Towards the Propagation of Model Updates along different Views in Multi-View Models

Susann Gottmann Nico Nachtigall Thomas Engel
Interdisciplinary Centre for Security, Reliability and Trust,
Université du Luxembourg, Luxembourg
firstname.lastname@uni.lu
Claudia Ermel
Technische Universität Berlin
firstname.lastname@tu-berlin.de
Frank Hermann
Carmeq GmbH
mail@firstnamelastname.eu

Abstract

Models are the keystones in model-driven systems development. They describe systems at different layers of abstraction and with a focus to different domains. For each domain, dedicated domain specific visual modelling languages are used for model definitions with the idea to separate concerns to different domain experts. This enables precise problem and requirement definitions and should decrease efforts in developing and validating systems. We focus on multi-view models that are in relationship with source models by triple graph grammars. A multi-view model provides different views of the source model at different layers of abstraction but within the same DSL which is typically different from the DSL of the source model. In practice, elements in different views may overlap. We present an informal methodology for consistently propagating updates from one view to the other views and also to the source domain. We motivate our approach by multi-view models in a hospital scenario.

1 Introduction

Models are the keystones in the model-driven development (MDD) of systems in general and of software systems (MDSD) in particular as proposed by the object management group (OMG) in the form of the model-driven architecture (MDA) [BBG05]. Models describe systems at different layers of abstraction and with a focus to different domains. For each domain, dedicated domain specific visual modelling languages (DSLs) are used for model definitions with the idea to separate concerns to different domain experts. This enables precise problem and requirement definitions for each domain and should decrease efforts in developing and validating systems.

Let us consider Fig. 1, first, which illustrates an abstract scheme of the runnning example in this paper. It shows two departments in a hospital: The management domain (= source domain, on the left) and the therapy domain (= target domain, on the right). Both domains differ in the data they are dealing with, but nethertheless, they are strongly interweaved with each other because they maintain the same patient in the hospital. In addition, in the therapy domain, nurses and doctors have different views on the data, due to different access permissions.

In: A. Anjorin, J. Gibbons (eds.): Proceedings of the Fifth International Workshop on Bidirectional Transformations (Bx 2016), Eindhoven, The Netherlands, April 8, 2016, published at http://ceur-ws.org

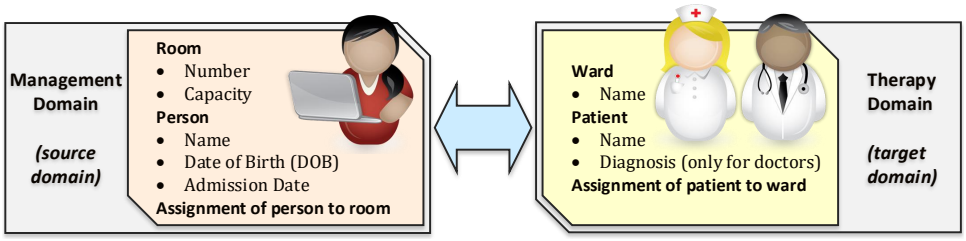


Figure 1: Scheme of hospital scenario

The running example is an example of a multi-view model in the target domain, that is in relationship with a model in the source domain. In practice, elements in different views may overlap. In the therapy domain multi-view models exist in which all views share the same DSL. All views in the therapy domain are in relationship with the management domain. The management domain belongs to another DSL.

The research question which we want to answer informally in this paper is: *If a model update in one view is performed, then how is it possible to consistently propagate this model update to all other views and also to the other domain?*

This research question evolved during the work on an ongoing industrial applied research project with our industrial partner SES[1] [GHE+13]. In this project, we apply triple graph grammars (TGGs) for translating source code of satellite control procedures to a visual model of the control flow of the same procedure. The visual model is an extended flow chart model that is extended to fulfill the needs of SES. The visualisation contains different abstraction layers leading to a multi-view visual model of the source code.

The hospital scenario in this paper illustrates the effects of our approach, because it has seperate views for nurses and doctors. While this example can be alternatively implemented based on a central database with appropriate access rights, we like to emphasize that the approach can be used for much more complex scenarios like the industrial application for satellite operations described above.

The applied research project is a follow-up project of the large-scale industrial project [HGN+13], in which we successfully applied TGGs and also the model synchronisation framework based on TGGs [HEEO12, HGN+14].

Due to the success of the first industrial project, we continued using triple graph grammars (TGGs) [SK08] as modelling technique in the second industrial project. The descision of using TGGs was made due to the following aspects: The formal framework of TGGs is comprehensive, well elaborated and it offers different analysis techniques for correctness and completeness of the model transformation [HEGO14, HEO+15]. TGGs are specified by a set of triple graph rules that establish the relationship between a model in the source domain and a model in the target domain by simultaneously creating the elements of the source model and the elements of the target model with correspondences between them. In our case, the target model is the multiple-view model. Note that formally, we assume multiple views to be contained in one "big" model where user restrictions for accessing the views are left to the implementation details. Now, we introduce our running example based on TGGs in greater detail.

Example 1.1 (Hospital Scenario) *Fig. 2 illustrates the detailed model of patients in a hospital and the correspondences between both. The model $M_{Management}$ on the left depicts the occupancy of the hospital and is focused on the* Management *domain of the hospital. Patients are represented as persons that are assigned (*asg*) to rooms of specific capacities. The model $M_{Therapy}$ on the right contains two views of the patients at different levels of abstraction and is focused to the* Therapy *domain. Nurses are able to access the names of the patients and the corresponding ward on which the patient is situated. Doctors have a more detailed view on the patients data. They can additionally access comments on the diagnosis of each patient. Moreover, dedicated nodes* : PP *define correspondences between the elements of the management model and elements of the therapy model. Each person corresponds to two patient nodes: One patient node in the nurses' view and one in the doctors' view. Rooms and wards are not related to each other in both domains. Furthermore, the doctors' view contains comments on the diagnosis of each patient that are neither contained in the nurses' view nor in the management model. The correspondences identify each patient in the nurses' view with a patient in the doctors' view and vice versa via a corresponding person node in the management model. The correspondences for a person are jointly created by applying a given triple graph rule. We assume that all elements that are jointly created by a rule, belong to each other.* △

[1] Société Européenne des Satellites, www.ses.com

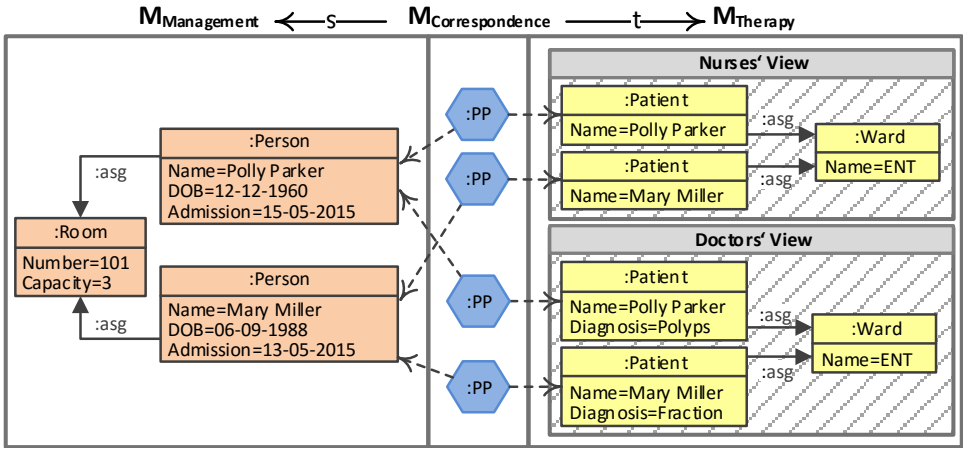


Figure 2: Correspondences between management & therapy model of patients

We informally introduce the *derived propagation framework* that is based on the model synchronisation framework using triple graph grammars [HEO+15], but first, we review the underlying formal framework, i.e., the theory behind triple graph grammars in Sec. 2. Then, we informally present the consistent propagation framework of model updates in a given multi-view model using the running example in Sec. 3. Finally, we review future work and conclude the paper in Sec. 4 and present perspectives for future work in Sec. 5.

2 Formal Framework

We review basic notions of the formal framework of our approach (cf. [EEPT06, EEGH15]) and address (multi-view) models that are represented by typed attributed graphs [GLEO12] and triple graph grammars [SK08].

Typed attributed graphs are composed of nodes, edges between nodes, node attributes and types for nodes, edges and attributes. The typing is defined by a type morphism between the typed graph (= instance graph) and a type meta-model, also called type graph.

A morphism $M \xrightarrow{m} M'$ between two graphs M and M' is a structure preserving mapping from nodes, edges and attributes of M to nodes, edges and attributes of M'. Structure preservation means that all edges in M are mapped to edges in M' such that the source and target nodes of the edge in M are mapped to the corresponding source and target nodes of the corresponding edge in M'. The same for attributes. A morpism $M \xrightarrow{m} M'$ between two typed graphs M and M' is additionally type preserving, i.e., nodes, edges and attributes are mapped to nodes, edges and attributes of the same type.

A triple graph $M = (M_{D_1} \xleftarrow{d_1} M_C \xrightarrow{d_2} M_{D_2})$ [SK08, EEGH15] consists of graphs M_{D_1}, M_{D_2} for domains D_1, D_2 and a correspondence graph M_C with morphisms $d_1 \colon M_C \to M_{D_1}, d_2 \colon M_C \to M_{D_2}$ for defining the correspondences between elements of M_{D_1} and M_{D_2}. In this paper, we use typed triple graphs, i.e., triple graphs typed over a triple type graph (cf. Ex. 2.1).

