Softw Syst Model (2014) 13:273–299
DOI 10.1007/s10270-012-0247-y

THEME SECTION PAPER

# Bridging the gap between formal semantics and implementation of triple graph grammars

## Ensuring conformance of relational model transformation specifications and implementations

**Holger Giese · Stephan Hildebrandt · Leen Lambers**

Received: 3 April 2011 / Revised: 20 March 2012 / Accepted: 5 April 2012 / Published online: 28 April 2012


**Abstract** The correctness of model transformations is a crucial element for model-driven engineering of high-quality software. A prerequisite to verify model transformations at the level of the model transformation specification is that an unambiguous formal semantics exists and that the implementation of the model transformation language adheres to this semantics. However, for existing relational model transformation approaches, it is usually not really clear under which constraints particular implementations really conform to the formal semantics. In this paper, we will bridge this gap for the formal semantics of triple graph grammars (TGG) and an existing efficient implementation. While the formal semantics assumes backtracking and ignores non-determinism, practical implementations do not support backtracking, require rule sets that ensure determinism, and include further optimizations. Therefore, we capture how the considered TGG implementation realizes the transformation by means of operational rules, define required criteria, and show conformance to the formal semantics if these criteria are fulfilled. We further outline how static and runtime checks can be employed to guarantee these criteria.

Communicated by Dr. Andy Schürr and Arend Rensink.

This work was developed in the course of the project—Correct Model Transformations—Hasso Plattner Institut, Universität Potsdam and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft. http://www.hpi.uni-potsdam.de/giese/projekte/kormoran.html?L=1.

H. Giese · S. Hildebrandt (✉) · L. Lambers
Hasso Plattner Institute at the University of Potsdam,
Prof.-Dr.-Helmert-Straße 2-3, 14482 Potsdam, Germany
e-mail: stephan.hildebrandt@hpi.uni-potsdam.de

H. Giese
e-mail: holger.giese@hpi.uni-potsdam.de

L. Lambers
e-mail: leen.lambers@hpi.uni-potsdam.de

## 1 Introduction

Model transformations are an important element of Model-Driven Engineering (MDE) [39] and allow several aspects of software development to be automated. Therefore, it is crucial that model transformations are correct and repeatable to support incremental development and maintenance of high-quality software. Like programming languages, model transformation languages require an unambiguous semantics as a reference to enable the verification of the outcome considering the model transformation specification (cf. [12]) and to ensure that different implementations result in the same outcome. In addition, an unambiguous formal semantics and clear understanding of how the relational specification is operationalized can help to identify which optimizations are really the most appropriate.

We will consider this challenge for the specific case of triple graph grammars (TGG) [38], which have a well-understood formal semantics and are quite similar to other relational approaches such as QVT Relational (c.f. [21]). For TGGs, there are different tools, which realize slightly different dialects, such as Fujaba TGG Engine [5], MOFLON [1], or ATOM3 [6]. Furthermore, even for a single tool it holds that different tool versions with different optimizations exist. For the Fujaba TGG Engine, there further exists a batch version with support for incremental synchronization [20], a version optimized for synchronizing multiple updates [13], and a version that further improves the runtime for synchronization [14] and can also be employed, for example, for runtime monitoring [42].

In this paper, we bridge the gap between the formal semantics of triple graph grammars and the batch model transformation of our related implementation [14] (cf. Fig. 1). The formal semantics of TGGs generating related source and target models simultaneously assumes backtracking and ignores non-determinism. Therefore, it cannot be used to build an efficient implementation. Practical implementations for TGGs in contrast should be efficient and therefore usually do not support backtracking, need to produce a unique transformation result, and include further optimizations in order to ensure an efficient solution. Closing this gap for this example is of general interest, as also for other TGG implementations (e.g., [1,26]) as well as other model transformation languages allowing for relational specifications (e.g., ATL [25], QVT [33]) a similar gap exists. Therefore, the outlined approach could serve as a scheme to also close this gap for other cases or to make a comparison with other approaches (e.g., [23,27]) presenting a solution to close this gap.[1]

An initial step towards closing this gap is concerned with allowing triple graphs with more flexible correspondences as in the original formalization of TGGs [37], where triple graphs are defined as morphism spans. When using morphism spans to formalize triple graphs, correspondence nodes need to be connected to exactly one source or target node, respectively. Moreover, these morphism spans need to be flattened [8] such that existing TGG implementations can interpret the morphism spans as specific graphs. We introduce a new category [15], where triple graphs are defined directly as specifically typed graphs such that correspondence nodes may be connected to more than one source or target node, respectively, and such that no flattening for the implementation is needed. Apart from the handling of more flexible triple graphs, our formalization follows the original one [37].

To further close the gap, we provide a first operationalization for TGGs and then step-by-step eliminate assumptions such as backtracking while adding constraints that the TGG rules have to fulfill in order to permit their proper and efficient operationalization: (1) As a starting point, we explain and formally define the formal semantics of TGGs and the related forward and backward transformations named *relational scheme* (see Fig. 1). (2) As a first step, we then derive a naive operationalization for TGGs in the form of the *operational scheme* that employs backtracking and bookkeeping for which we can show conformance to the formal semantics by demonstrating consistency—each transformation result of the implementation must fit to the semantics—and completeness—all possible transformations for the semantics are also covered by the operationalization. (3) Then, we define a *deterministic operational scheme* via suitable criteria for determinism and show that for these criteria the operational rules of (2) can guarantee a deterministic result of the transformation. We thus can exclude non-determinism, which is not ruled out by the original TGG semantics, but is necessary as in practice a model transformation implementation needs to produce a unique result. Furthermore, we can still show conformance to the formal semantics via the operationalization introduced before (see step (2)). (4) There is the limitation that the considered TGG implementation only employs a bookkeeping approach for nodes but not for edges. Again we can define a related *implementation scheme* via adjusting the rules and criteria that if fulfilled guarantee that also this scheme conforms to the formal semantics referring to the operationalization introduced before (see step (3)). The provided bridge of steps (1)–(4) only closes the gap between the formal semantics of TGGs and the implementation at the level of abstraction related to the standard graph transformation system semantics. (5) In a final step, we present an *efficient implementation scheme* covering several additional optimization tricks employed in the implementation that go beyond this abstraction. These are, in particular, the strategy to avoid searching for matches in the whole source graph and the way control flow constructs are used in the implementation to realize the transformation on top of the graph transformation language Story Diagrams [11]. This *efficient implementation scheme* is based on the model transformation algorithm presented in [20]. In the meantime, the algorithm's implementation has been improved, especially regarding model synchronization (although synchronization is not considered in this paper). Note that in [16] we have already sketched steps (1)–(4). In this paper, we further close the gap by also focusing on the optimizations (step (5)) used in our implementation and presenting a run-time conflict check. Moreover, we add explanations on attribute handling and, finally, we present an evaluation of our approach on an industrial case study [17,19].

Summarizing, the contribution of this paper is that it closes the gap between the formal semantics of triple graph grammars and the batch model transformation of our related implementation [14]. Note in particular that when developing the bridge for this gap, we assumed our implementation to be fix in the sense that we did not change or simplify anything in our implementation to simplify this task. This means that our research resolves the following question: given a TGG implementation holding different optimizations[2] necessary to make the tool scalable to case studies of an industrial size, is it possible to formally prove that the operationalization including optimizations implemented in this tool conform

[1] For existing relational model transformation approaches such as QVT [33] conformance and how to statically check related constraints is an open problem [21,40].

[2] Comparing the optimizations in our implementation with other TGG tools is not in the scope of this paper, instead we focus on proving correctness of our optimizations with respect to TGG conformance.

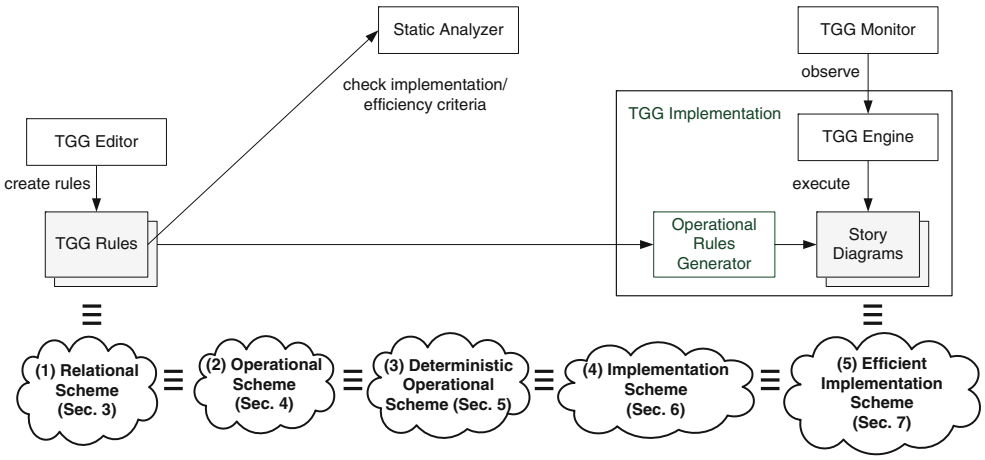



**Fig. 1** Bridging the gap between formal semantics and implementation of TGGs

to the original TGG formal semantics (relational scheme)? In this paper, we show that it is possible to close this gap stepwise by proving that specific assumptions and optimizations in the tool conform to the TGG formal semantics when they adhere to specific criteria. We show how to check these criteria in a static manner. It turned out that the check for one of these criteria does not scale very well. For this case, we have developed a runtime check and integrated it into our tool.

Related work

The relation of the derived forward transformations to the formal semantics of TGGs has been the subject of several research works already: In [38], a mechanism for controlling the application of the rules of the operationalization of TGGs was proposed, for which consistency could be shown, but completeness is not warranted. Based on the same idea, an approach is presented in [9], which checks the consistency of a forward transformation on-the-fly, maintains completeness, but still involves parsing. In [10], from consistent forward transformations so-called terminating NAC-consistent forward ones are derived and checked for determinism. It is not shown, as done in [16] and in this paper, that having a deterministic set of forward rules with integrated mechanisms for controlling the rules means that consistency follows.

In [23], an internal control mechanism[3] for the operational rules of TGGs is presented and its conformance is shown with the external control mechanism previously introduced in [9]. Having this internal control, it is shown that functional behavior of the model transformation can be analyzed statically. In particular, it is argued that non-determinism can be disregarded if it leads to incompletely translated models. No TGG implementation is mentioned validating this approach nor it is mentioned whether it would be possible to build one working without backtracking.[4] The static analysis is only partially implemented, little information is given on its scalability, and it is unclear how to automate the remaining parts of the static analysis. As a follow-up paper for [9], [22] tackles the backtracking issue by introducing so-called filter NACs that to some extent can be derived automatically and that are added to the operational rules without losing conformance. Filter NACs help to reduce, but in general do not completely avoid backtracking. In this paper, we show that if the TGG fulfills some specific criteria, our TGG implementation using an efficient matching strategy guarantees a unique transformation result without having to revert to backtracking. We also show how the criteria guaranteeing uniqueness of the transformation result can be analyzed statically within reasonable constraints. Also in [27], it is shown that a conform as well as an efficient operationalization for a new characterization of well-formed TGGs exists. Here, TGGs do not have to be deterministic as required in our approach. However, requiring determinism has the advantage that if our translation algorithm aborts with an error, then we can be sure that a conforming translation would not exist anyway. Moreover, in contrast to the criteria we propose, the so-called integrity-preserving properties that TGGs need to fulfill in [27] in order to obtain conformance cannot be checked automatically yet. This complex design task is left to the TGG developer.

Using TGGs to specify transformations between metamodels with big structural and/or semantic differences can be quite problematic. This is a general problem of relational model transformation approaches and not the focus of the

[3] It adds special attributes to model elements, storing if the corresponding elements have already been translated. We use so-called bookkeeping edges pointing to model elements that still need to be translated (see also [16]), which does not require changing meta models.

[4] At first sight, this seems difficult since the approach allows for branches leading to incompletely translated models besides branches leading to the unique transformation result.

paper. Enhancing the expressiveness of TGGs is subject to current research. For example, application conditions can be used to enhance expressiveness as described in [22,27]. Note that our implementation does not support general (negative) application conditions. Including them into the formal framework presented in this paper is part of future work. Also [21] is concerned with expressiveness of TGGs and compares them to the QVT Relational standard. Note that adding new features to TGGs enhancing their expressiveness makes the task of proving conformance more complex.

Finally, in contrast to [9,38], where triple graphs are defined as spans of injective morphisms, we developed a formalization of triple graphs [15], based on plain graphs typed over a suitable type triple graph, which is nearer to our implementation. Moreover, we do not assume that the TGG axiom is empty, since this does not seem very realistic in praxis. Our implementation is based on the Eclipse Modeling Framework (EMF).[5] However, in our formalization we abstract from specific EMF features such as navigability, compositions, or metamodel constraints. Supporting these features in the formal semantics of triple graph grammars is a different task and, therefore, we abstract from this problem in this paper. However, there is already related work [4] which is concerned with the problem of describing EMF transformations by a specific class of graphs and graph transformations in a faithful way.

Another approach to relational, bidirectional transformations with a formal basis is [34] where a terminating, correct, and complete operationalization for so-called patterns, some kind of graph constraints for triple graphs, is derived. Large sets of valid transformation results (i.e. non-determinism) may occur, leading to efficiency problems concerning implementation.

In [28], an approach is presented to systematically validate model transformations with respect to termination and confluence, and hence uniqueness of the transformation result. Graph transformation theory is used to formally specify and statically analyze model transformations. However, this approach uses an operational specification and does not examine the conformance of the relational specification with its operational counterpart.

Summarizing, as far as we know, there is no other approach guaranteeing consistency, completeness, and determinism all at once for a specific subset of TGGs or related relational approaches on a formal as well as efficient and thus practical implementation level.

Outline of the paper

We begin with an informal introduction on TGGs in Sect. 2, introduce our running example and our tool support for TGGs. In Sect. 3, we describe TGGs in a more detailed and formal way as a relational specification scheme for model transformations. Then, we outline a naive operational computation scheme in Sect. 4, and introduce a bookkeeping mechanism and prove conformance to the formal semantics in Sect. 5. This scheme is further refined towards a deterministic computation scheme in Sect. 6, where proper restrictions for the rules are introduced that ensure determinism. Then, we introduce minor derivations in the computation scheme in Sect. 7 that hold for the implementation, show how the corresponding restrictions for the rules can be checked statically, and prove that conformance and determinism also hold for the implementation. In Sect. 8, we introduce a so-called efficient implementation scheme containing several optimizations with regard to match search strategies and control flow constructs with regard to rule application. We show conformance of the efficient implementation scheme with the implementation scheme without optimizations. In Sect. 9, we present the provided tool support including a new runtime check that we have implemented to verify the uniqueness of the transformation result and evaluate our approach on our running example and on an industrial case study. Finally, we close the paper with our conclusions and an outlook on planned future work.

## 2 Introduction to triple graph grammars and our TGG implementation

As a running example we will use a model transformation[6] from SDL block diagrams [24] to UML class diagrams. The metamodels of these languages are shown in Fig. 2. Block diagrams are hierarchical structures, where a *BlockDiagram* contains *SystemBlocks* which in turn contain *Blocks*. In the class diagram, a *ClassDiagram* contains all other elements. These are *Classes* that can be connected to each other via *Associations*. Furthermore, *Stereotypes* can be attached to classes. All these elements, except *Stereotypes*, can have a name. There is also a correspondence metamodel. Its elements connect elements of the other two metamodels. This way, the correspondence model stores traceability information, which allows to find elements of one model that correspond to an element of the other model.

