict resolution result graph is replaced by the TGG-derived graph that has the least differences compared to the conflict resolution result graph. In our case, this is graph $G^T_{2,FCB}$ that equals $G^T_{2,FC}$ with the small difference that it contains only one married-edge between the nodes of Bill Gates and Melinda Gates.

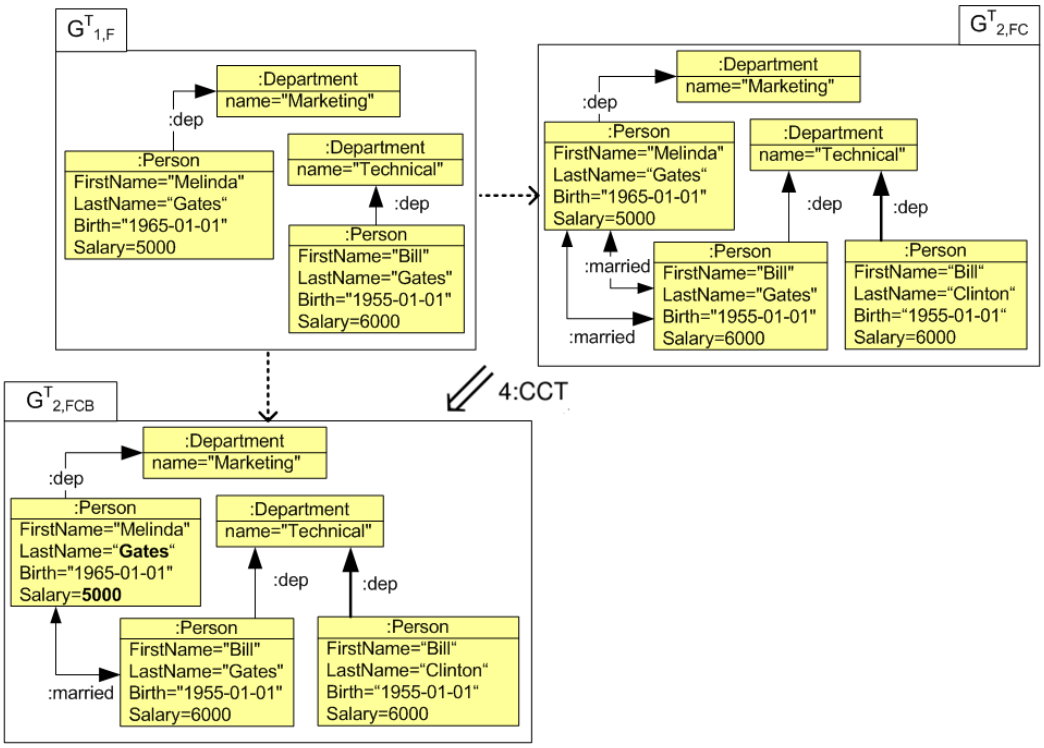


Figure 23: Operation 4:CCT for consistency creation on the target domain

C.5 Operation 5:bPpg: Backward Propagation

Finally, operation 5:bPpg is explained, which is constructed (symmetrically to operation 2:fPpg) from auxiliary operations ⟨bAln, bDel, bAdd⟩. In essence, this operation propagates back all changes that occurred on the target domain to the triple and adapts all elements on the source domain that are concerned by these changes. Fig. 24 shows the backward propagation square for operation 5:bPpg from Fig. 8 in abstract syntax. Since none of the changes on the target domain concern the source domain model, the source domain is not changed but just connected to the changed target part of the model.

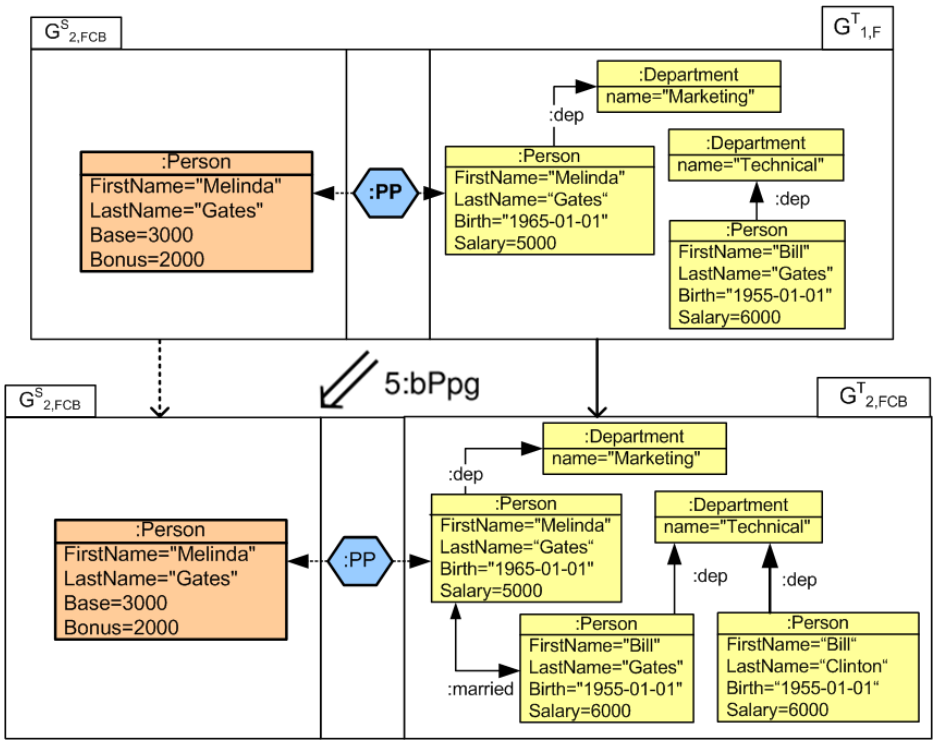


Figure 24: Operation 5:bPpg for backward propagation to the source domain