A triple graph morphism $M \xrightarrow{m} M'$ between triple graphs $M = (M_{D_1} \xleftarrow{d_1} M_C \xrightarrow{d_2} M_{D_2})$ and $M' = (M'_{D_1} \xleftarrow{d'_1} M'_C \xrightarrow{d'_2} M'_{D_2})$ is given by $m = (m_{D_1}, m_C, m_{D_2})$ with morphisms $M_{D_1} \xrightarrow{m_{D_1}} M'_{D_1}, M_C \xrightarrow{m_C} M'_C$ and $M_{D_2} \xrightarrow{m_{D_2}} M'_{D_2}$, i.e., m consists of three morphisms for each of the triples components. For morphisms m it holds that $m_{D_1} \circ d_1 = d'_1 \circ m_C$ and $m_{D_2} \circ d_2 = d'_2 \circ m_C$, i.e., both parts of the diagram commute (annotated in the diagram with (=)), which means that correspondences between elements are preserved.

$$\begin{array}{ccccc} M = M_{D_1} & \xleftarrow{d_1} & M_C & \xrightarrow{d_2} & M_{D_2} \\ \downarrow m & & \downarrow m_{D_1}\ (=) & & \downarrow m_C\ (=) \downarrow m_{D_2} \\ M' = M'_{D_1} & \xleftarrow[d'_1]{} & M'_C & \xrightarrow[d'_2]{} & M'_{D_2} \end{array}$$

Example 2.1 (Attributed Triple Graphs & Typing) *The attributed triple graph in Fig. 3 depicts the triple type graph for the models in our running example (Ex. 1.1). It consists of attributed graphs* $MM_{Managenemt}$ *and* $MM_{Therapy}$ *which are the type graphs for the two domains as well as the correspondence graph* $MM_{Correspondence}$*. Morphisms* s *and* t *relate* Persons *with* Patients *via correspondence node* PP*. Each model of the management domain may contain several* Rooms *and* Persons*.* Persons *may be assigned to* Rooms *(edge* asg*). Each* Room *may have a* Number *and a* Capacity*. Each* Person *may have a* Name*, date of birth (attribute* DOB*) and date of* Admission*.*

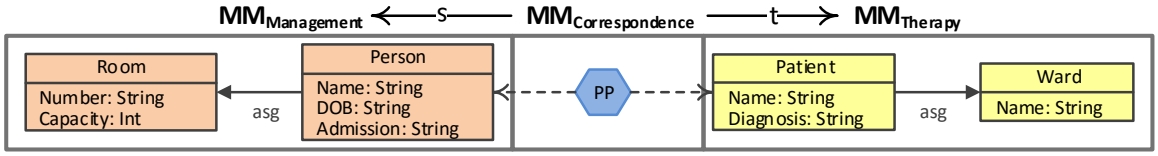
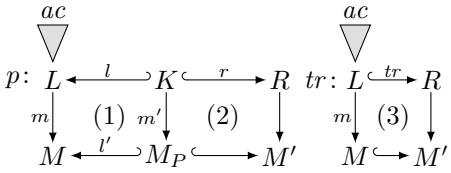


Figure 3: Meta-model (type graph) of models in the hospital scenario

Each model of the therapy domain may contain several Wards *and* Patients. Patients *may be assigned to* Wards *(edge* asg*). Furthermore, each* Ward *and* Patient *may have a* Name. *Each* Patient *additionally may have comments on his* Diagnosis. *The attributed triple graph of Fig. 2 is typed over the triple type graph in Fig. 3 by a type triple graph morphism. In the visual notation, the typing is made explicit by a leading colon where each element* : Type *in Fig. 2 is mapped to element* Type *in Fig. 3 by the type morphism.*

Note that formally, the type graph in Fig. 3 is incomplete, because we omitted the additional node type View *with attribute* Name *(of values* Nurse *or* Doctor*) and edges to types* Ward *and* Patient. *That node type is used to flag whether model elements belong to the doctors' view or the nurses' view. In the remainder of the paper, we have chosen the following short-notation for model* $M_{Therapy}$*: boxes* Nurses' View *and* Doctors' View *indicate the corresponding flags.* △

A graph transformation rule $p\colon L \xleftarrow{l} K \xrightarrow{r} R$ contains a left-hand side (LHS) graph L, a right-hand side (RHS) graph R, a context graph K and a span of inclusions l and r. Intuitively, l specifies the deletions and r the additions of the rule. An inclusion $G \xrightarrow{i} G'$ is a morphism with $G \subseteq G'$.



A transformation step $M \xRightarrow{(p,m)} M'$ is given by the application of p to a graph M via a match morphism m leading to the diagram on the right with commuting pushouts (1) and (2) and the result graph M'. Intuitively, a pushout (1) over morphisms l, m' is the gluing of graphs L and M_P via common K resulting in graph M with morphisms m, l'. Rule p is applicable to M via match m, if pushouts (1) and (2) exist. Intuitively, the application of p to M via match m leads to M' which is obtained from M by deleting all elements M that are in L but not in K while preserving all elements of K in M leading to intermediate graph M_P and finally, creating all elements that are in R but not in K.

A triple graph transformation rule $L \xrightarrow{tr} R$, in short triple rule, contains LHS triple graph L, RHS triple graph R and inclusion tr. From that it follows, that triple rules are only creating. The application of triple rules is analogue to general rules but restricted to a single pushout (3). Additionally, a (triple) rule may be equipped with negative (positive) nested application conditions, in short NACs (PACs), in order to restrict the application of the rule [HP09, EEGH15]. For a given rule and a match from the rule to a graph, a NAC (PAC) ac defines a context for the matched graph part that is forbidden to exist (must exist) in order that the rule is applicable to the graph via the match.

Remark 2.1 (Short Visual Notation) *In the remainder of the paper, we use the short notation of (triple) graph transformation rules. In detail, it means that elements, that will be added are marked with ++ and are annotated in green. Those elements are only contained in the RHS. All unmarked elements are contained in the LHS and RHS of the rule and will be preserved by the (triple) graph transformation rule. Elements that will be deleted are marked with −− and are annotated in red. Those elements are only contained in the LHS of the rule. Triple graph rules are non-deleting, i.e., the triple rules in Ex. 2.2 do not delete any element. Yet, we chose the same short notation for visualising the model update (which is deleting in our running example).* △

Example 2.2 (Triple Rules) *The triple rules in Fig. 4 define how integrated models (triple graphs) of our hospital scenario are created. All elements that are enclosed by a NAC box are forbidden to exist in M so that the rule is applicable to M.*

Rule 1 Ward *creates a ward with name for the nurses' and doctors' view, but only if there does not already exist a ward of the same name (cf. NAC).*

Rule 2 Diagnosis *adds a diagnosis to an existing patient in the doctors' view.*

Rule 3 Person-2-Patient *assumes that there already exists a room in the management model and a ward with the given name in the nurses' and doctors' views of the therapy model. Then, the rule simoultaneously creates a*

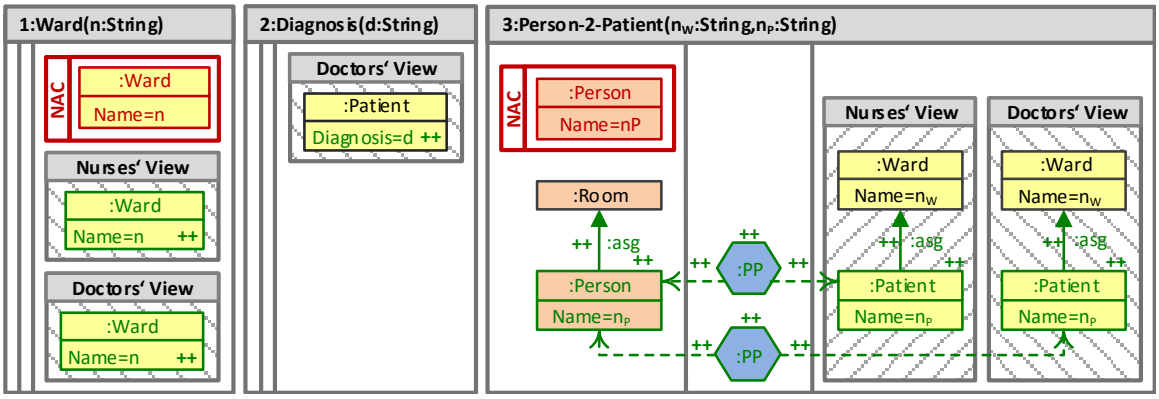


Figure 4: Triple rules of our running example

person with a name and its corresponding patient nodes with the same name in both views and its correspondences. Finally, the person node (and patient nodes) are assigned to a room (ward). The rule is only applicable, if there does not already exist a person of the same name in the management model.