*Triple graph grammars* (TGG) relate three different models: A source model, a target model, and a correspondence model. A TGG consists of an axiom (the grammar's start graph) and several TGG rules. The TGG for the transformation of block and class diagrams is shown in Fig. 3.[7]

[5] http://www.eclipse.org/emf.

[6] This model transformation is a simplified version of a transformation used in the industrial case study on flexible production control systems [36].

[7] Note, that the types defined in Fig. 2 are abbreviated in Fig. 3.

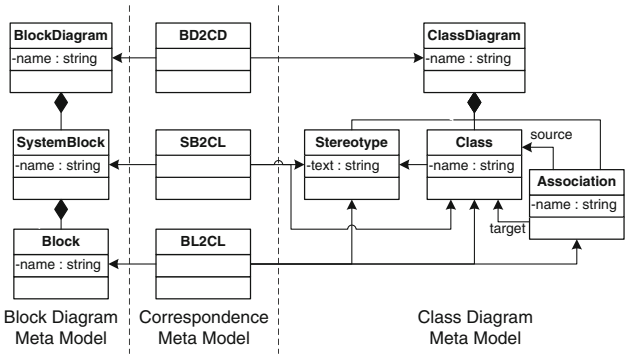



**Fig. 2** Example metamodels

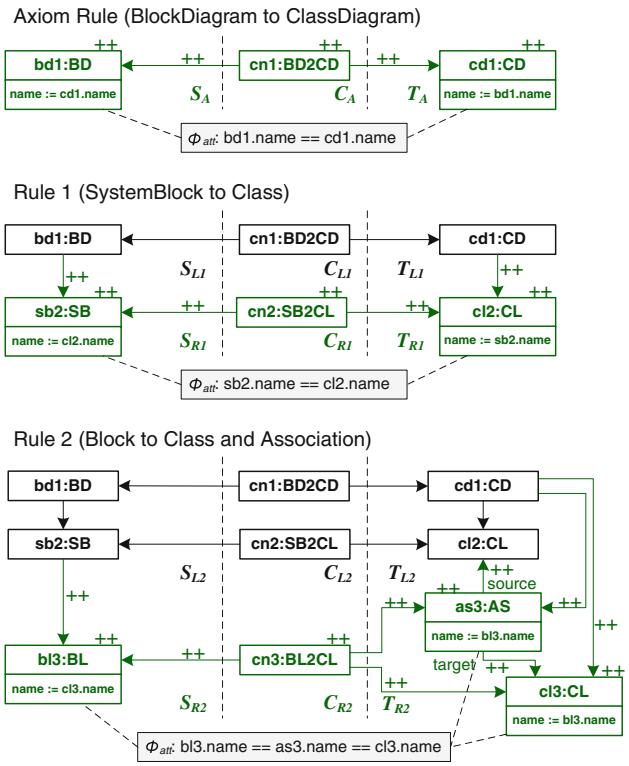



**Fig. 3** Example TGG rules and axiom rule

Here, we use a short notation that combines the left-hand (LHS) and right-hand sides (RHS) of the graph transformation rule. Elements that belong to the LHS and RHS are drawn in black; elements that belong only to the RHS (i.e. which are created by the rule) are drawn green and marked with "**++**". TGG rules are divided into three domains: The source model domain (left), target model domain (right), and the correspondence model domain (middle). The axiom in Fig. 3 relates the root elements of the source and target models with the axiom correspondence node.[8] The attribute assignments, defined through OCL expressions in our tool environment, state that the names of the block and class diagrams must be equal. Rule 1 creates a *SystemBlock* and a corresponding *Class*. The *BlockDiagram* and *ClassDiagram* must already exist. Rule 2 creates a *Block* in the block diagram domain and connects it to the *SystemBlock*. In the class diagram domain, a class is created and connected to the *SystemBlock*'s *Class* with an *Association*.

TGGs cover three kinds of model transformation directions: Forward, backward, and correspondence transformations. A forward (backward) transformation takes a source (target) model as input and creates the correspondence and target (source) model. A correspondence transformation[9] requires a source and target model and creates only the correspondence model. This paper's main concern is how to come from the relational TGG description to a conforming and efficient transformation covering one of these directions.

Thereby, we concentrate on showing conformance for batch transformations, not for synchronizations [14,20].

We have developed an implementation of TGGs based on Eclipse and the Eclipse Modeling Framework. The system can perform model transformation and model synchronization.[10] It utilizes several optimizations to increase the performance of model transformations. For example, these optimizations limit the pattern matching process to a subset of the source model (cf. Sect. 8). As shown in Fig. 1, our TGG implementation includes an editor to create TGG rules. Connected to the editor is a static analyzer that checks the various criteria that will be presented in this paper. The TGG rules are relational in nature. Therefore, operational rules have to be derived in the form of Story Diagrams. These Story Diagrams combine graph transformation rules, which realize operational transformation rules with bookkeeping (cf. Sect. 6) with control flow constructs for controlling rule applications. A TGG engine executes the Story Diagrams to perform model transformations. With the TGG Monitor (see. Sect. 8), the execution of a model transformation can be observed to aid the user in debugging TGG rules. In this paper, we bridge the gap between the formal relational semantics of TGGs and our TGG implementation.

## 3 Relational scheme

The relational scheme starts with an initial triple graph, called the axiom. After that, TGG rules are applied to extend it. The example in Fig. 4 was created in this manner. The block and class diagrams are the source and target graphs connected by a correspondence graph, constituting a triple graph. Further on, we use a triple of variables $SCT$ to denote one triple graph,

[8] EMF requires all models to have a root node that contains all other model elements.

[9] The correspondence transformation is also known as mapping transformation or model integration.

[10] It can be downloaded from our Eclipse update site http://www.mdelab.de/update-site.

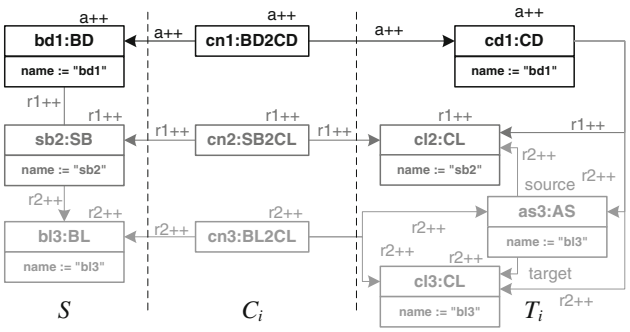
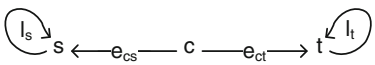



**Fig. 4** Example models connected by a correspondence model

where $S$ denotes its source component, $C$ its correspondence component, and $T$ its target component.

Formally,[11] models can be interpreted as typed graphs according to some given type graph (the model's metamodel). In case of triple graphs, the type graph in turn adheres to a particular structure reflecting its three components (the "metametamodel" of triple graphs). A *triple graph* $SCT$ is a graph typed over

TRIPLE: 


Therefore, a triple graph $SCT$ consists of a *source component* $S$, containing all elements of type $s$ and $l_s$; a *correspondence component* $C$, containing all elements of type $s$, $t$, $e_{cs}$,$e_{ct}$, and $c$; and a *target component* $T$, containing all elements of type $t$ and $l_t$. A *triple type graph* $S_T C_T T_T$ is a special triple graph defining node and edge types for the source component, correspondence component, and target component of triple graphs. In particular, this means that the correspondence component and source component (target component) of a triple graph overlap in the source nodes (target nodes) connected to correspondences. Thus, using this notation, a triple graph $SCT$ coincides with the union of its source component $S$, correspondence component $C$ and target component $T$. The metamodels of Fig. 2 can be interpreted as a *triple type graph*. A *typed triple graph* is a triple graph typed over $S_T C_T T_T$, e.g., concrete block and class diagrams connected by a correspondence model like shown in Fig. 4. We say that a finite graph $S$ ($T$ or $C$) typed over $S_T$ ($T_T$ or $C_T$) is a source graph (target graph or correspondence graph, respectively) and belongs to the language $\mathscr{L}(S_T)$ ($\mathscr{L}(T_T)$ or $\mathscr{L}(C_T)$, respectively). We extend these kinds of triple graphs with attributes formalized using the symbolic approach [35]. Basically, in this approach first-order formulas over attribute labels and values of the attribute domain are used to constrain the possible attribute values.

Formally, a *triple graph grammar* TGG $= (S_A C_A T_A, \mathscr{R})$ consists of an axiom $S_A C_A T_A$ and a set of non-deleting rules $\mathscr{R}$ for triple graphs. Each rule consists of an inclusion from the LHS $S_L C_L T_L$ to the RHS $S_R C_R T_R$ of the rule. Each element in $S_L C_L T_L$ corresponds to an element in $S_R C_R T_R$ and is preserved by the rule. Moreover, each rule is equipped with a first-order formula $\Phi_{\text{att}}$ (attribute formula) over attribute labels occurring in the rule and values of the attribute domain, expressing valid attribute value assignments for attributes in the LHS and RHS of the rule.

All elements in $S_R C_R T_R \setminus S_L C_L T_L$ are created when the rule is applied. A rule $r$ can be applied to a triple graph $S_G C_G T_G$ if an injective morphism $m$ of its LHS $S_L C_L T_L$ into $S_G C_G T_G$ can be found such that corresponding attribute values are valid assignments according to $\Phi_{\text{att}}$. We say that $m$ is a *match* from $S_L C_L T_L$ into $S_G C_G T_G$. The result $S_H C_H T_H$ of the rule application $S_G C_G T_G \overset{m,r}{\rightarrow} S_H C_H T_H$ via rule $r$ and match $m$ consists of the gluing of $S_G C_G T_G$ with elements in $S_R C_R T_R \setminus S_L C_L T_L$ via match $m$. This means that all elements created by $r$ are added to the graph $S_G C_G T_G$. Moreover, attribute values of created elements are set according to $\Phi_{\text{att}}$.

Figure 4 shows in a compact way how rules 1 and 2 of our TGG in Fig. 3 can be applied to the axiom obtaining $SC_i T_i$. The elements created by rules 1 and 2 are marked with "r1++" and "r2++", respectively. Rule 1 requires the presence of a *BlockDiagram* connected with a *ClassDiagram* and thus can be matched to the already present axiom such that a *SystemBlock* connected with a *Class* can be added as children to the already present *BlockDiagram* and *ClassDiagram*. The attribute formula `sb2.name==cl2.name` derived from rule 1 specifies that *SystemBlock* and *Class* must have equal names.[12] In this case, the name of the added *SystemBlock sb2* and *Class cl2* are both set to "sb2". After rule 1 has been applied, analogously rule 2 can be applied obtaining $SC_i T_i$, consisting of a block and class diagram related according to our example TGG. The *BlockDiagram* contains a *SystemBlock* containing a *Block* and the corresponding *ClassDiagram* contains two *Class*es connected by an *Association*

[11] As mentioned in Sect. 1, we formalize triple graphs based on a new category [15], allowing triple graphs with correspondence nodes that do not necessarily need to be connected to exactly one source or target node, respectively.

[12] In our tool as shown in Fig. 3, the user inputs the attribute assignments for the corresponding forward (`cl2.name:=sb2.name`) and backward transformation (`sb2.name:=cl2.name`) directly using OCL expressions such that the (more conceptual) relational attribute formula `sb2.name==cl2.name` is derived here. Note that the user himself needs to take care of the fact that the attribute assignments for the forward and backward direction are consistent to each other. More detailed explanations to the specifics of attribute handling for TGGs can be found in [31].

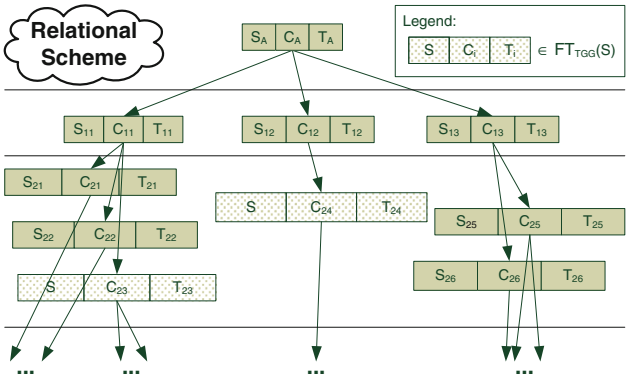



**Fig. 5** Relational scheme: $FT_{\mathrm{TGG}}$

from the *Class* corresponding to the *SystemBlock* to the *Class* corresponding to the *Block*.

We say that $\mathscr{L}(TGG)$ is the set of all triple graphs that can be derived from $S_A C_A T_A$ using rule applications via rules in $\mathscr{R}$. Thereby, $\rightarrow^*_{\mathrm{TGG}}$ denotes the reflexive and transitive closure of a rule application via some rule in $TGG$. Note, that the standard results on graph transformation such as the Concurrency Theorem, Local-Church Rosser Theorem, and Critical Pair Lemma [7,30] hold for typed triple graph transformations.[13]

As mentioned before, we can perform forward, backward, and correspondence transformations based on a TGG. Subsequently, we focus on forward transformations. Analogous results can be derived for the correspondence case. For the backward case we obtain similar results straightforwardly because of symmetry reasons with the forward case.

We introduce the **relational scheme** $FT_{\mathrm{TGG}}$ as follows:

$FT_{\mathrm{TGG}}$ of a source graph $S$ returns the set $FT_{\mathrm{TGG}}(S)$ of triple graphs $SC_iT_i$ that can be generated by the $TGG$ starting from the axiom $S_A C_A T_A$ and that have $S$ as a source component. This is illustrated in Fig. 5, where each arrow depicts a rule application via some $TGG$ rule. In other words, we apply the triple graph grammar like an ordinary grammar to generate the language of triple graphs defined by the TGG and select those triple graphs that have $S$ as a source component.

**Definition 1 (relational scheme: $FT_{\mathrm{TGG}}$)**
$FT_{\mathrm{TGG}} : \mathscr{L}(S_T) \rightarrow \mathscr{P}(\mathscr{L}(TGG))$ is defined as follows:
$FT_{\mathrm{TGG}}(S) := \{SC_iT_i \mid S_A C_A T_A \rightarrow^*_{\mathrm{TGG}} SC_iT_i\}$.

Figure 4 shows a block diagram $S$ with a corresponding class diagram $T_i$. Both models are connected by a correspondence model $C_i$. Assuming that we want to transform $S$, this is a valid forward transformation result in $FT_{\mathrm{TGG}}(S)$ because there is a corresponding transformation producing a triple graph $SC_iT_i$ according to the $TGG$ in Fig. 3. The attribute formulae ensure equality of the name attributes of corresponding elements. However, while all three models are created in parallel, the actual name values can be chosen arbitrarily. The annotations in Fig. 4 indicate which rules create the corresponding elements. Note that the generation of $SC_iT_i$ corresponds conceptually to one path in the derivation tree of Fig. 5. Each path in this tree ending up with a triple graph having $S$ as a source component belongs to $FT_{\mathrm{TGG}}(S)$.

## 4 Naive operationalization

The relational scheme creates all three models in parallel. However, in practice only the source or target model exists and the correspondence and target or source model should be created by the forward or backward model transformation, respectively. Alternatively, the source and target model might exist and the correspondence model should be created by a so-called correspondence transformation.