Note that in order to create the graph of Fig. 2, additional rules are required that we implicitly assume but do not explicitly present. This comprises rules for creating rooms, DOB, and admission node attributes. △

A triple graph grammar $TGG = (MM, \emptyset, TR)$ over domains D_1 and D_2 consists of a triple type graph $MM = (MM_{D_1} \xleftarrow{d_1} MM_C \xrightarrow{d_2} MM_{D_2})$ for domains D_1 and D_2 (cf. Fig. 3), the empty start graph and a set of triple rules TR (cf. Fig. 4). The language $\mathcal{L}(TGG) = \{M' \mid \text{there exists } \emptyset \xRightarrow{tr^*} M'\}$ that is generated by a TGG is defined by all triple graphs M' that are typed over MM and can be created by successively applying the rules $tr \in TR$ starting at the empty graph. The language $\mathcal{L}(TGG)^{D_1}$ comprises all graphs of $\mathcal{L}(TGG)$ but restricted to the graph component for domain D_1. (And similar for $\mathcal{L}(TGG)^{D_2}$.) With $\mathcal{L}(MM)$ ($\mathcal{L}(MM_{D_i})$) we define the set of graphs that are typed over type graph MM (MM_{D_i} for $i = 1, 2$). In general, $\mathcal{L}(TGG) \subseteq \mathcal{L}(MM)$, since, the graphs in $\mathcal{L}(TGG)$ are additionally restricted by the rules of the TGG.

Example 2.3 (Languages of Triple Graph Grammars) *Given the $TGG = (MM, \emptyset, TR)$ over type graph MM of Fig. 3 and with rules TR of Ex. 2.2, then the triple graph of Fig. 2 is contained in $\mathcal{L}(TGG)$. Therefore, graph $M_{Management}$ is contained in language $\mathcal{L}(TGG)^{Management}$ and graph $M_{Therapy}$ is contained in language $\mathcal{L}(TGG)^{Therapy}$. Furthermore, graph $M_{Management}$ is also contained in $\mathcal{L}(MM_{Management})$ and graph $M_{Therapy}$ is also contained in $\mathcal{L}(MM_{Therapy})$.* △

3 Propagation of Model Updates

We focus on consistent propagations of model updates. Therefore, we clarify the notion of consistency first. Given a TGG over type graph $MM = (MM_{D_1} \xleftarrow{s} MM_C \xrightarrow{t} MM_{D_2})$, then all graphs in $\mathcal{L}(TGG)$ are called consistent integrated models. All graphs in $\mathcal{L}(TGG)^{D_1}$ are called consistent models of domain D_1 and all graphs in $\mathcal{L}(TGG)^{D_2}$ are called consistent models of domain D_2. A model update leads to state changes in models and is given by a span of inclusions $u\colon M \xleftarrow{u_1} M_P \xrightarrow{u_2} M'$. All elements in $M \setminus u_1(M_P)$ are deleted, all elements in M_P are preserved and all elements in $M' \setminus u_2(M_P)$ are created. An update is consistent, if M' is a consistent model. An update is called D_1-update, if it leads to changes in models of domain D_1 only and it is called D_2-update, if it leads to changes in models of domain D_2 only. If the model update is unknown, existing methods of difference compution [EEGH15] can be used to obtain the update from the state-changes of models.

Definition 3.1 ((Domain) Model Update) *Given models M, M', then the update $u = (u_1, u_2)$ between M and M' is given by a span of inclusions $u\colon M \xleftarrow{u_1} M_P \xrightarrow{u_2} M'$. Given a TGG over triple type-graph $MM = (MM_{D_1} \xleftarrow{s} MM_C \xrightarrow{t} MM_{D_2})$, then u is called a D_i-domain update, short D_i-update, if $M, M_P, M' \in \mathcal{L}(MM_{D_i})$ ($i = 1, 2$). A D_i-update is consistent, if $M' \in \mathcal{L}(TGG)^{D_i}$. With $\Delta_{D_i} = \{u \mid u \text{ is a } D_i\text{-update}\}$ we denote all updates in domain D_i.* △

Example 3.1 ((Domain) Model Update) *We consider the update illustrated in Fig. 5 (top) in the therapy domain of model $M = M_{Therapy}$ (cf. Fig. 2). Due to readability we present an extract of the model but implicitly*

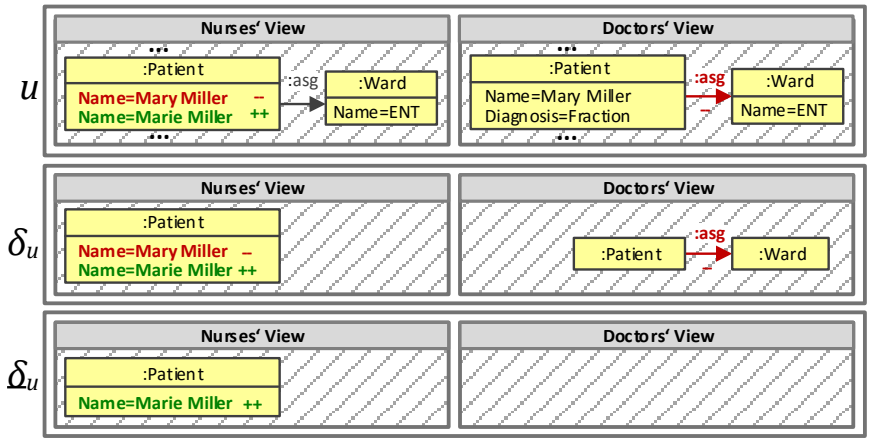


Figure 5: Therapy domain model update (u), delta (δ_u) & creating delta ($\underline{\delta}_u$)

assume the whole model. The update consists of the following steps: A nurse is correcting the name of patient Mary Miller *to* Marie Miller*. Simultaneously, a doctor removes the assignment of this patient to ward* ENT*. Note that the update is not consistent w.r.t. the TGG of Ex. 2.3, i.e.,* $M' \notin \mathcal{L}(TGG)^{Therapy}$*. To be consistent, the name of* Mary *also needs to be updated to the same new name in the doctors' view and the assignment to the ward should not be deleted following rule 3* Person-2-Patient *of Fig. 4.* △

The delta δ_u of an update u specifies the update in mimimal context and is given by u restricted to those elements only that are touched by the update. This includes all elements that are created and deleted by u as well as the elements that are directly connected to them. The creating delta $\underline{\delta}_u$ of an update u is delta δ_u but restricted to the created elements of u only.

Example 3.2 (Delta & Creating Delta) *The delta* δ_u *of update* u *Fig. 5 (middle) only contains those elements of* u *that are affected by the update. The corresponding creating delta* $\underline{\delta}_u$ *of* u *is* δ_u *but restricted to the created elements of* u *only.* △

The problem of propagating domain model updates according to a given TGG, is to extend them such that they fit to the TGG, e.g., the update of Fig. 5 needs to be extended in order to also cover the change of Mary's name in the doctors' view. The extension is performed by applying propagation operations. The propagation framework according to a given TGG is given by two total and deterministic propagation operations, one for each domain of the TGG. An operation is total, if for each valid input it leads to a result. An operation is determistic, if it has functional behaviour, i.e., it terminates and leads to a unique result for each valid input, and the operation does not require backtracking.

The propagation operation Ppg_{D_i} for one domain D_i ($i \in \{1, 2\}$) consists of the two steps Del and Add that are illustrated in Fig. 6. The following listing summarises the two steps of Ppg_{D_i}:

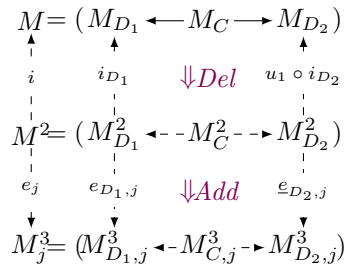


Figure 6: Propagation operation Ppg_{D_2} in domain D_2

- **Step Deletion** (Del)**:** Given a model M and a model update u, then the Del step deletes everything from M that is deleted by update u, first. Then, Del deletes everything that is related to the update in order to obtain a maximal consistent integrated sub-model (namely M') of M w.r.t. the given TGG. Thus, step Del propagates the deletion of elements along different views.

- **Step Addition** (Add)**:** The Add step works with two models: Model M (no deletion performed) and M' (the result of the Del step).
First of all, the Add step adds everything to M and M' that is created by update u. Then, markings are added to both models. Each element that is added by update u is marked with **F** (False, means: not translated). All other (untouched) elements are marked with **T** (True, means: translated). Afterwards, in the Ext sub-step, two special kinds of triple rules, are iteratively applied to M' and M, in order to change the **F** markers are to **T**. At the same time, elements

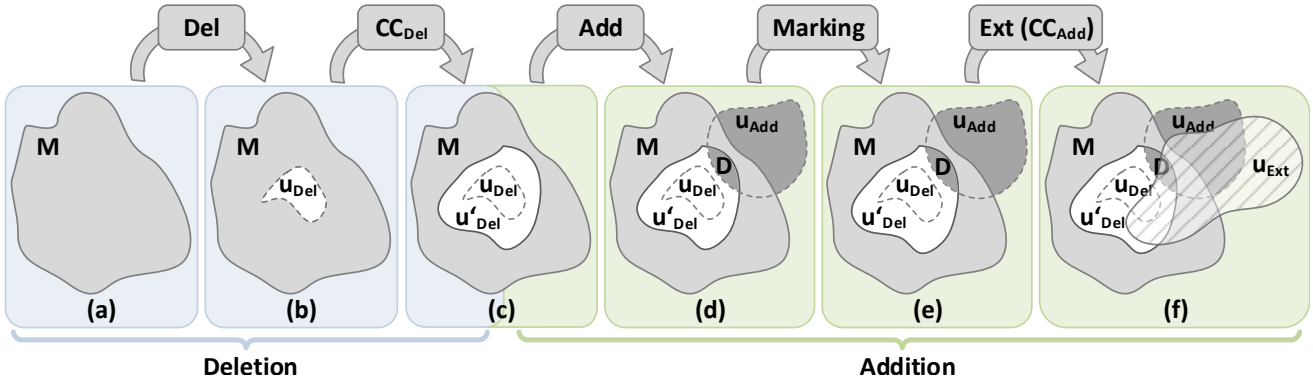


Figure 7: Propagation of domain model updates: (a) - (f) illustrating sub-steps

may be reconstructed that were deleted during the Del step, but that are necessary to derive a consistent model. The special kinds of rules used are called *shifted rules* and *consistency creating rules* (CC rules). The latter are a special kind of shifted rules, which only change markers. The iteration stops when no **F** marker is available anymore (successful), or when no shifted (or CC) rule could be found that is applicable (abort). The derived propagation framework $PPG(TGG)$ returns the modified model M'. If the derived propagation framework finishes successfully, then the returned model is consistent.

The derived propagation framework of model updates $PPG(TGG)$ summarises all propagation operations Ppg_{D_i} $(i \in \{1,2\})$. It is defined as follows:

Definition 3.2 (Propagation Problem & Framework) *Let $TGG = (MM, \emptyset, TR)$ be a triple graph grammar over triple type graph $MM = (MM_{D_1} \xleftarrow{s} MM_C \xrightarrow{t} MM_{D_2})$ for domains D_1, D_2. A triple graph $M = (M_{D_1} \leftarrow M_C \rightarrow M_{D_2}) \in \mathcal{L}(MM)$ coincides with an update $u\colon M \leftarrow M_P \rightarrow M' \in \Delta_{D_i}$, if $M = M_{D_i}$ $(i = 1,2)$.*

The D_i-propagation problem is to construct an operation $Ppg_{D_i}\colon UD \rightarrow \Delta_{D_i}$ from tuples $UD = \{(M, u, \delta_u) \mid u \in \Delta_{D_i}, M$ coincides with u, δ_u is delta of $u\}$ of D_i-updates Δ_{D_i} on triple graphs with corresponding deltas to extended D_i-updates $(i = 1,2)$.

The propagation framework $PPG(TGG) = (\Delta_{D_1}, \Delta_{D_2}, Ppg_{D_1}, Ppg_{D_2})$ is given by all updates $\Delta_{D_1}, \Delta_{D_2}$ of domains D_1 and D_2 as well as total and deterministic propagation operations Ppg_{D_1}, Ppg_{D_2} for both domains. △

As illustrated in Fig. 7, intuitively, given a model M **(a)** and a model update $u\colon M \xleftarrow{u_1} M_P \xrightarrow{u_2} M'$, then step Del first deletes everything $u_{Del} = M \setminus u_1(M_P)$ from M that is deleted by u **(b)**. Del additionally deletes everything u'_{Del} from M that is related to u_{Del} in order to obtain a maximal consistent integrated sub-model $M \setminus u'_{Del}$ of M w.r.t. the given TGG **(c)**. Thus, step Del propagates the deletion of elements along different views. In a first sub-step of step Add those elements $u_{Add} = \underline{R} \setminus (M \setminus u'_{Del})$ are added to the model that are created by the update leading to model $(M \setminus u'_{Del}) \cup u_{Add}$ **(d)**. Previously deleted elements D by Del may be recreated by this sub-step. Then, the models will be marked **(e)**, which is used in the last sub-step, the extension sub-step. In applying shifted rules and CC rules, the model will be extended in order to derive a consistent integrated model **(f)**. We will review all steps in detail using the running exmaple.

Example 3.3 (Del Step) *Applying the update in Fig. 5 to model M in Fig. 2 will delete the name of patient* Mary Miller *in the nurses' view as well as her assignment to ward* ENT *in the doctors' view of the therapy domain. The result of this first sub-step is visualised in Fig. 8 (left). This completes sub-step* Del *in Fig. 7 (b).*

Model I is a triple graph, where the Therapy *domain is modified in the sense that the deletion part of the update u is performed on that domain, wheras the* Management *domain and the correspondence domain stay unchanged.*

Afterwards, by following Rule 3 Person-2-Patient, Del *additionally deletes the whole patient node of* Mary Miller *not only in the nurses' view but also in the doctors' view plus the corresponding* : Person *node in the management domain and the affected correspondences* : PP. *This leads to the maximal consistent integrated sub-model M^2 (cf. Fig. 8) (right). (Note, the red elements marked with $--$ are deleted, i.e., they are not part of M^2 anymore. In Fig. 8) (right) they are visualised for better understanding of that step.)*

Model M^2 can be created by a terminating sequence of rule applications with the triple rules in Fig. 4. Thus, the sub-model is consistent w.r.t. the given TGG ($M^2 \in \mathcal{L}(TGG)$). This completes sub-step CC_{Del} *in Fig. 7 (c).* △

The Add step is divided into three sub-steps: The first sub-step extends the model I (intermediate result from Del) and M^2 (result from Del) by the update delta $\underline{\delta_u}$ resulting in models M' and M^R. This completes sub-step Add in Fig. 7 (d).

Afterwards, in the second sub-step, the extended models M^R and M' will be marked via model M^2 (c.f. Fig. 10). This completes sub-step Marking in Fig. 7 (e).

In the third sub-step, the framework iteratively extends both models from the second sub-step according to the given set of triple rules TR in order to derive a consistent model satisfying the model update u and the corresponding TGG. Note that it may occur, that no consitent model can be derived. Then, the propagation will abort returning an inconsistent model. This completes sub-step Ext in Fig. 7 (f).

Example 3.4 (Add Step - Sub-Step 1) *In detail, the update creates the attribute* Name *with updated value* Marie Miller *in the nurses' view (c.f. Fig. 9 (left)). As the corresponding* : Patient *node has been previously deleted by* Del*, the node together with the updated name attribute are added to the nurses' view. Therefore, the* : Patient *node is recreated. Furthermore, the deletion of the* : asg *edge is also applied to the nurses' view. The resulting model of the first sub-step is shown in Fig. 9 (right).* △

Example 3.5 (Sub-step 2: Markings) *Let us consider Fig. 10. Triple graphs M' and M^R are marked via interface graph M^2 that is illustrated on the top. During marking, triple graph M' is enriched by the following markers: All elements that can be mapped by M^2 are marked with* **T***, i.e., these are all elements that are not touched by update u. In detail, the* : Person *node of* Polly Parker *and the corresponding* : Patient *nodes including the correspondence nodes* : PP *of that person are marked with* **T***. Additionally the* : Room *node and both* : Ward *nodes are marked with* **T***. Furthermore, all attributes included by those nodes and all necessary* : asg *edges between those nodes and also from and to the corresponding* : PP *nodes are marked with* **T***, too. All other elements of M' are marked with* **F***, i.e.,* : Person *and* : Patient *nodes of* Mary Miller *and the modified* : Patient *node of* Marie Miller*, including the contained attributes and the correspondence nodes* : PP *and the necessary* : asg *edges and edges between from and to the corresponding* : PP*.*

Triple graph M^R is enriched by the **T** *and* **F** *markers accordingly. In detail, all nodes including the contained attributes concerning* : Person *or* : Patient Polly Parker*, respectievly, are marked with* **T***. Beyond, all* : Room *and* : Ward *nodes and their attributes are marked with* **T** *and all edges between the mentioned nodes. In contrast, node* : Patient *and its attribute* Name = Marie Miller *get marker* **F***.* △

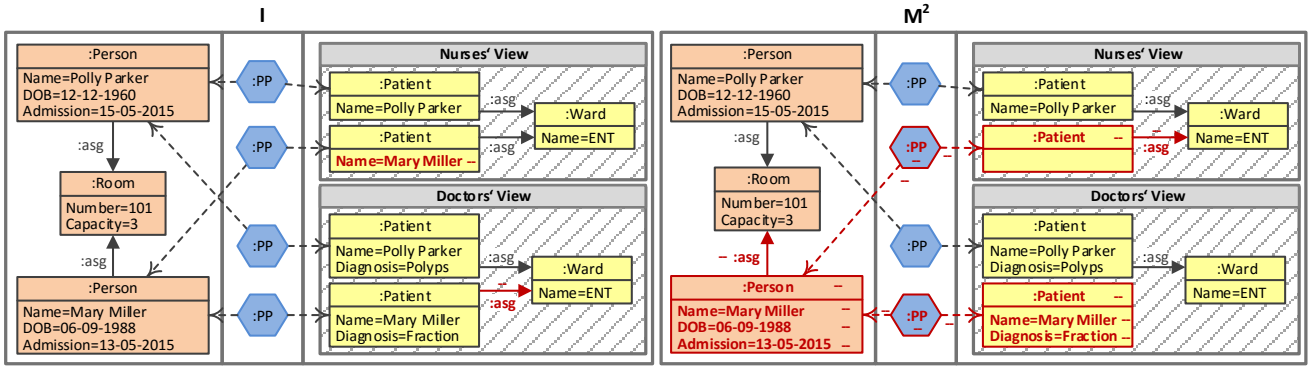


Figure 8: Intermediate model of Del step (left) and result of Del step (right)