To this extent, *operational transformation rules* from the relational TGG rules can be derived as introduced in [37]: For the forward transformation, all elements belonging to the source domain that were previously created are added to the LHS of the rule. A forward rule then specifies how source elements can be translated into target elements according to the TGG (see Fig. 6 depicting the forward rule $r_1^{\mathrm{F}}$ derived from rule $r_1$ of our running example TGG). More formally, given a triple graph rule $r : S_L C_L T_L \rightarrow S_R C_R T_R$ a *forward rule* $r^{\mathrm{F}} : S_R C_L T_L \rightarrow S_R C_R T_R$ is derived. In particular, the elements belonging to $S_R \setminus S_L$ are added to $S_L C_L T_L$. Since these are all source elements, the source component then consists of $S_R \setminus S_L \cup S_L = S_R$ and we obtain a well-defined triple graph[14] $S_R C_L T_L$ since $S_R \cap C_L = S_L \cap C_L \subseteq C_L$. The attribute formula of $r^{\mathrm{F}}$ is identical to the attribute formula of $r$, since the same constraints should still hold for the corresponding attribute values. The only difference is that when applying $r^{\mathrm{F}}$ attribute values of elements in $S_R \setminus S_L$ are now also determined by matching them to elements with concrete attribute values in the instance graph. Analogously, a so-called backward rule and

[13] As explained in more detail in [15], we can define the category $\mathbf{TripleGraphs}_{S_T C_T T_T}$, having typed triple graphs as objects and typed triple graph morphisms as arrows, such that $\mathbf{TripleGraphs}_{S_T C_T T_T}$ with the set $\mathscr{M}$ of injective morphisms (isomorphic on the data type part) is an adhesive HLR category [7,30]. We extend these triple graphs with attributes formalized using the symbolic approach [35]. As explained in [35] this kind of attribute formalization fits into the adhesive HLR framework as well.

[14] The union of $S_R$,$C_L$, and $T_L$ defines in this special case the intended triple graph. Actually, the source nodes in $S_R \setminus S_L$ (typed over $s$ in $TRIPLE$) should be added to the correspondence component $C_L$. We use this slight abuse of notation in case that the correspondence component of a triple graph is not connected via correspondence edges to each source or target node in the source or target component, respectively. It is part of future work to introduce our triple notation in such a way that it fits better also for these special cases.

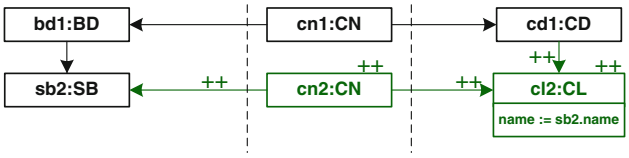
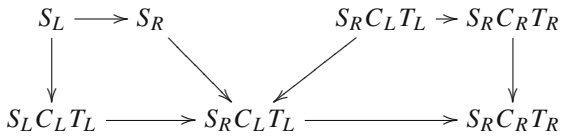



**Fig. 6** Operational forward rule $r_1^{\mathrm{F}}$ derived from rule $r_1$

correspondence rule can be derived. A *backward rule* specifies how target elements can be translated into source elements according to the TGG. More formally, given a triple graph rule $r : S_L C_L T_L \rightarrow S_R C_R T_R$, a backward rule $r^{\mathrm{B}} : S_L C_L T_R \rightarrow S_R C_R T_R$ is derived. A *correspondence rule* specifies how source and target elements can be connected to each other according to the TGG. More formally, given a triple graph rule $r : S_L C_L T_L \rightarrow S_R C_R T_R$, a correspondence rule $r^{\mathrm{C}} : S_R C_L T_R \rightarrow S_R C_R T_R$ is derived. From this point, we concentrate on forward transformations, since the considerations for the other transformation directions are analogous.

A *decomposition and composition* result as given in [9,37] shows that a triple graph transformation creating a triple graph $SCT$ can be decomposed into a so-called source and forward transformation, and vice versa.[15] First, the source transformation creates the source graph $S$ to be translated by applying source rules and the corresponding forward transformation performs the actual translation into $SCT$ via forward rules. Source rules are rules that correspond to the source projection of a TGG rule. More formally, given a triple graph rule $r : S_L C_L T_L \rightarrow S_R C_R T_R$, then a *source rule* $r^{\mathrm{S}} : S_L \rightarrow S_R$ can be derived. The decomposition and composition result relies on the fact that a triple graph rule $r : S_L C_L T_L \rightarrow S_R C_R T_R$ can be understood as a sequential application (denoted by $*$) of the corresponding source and forward rule $r = r^{\mathrm{S}} *_{L^F} r^{\mathrm{F}}$ as shown in the following diagram, where all arrows denote inclusions:




Consequently, a triple graph transformation creating $SCT$ can be decomposed into a sequence of steps, where each step consists of a source transformation creating a part of the source graph $S$ to be translated and the corresponding forward transformation translating these created elements. This sequence can then be rearranged such that first all source graph elements are created by the source transformations and afterwards the created source graph is translated by the corresponding forward transformations. The decomposition and composition result represents a *first step to the operationalization of TGGs*, since it demonstrates that it is possible to find for each triple graph $SCT$ generated by the TGG a forward transformation translating $S$ into $SCT$. The question remains of how to apply the forward rules without having to investigate how the source graph might have been created by the corresponding source rules. The next sections describe how to overcome this problem.

## 5 Operational scheme

In order to be able to guarantee consistency with the TGG, the transformation implementation has to keep track of those model elements that were already transformed in a previous transformation step and which elements have not been transformed yet. For this purpose, we introduce a special *bookkeeping node b*, which keeps a *bookkeeping edge* to each source model element that has not been transformed yet (cf. Fig. 7).[16] When some source model element is transformed, the corresponding bookkeeping edge is deleted. This implies that the bookkeeping edges have to be added to $S$, which is the source graph to be translated, once before executing the model transformation, obtaining the *initial source graph* $B_{\mathrm{init}} S$. In contrast to [23], where a bookkeeping mechanism is proposed using a special attribute added to the elements to be translated, our mechanism has the advantage that the bookkeeping structures are stored as extensions referring to the original source and target language elements, i.e. the metamodels of the source or target language do not have to be changed.[17] Note that we do not introduce a bookkeeping mechanism for attributes, since we assume that if a node is created in a TGG it should be created together with all its attributes and it does not make sense from a practical point of view to do this in different TGG rules.

Furthermore, we extend each *forward rule* $r^{\mathrm{F}}$ with a *bookkeeping mechanism* to $r^{\mathrm{BF}}$. Figure 8 shows the operational bookkeeping forward rule derived from rule 1 in Fig. 3. When it is applied, the bookkeeping edges pointing to model elements of $S_R \setminus S_L$ are deleted. This is indicated by the "**- -**" annotation and the red color of these edges. The model elements of $S_L$ must have been transformed already. This is ensured by negative application conditions, which prohibit bookkeeping edges to these elements. The attributes of created target model elements are set according to $\Phi_{\mathrm{att}}$. The advantage of integrating the bookkeeping mechanism into the forward rules is that, on the one hand, rule application itself makes sure that consistency with the TGG is preserved

[15] We informally reintroduce this result here, since it is analogous to [9, 37].

[16] Note, that we use a graph model, where edges from nodes to edges are allowed, as explained in more detail in [15].

[17] [5] presents more detailed explanations on this kind of meta-model integration pattern for the FUJABA Tool suite.

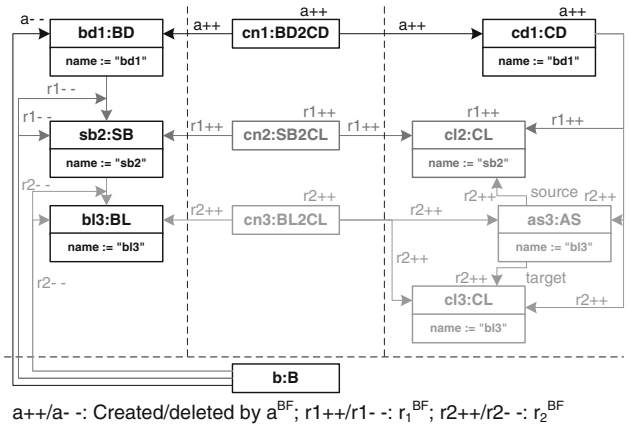



**Fig. 7** Forward transformation with bookkeeping

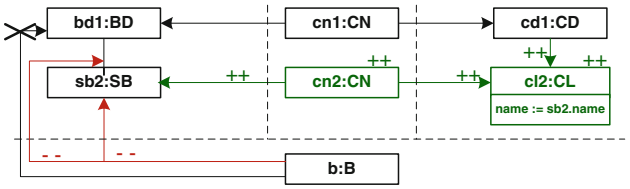



**Fig. 8** Operational forward rule $r_1^{\mathrm{BF}}$ with bookkeeping derived from rule $r_1$

since translated elements are marked as such and, on the other hand, we can use this information (see Sect. 6) to predict at design time potential conflicts between forward transformation steps competing to translate the same elements.

Formally, $r^{\mathrm{F}}$ is extended[18] to $r^{\mathrm{BF}} : B_{S_R \setminus S_L} S_R C_L T_L \leftarrow bS_R C_L T_L \rightarrow bS_R C_R T_R$, a span of inclusions with $NAC^{\mathrm{BF}}$ defined as follows: A special bookkeeping node $b$ is added, which is preserved. For each node and edge $x \in S_R \setminus S_L$, we add a bookkeeping edge from $b$ to $x$, which is deleted by the rule. The set $B_{S_R \setminus S_L}$ consists of these bookkeeping edges and the bookkeeping node $b$. $NAC^{\mathrm{BF}}$ is a set of negative application conditions, which forbid for each element $x \in S_L$ an incoming bookkeeping edge from $b$. In the example in Fig. 8, $B_{S_R \setminus S_L}$ contains $b$, the bookkeeping edges to *sb2*, and to the link between *bd1* and *sb2*. $NAC^{\mathrm{BF}}$ forbids the edge from $b$ to *bd1*.

We can apply a bookkeeping forward rule $r^{\mathrm{BF}}$ to a triple graph with bookkeeping $B_G S_G C_G T_G$ if there exists a match $m : B_{S_R \setminus S_L} S_R C_L T_L \rightarrow B_G S_G C_G T_G$, fulfilling $NAC^{\mathrm{BF}}$, meaning that each node and edge in $m(S_L)$ has *no* incoming edge from $b$ (translated already), and each node and edge in $m(S_R \setminus S_L)$ has an incoming edge from $b$ (to be translated). The application of rule $r^{\mathrm{BF}}$ to $B_G S_G C_G T_G$ via $m$ deletes all bookkeeping edges in $m(B_{S_R \setminus S_L})$ and adds the translation $S_R C_R T_R \setminus S_R C_L T_L$.

[18] We therefore extend also the graph $TRIPLE$ and $S_T C_T T_T$ with corresponding bookkeeping node and edge types as presented in [15].

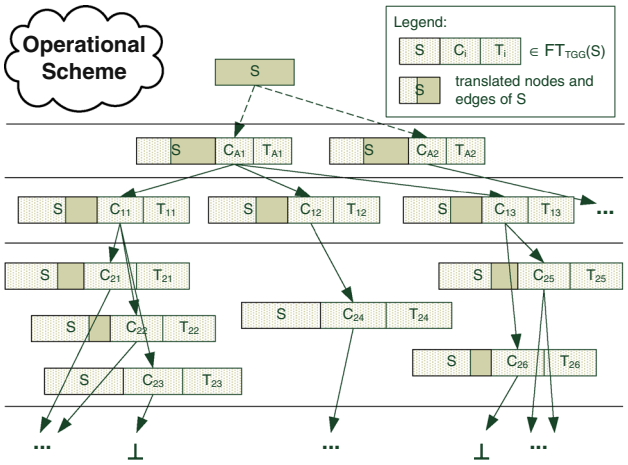



**Fig. 9** Operational scheme: $FT_{\mathrm{CON}}$

We can interpret the axiom $S_A C_A T_A$ of a $TGG$ as a triple graph rule, called *axiom rule*, $a : \emptyset \rightarrow S_A C_A T_A$. The bookkeeping forward rule $a^{\mathrm{BF}}$ of $a$ is built analogously to regular TGG rules. However, the bookkeeping is not sufficient to exclude that it is applied multiple times. Thus, this rule must be controlled separately at the beginning of each forward transformation. We define $CON = OP_{FT}(TGG)$ as the rule set consisting of a bookkeeping forward rule $r^{\mathrm{BF}}$ for each rule $r$ of the $TGG$ ($CON$ because we will show that it conforms to the TGG). Moreover, we have a mapping $Trans^F$ computing for each triple graph with bookkeeping the part of $S$ that has already been translated by some bookkeeping forward rule. In particular, it consists of all nodes and edges with no incoming bookkeeping edges. Finally, we define $B_{\mathrm{init}} S$ as the initial source graph consisting of $S$ and $b$ equipped with one bookkeeping edge for each graph element in $S$.

We introduce the **operational scheme** $FT_{\mathrm{CON}}$ as follows: Given a TGG and its operationalization $CON$, then $FT_{\mathrm{CON}}$ of a source graph $S$ returns all triple graphs $SC_i T_i$ that can be reached by first applying the forward axiom rule $a^{\mathrm{BF}}$ to $B_{\mathrm{init}} S$ once, and afterwards applying bookkeeping forward rules in $CON$ up until all nodes and edges in $S$ have been translated such that $S = Trans^F(B_i SC_i T_i)$. This is illustrated by Fig. 9, where the application of $a^{\mathrm{BF}}$ is depicted by a dashed arrow and all other arrows depict a rule application via a bookkeeping forward rule in $CON$. Derivation paths ending up with $\perp$ denote terminating paths, where no rule is applicable anymore.

**Definition 2** (*operational scheme:* $FT_{\mathrm{CON}}$)
Given a TGG and its operationalization $CON$, then $FT_{\mathrm{CON}} : \mathscr{L}(S_T) \rightarrow \mathscr{P}(\mathscr{L}(TGG))$ of a source graph $S$ is defined as follows: $FT_{\mathrm{CON}}(S) := \{SC_i T_i \mid B_{\mathrm{init}} S \rightarrow_{a^{\mathrm{BF}}} B_A SC_A T_A \rightarrow^*_{\mathrm{CON}} B_i SC_i T_i \wedge S = Trans^F(B_i SC_i T_i)\}$.

This implies that all elements in $S$ have been transformed exactly once. We call this a *valid transformation*.

Figure 7 shows the valid forward transformation of our example block diagram. Initially, there is a link from the bookkeeping node *b* to each source model node and edge. The bookkeeping forward axiom rule transforms *bd1* and produces *cn1* and *cd1*. The bookkeeping edge to *bd1* is deleted. Next, bookkeeping forward rule $r_1^{\mathrm{BF}}$ is applied to create *cn2* and *cl2*. It also deletes the bookkeeping edges to *sb2* and to the link between *bd1* and *sb2*. Finally, bookkeeping forward rule $r_2^{\mathrm{BF}}$ is applied, analogously. The name attributes of target model elements are set according to the attribute formulae. For example, when applying $r_1^{\mathrm{BF}}$ with formula `sb2.name == cl2.name` the attribute value of `sb2.name` is determined by matching `sb2` such that the value of the attribute `cl2.name` is determined according to `cl2.name:= sb2.name`. Thus, when creating *cl2*, the name of *cl2* is set to the name of *sb2*. In Fig. 9, this bookkeeping forward transformation corresponds conceptually to one path in the derivation tree ending up with a triple graph, where $S$ is completely translated. Because of conformance, this resulting triple graph then belongs to $FT_{\mathrm{TGG}}(S)$.

In the following, we show that $FT_{\mathrm{TGG}}$ conforms with $FT_{\mathrm{CON}}$, meaning that for each forward transformation via bookkeeping rules a corresponding forward transformation according to the $TGG$ exists (*consistency*) and the other way round (*completeness*). Intuitively, this means that each triple graph result of the operational scheme $SC_iT_i$ (cf. Fig. 9), where all elements of $S$ have been translated, can be found as an element of the triple graph language (cf. Fig. 5), and vice versa.

In [8,37], it is argued already in detail that a TGG rule application can be decomposed into a sequence of transformations via the corresponding source rule,[19] where the correspondence and target component of the rule are empty, followed by a transformation via the corresponding forward rule and the other way round (composition). Since our forward and source rules are constructed analogously to [8,37], in the following proof ideas, we assume these results and concentrate on arguing that the bookkeeping mechanism as added in this paper to the forward rules leads to consistency and completeness as described above. Complete proofs can be found in [15].

As an auxiliary result, we show that each application of bookkeeping forward rules is backed up by a corresponding TGG rule application (partial consistency). Then, it follows directly that also in the special case that a complete source graph is translated, the correspondence must hold and thus we have *consistency*. Conversely, we show that a forward transformation via TGG rules guarantees that a related transformation via bookkeeping rules can be found, leading to *completeness*.

**Lemma 1** (partial consistency)
*For a $TGG$ and its operationalization $CON = OP_{FT}(TGG)$ holds that $B_{\mathrm{init}}S \rightarrow_{a^{\mathrm{BF}}} B_ASC_AT_A \rightarrow^*_{\mathrm{CON}} B_iSC_iT_i \wedge Trans^F(B_iSC_iT_i) = S_i$ implies $S_AC_AT_A \rightarrow^*_{\mathrm{TGG}} S_iC_iT_i$ via the related TGG rules.*

*Proof* (Proof idea) The bookkeeping forward axiom rule $a^{\mathrm{BF}}$ is applied to $B_{\mathrm{init}}S$ via some match $m_A$ conforming to $S_AC_AT_A$ only once. The set of translated elements after this first step $Trans^F(B_ASC_AT_A) = m_A(S_A)$. Furthermore, bookkeeping of the rules in $CON$ implies that during the transformation of $B_ASC_AT_A$ each node and edge of $S \setminus m_A(S_A)$ is translated at most once conform to the corresponding TGG rules. Each rule application of $CON$ via some rule $r^{\mathrm{BF}}$ enlarges the set of translated elements in $S$ with the matched elements of $S_R \setminus S_L$. Accordingly, when a series of rule applications via $CON$ starting with $B_ASC_AT_A$ delivers $B_iSC_iT_i$ such that $Trans^F(B_iSC_iT_i) = S_i$, then applying the related $TGG$ rules generates $S_iC_iT_i$ from $S_AC_AT_A$. □

**Lemma 2** (completeness)
*For a $TGG$ and its operationalization $CON = OP_{FT}(TGG)$, it holds that $FT_{\mathrm{TGG}}(S) \subseteq FT_{\mathrm{CON}}(S)$. In particular, $B_{\mathrm{init}}S \rightarrow_{a^{\mathrm{BF}}} B_ASC_AT_A \rightarrow^*_{\mathrm{CON}} B_iSC_iT_i$ and $Trans^F(B_iSC_iT_i) = S$ if $S_AC_AT_A \rightarrow^*_{\mathrm{TGG}} SC_iT_i$ via the related TGG rules.*

*Proof* (Proof idea) The forward axiom rule $a^{\mathrm{BF}}$ can be applied to $B_{\mathrm{init}}S$ such that $B_{\mathrm{init}}S \rightarrow_{a^{\mathrm{BF}}} B_ASC_AT_A$ because the source axiom $S_A$ is contained in $S$. Moreover, for each $TGG$ rule application via $r$ generating the graph elements $S_R \setminus S_L$ in $S$, the related bookkeeping forward rule $r^{\mathrm{BF}}$ of $CON$ can be applied, translating exactly those elements in $S$ conforming to $r$. Since each element in $S$, except the axiom elements, is generated by such a TGG rule application, we have that $B_{\mathrm{init}}S \rightarrow_{a^{\mathrm{BF}}} B_ASC_AT_A \rightarrow^*_{\mathrm{CON}} B_iSC_iT_i$ such that $Trans^F(B_iSC_iT_i) = S$. □

**Theorem 1** (conformance)
*For a $TGG$ and its operationalization $CON = OP_{FT}(TGG)$, it holds that $FT_{\mathrm{TGG}}(S) = FT_{\mathrm{CON}}(S)$. In particular, $B_{\mathrm{init}}S \rightarrow_{a^{\mathrm{BF}}} B_ASC_AT_A \rightarrow^*_{\mathrm{CON}} B_iSC_iT_i$ and $Trans^F(B_iSC_iT_i) = S$ if and only if $S_AC_AT_A \rightarrow^*_{\mathrm{TGG}} SC_iT_i$ via the related TGG rules.*

*Proof* $FT_{\mathrm{TGG}}(S) \supseteq FT_{\mathrm{CON}}(S)$ (consistency) follows from Lemma 1 for the special case that $Trans^F(B_iSC_iT_i) = S$.

$FT_{\mathrm{TGG}}(S) \subseteq FT_{\mathrm{CON}}(S)$ (completeness) holds because of Lemma 2. □

[19] Given a TGG rule $r : S_LC_LT_L \rightarrow S_RC_RT_R$, then we have the following corresponding source rule $r^S : S_L \rightarrow S_R$.

## 6 Deterministic operational scheme

As can be seen in Fig. 9, it is not guaranteed that whenever a valid transformation result exists, it can be found *without backtracking*. The determinism criteria studied in this section restrict the TGGs to those ones where backtracking can be safely avoided. These criteria ensure, on one hand, that whenever a valid transformation result exists, it can be found without backtracking. On the other hand, if no valid transformation result exists, then we can find this out without backtracking, as well.

In order to avoid backtracking, we need to show that applying bookkeeping forward rules as long as possible always terminates with a *unique result*. To this extent, we use the theory of critical pairs guaranteeing that under specific conditions a set of bookkeeping forward rules is locally confluent [7,30]. A critical pair describes a conflict in a minimal context. Conflicts arise for bookkeeping forward rules if one rule deletes a bookkeeping edge marked for deletion also by the other rule.[20] This is because after applying the first rule and deleting the bookkeeping edge which is marked for deletion also by the other rule, this rule cannot be applied anymore. We ignore critical pairs with the same rules and same matches, since they represent a confluent situation in a trivial way. We forbid the existence of critical pairs (conflict-freeness criterion) since the corresponding conflicts may lead to a result that is not unique. Note that this criterion could be relaxed by allowing for critical pairs that are strictly NAC-confluent [29,30]. Since we want to provide feasible practical tool support for our approach, we have opted however for the more severe conflict-freeness criterion.[21]

Moreover, we introduce a termination criterion ensuring that each application of a bookkeeping forward rule indeed diminishes the number of translated elements. In order to ensure that new attribute values are set in a unique way such that single rule applications have a unique transformation result, we introduce an attribute criterion. Finally, we make sure by the domain restriction criterion that the forward rule of the axiom can only be applied in a unique way.

The **forward determinism criteria** are as follows: (1) Each $TGG$ rule creates at least one graph element on the source part (*termination criterion*). (2) For the rules in $CON = OP_{FT}(TGG)$ there exist no critical pairs, ignoring pairs with same rules and same matches (*conflict-freeness criterion*). (3) For the rules $CON = OP_{FT}(TGG)$, we require for the attribute formula $\Phi_{\text{att}}$ that attribute values of preserved elements remain unchanged and that attribute value assignments of created elements are uniquely determined as soon as the value assignments of attributes of preserved elements are set (*attribute criterion*).

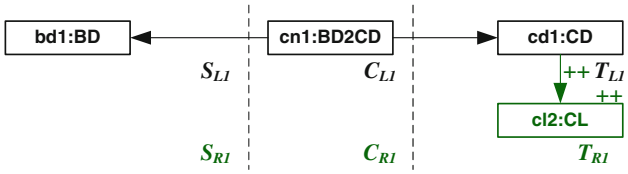



**Fig. 10** Violation of forward termination criterion

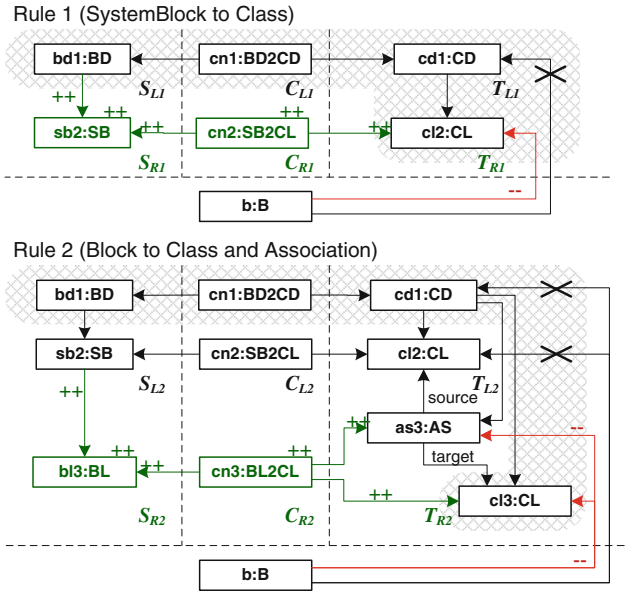



**Fig. 11** Operational backward rules $r1^{BB}$ and $r2^{BB}$ in conflict

Moreover, we have the following **domain restriction criterion**: Only source graphs, containing the source component of the TGG axiom only once, are translated. We say that these source graphs belong to $\mathscr{L}(S_T^A) \subseteq \mathscr{L}(S_T)$.

The example rule in Fig. 10 illustrates a violation of the *termination criterion* (forward). It depicts a TGG rule that would lead to a non-terminating forward transformation, because the corresponding bookkeeping forward rule would create classes in a class diagram already connected with a block diagram. No source element at all is created by the TGG rule such that during the forward transformation target elements are created without a corresponding source element. No bookkeeping edges are consumed that would reduce the number of source elements to be translated, leading to non-termination.

The example transformation rules (cf. Fig. 3) illustrate a violation of the *conflict-freeness criterion* because a conflict exists between the bookkeeping rules 1 and 2 for the backward direction. Figure 11 shows these rules. The LHS of rule 1 is completely contained in the LHS of rule 2 (shaded back-

[20] Note that neither produce-forbid conflicts can occur, since no bookkeeping edges are produced, nor can attribute conflicts occur, since we assume that attributes are only written if the corresponding node is created (attribute criterion).

[21] Currently, there is no tool support for computing if critical pairs are strictly NAC-confluent. Implementing such an algorithm would involve exponential complexity with respect to the depth of the search tree.

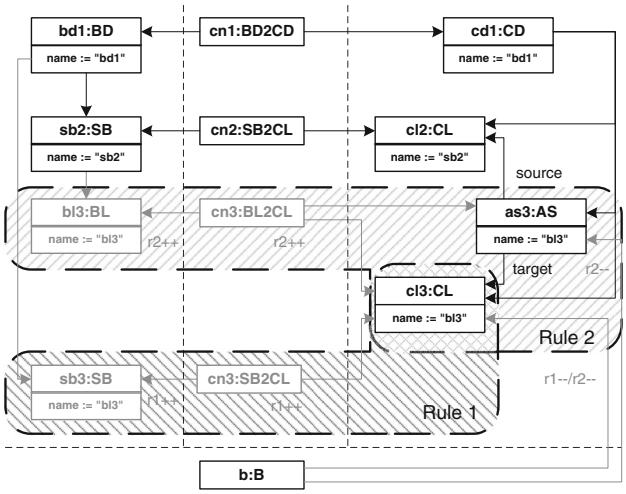



**Fig. 12** Operational backward rules $r1^{BB}$ and $r2^{BB}$ competing for the translation of *cl3*

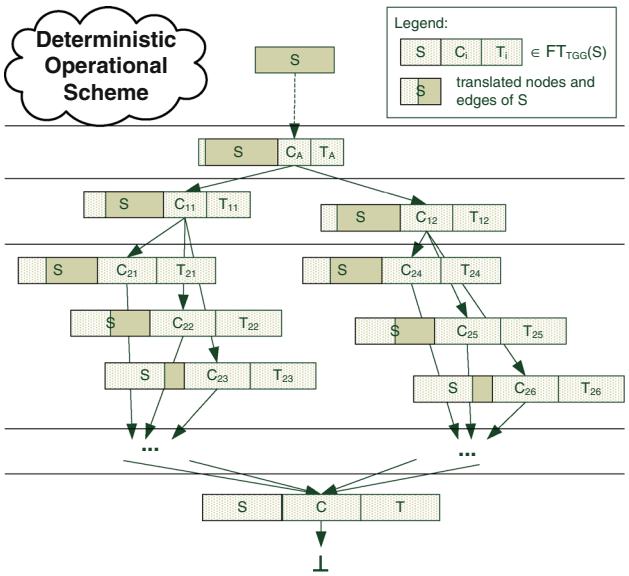



**Fig. 13** Deterministic operational scheme: $FT_{\mathrm{DET}}$

ground). Therefore, both rules can be applied in the same context and compete for the translation of the same *Class*, namely *cl2* in rule 1 and *cl3* in rule 2, respectively. Figure 12 shows the backward transformation of a class diagram model with both alternatives. In particular, *cl3* can be translated by rules 1 and 2 but with different results, which are both shown in the figure.[22] Rule 1 creates a second *SystemBlock* in the block diagram model, rule 2 creates a *Block*. In particular, we have a delete-use-conflict because if the bookkeeping edge to the instance class *cl3* is deleted by rule 1, then it cannot be matched anymore by rule 2 and the other way round. In addition, rule 1 leaves *as3* untranslated. After applying rule 1 to translate *cl3*, the bookkeeping edge to *as3* still exists. Therefore, the transformation result is not unique and the transformation not deterministic.

**Theorem 2** ($FT_{\mathrm{CON}}$ forward deterministic)
*For a $TGG$ and its operationalization $CON = OP_{FT}(TGG)$ fulfilling the forward determinism criteria, it holds that for each $S \in \mathscr{L}(S_T^A)$ either some $SCT$ exists such that $FT_{\mathrm{CON}}(S) = \{SCT\}$ or $FT_{\mathrm{CON}}(S) = \emptyset$. We say that $FT_{\mathrm{CON}}$ is forward deterministic.* (*Proof see Appendix A*)

Consequently, we can introduce the **deterministic operational scheme** $FT_{\mathrm{DET}}$ that works without backtracking: Given a $TGG$ and its operationalization $CON = OP_{FT}(TGG)$ fulfilling the forward determinism criteria, and a transformation domain $\mathscr{L}(S_T^A)$ fulfilling the domain restriction criterion, then $FT_{\mathrm{DET}}$ returns a valid transformation result $SCT$ of $S$ if after having applied the forward axiom rule $a^{\mathrm{BF}}$ to $B_{\mathrm{init}}S$ once, it is possible to apply bookkeeping forward rules in $CON$ up until each node and edge in $S$ have been translated (see Fig. 13). Otherwise, $FT_{\mathrm{DET}}$ returns undefined.

**Definition 3** (*deterministic operational scheme: $FT_{\mathrm{DET}}$*) Given a $TGG$ and its operationalization $CON = OP_{FT}(TGG)$ fulfilling the *forward determinism criteria*, then $FT_{\mathrm{DET}} : \mathscr{L}(S_T^A) \to \mathscr{L}(TGG)$ is a partial mapping such that $FT_{\mathrm{DET}}(S) := SCT$ if $FT_{\mathrm{CON}}(S) = \{SCT\}$, else $FT_{\mathrm{DET}}(S)$ is undefined.

As can be seen in Fig. 13, there might be several bookkeeping transformations leading to a transformation result where the complete source graph has been translated, but if such a valid transformation result exists, then it is unique.