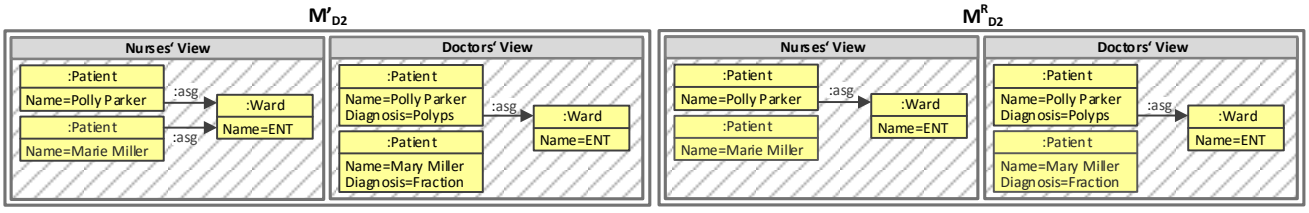


Figure 9: Models M'_{D_2} and $M^R_{D_2}$, illustration of *Therapy* domain only

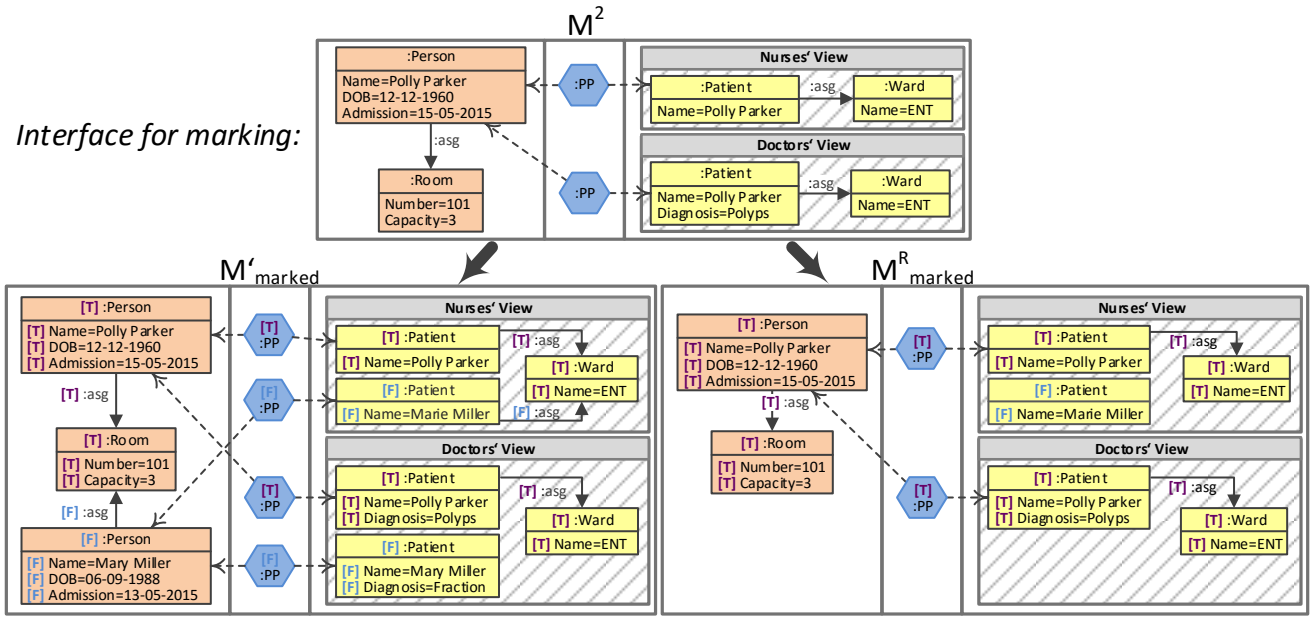


Figure 10: M' and M^R are marked resulting in M'_{marked} and M^R_{marked} (bottom) via interface M^2 (top)

The third sub-step Ext extends the given domain update u such that the update fits to the given TGG. Therefore, Ext takes models M^R_{marked} and M'_{marked} from the second sub-step of Add as input.

Sub-step Ext analyses the applicability of the triple rules $tr \in TR$ from the given TGG to M^R_{marked} such that each application maximally overlaps with that part of M^R_{marked} which is marked with **F**. From each overlapping triple rule that is applicable via a match, an operational rule, called shifted rule sr_1, is derived which creates only those elements of the triple rule that do not overlap. Afterwards, Ext extends the maximal overlapping so that it "fits" to M'_{marked} and consequently, a second shifted rule sr_2 is derived. This will recreate necessary elements. With the help of the shifted rule sr_2 and model M^R_{marked} an intermediate model $\underline{M}^R_{marked}$ is derived. Finally, sr_2 will be applied to $\underline{M}^R_{marked}$ and also to M'_{marked} leading to models $M^{R'}_{marked}$ and M''_{marked}, where the markings of overlapping elements will be changed from **F** to **T** ([**F** > **T**]). (For details, see Ex. 3.6). The markings **T** and **F** are introduced in order to control which part (elements) of an update already have been extended.

If such a shifted rule cannot be found, then the Ext step checks for complete overlappings, i.e., no elements will be created. In that case, a special kind of operational rule is applied which is called consistency creating rule (CC rule) [HEO+15]. Those rules only change markers from **F** to **T**.

Applications with empty overlappings are omitted unless they completely overlap with previously deleted elements. If the overlapping contains previously deleted elements, then the application is additionally maximally overlapped with previously deleted elements in order to guide the search for applicable matches. Therefore, previously deleted elements may be recreated in the extension process.

Then, models $M^{R'}_{marked}$ and M''_{marked} are taken as input for sub-step Ext again, as long as all elements are not marked with **T** or no rules are applicable any more. The successive application of shifted rules extends an update step-wise in the sense that the elements which are created by the update are complemented by those elements of the underlying triple rules which do not overlap with the update and therefore, could not be shifted but would also be created when applying the triple rules.

As, in all sub-steps of Add, previously deleted elements may be recreated, the proposed propagation framework prioritises creation over deletion. In contrast to propagating the deletion of elements by Del, step Add propagates the creation of elements along different views.

A *shifted rule* is derived from a triple rule by shifting elements from R to L resulting in new left- and right-hand sides $\underline{L}$ and $\underline{R}$. All elements of L in $\underline{L}$ and all elements of $\underline{R}$ are marked with **T** while the markings of the shifted elements is changed from **F** to **T**, i.e., $\underline{K}$ is $\underline{L}$ without F-markings. Shifted rules are a generalisation of *consistency creating (CC) rules* (c.f. Ex. 3.6). CC rules were defined in the theory of model synchronisations to mark consistent integrated sub-models of possibly inconsistent models [HEO+15]. Given a triple rule $tr\colon L \to R$, the CC rule of tr is derived by shifting all elements from R to L. CC rules are used for applications (update extensions) based on complete overlappings of the underlying triple rule with previously deleted elements, i.e.,

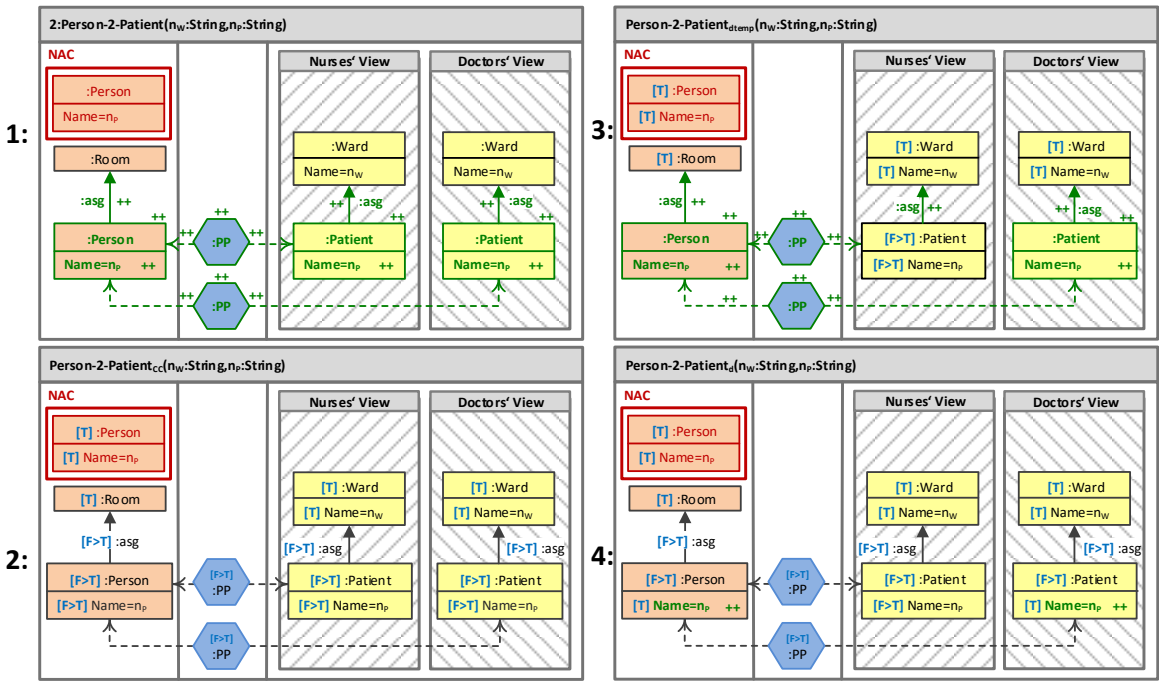


Figure 11: Triple rule Person-2-Patient (1), CC rule (2), & two relevant maximal shifted rules (3) and (4)

CC rules only change markings from **F** (not translated) to **T** (translated). For update extensions, only shifted rules are of relevance where at least one element is shifted, i.e., the shifted rule is *relevant*. This omits applications of shifted rules which do not overlap with the elements that are created by the update and therefore are of no importance concerning the update. Furthermore, shifted rules have to be *maximal* in the sense that they should shift and match as much as possible elements such that the overlappings are maximal and only the non-existing (non-overlapping) elements of the underlying triple rules are created by their application.