## 7 Implementation scheme

Our implementation [14] is based on the Eclipse Modeling Framework[23] (EMF). Currently, this implementation only provides bookkeeping on nodes and does not provide bookkeeping for edges, because edges do not have an identity in EMF-based models.[24]

The operational rules for the implementation, $IMP$, are analogous to the bookkeeping forward rules in $CON$, apart from the fact that the bookkeeping for edges is omitted. Given a TGG rule $r$, we have a *node bookkeeping forward rule* $r^{\mathrm{IF}}$. First, let us assume that $S_{R_N}$ and $S_{L_N}$ denotes the set of nodes in $S_R$ and $S_L$, respectively. Given a $TGG$ rule $r$, then a node bookkeeping forward rule $r^{\mathrm{IF}} : B^N_{S_{R_N} \setminus S_{L_N}} S_R C_L T_L \leftarrow$

[22] Thereby, *cl2* of rule 1 as well as *cl3* of rule 2 are mapped to the instance *Class cl3*.

[23] http://www.eclipse.org/modeling/emf/.

[24] A helper structure would be required for edge bookkeeping.

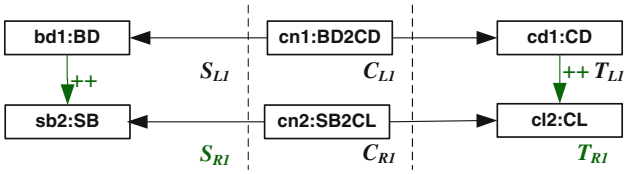



**Fig. 14** Violation of refined forward termination criterion

$bS_RC_LT_L \rightarrow bS_RC_RT_R$ is a span of inclusions, deleting for each node in $S_{R_N} \setminus S_{L_N}$ the corresponding bookkeeping edges, together with $NAC^{\text{IF}}$ a set of NACs forbidding for each node $n$ in $S_L$ an incoming bookkeeping edge from $b$, expressing that the node has been translated already. For example, $r_1^{\text{IF}}$ is equal to $r_1^{\text{BF}}$ (see Fig. 8) apart from the bookkeeping edge to the edge between $bd1$ and $sb2$. Given a $TGG$, then $IMP = OP_{FT}^{\text{IMP}}(TGG)$ is the rule set consisting of a node bookkeeping forward rule $r^{\text{IF}}$ for each rule $r$ of the $TGG$. Because we now only do bookkeeping on nodes but not on edges, we adapt the forward determinism criteria to forward implementation criteria such that *uniqueness of the transformation result* can still be guaranteed [15]. The *domain restriction criterion* remains unchanged. Moreover, we start each node bookkeeping forward transformation with the initial graph $B_{\text{init}}^N S$ (instead of $B_{\text{init}} S$) where bookkeeping edges to each node (and not to the edges) in $S$ were added to $S$.

The **forward implementation criteria** are given as follows: (1) Each $TGG$ rule creates at least one graph node on the source part (*refined termination criterion*). (2) For the rules in $IMP = OP_{FT}^{\text{IMP}}(TGG)$ there exist no critical pairs, ignoring pairs with same rules and same matches (*conflict-freeness criterion*). (3) For the rules $IMP = OP_{\text{IMP}}(TGG)$, we require for the attribute formula $\Phi$ that attribute values of preserved elements remain unchanged and that attribute value assignments of created elements are uniquely determined as soon as the value assignments of attributes of preserved elements are set (*attribute criterion*).

The example rule in Fig. 14 illustrates a violation of the refined termination criterion (forward). It depicts a TGG rule that would lead to a non-terminating node bookkeeping forward transformation. The corresponding node bookkeeping forward rule would create for each association between some block diagram and system block a corresponding association between a class diagram and a class. The problem is that since a node bookkeeping forward rule does not perform bookkeeping on edges this rule can be applied infinitely many times to the same association. Concluding, no bookkeeping is done reducing the number of source elements to be translated leading to non-termination.

The mapping $Trans^{FN}$ computes for each triple graph with node bookkeeping $B^N SCT$ the nodes that have already been translated by some bookkeeping forward rule before:

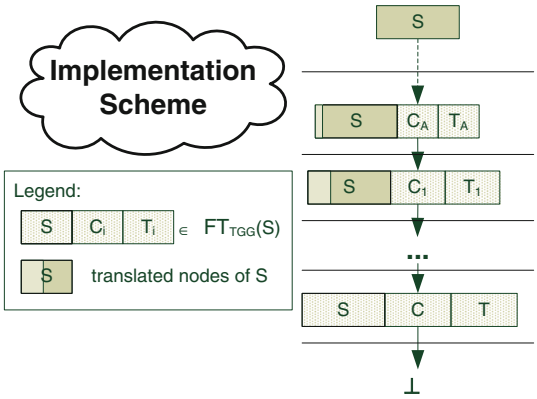



**Fig. 15** Implementation scheme: $FT_{\text{IMPd}}$

$Trans^{FN}(B^N SCT)$ is a subgraph of $S$, consisting of all nodes with no incoming bookkeeping edge. We define $B_{\text{init}}^N S$ as the initial source graph, consisting of $S$ and $b$ equipped with one bookkeeping edge for each graph node in $S$. Given a $TGG$ and its operationalization $IMP = OP_{FT}^{\text{IMP}}(TGG)$, then we can define $FT_{\text{IMP}} : \mathscr{L}(S_T) \rightarrow \mathscr{P}(\mathscr{L}(TGG))$ as follows: $FT_{\text{IMP}}(S) := \{SC_iT_i | B_{\text{init}}^N S \rightarrow_{a^{\text{IF}}} B_A^N SC_AT_A \rightarrow_{\text{IMP}}^* B_i^N SC_iT_i \wedge S_N = Trans^{FN}(B_i^N SC_iT_i)\}$, where $S_N$ is the set of all nodes in $S$. We prove that $FT_{\text{IMP}}$ is forward deterministic if it fulfills the forward implementation criteria and domain restriction criterion. Afterwards, we can define the implementation scheme $FT_{\text{IMPd}}$ that works without backtracking.

**Theorem 3** ($FT_{\text{IMP}}$ forward deterministic)
*For a $TGG$ and its operationalization $IMP = OP_{FT}^{\text{IMP}}(TGG)$ fulfilling the forward implementation criteria, it holds that for each $S \in \mathscr{L}(S_T^A)$ either some $SCT$ exists such that $FT_{\text{IMP}}(S) = \{SCT\}$ or $FT_{\text{IMP}}(S) = \emptyset$. We say that $FT_{\text{IMP}}$ is forward deterministic. (Proof see Appendix A)*

We introduce an **implementation scheme** $FT_{\text{IMPd}}$ that works without backtracking because of determinism of the transformation result: Given a $TGG$ and its operationalization $IMP = OP_{FT}^{\text{IMP}}(TGG)$ fulfilling the forward implementation criteria, and a transformation domain $\mathscr{L}(S_T^A)$ fulfilling the restricted domain criterion, then $FT_{\text{IMPd}}$ returns a valid transformation result $SCT$ for $S$ if after applying the forward axiom rule $a^{\text{IF}}$ to $B_{\text{init}}^N S$ once, node bookkeeping forward rules in $IMP$ can be applied up until all nodes in $S$ have been translated (see Fig. 15). Otherwise, $FT_{\text{IMPd}}$ returns undefined.

**Definition 4** (*implementation scheme: $FT_{\text{IMPd}}$*)
Given a $TGG$ and its operationalization $IMP = OP_{FT}^{\text{IMP}}(TGG)$ fulfilling the *forward implementation criteria*, then $FT_{\text{IMPd}} : \mathscr{L}(S_T^A) \rightarrow \mathscr{L}(TGG)$ is a partial mapping such that $FT_{\text{IMPd}}(S) := SCT$ if $FT_{\text{IMP}}(S) = \{SCT\}$, otherwise $FT_{\text{IMPd}}(S)$ is undefined.

For valid source models, we can prove *conformance* of $FT_{\text{IMPd}}$ with the TGG. A source model $S$ is *valid* if there exists a forward translation into a triple graph $SCT$ according to the TGG. More formally, $S$ is valid if a triple graph $SCT \in \mathscr{L}(TGG)$ with source component $S$ exists. Note that conformance for valid source graphs is obtained although we only do bookkeeping on nodes. This is because if each node in the source graph has been translated, we know that the translation has achieved its unique result consistent with the TGG and not only the nodes, but also all edges must have been translated in a consistent and unique way.

**Theorem 4** ($FT_{\text{IMPd}}$ conform with $FT_{\text{TGG}}$)
*Given a TGG with operationalization* $IMP = OP_{FT}^{\text{IMP}}(TGG)$ *fulfilling the forward implementation criteria and some valid* $S \in \mathscr{L}(S_T^A)$*, it holds that* $\{FT_{\text{IMPd}}(S)\} = FT_{\text{IMP}}(S) = FT_{\text{TGG}}(S)$. *(Proof see Appendix A)*

Note that only with the restriction that $S$ is valid, we can conclude conformance of $FT_{\text{IMPd}}$ with the TGG. It would be better to have an implementation, which checks this while transforming $S$. The deterministic operational scheme $FT_{\text{DET}}$, with bookkeeping also for edges, provides such a solution. Therefore, it is currently used to realize an improved implementation providing this feature.

*Example 1* (checking implementation criteria) When we analyze the forward transformation of our example TGG, we see that the forward axiom rule in Fig. 3 can only be matched in a unique way to $S$, so the *restricted domain criterion* is fulfilled. Moreover, it is obvious that each TGG rule in Fig. 3 creates at least one source node, which satisfies the *refined termination criterion*. Furthermore, there are no rules that change attribute values of elements of the LHS of the bookkeeping rule. Only attribute values of created elements are set. These values are uniquely determined by the name attribute value of the corresponding source element. Therefore, the *attribute criterion* is also fulfilled.

For the *conflict-freeness criterion*, we need to compute critical pairs for the node bookkeeping forward rules $r_1^{\text{IF}}$ and $r_2^{\text{IF}}$. AGG [41] is a graph transformation tool able to compute this set for a given pair of rules. In particular, it computes that indeed there exist no critical pairs such that the conflict-freeness criterion is fulfilled.

On the contrary, analyzing the backward transformation, the conflict-freeness criterion is not fulfilled, since AGG [41] computes a critical pair for the node bookkeeping backward rules $r_1^{\text{IB}}$ and $r_2^{\text{IB}}$. Both rules compete to translate the same class, being the target of an association, see the more detailed explanation in Sect. 6 and Fig. 11. Therefore, we have two cases: (a) If the encoding of the target cannot be changed, the backward transformation cannot be used. We can conclude that the criteria implicitly allow us to check whether the rule set is bidirectional or not. (b) We correct the encoding of the target if necessary and also the TGG rules in order to obtain backward determinism. In our example, this can be achieved by adding a stereotype in $r1$ to classes corresponding to system blocks and another stereotype to classes being target of an association as in rule $r2$. The *text* attribute of the stereotype is set to "system" and "block", respectively. Figure 16 shows the corrected TGG rule $r1$. For these corrected backward rules, AGG computes that indeed there exist no critical pairs. In Sect. 9.4, we provide more information on tool support for checking the implementation criteria.

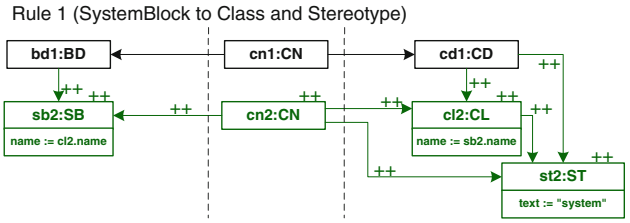



**Fig. 16** Corrected TGG rule $r_1$ with a stereotype

Concluding, suppose that we would not be able to rely on determinism of the model transformation result in our implementation. In this case, we would need to apply backtracking in order to find all possible transformation results. The complexity of such a backtracking algorithm would be exponential with respect to the depth of the derivation tree. On the contrary, if determinism can be assumed because it has been computed statically beforehand, then performing a model transformation becomes linear with respect to the depth of the derivation tree (abstracting from the complexity of matching rules, which would be comparable in both algorithms).

## 8 Efficient implementation scheme

As mentioned in Sect. 1, our implementation covers several additional optimizations that go beyond the abstraction presented in the previous section. These optimizations are based on the model transformation algorithm presented in [20]. In the meantime, the algorithm's implementation has been migrated to EMF and improved, especially regarding model synchronization (although synchronization is not considered in this paper). The optimizations in particular include the strategy to avoid searching for matches in the whole source graph and the way control flow constructs are used in the implementation to realize the transformation on top of the graph transformation language Story Diagrams [11] (see step (5) in Fig. 1). Therefore, we present the optimizations with regard to efficiency, implemented in the transformation engine when applying node bookkeeping operational rules. Thereafter, we prove that these optimizations preserve conformance with the TGG rules if they fulfill some additional

syntactical constraints. In fact, our efficient implementation keeps track of correspondence nodes that may still lead to new transformations because they were created by some previous transformation step. We use this *tracking mechanism* in order to optimize matching by searching only for matches including such a tracked correspondence node. We are even able to perform a local search starting from the tracked correspondence node in most cases.

To execute a model transformation, the engine is started with the root elements of the source and target models as parameters, as well as the desired transformation direction (i.e. forward or backward). First, the axiom rule is applied to transform the source axiom. The correspondence node that was created by the axiom rule is the first one to be tracked and it is saved into the so-called *active correspondence set* $P$. One node in $P$ at a time may be activated, and it is called the *active correspondence node*. We can then apply transformation rules that expect this type of correspondence node in their LHSs. If no more matches can be found for the active correspondence node, it is not tracked further, deactivated and removed from $P$. If during the search for matches a rule has been applied successfully, then the created correspondence node is marked to be tracked and added to the active correspondence set $P$ on his turn. This procedure is followed until the active correspondence set $P$ is empty and each correspondence node has been deactivated.

The following algorithm in pseudo-code describes the above procedure:

```
// Batch transformation algorithm with efficient
// matching
Apply axiom rule to create axiom correspondence
  node
Create empty set P and add created axiom
  correspondence node to P
Create empty set U

while (P is not empty)
{
  Select a correspondence node c from P
  U := rules in IMP with some node of same type
  as c in LHS
  if (some r in U applicable via match including
  c)
  {
    Apply r
    Add correspondence nodes created by r to P
  }
  else
  {
    Remove c from P
    Empty U
  }
}
```

In the following, we formalize this procedure and show that using such an efficient matching algorithm, we do not "miss" any matches. Therefore, we equip each graph with node bookkeeping $B^N SCT$ of a forward transformation with a set $P$ of correspondence nodes created and still to be processed during the forward transformation, also called *active correspondence set*, a node $c$, also called *active correspondence node*, and a set $U$ of all rules in $IMP$, holding a node of the same type as $c$ in their LHS, also called *active rule set*. We call these 4-tuples $(B^N SCT, P, c, U)$ also *efficient matching states*.

First, we define a new transition $\rightarrow_{\mathrm{IMP},c}$ where a rule application via some rule in $IMP$ on a graph with node bookkeeping takes place only if it matches the active correspondence node $c$.

**Definition 5** $(\rightarrow_{\mathrm{IMP},c})$
Given a $TGG$ and the relation $\rightarrow_{\mathrm{IMP}}$, then we define $\rightarrow_{\mathrm{IMP},c}$ as a relation over triple graphs with node bookkeeping and one distinguished correspondence node $c$ as follows: $B_i^N SC_i T_i \rightarrow_{\mathrm{IMP,c}} B_{i+1}^N SC_{i+1} T_{i+1}$ if $B_i^N SC_i T_i \rightarrow_{\mathrm{IMP}} B_{i+1}^N SC_{i+1} T_{i+1}$ via a match including correspondence node $c$.

Now we define transitions over efficient matching states performing analogous computations as given in the algorithm described above.

**Definition 6** $(\rightsquigarrow_{a^{\mathrm{IF}}}, \rightsquigarrow_{\mathrm{IMP}})$
Given a $TGG$ and the relation $\rightarrow_{\mathrm{IMP}}$ and corresponding $\rightarrow_{\mathrm{IMP},c}$, then we define $\rightsquigarrow_{a^{\mathrm{IF}}}$ and $\rightsquigarrow_{\mathrm{IMP}}$ as a relation over efficient matching states, being triple graphs with node bookkeeping equipped with an active correspondence set $P$, an active correspondence node $c$, and an active rule set $U$, where in particular the active correspondence node might be undefined $\perp$:

1. $(B_{\mathrm{init}}^N S, \emptyset, \perp, \emptyset) \rightsquigarrow_{a^{\mathrm{IF}}} (B_a^N SC_a T_a, P_a, \perp, \emptyset)$ if $B_{\mathrm{init}}^N S \rightarrow_{a^{\mathrm{IF}}} B_a^N SC_a T_a$, and $P_a$ is the set of created correspondence nodes in this step (axiom initialization).
2. $(B_i^N SC_i T_i, P, \perp, \emptyset) \rightsquigarrow_{\mathrm{IMP}} (B_i^N SC_i T_i, P, c, U)$ if $c \in P$, and $U$ consists of all rules in $IMP$ that contain in their LHS a correspondence node of the node type of $c$ (new active correspondence node).
3. $(B_i^N SC_i T_i, P, c, U) \rightsquigarrow_{\mathrm{IMP}} (B_{i+1}^N SC_{i+1} T_{i+1}, P', c, U)$ if $B_i^N SC_i T_i \rightarrow_{\mathrm{IMP},c} B_{i+1}^N SC_{i+1} T_{i+1}$ via some rule $r^{\mathrm{IF}}$ in $U$, and $P'$ is the set obtained by adding to $P$ all created correspondence nodes in this step (apply transformation and update $P$).
4. $(B_i^N SC_i T_i, P, c, U) \rightsquigarrow_{\mathrm{IMP}} (B_i^N SC_i T_i, P, c', U')$ if $c' \in P$, and $U'$ consists of all rules in $IMP$ that contain in their LHS a correspondence node of the node type of $c'$ (switch active correspondence node).
5. $(B_i^N SC_i T_i, P, c, U) \rightsquigarrow_{\mathrm{IMP}} (B_i^N SC_i T_i, P', \perp, \emptyset)$ if no $B_i^N SC_i T_i \rightarrow_{\mathrm{IMP},c} B_{i+1}^N SC_{i+1} T_{i+1}$ via some rule $r^{\mathrm{IF}}$ in

$U$ exists and $P' = P \setminus \{c\}$ (end active correspondence node).

In order for this efficient matching strategy to still conform with the TGG rules, we need them to obey some additional syntactic constraints. Given a $TGG$, then the **forward efficiency criteria** consist of the following syntactical constraints on each $TGG$ rule:

(a) Each TGG rule except for the axiom rule contains at least one correspondence node in its LHS (restricted match search).
(b) Each target node in the LHS of a TGG rule is connected with an incoming edge from exactly one correspondence node in the LHS (restricted match search).
(c) The axiom rule and each TGG rule create exactly one correspondence node (create new matches via correspondence node only).
(d) If the axiom rule or some TGG rule create a target node, then it is created together with an incoming edge from the created correspondence node (create new matches via correspondence node only).
(e) If the axiom rule or some TGG rule create a target edge, then at least one of its end nodes is created as well (create new matches via correspondence node only).
(f) If the axiom rule or some TGG rule create a correspondence-source or correspondence-target edge, then it is an outgoing edge from the created correspondence node (create new matches via correspondence node only).

The forward efficiency criteria (a-f) ensure that each rule application via the axiom rule $a^{\mathrm{IF}}$ or some rule $r^{\mathrm{IF}}$ in $IMP$ may only create new matches for some rule in $IMP$ that include the correspondence node $c$ just created by $a^{\mathrm{IF}}$ or $r^{\mathrm{IF}}$, respectively. Assuming that a new match not including the correspondence node $c$ would arise. Then, $a^{\mathrm{IF}}$ or $r^{\mathrm{IF}}$ must have created some elements other than $c$, leading to a new match for some rule in $IMP$ not including $c$. Therefore, this element cannot be a correspondence-source or correspondence-target edge, since then $c$ would be included in the match because of criterion (f). Moreover, it cannot be a target node, since it has been created with an edge from $c$ because of criterion (d) and each target node must be matched together with an edge from exactly one correspondence node because of criterion (b). Thus, either it matches $c$ or it does not match at all. It cannot be a target edge either, because at least one of its ends is a created target node because of criterion (e). This target node was created with an edge from $c$ because of criterion (d) which must be matched together with an edge from exactly one correspondence node because of criterion (b). Thus, either it matches $c$ or it does not match at all.

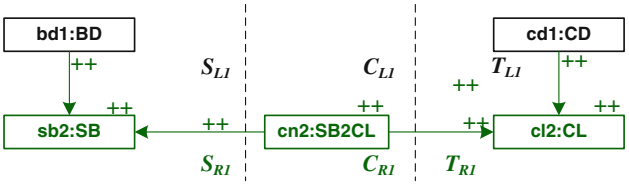



**Fig. 17** Example rule violating the forward efficiency criteria (a) and (b)

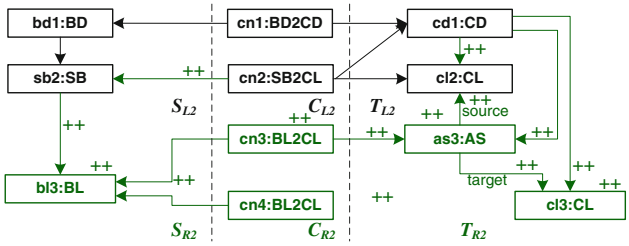



**Fig. 18** Example rule violating the forward efficiency criteria (b), (c), (d), (e), and (f)

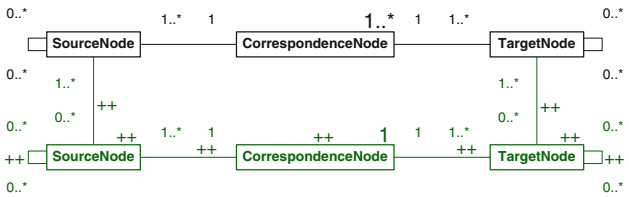



**Fig. 19** Prototypical structure of a TGG rule

Figure 17 shows an example rule, which violates forward efficiency criteria (a-b). There is no correspondence node in its LHS (a) and *cd1* does not have an incoming edge (b).

Another invalid TGG rule is shown in Fig. 18. The node *cd1* is connected to two correspondence nodes of the LHS (criterion (b)). The rule creates two correspondence nodes, *cn3* and *cn4* (criterion (c)). *cl3* has no incoming correspondence edge (criterion (d)). Finally, the edge between *cd1* and *cl2* is created but its end nodes are not (criterion (e)). The created correspondence-source edge between *cn2* and *sb2* does not start from the created correspondence node (criterion (f)).

We tried to keep the efficiency criteria only as strict as necessary such that conformance with the TGG is still guaranteed. However, it is probably not that easy for an average model transformation developer to keep all these criteria in mind. In practice, we employ the following prototypical structure for TGG rules as shown in Fig. 19 as guideline. The prototypical structure fulfills slightly stricter criteria, which implies satisfaction of the forward and backward efficiency criteria. Rules that adhere to this prototypical structure create source and target models, where source and target patterns that correspond to each other are connected by exactly one

correspondence node.[25] The TGG rules in our running example (see Fig. 3) adhere to the above prototypical structure and, therefore, fulfill the efficiency criteria.

We introduced the efficiency criteria to be able to work with the following efficient forward scheme: Given a $TGG$ and its operationalization $IMP = OP_{FT}^{\text{IMP}}(TGG)$, then we can define $FT_{\text{EFF}} : \mathscr{L}(S_T) \to \mathscr{P}(\mathscr{L}(TGG))$ as follows: $FT_{\text{EFF}}(S) := \{SC_iT_i | (B_{\text{init}}^N S, \emptyset, \perp, \emptyset) \rightsquigarrow_{a^{\text{IF}}} (B_a^N SC_aT_a, P_a, \perp, \emptyset) \rightsquigarrow_{\text{IMP}}^* (B_i^N SC_iT_i, \emptyset, \perp, \emptyset) \wedge S_N = Trans^{FN}(B_i^N SC_iT_i)\}$, where $S_N$ is the set of all nodes in $S$. $FT_{\text{EFF}}$ uses the relations $\rightsquigarrow_{a^{\text{IF}}}$ and $\rightsquigarrow_{\text{IMP}}$ over efficient matching states and following our efficient matching strategy in order to compute a transformation result.

We do not want conformance with the TGG to be disturbed by applying this efficient matching strategy. Therefore, we show that $FT_{\text{EFF}}$ conforms to $FT_{\text{IMP}}$, without efficient matching strategy, if the forward implementation criteria and forward efficiency criteria are fulfilled. To this extent, we argue in the following lemma that no matches can be "missed" by the relation $\rightsquigarrow_{\text{IMP}}$ used by $FT_{\text{EFF}}$. Afterwards, we can follow directly that because of conformance of $FT_{\text{EFF}}$ with $FT_{\text{IMP}}$ and the fact that $FT_{\text{IMP}}$ is forward deterministic, also $FT_{\text{EFF}}$ is forward deterministic.

**Lemma 3** (conformance of $\to_{\text{IMP}}$ and $\rightsquigarrow_{\text{IMP}}$)
*For a $TGG$ with operationalization $IMP = OP_{FT}^{\text{IMP}}(TGG)$ satisfying the forward implementation and forward efficiency criteria, it holds that $B_{\text{init}}^N S \to_{a^{\text{IF}}} B_a^N SC_aT_a \to_{\text{IMP}}^* B_i^N SC_iT_i$ iff $(B_{\text{init}}^N S, \emptyset, \perp, \emptyset) \rightsquigarrow_{a^{\text{IF}}} (B_a^N SC_aT_a, P_a, \perp, \emptyset) \rightsquigarrow_{\text{IMP}}^* (B_i^N SC_iT_i, P, c, U)$, where $P_a$ consists of the correspondence node created by $a^{\text{IF}}$ and the last transition in this sequence via $\rightsquigarrow_{\text{IMP}}$ is one of type (3) in Definition 6.* (*Proof see Appendix A*)

**Theorem 5** (conformance of $FT_{\text{EFF}}$ and $FT_{\text{IMP}}$)
*For a $TGG$ with operationalization $IMP = OP_{FT}^{\text{IMP}}(TGG)$ satisfying the forward implementation and forward efficiency criteria, it holds that $FT_{\text{EFF}}(S) = FT_{\text{IMP}}(S)$.* (*Proof see Appendix A*)

However, we do not only want $FT_{\text{EFF}}$ to be conform with the TGG, but we also still aim for a scheme working without backtracking. Therefore, we prove that $FT_{\text{EFF}}$ is forward deterministic if it fulfills the forward implementation criteria, forward efficiency criteria and the domain restriction criterion.

**Theorem 6** ($FT_{\text{EFF}}$ forward deterministic)
*For a $TGG$ and its operationalization $IMP = OP_{FT}^{\text{IMP}}(TGG)$ fulfilling the forward implementation and forward efficiency criteria, it holds that for each $S \in \mathscr{L}(S_T^A)$ either some $SCT$ exists such that $FT_{\text{EFF}}(S) = \{SCT\}$ or $FT_{\text{EFF}}(S) = \emptyset$. We say that $FT_{\text{EFF}}$ is forward deterministic.*

*Proof* This follows directly from the fact that $FT_{\text{IMP}}$ and $FT_{\text{EFF}}$ are conform (Theorem 5) and the fact that $FT_{\text{IMP}}$ is forward-deterministic (Theorem 3). □

Consequently, we can introduce the **efficient implementation scheme** $FT_{\text{EFFd}}$ that works without backtracking and conforms to $FT_{\text{IMPd}}$: Given a $TGG$ and its operationalization $IMP = OP_{FT}^{\text{IMP}}(TGG)$ fulfilling the forward implementation criteria and the forward efficiency criteria, and a transformation domain $\mathscr{L}(S_T^A)$ fulfilling the domain restriction criterion, then $FT_{\text{EFFd}}$ returns a transformation result $SCT$ of $S$ if after having applied the forward axiom rule $a^{\text{BF}}$ to $B_{\text{init}} S$ via $\rightsquigarrow_{\text{IMP}}$, it is possible to apply bookkeeping forward rules in $IMP$ via $\rightsquigarrow_{\text{IMP}}$ up until each node in $S$ has been translated. Otherwise, $FT_{\text{EFFd}}$ returns undefined.

**Definition 7** (*efficient implementation scheme: $FT_{\text{EFFd}}$*)
Given a $TGG$ and its operationalization $IMP = OP_{FT}^{\text{IMP}}(TGG)$ fulfilling the *forward implementation* and *forward efficiency criteria*, then $FT_{\text{EFFd}} : \mathscr{L}(S_T^A) \to \mathscr{L}(TGG)$ is a partial mapping such that $FT_{\text{EFFd}}(S) := SCT$ if $FT_{\text{EFF}}(S) = \{SCT\}$, else $FT_{\text{EFFd}}(S)$ is undefined.

**Theorem 7** ($FT_{\text{EFFd}}$ and $FT_{\text{TGG}}$)
*Given a $TGG$ with operationalization $IMP = OP_{FT}^{\text{IMP}}(TGG)$ fulfilling the forward implementation and forward efficiency criteria and some valid $S \in \mathscr{L}(S_T^A)$, it holds that $\{FT_{\text{EFFd}}(S)\} = FT_{\text{EFF}}(S) = FT_{\text{TGG}}(S)$.*

*Proof* This follows directly from conformance of $FT_{\text{EFF}}$ with $FT_{\text{IMP}}$ (Theorem 5) such that $FT_{\text{EFF}}(S) = FT_{\text{IMP}}(S)$. Moreover, from conformance of $FT_{\text{IMPd}}$ with $FT_{\text{TGG}}$ (Theorem 4) it follows that $FT_{\text{TGG}}(S) = FT_{\text{IMP}}(S) = \{FT_{\text{IMPd}}(S)\}$ such that $FT_{\text{EFF}}(S) = FT_{\text{TGG}}(S)$. Since $S$ is valid, $FT_{\text{TGG}}(S)$ is not empty and therefore, $\{FT_{\text{EFFd}}(S)\} = FT_{\text{EFF}}(S) = FT_{\text{TGG}}(S)$. □

Finally, note that our implementation uses a queue instead of a set to store the correspondence nodes contained in $P$ and a list $L$ instead of a set to store the rules contained in $U$. Moreover, a correspondence node $c$ stays active up until all applicable rules in $L$ have been applied. This more concrete efficient matching strategy is described by the following algorithm in pseudo-code:

```
// Batch transformation algorithm with
   // efficient matching

// Tracking mechanism as a queue
Apply axiom rule to create axiom correspondence
  node
```

[25] This prototypical structure allows to automatically derive the correspondence structure in TGG rules if the transformation developer specifies corresponding source and target patterns (although not implemented in our tool yet). This would be similar to QVT Relational, where such correspondences are hidden from the user.

```
Create a queue Q and push the created axiom
  correspondence node into Q
Create empty list L

while (Q is not empty)
{
  Pop first correspondence node c from queue
  L := rules in IMP having a node of same type
  as c in LHS
  foreach rule r in L
  {
    if (r applicable via match including c)
    {
      Apply r
      Push correspondence nodes created by r
      into Q
    }
  }
}
```

This transformation mechanism activates a new correspondence node only if all rules via the active correspondence node *c* have been applied already. Moreover, correspondence nodes are activated and rules are applied in a specific order because of the queue and list structure of tracked correspondence nodes and rules, respectively. Because of Theorem 6 we know that the relation $FT_{\text{EFF}}$ is deterministic provided that the forward implementation and efficiency criteria are fulfilled. Therefore, we know that the algorithm as presented in the beginning of this section allows for several different computations, but always obtains the same transformation result. Our implementation now realizes one of these computations by choosing a specific order for which rule to apply and where to apply it, and at the same time leading to the same result.