Example 3.6 (Relevant Maximal Shifted Rule & CC Rule) *Given model* M^R_{marked} *as illustrated in Fig. 10 (bottom, right), triple rule* Person-2-Patient *(Rule 3) from Fig. 4 and Fig. 11 (1), respectively, and a decomposition* d *of Rule* Person-2-Patient*. Intuitively, the shifted rule that results out of the decomposition* d *shall "fit" to triple graph* M^R_{marked}*, i.e., additionally to all elements from the LHS of triple rule* Person-2-Patient*, it shall contain the* : Patient *node including the* Name = Marie Miller *attribute in the nurses' view. Technically, this is achieved in the following way: Let* $d\colon L \to L' \to R$ *be given by the LHS* L *of Rule* Person-2-Patient *and* L' *which additionally contains node* : Patient *and its attribute* Name = Marie Miller *in the nurses' view. Therefore, decomposition* d *controls the shifting of elements from* R *towards* L*. The rule* Person-2-Patient$_{dtemp}\colon \underline{L} \leftarrow \underline{K} \rightarrow \underline{R}$ *in Fig. 11 (3) is a relevant maximal shifted applicable rule of rule* Person-2-Patient *w.r.t.* d *and* M^R_{marked}*. It is an operational rule derived from triple rule* Person-2-Patient *(cf. Fig. 11 (1)).*

The application of the shifted rule updates the markings of the two shifted elements from **F** *to* **T** *(***F** > **T***). The additional elements in the NAC are all marked with* **T***. The shifted rule is relevant, as, two elements were shifted, i.e.,* L *and* L' *are not isomorphic (*$L \ncong L'$*). Beyond, the rule is maximal shifted and applicable w.r.t.* M^R_{marked}*, as, there exists an applicable match from* $\underline{L}$ *to* M^R_{marked} *and no further elements can be shifted such that a new "extended" applicable match is obtained.*

In Fig. 11 (4) a second shifted rule Person-2-Patient$_d\colon \underline{L} \leftarrow \underline{K} \rightarrow \underline{R}$ *is illustrated. It is a relevant maximal shifted rule w.r.t.* O^R_{marked} *(c.f. Fig. 13). There, the following elements were shifted to* $\underline{L}$*:*

1. *the two* : Patient *nodes,*
2. *the* Name *attribute with value* n_P *of the nurses' view,*
3. *the* : Person *node,*
4. *the three* : asg *edges,*
5. *the two* : PP *nodes and*
6. *the four edges starting at the two* : PP *nodes.*

Again, the RHS $\underline{R}$ *of rule* Person-2-Patient$_d$ *is* $\underline{K}$*, but extended by the* Name *attribute of node* : Person *of the*

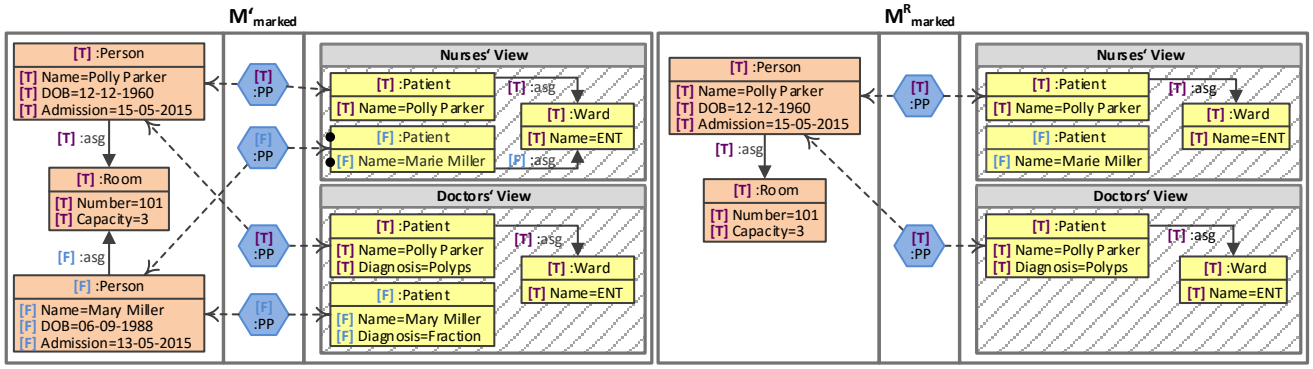


Figure 12: Initial situation, including match from Person-2-Patient$_{\text{dtemp}}$ to M'_{marked} and M^R_{marked}

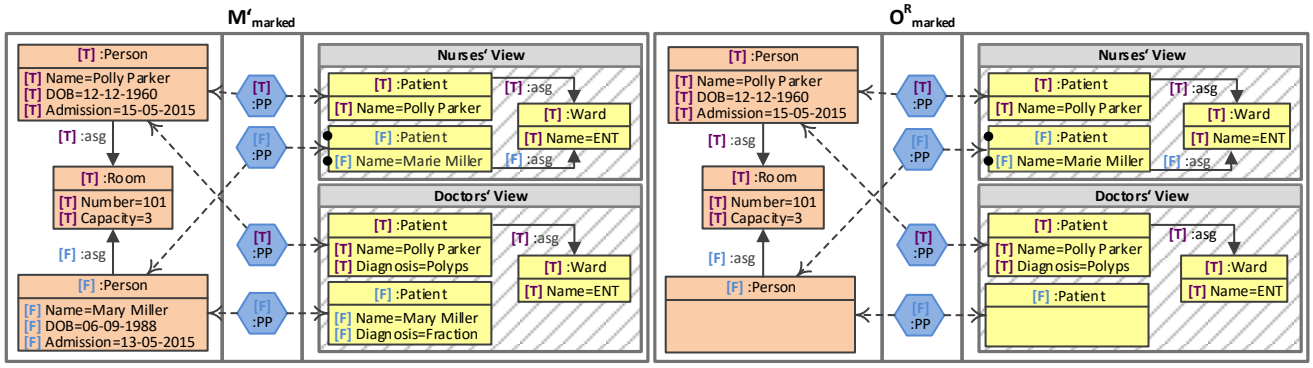


Figure 13: Intermediate model before shifted rule application

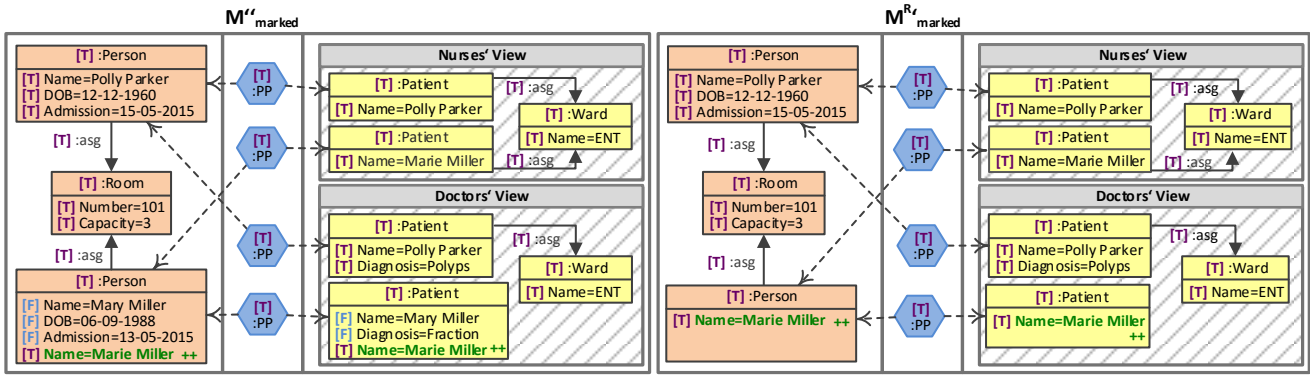


Figure 14: Result of shifted rule application - new input to next Ext iteration

management model and the Name *attribute of node* : Patient *in the doctors' view.*

The rule Person-2-Patient$_{CC}$ *in Fig. 11 (2) is the CC rule of rule* Person-2-Patient. *All elements are shifted from the RHS towards the LHS. Therefore, when applying a CC rule no elements are created but the markings of the shifted elements are updated from* **F** *to* **T**. *The markings of all other elements remain* **T**. △

Example 3.7 (Sub-step Ext **of** Add **in Great Detail)** *The* Ext *sub-step uses the marked models from the second sub-step of* Add *as input. This initial situation is illustrated in Fig. 10.*

1st iteration: *First,* Ext *checks, if a shifted rule can be found with regard to the models M'_{marked} and M^R_{marked} (c.f. Fig. 10). This is not possible. A shifted rule can be found with match to M^R_{marked} which is shifted rule* Person-2-Patient$_{\text{dtemp}}$ *(c.f. Fig. 11 (3)). This situation is shown in Fig. 12, where the match is explcitly illustrated with black dots. Based on this "small" shifted rule, an "extended" shifted rule* Person-2-Patient$_{\text{d}}$ *(c.f. Fig. 11*