*Example 2* The running example in Fig. 3 obviously fulfills the forward efficiency criteria. All rules contain correspondence nodes in their LHS (except the axiom rule) and create exactly one correspondence node, all target nodes are created with a correspondence edge, and target edges are created along with target nodes. Therefore, the running example is suitable for applying the transformation algorithm according to the efficient implementation scheme.

We have implemented a TGG Monitor (cf. Fig. 1) that allows to observe the model transformation process. Some screenshots of this monitor will be used in the following to illustrate the forward transformation of the running example. In the first step, the forward bookkeeping axiom rule is applied (see Fig. 20). The source model's root element is provided as a parameter to the axiom rule. It creates a correspondence node of type *CorrAxiom*, which is added to the transformation engine's queue $Q$, and a *ClassDiagram* in the target model. The *ClassDiagram*'s name is set to the *BlockDiagram*'s name.

In the next step, *CorrAxiom* is removed from the queue. It is now the active correspondence node. The applicability of all forward bookkeeping rules that contain a correspondence node of this type in their LHS, which applies to both rules of the running example, needs to be checked. Therefore, the list of rules $L$ contains these two rules. The application order of the rules is not explicitly specified, so let us assume that rule 2 is tried to be applied first. The active correspondence node is provided as a parameter to the rule. Rule 2 matches it to the appropriate correspondence node in its LHS, *cn1*, and searches for matches of the other elements of its LHS. The *SystemBlock* and its corresponding *Class* are required but the *Class* does not exist because the *SystemBlock* has not been transformed yet. Therefore, application of rule 2 fails and the next rule is applied.

Now, a match for the application condition of rule 1 can be found. Rule 1 transforms the *SystemBlock* and creates a correspondence node and target elements (see Fig. 21).

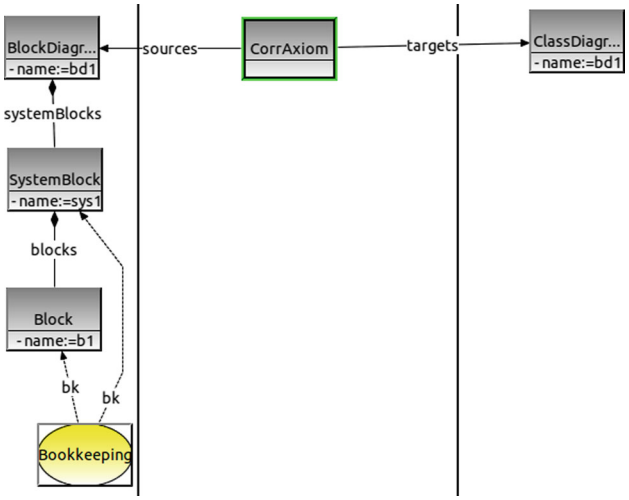



**Fig. 20** State of the model transformation after applying the axiom rule

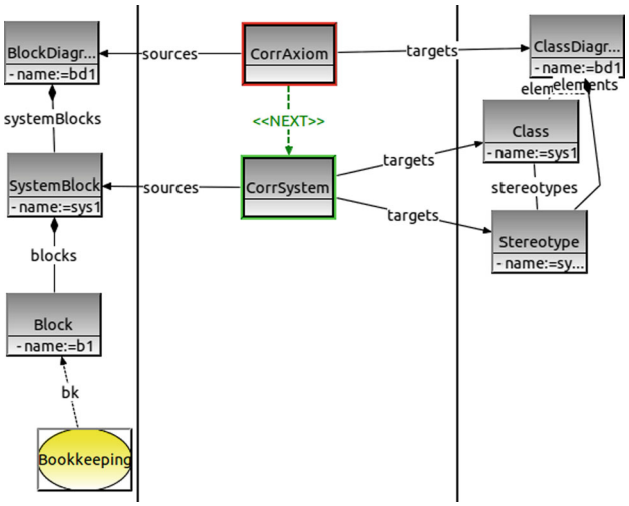



**Fig. 21** State of the model transformation after applying rule $r_1^{\text{BF}}$ via *CorrAxiom*

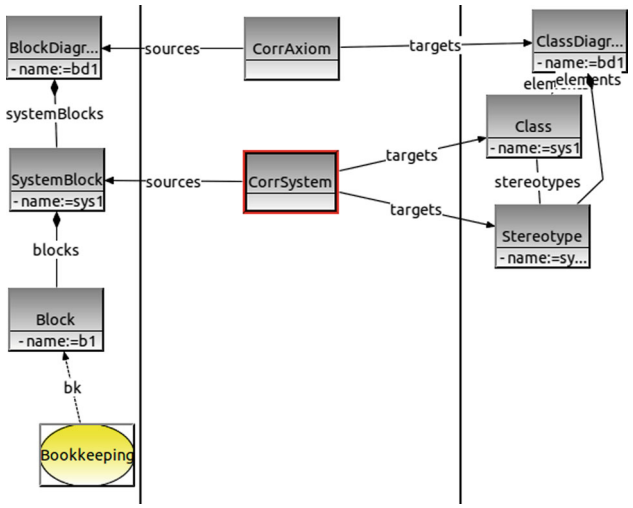



**Fig. 22** State of the model transformation after *CorrAxiom* has been deactivated

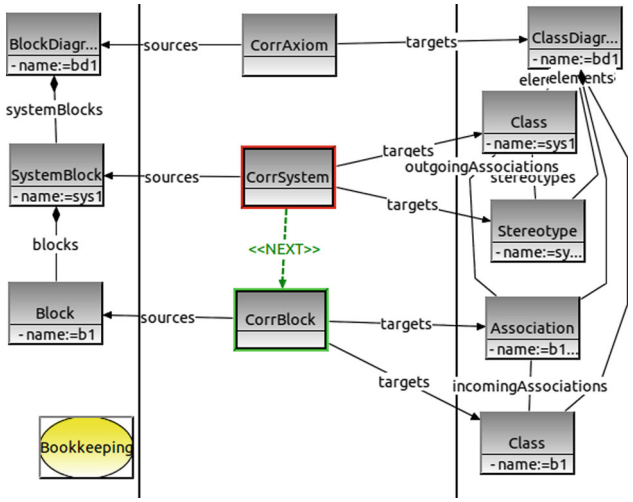



**Fig. 23** State of the model transformation after applying rule $r_2^{\mathrm{BF}}$ with *CorrSystem* as the active correspondence node

The new correspondence node is added to $Q$ (indicated by the $\langle\langle NEXT\rangle\rangle$ edge between the correspondence nodes).

After that, no more rules can be applied via the active correspondence node *CorrAxiom*. Therefore, *CorrAxiom* is deactivated and the next correspondence node, *CorrSystem*, is removed from the queue and activated. This is shown in Fig. 22.

Again, all rules are applied that contain a correspondence node of this type in their LHS. In the example, only rule 2 requires a node of type *CorrSystem* so its application is checked. Indeed, a match can be found for rule 2. It transforms the *Block* and creates a correspondence node and target elements. This correspondence node is also added to the queue (see. Fig. 23).

After rule 2, there are no more rules to be applied. Rule 1 does not require a correspondence node of type *CorrSystem*. Therefore, the transformation engine removes the next correspondence node from the queue again, which is a *CorrBlock* node, and turns it into the active correspondence node. Again, all forward bookkeeping rules are tried to be applied that contain a node of this type in their LHS. However, there are no rules with this kind of correspondence node in their LHS. Now, the queue has run empty and the transformation process has finished.

## 9 Evaluation and tool support

The preceding sections presented the so-called implementation and efficiency criteria, which ensure conformance of a model transformation following the efficient implementation scheme with the TGG. To be useful to a model transformation developer, tools are needed that can check these criteria automatically. The tools for editing TGGs and performing model transformations will be briefly presented in Sect. 9.1. Section 9.2 describes how checking the aforementioned criteria is implemented. Finally, in Sect. 9.4 we recall the results obtained for our running example. In Sect. 9.5, we present and evaluate the results that we have obtained on an industrial case study.

### 9.1 Tool support

With our tool framework,[26] a model transformation developer can create TGG rules using a graphical TGG editor (see Fig. 24). Using EMF's validation mechanism, most of the presented criteria can be checked directly within the editor (cf. Sect. 9.2). Elements violating these criteria are highlighted in the editor so the user can easily fix these problems.

The generator (cf. Fig. 1) creates Story Diagrams from the TGG. These Story Diagrams are executed by a Story Diagram interpreter [18]. This interpreter has a dynamic pattern matching strategy, which adapts to the specifics of the instance model. This ensures good pattern matching performance in many cases. The Story Diagram interpreter is invoked by the TGG Engine to execute a model transformation. The efficient implementation scheme (see Sect. 8) is realized by the TGG Engine and the generated Story Diagrams. The TGG Engine contains those parts that are independent of the specific source and target metamodels, e.g., the transformation queue and the overall algorithm that controls rule applications. The generated Story Diagrams contain the operational rules for the forward, backward, and mapping transformations. Besides transformations, also online model

[26] It can be downloaded from our Eclipse update site http://www.mdelab.de/update-site.

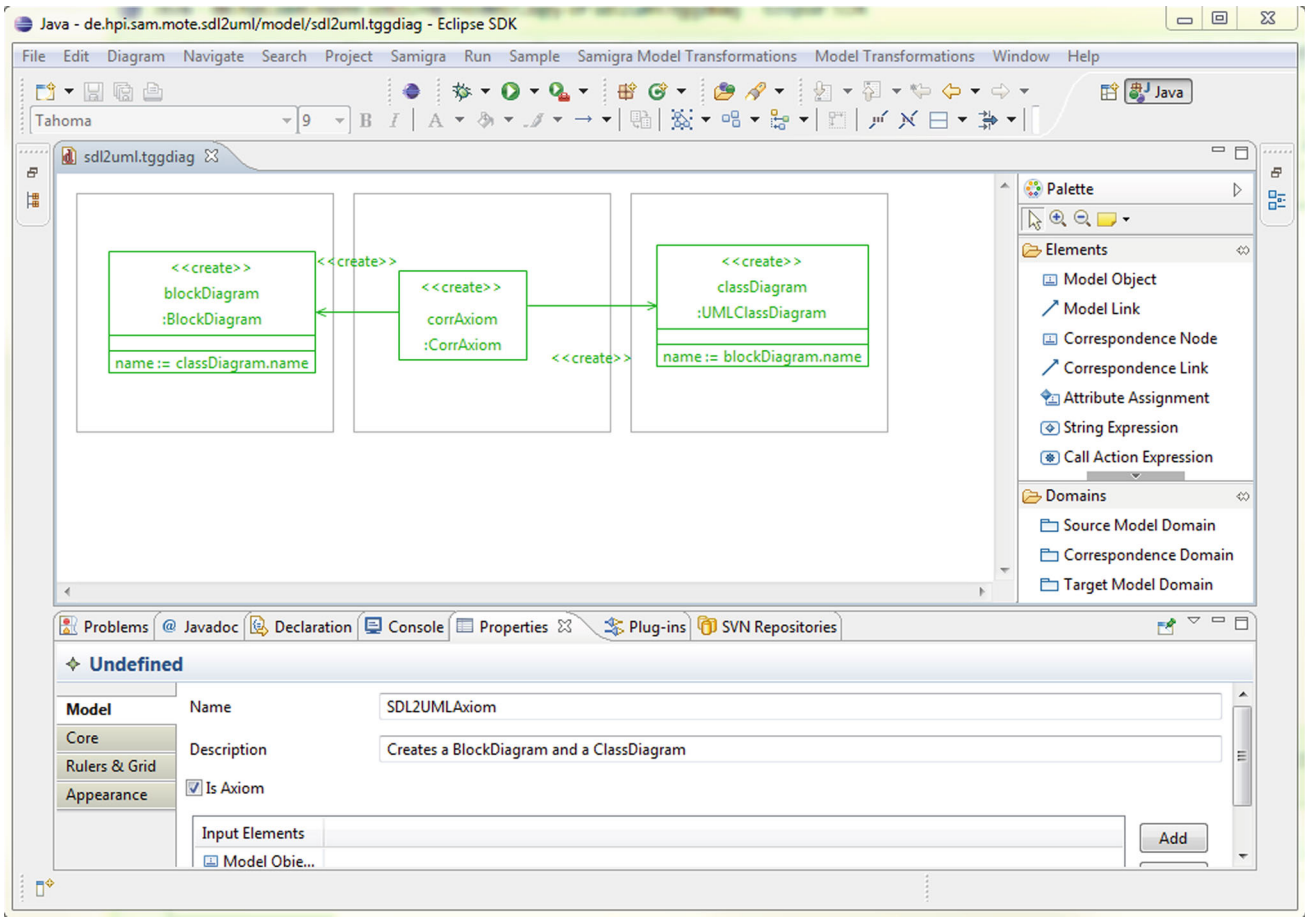



**Fig. 24** Screenshot of the TGG Editor showing the axiom of the example TGG

synchronization is supported by recording model changes and propagating these changes to the other model according to the TGG.

Recently, also a TGG Monitor was developed, which allows the execution of a model transformation to be monitored step by step. It can show the current state of the instance models and the content of the TGG Engine's queue with tracked correspondence nodes. The model elements can be colored depending on the order in which they were created, or depending on which rule created them. This helps in debugging model transformations.

## 9.2 Checking criteria

The implementation and efficiency criteria (apart from the conflict-freeness criterion as explained in the next section) as presented in Sects. 7 and 8, respectively, only impose some simple constraints on the structure of TGG rules. For example, the refined termination criterion demands that each TGG rule creates at least one graph node on the source part. This can be checked easily using constraint languages like OCL. In our implementation, we use Check,[27] a constraint language very similar to OCL. The constraint for checking the refined termination criterion looks like this:

```
context TGGRule ERROR
``A TGG rule must create at least one source
 graph node'':
 this.sourceDomain.modelElements.
 exists(e|e.modifier ==
  TGGModifierEnumeration::CREATE);
```

These checks are integrated with the EMF validation framework and can be easily invoked from the TGG editor. Elements violating a criterion are tagged and the appropriate error message is displayed to the user.

## 9.3 Checking for conflicts

*Static conflict detection* is enabled by so-called critical pair detection. A critical pair is a pair of conflicting transformations in a minimal context. In general, computing the complete set of critical pairs for a given pair of rules is exponential

[27] http://www.eclipse.org/modeling/m2t/?project=xpand.

in the number of rule elements in the LHSs of these rules. This is because so-called overlaps of the rules' preconditions need to be built in order to compute all possible minimal contexts of rule applications. AGG is a graph transformation tool [41] able to compute the complete set of critical pairs for a given set of graph transformation rules. We have developed a prototypical implementation, translating TGG bookkeeping operational rules into AGG rules.[28] Unfortunately, the AGG critical pair algorithm does not scale for the rules used in our industrial case study (see Sect. 9.5) because of run-out-of-memory problems. Similar problems have been reported, for example, in [2,32].