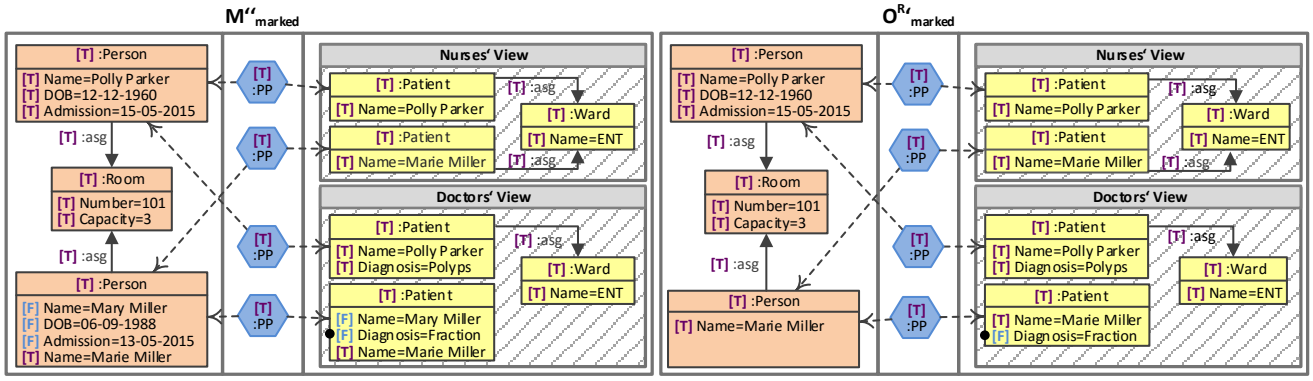


Figure 15: Intermediate model, including match from $\mathsf{Diagnosis}_{\mathsf{CC}}$ to M''_{marked} and $O^{R'}_{marked}$

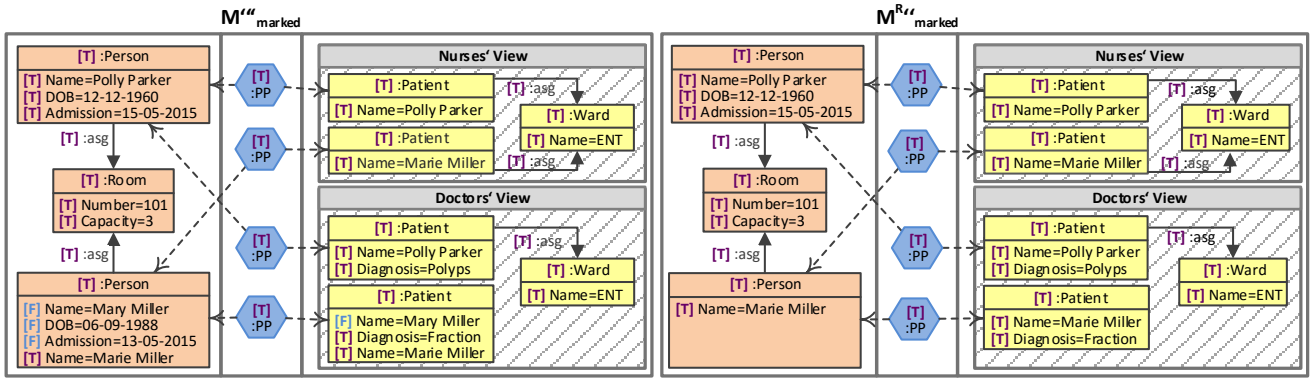


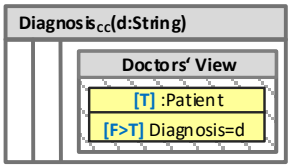
Figure 16: Result of CC rule application

(4)) is derived. Then, an intermediate model O^{R}_{marked} *(c.f. Fig. 13) is constructed. (For details: We create an effective pushout (c.f. Def. 4.23 in [EEGH15]) via* M'_{marked}, M^{R}_{marked} *and* $\mathsf{Person\text{-}2\text{-}Patient}_{\mathsf{dtemp}}$. *Intuitively in sets, an effective pushout* E *is the union of* B *and* C *over a common* D, *whereas* D *is the intersection of* B *and* C *over a common* A.*)*

There, we can see that all elements are reconstructed that are necessary for the application of shifted rule $\mathsf{Person\text{-}2\text{-}Patient}_{\mathsf{d}}$, *i.e., elements that we want to restore. In detail, these elements are:*

- *The* : Person *node and its assignement* : asg *to the correct* : Room *in the* Management *domain.*
- *The empty* : Patient *node in the doctors' view. Note, the assignement to the correct* : Ward *is not reconstructed, because the deletion of this edge is part of upadate* u.
- *The correspondence nodes* : PP *between the* : Person *and both corresponding* : Patient *nodes in the* Therapy *domain.*

Note, all reconstructed elements are marked with $\mathbf{F}$. *Now, shifted rule* $\mathsf{Person\text{-}2\text{-}Patient}_{\mathsf{d}}$ *will be applied resulting in triple graphs* M''_{marked} *and* $M^{R'}_{marked}$ *(c.f. Fig. 14). Both models are used as input for the next iteration of the* Ext *sub-step.*



2nd iteration: *Again,* Ext *checks successively, if a shifted rule can be found. Now, this is not applicable. Instead, we can find a CC rule that is applicable. We consider CC rule* $\mathsf{Diagnosis}_{\mathsf{CC}}$ *shown on the right that is derived out of triple rule 2* Diagnosis *(c.f Fig. 4). Again, we calculate an intermediate triple graph* $O^{R'}_{marked}$ *(using effective pushout) that is extended by the* Diagnosis *attribute with marker* $\mathbf{F}$. *Then, CC rule* $\mathsf{Diagnosis}_{\mathsf{CC}}$ *will be applied to both models in Fig. 15 resulting in triple graphs* M'''_{marked} *and* $M^{R''}_{marked}$ *(c.f. Fig. 16). The latter models are used as input for the next iteration phase of the* Ext *sub-step.*

Next iterations: *We combine the next steps, because in our running example, they are similar to the 2nd iteration phase, because only CC rules are still applicable. They handle the following two attributes contained in node* : Person *of* Marie Miller *in the* Management *domain: a)* $\mathsf{DOB} = 06 - 09 - 1988$,

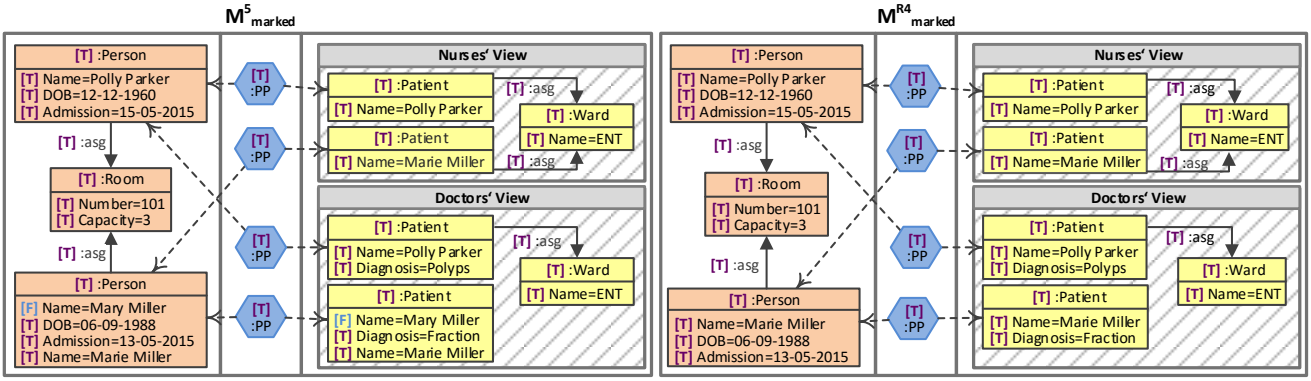


Figure 17: Final result

and b) Admission $= 13 - 05 - 2013$. *Finally, the* Ext *sub-step finishes successfully and returns model* M^{R4}_{marked} *(c.f. Fig. 17 (right)). The other model* M^{5}_{marked} *(left) will be discarded. The propagation ends successfully, because all translation markers are set to* $\mathbf{T}$*, i.e., a consistent triple graph could be derived. As visualised in model* M^{R4}_{marked}*, the derived propagation framework* $PPG(TGG)$ *reconstructed all necessary elements that were deleted in the* Del *step, although deleting those items was not intended by the user. Note, that the* : asg *edge from the* : Patient Marie Miller *to the corresponding* : Ward *in the doctors' view is restored, too, even though the deletion of this edge was part of update* u*. This is an example for the priorisation of addition over deletion of the* $PPG(TGG)$ *framework.* △

Finally, in combining all steps that were informally introduced in this section, the derived propagation framework $PPG(TGG)$ can be defined.

Definition 3.3 (Derived Propagation Framework) *Given a TGG over domains* D_1 *and* D_2*, then each propagation operation* $(Ppg_{D_i})_{i=1,2}$ *of the derived propagation framework* $PPG(TGG)$ *is defined by the sequence of steps according to Rem. 3.1.* △

Remark 3.1 (Derived Propagation Framework) *Given a* $TGG = (MM, \emptyset, TR)$ *over type graph* MM *and with a set of triple rules* TR*. Furthermore, given a triple graph* M *and a consistent domain model update* u*.*

Then, the derived propagation framework $PPG(TGG)$ *applies update* u *on* M *and derives a consistent model* M' *in executiong the following steps:*

1. Del *step,*
2. Add *step consiting of sub-steps:*
   (a) *sub-step 1, first addition,*
   (b) *sub-step 2, i.e., marking,*
   (c) *sub-step 3, extension* Ext *is executed iteratively.*

△

4 Related Work

The presented propagation framework assumes that the underlying TGG is deterministic. As TGGs are non-deterministic in general [HEO+15], we introduced a technique for obtaining deterministic sets of operational rules in previous work. The framework of concurrent model synchronisations [HEEO12, GHN+13] is equipped with an additional first step for deriving consistent model updates from inconsistent ones. In contrast to the present paper, model synchronisations do not deal with propagating model updates across one model but propagating model updates between two models of typically different domains, called source and target domains. A consistency creating operation CCS (CCT) is used for converting inconsistent updates in the source (target) domain to consistent updates. However, in contrast to our approach, both operations simply neglect inconsistent model updates u without trying to propagate them across the model, i.e., both operations are not delta-preserving. We have motivated our approach by multi-view models but the approach can also be applied in other contexts where consistent model updates and models are requested and consistency is defined in terms of the language $\mathcal{L}(TGG)$ that is induced by a TGG. The theory of correct and complete model transformations [DXC+11, HEGO14] and

model synchronisations [HEEO12, GHN+13, HEO+15] rely on consistent domain models and consistent model updates based on TGGs.

Independently from the present approach, another solution for multiple views based on the existing theory of model synchronisations [HEO+15, EEGH15] may be feasible. Assuming a separate model and TGG for each view, then model updates in one view may be propagated back to the source model at first, followed by a forward propagation from the source model to the other view. This solution does not only differ in the assumption of separate models and TGGs for each view from the presented approach, but also rises questions concerning the conflict resolution of simoultaneous updates in several views. Furthermore, it is unclear how to deal with updates of elements that are only contained in the views but not reflected by the source model. The propagation of an update is performed over the source model and therefore, the update would be lost. Developing such a solution and relating it to the presented approach is topic of future work.

Another approach that is related to multi-view models are view triple graph grammars (ViewTGGs) [JKS06, JS08, ARDS14]. In our presented running example, the target domain can be interpreted as ViewTGG. The nurses' view is a view of the doctors' view. In order to apply the theory of ViewTGGs to our running example, we should divide the information in the target model in a second triple graph grammar: one domain contains the doctors' view, the other the nurses' view. Then, we can apply the model synchronisation techniques in [ARDS14]. However, the approach we presented in this paper is more flexible and is also able to deal with models, where the concept of ViewTGGs cannot be applied, but where elements in one domain are still strongly interweaved with each other.

5 Conclusion & Future Work

We presented a first informal idea of a derived propagation framework for consistently propagation model updates along different views in multi-view models. This framework extends the existing TGG-based model synchronisation framework by propagating changes also between related elements within one domain. We plan to present a formal elaboration of this framework in future work. Furthermore, we want the framework to fulfill the following properties that we will prove using the formalization of the presented propagation framework.

1. The propagation shall preserve **identity**.
2. The propagation shall always yield **consistent** results.
3. If the given model update is consistent, then the propagation shall yield a consistent result.
4. The propagation shall always return a **single result**.
5. The propagation shall have **functional behaviour**, i.e., the propagation shall terminate and shall be deterministic [EEPT06]. Termination and determinism lead to functional behaviour, i.e., for the same model M, the propagation shall always produce the same result.

It is also desired, to implement the approach as an extension to the existing model transformation tool HenshinTGG [Hen15]. HenshinTGG allows the specification and execution of model transformations between source and target languages based on TGGs. It also provides an implementation of synchronising consistent model updates from source to target and vice versa based on the theory in [HEEO12, GHN+13, EEGH15]. Therefore, an implementation of the approach would enable the synchronisation (complete transformation) of inconsistent source and target updates (models) in HenshinTGG.

Furthermore, we plan an evaluation of the presented technique on the real world scenario which we introduced in Sec. 1 where source code of satellite procedures is transformed to multi-layered statecharts [GHE+13]. Each layer represents one view of the source code at a different level of abstraction. As views of adjacent levels of abstraction may contain overlapping elements, updates made in one view should be consistently propagated to adjacent views. This enables the synchronisation of the extended update back to the source code by using the implementation mentioned above.

Furthermore, the derived propagation framework, we introduced in this work, considers only model updates in the same domain and also in the same view at this stage. We want to investigate more complex updates in future work. There, it is also relevant to anaylse the propagation of conflicting model updates. These conflicts may lead to situations where no consistent model update can be obtained by the update propagation and a manual conflict resolution is required. This leads to the notion of conflicting updates that may be forbidden for users as their propagation may yield side effects that are not transparent to the user. For a given model transformation, we plan to provide a construction for obtaining a set of constraints that specify forbidden updates on consistent multi-view models. These constraints can be used directly in order to restrict the user's behaviour.

In addition, we plan to investigate to what extent an additional step for automatic conflict resolution [EET11] as used in concurrent model synchronisations [HEEO12] is beneficial for our approach.

Furthermore, the alternative approach for update propagation between views from Sec. 4 is promising enough to be investigated in future work.

Acknowledgments

Supported by the Fonds National de la Recherche, Luxembourg (3968135, 4895603).

References

[ARDS14] Anthony Anjorin, Sebastian Rose, Frederik Deckwerth, and Andy Schürr. Efficient model synchronization with view triple graph grammars. In Jordi Cabot and Julia Rubin, editors, *Modelling Foundations and Applications*, volume 8569 of *LNCS*, pages 1–17. Springer, 2014.

[BBG05] Sami Beydeda, Matthias Book, and Volker Gruhn. *Model-Driven Software Development*. Springer, 2005.

[DXC+11] Zinovy Diskin, Yingfei Xiong, Krzysztof Czarnecki, Hartmut Ehrig, Frank Hermann, and Fernando Orejas. From state- to delta-based bidirectional model transformations: The symmetric case. In *Model Driven Engineering Languages and Systems, Proc. of the 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011*, pages 304–318, 2011.

[EEGH15] Hartmut Ehrig, Claudia Ermel, Ulrike Golas, and Frank Hermann. *Graph and Model Transformation: General Framework and Applications*. Springer, 2015.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, 2006.

[EET11] Hartmut Ehrig, Claudia Ermel, and Gabriele Taentzer. A formal resolution strategy for operation-based conflicts in model versioning using graph modifications. In *Proc. of the 14th International Conference on Fundamental Approaches to Software Engineering, FASE 2011, Saarbrücken, Germany, March 26-April 3, 2011*, pages 202–216, 2011.

[GHE+13] Susann Gottmann, Frank Hermann, Claudia Ermel, Thomas Engel, and Gianluigi Morelli. Towards bidirectional engineering of satellite control procedures using triple graph grammars. In *Proc. of the 7th Workshop on Multi-Paradigm Modeling, MPM 2013, Miami, FL, September 30, 2013*, pages 67–76, 2013.

[GHN+13] Susann Gottmann, Frank Hermann, Nico Nachtigall, Benjamin Braatz, Claudia Ermel, Hartmut Ehrig, and Thomas Engel. Correctness and completeness of generalised concurrent model synchronisation based on triple graph grammars. In *Proc. of the 2nd Workshop on the Analysis of Model Transformations (AMT 2013), Miami, FL, USA, September 29, 2013*, 2013.

[GLEO12] Ulrike Golas, Leen Lambers, Hartmut Ehrig, and Fernando Orejas. Attributed graph transformation with inheritance: Efficient conflict detection and local confluence analysis using abstract critical pairs. *Theor. Comput. Sci.*, 424:46–68, 2012.

[HEEO12] Frank Hermann, Hartmut Ehrig, Claudia Ermel, and Fernando Orejas. Concurrent model synchronization with conflict resolution based on triple graph grammars. In *Proc. of the 15th International Conference on Fundamental Approaches to Software Engineering, FASE 2012, Tallinn, Estonia, March 24 - April 1, 2012*, pages 178–193, 2012.

[HEGO14] Frank Hermann, Hartmut Ehrig, Ulrike Golas, and Fernando Orejas. Formal analysis of model transformations based on triple graph grammars. *Mathematical Structures in Computer Science*, 24(4), 2014.

[Hen15] HenshinTGG. https://github.com/de-tu-berlin-tfs/Henshin-Editor, 2015.

[HEO+15] Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin, Yingfei Xiong, Susann Gottmann, and Thomas Engel. Model synchronization based on triple graph grammars: correctness, completeness and invertibility. *Software and System Modeling*, 14(1):241–269, 2015.

[HGN+13] Frank Hermann, Susann Gottmann, Nico Nachtigall, Benjamin Braatz, Gianluigi Morelli, Alain Pierre, and Thomas Engel. On an automated translation of satellite procedures using triple graph grammars. In Keith Duddy and Gerti Kappel, editors, *Theory and Practice of Model Transformations*, volume 7909 of *LNCS*, pages 50–51. Springer, 2013.

[HGN+14] Frank Hermann, Susann Gottmann, Nico Nachtigall, Hartmut Ehrig, Benjamin Braatz, Gianluigi Morelli, Alain Pierre, Thomas Engel, and Claudia Ermel. Triple graph grammars in the large for translating satellite procedures. In Davide Di Ruscio and Dniel Varr, editors, *Theory and Practice of Model Transformations*, volume 8568 of *LNCS*, pages 122–137. Springer, 2014.

[HP09] Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(2):245–296, 2009.

[JKS06] Johannes Jakob, Alexander Knigs, and Andy Schürr. Non-materialized model view specification with triple graph grammars. In Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro, and Grzegorz Rozenberg, editors, *Graph Transformations*, volume 4178 of *LNCS*, pages 321–335. Springer, 2006.

[JS08] Johannes Jakob and Andy Schürr. View creation of meta models by using modified triple graph grammars. *Electronic Notes in Theoretical Computer Science*, 211:181 – 190, 2008. Proc. of the 5th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2006).

[SK08] Andy Schürr and Felix Klar. 15 years of triple graph grammars. In *Proc. of the 4th International Conference on Graph Transformations, ICGT 2008, Leicester, UK, September 7-13, 2008*, pages 411–425, 2008.