Because of these scalability problems, we decided to implement a *runtime check* reporting a uniqueness violation if the model transformation result obtained by our efficient implementation scheme is not likely to be the only one. Executing the implemented runtime check is roughly as time-consuming as the transformation itself. Therefore, the runtime check does not add any *scalability* problems. Note that the results of such a runtime check are only valid for the translation of a specific source model. On the contrary, static conflict detection analysis is able to provide a uniqueness result for any transformed model via the analyzed TGG. The following pseudo-code describes the runtime check for a given source model $S$ and a TGG with the associated forward rules without bookkeeping:

```
//Conflict check at runtime
//Given SCT, a transformation result of S
//according to the efficient implementation
//scheme

forall (forward rules without bookkeeping r^F)
{
  forall (matches m of r^F in SCT)
  {
    if (application result of r^F via m not
        found)
      report violation
  }
}
```

This runtime check takes as input a transformation result $SCT$ of $S$ obtained by the efficient implementation scheme. First, it checks for each forward rule without bookkeeping if it is applicable to $SCT$, i.e. it is searched for matches of $r^{\mathrm{F}}$ (cf. Sect. 5). Afterwards, it checks for each match if $r^{\mathrm{F}}$ has been applied. If not, a violation is reported, since in this case it could have been possible to obtain a different transformation result.

The core idea of the runtime check is that if a match for the LHS of a forward rule without bookkeeping exists, then the preceding transformation must have created all elements specified by the RHS of the rule. If this is not the case, then these matched elements must have been transformed by another rule whose LHS overlaps with the LHS of the current rule. This would be a violation of the conflict-freeness criterion.

If the runtime check does not report a violation, then we can be sure that the transformation result $SCT$ of $S$ is unique in the sense that there exists no other transformation result $SC'T'$ because of the following argumentation: Suppose that we obtain a different transformation result $SC'T'$ without a violation report. Then we would have another transformation instance reaching some state $SC'_{i-1}T'_{i-1}$ that can still be mapped to $SCT$, but after applying some operational rule $r^{\mathrm{F}}$ the state $SC'_iT'_i$ is obtained containing elements that cannot be mapped anymore to $SCT$. In other words, $r^{\mathrm{F}}$ is applicable to $SCT$, although the application result cannot be found in $SCT$. Consequently, the runtime check should have reported a violation which is a contradiction. In order to also check that no results $SC'T'$ may arise that can be mapped to $SCT$ because they are smaller than $SCT$ (i.e. only create a subset of $CT$), it is stored during the transformation which rule translates which elements into which elements. This is ensured by marking the created correspondence node with the rule name creating it together with the assumption that our TGG rules adhere to the prototypical structure as presented in Fig. 19. The runtime check then does not only check if the application result for each match $m$ of $r^{\mathrm{F}}$ can be found in $SCT$. In addition, it checks if the application result has indeed been generated via match $m$ and rule $r^{\mathrm{F}}$. Since elements can only be translated once, this check is sufficient. Concluding, our runtime check also excludes the case that a smaller result could have been obtained.

### 9.4 Example: block diagrams—class diagrams

Our running example (see Fig. 3) consists of an axiom and two TGG rules specifying the translation of block diagrams into class diagrams and the other way round.

In Example 1, we have described how to check the implementation criteria. Checking the forward efficiency criteria is described in Example 2. Executing the EMF validation as described in Sect. 9.2 to check these criteria takes 8 s. However, the checks include checking the forward and backward criteria, as well as several rather trivial checks (e.g., check if the name and type of a node are set).

We used AGG to statically check that the node bookkeeping forward rules $IMP$ of our example are conflict-free. In particular, AGG is able to verify this in 1 s.[29] Note that AGG

[28] Since stereotypes are not supported by AGG, we need to translate them into regular types, an automatic handling of this feature is not supported yet.

[29] All performance measurements were executed on a PC with the following configuration: Pentium Core i5 750 processor with 2.67 GHz, 6.00 GB RAM, Ubuntu 10.10 64bit, OpenJDK 6b20-1.9.7.

does not provide the possibility to specify edges from nodes to edges. However, since we do critical pair analysis on rules in $IMP$, doing only bookkeeping on nodes, we do not need this possibility. Moreover, note that for computing critical pairs in AGG we interpret stereotypes with different text attributes as distinct types and use maximal multiplicities in the type graph ruling out critical pairs describing conflicts that would never occur anyway. In [30], it is proven that ruling out critical pairs by maximal multiplicities does not affect the completeness of critical pairs.

On the contrary, analyzing the backward transformation, we have seen that the conflict-freeness criterion was not fulfilled. Figure 16 shows the corrected TGG rule $r1$. For these corrected backward rules, AGG computes in 3 s. that indeed there exist no critical pairs.

### 9.5 Industrial case study: SysML-AUTOSAR

We have employed the model transformation system in an industrial case study [17,19] to integrate the open-source SysML modeling tool TOPCASED[30] and the commercial modeling tool SystemDesk from dSPACE[31] for AUTOSAR modeling. Both modeling languages are widely used in the automotive industry to model embedded systems. SysML is less detailed but includes modeling hardware components while AUTOSAR focusses on modeling software components. The model transformation system was used to transform SysML to AUTOSAR models and then to propagate changes made in one model to the other. For this, we needed the model synchronization feature of the transformation system.

In the case study, two TGGs were developed: one for transforming SysML models to AUTOSAR models, and the other for transforming AUTOSAR models to SystemDesk models. TOPCASED is based on Eclipse and EMF so accessing its models is very easy. SystemDesk only provides a COM API to access the model that the user is currently editing. In addition, the internal representation of SystemDesk differs slightly from the official AUTOSAR standard. Therefore, we decided to split the problem of propagating the AUTOSAR model to SystemDesk into two parts: First, the second model transformation converts the AUTOSAR model to the SystemDesk representation, then a COM adapter synchronizes this model with SystemDesk's internal model. Due to the complexetiy of both languages, only significant parts of their metamodels are covered by the developed transformation rules. For more information, we refer to [17,19].

The SysML-to-AUTOSAR TGG consists of 26 rules (including axiom), where the largest rules contain 21 nodes. Executing the EMF validation rules on the whole TGG to check the implementation and efficiency criteria takes 24 s.[32] This transformation maps many different AUTOSAR elements to the same SysML elements, especially SysML blocks. To make the transformation conflict-free, we used stereotypes in the SysML model, similar to the running example (cf. Sect. 7). These stereotypes are already used by system engineers in their SysML models. As mentioned before, the static conflict detection check in AGG does not scale for TGG rules of this size. Therefore, we only executed our runtime check, which is roughly as time-consuming as the model transformation itself.

The AUTOSAR-to-SystemDesk TGG contains 51 rules, with a maximum of 31 nodes per rule. Here, checking the implementation and efficiency criteria takes 38 s. Also in this case, we only performed runtime checks showing uniqueness of the transformation result. These numbers illustrate that checking the implementation and efficiency criteria can be done in quite a short time, even for very large transformations.

## 10 Conclusion

In this paper, we have closed the gap between the formal semantics of TGGs and our implementation. This not only ensures that a valid rule set results in a unique and semantically correct outcome, it also permits to decide whether a TGG can be applied in a conform way in the forward or backward direction, or in particular, in both directions.

We learned from this work that it is possible to close the gap between the formal semantics of TGGs and an already existing implementation. As a guideline for similar work it turned out to be successful to divide the proof into several steps. In each step, an ideal implementation was presented including more aspects of the real implementation. Corresponding criteria still ensuring conformance with the TGG were developed for each step. The first steps for bridging the gap were mainly concerned with coming from the relational semantics to the bookkeeping mechanism as well as the determinism assumption used in our tool. The last step was concerned with showing that additional optimizations used for efficiency reasons still conform with the relational TGG semantics. Having formalized the assumptions and restrictions of our tool ensuring conformance, it makes our tool comparable to other approaches with the same goal. Moreover, our criteria could serve as a basis for developing similar criteria for other relational approaches such as QVT.

Concerning future work, this contribution links a practical implementation with a suitable formal semantics such that based on this sound foundation and former work [3,12,30] we can now study the verification of model transformations

[30] http://www.topcased.org.

[31] http://dspace.de.

[32] Intel Core i5 750, 2.67 GHz, 6GB RAM.

exploiting the identified criteria. In addition, it is planned to also cover the sophisticated model synchronization schemes that have been developed [14,20] in the same manner in order to prove their correctness, define required constraints, and maybe also identify further potential for optimization.

**Acknowledgments** We would like to thank dSPACE for the permission to use the TGG model transformation case study for this research and Johann Schmidt for his work on the TGG Monitor implementation.

## A Proofs

Theorem 2

*Proof* In case that no transformation of $S$ exists such that all elements can be translated, $FT_{\mathrm{CON}}(S)$ is empty.

Suppose that $FT_{\mathrm{CON}}(S)$ is not empty and that $SCT$ belongs to $FT_{\mathrm{CON}}(S)$. Then, we have that $B_{\mathrm{init}}S \rightarrow_{a^{\mathrm{BF}}} B_A SC_A T_A \rightarrow^*_{\mathrm{CON}} BSCT$ such that $Trans^F(BSCT) = S$. It follows that $SCT$ is the only element belonging to $FT_{\mathrm{CON}}(S)$ because of the following argumentation: Since $S \in \mathscr{L}(S_T^A)$, the match of the forward axiom rule $a^{\mathrm{BF}}$ is uniquely fixed (domain restriction criterion). Because of the termination criterion, it holds that each bookkeeping forward rule deletes at least one bookkeeping edge to a source element in $S$. Because bookkeeping forward rules are not producing any source elements and $S$ is finite, this means that applying bookkeeping forward rules as long as possible always terminates. The attribute criterion ensures that each rule application ends up with a unique transformation result, since attribute values of created elements are uniquely determined. Moreover, it follows from the Critical Pair Lemma in [7,29,30], the conflict-freeness criterion and attribute criterion (attribute values of preserved elements remain unchanged avoiding attribute conflicts) that $CON$ is locally confluent. In particular, if there are no critical pairs, ignoring pairs with same rules and same matches, then we can conclude that for each pair of transformations $H_1 \overset{r_1,m_1}{\Leftarrow} G \overset{r_2,m_2}{\Rightarrow} H_2$ either $H_1 \cong H_2$ or there exist transformations $H_1 \overset{r_2,m_1'}{\Rightarrow} X \overset{r_1,m_2'}{\Leftarrow} H_2$.

Together with termination this means that $\rightarrow^*_{\mathrm{CON}}$ is confluent and thus, the application of rules in $\rightarrow^*_{\mathrm{CON}}$ as long as possible terminates with a unique result. If all elements in $S$ have been translated, no rule in $CON$ is applicable anymore, since the application of any other bookkeeping rule would need at least one non-translated element (see termination criterion). Therefore, the result $BSCT$ is a terminal state, which is unique such that $FT_{\mathrm{CON}}(S) = \{SCT\}$. □

Theorem 3

*Proof* In case that no transformation via node bookkeeping rules exists such that all nodes of $S$ can be translated, $FT_{\mathrm{IMP}}(S)$ is empty.

Suppose that $FT_{\mathrm{IMP}}(S)$ is not empty and that $SCT$ belongs to $FT_{\mathrm{IMP}}(S)$. Then, we have that $B^N_{\mathrm{init}}S \rightarrow_{a^{\mathrm{IF}}} B^N_A SC_A T_A \rightarrow^*_{\mathrm{IMP}} B^N SCT$ such that $S_N = Trans^{FN}(B^N SCT)$. It follows that $SCT$ is the only element belonging to $FT_{\mathrm{IMP}}(S)$ because of the following argumentation. Recall that since $S \in \mathscr{L}(S_T^A)$, the way to match the forward axiom rule $a^{\mathrm{IF}}$ is uniquely fixed. The attribute criterion ensures that each rule application ends up with a unique transformation result, since attribute values of created elements are uniquely determined. Moreover, analogous to the proof of Theorem 2, it follows that the application of node bookkeeping forward rules, fulfilling the forward implementation criteria and attribute criterion, as long as possible terminates with a unique result. Thereby note that rules in $IMP$ only delete bookkeeping edges to nodes (not to edges) and therefore, we need the refined termination criterion. If all nodes in $S$ have been translated, no rule in $IMP$ is applicable anymore, since the application of any other node bookkeeping rule would need at least one non-translated node (again because of the refined termination criterion). Therefore, the result $B^N SCT$ is a terminal state, which is unique such that $FT_{\mathrm{IMP}}(S) = \{SCT\}$. □

Theorem 4

*Proof* Because each rule application via $r^{\mathrm{BF}}$ doing bookkeeping on nodes and edges implies a rule application via $r^{\mathrm{IF}}$, where bookkeeping on edges is disregarded, we can conclude that $FT_{\mathrm{CON}}(S) \subseteq FT_{\mathrm{IMP}}(S)$.

Moreover, we can prove that $FT_{\mathrm{CON}}(S) \supseteq FT_{\mathrm{IMP}}(S)$. Suppose that $SCT$ belongs to $FT_{\mathrm{IMP}}(S)$ and therefore, $B^N SCT$ exists such that $S_N = Trans^{FN}(B^N SCT)$. We know by assumption that $S$ is valid and therefore, it follows that there exists $SC_*T_* \in \mathscr{L}(TGG)$ and consequently, $SC_*T_* \in FT_{\mathrm{TGG}}(S)$. Then, it follows from completeness of $FT_{\mathrm{CON}}$ that $SC_*T_* \in FT_{\mathrm{CON}}(S)$. Therefore, there exists $B_*SC_*T_*$ such that $Trans^F(B_*SC_*T_*) = S$. Since $FT_{\mathrm{CON}}(S) \subseteq FT_{\mathrm{IMP}}(S)$, it follows that $SC_*T_* \in FT_{\mathrm{IMP}}(S)$ with $Trans^{FN}(B^N_* SC_*T_*) = S_N$. Because of determinism of $IMP$, it follows that $SCT = SC_*T_*$. Therefore, it follows that there exists $B_*SC_*T_* = B_*SCT$ such that $S = Trans^F(B_*SC_*T_*) = Trans^F(B_*SCT)$. Consequently, $SCT$ belongs to $FT_{\mathrm{CON}}(S)$.

Concluding, $FT_{\mathrm{IMP}}(S) = FT_{\mathrm{CON}}(S)$ and because of conformance of $FT_{\mathrm{CON}}$ with $FT_{\mathrm{TGG}}$ also $FT_{\mathrm{IMP}}(S) = FT_{\mathrm{TGG}}(S)$. Since $S$ is valid, $FT_{\mathrm{TGG}}(S)$ is not empty and therefore $FT_{\mathrm{IMP}}(S) = \{FT_{\mathrm{IMPd}}(S)\}$. □

Lemma 3

*Proof* It follows straightforwardly from Definition 6 that we can restrict a sequence keeping track of active correspondence nodes and rules to a corresponding sequence without keeping track of active correspondence nodes and rules.

For the opposite direction, we need to show that $\rightsquigarrow_{\mathrm{IMP}}$ as given in Definition 6 can manipulate the active correspondence nodes and rules in such a way that the transitions in $B^N_{\mathrm{init}} S \rightarrow_{a^{\mathrm{IF}}} B^N_a SC_a T_a \rightarrow^+_{\mathrm{IMP}} B^N_i SC_i T_i$ take place correspondingly via $\rightsquigarrow_{a^{\mathrm{IF}}}$ and $\rightsquigarrow_{\mathrm{IMP}}$ by arguing that no matches can be "missed". First, it follows straightforwardly from Definition 6 that $(B^N_{\mathrm{init}} S, \emptyset, \perp, \emptyset) \rightsquigarrow_{a^{\mathrm{IF}}} (B^N_a SC_a T_a, P_a, \perp, \emptyset)$.

Further on, we can follow from the forward efficiency criteria (a-f) that each rule application via the axiom rule $a^{\mathrm{IF}}$ or some rule $r^{\mathrm{IF}}$ in $IMP$ may only create new matches for some rule in $IMP$ that include the correspondence node $c$ just created by $a^{\mathrm{IF}}$ or $r^{\mathrm{IF}}$, respectively. This is because, assume that a new match not including the correspondence node $c$ would arise. Then, $a^{\mathrm{IF}}$ or $r^{\mathrm{IF}}$ must have created some elements other than $c$, leading to a new match for some rule in $IMP$ not including $c$. This element can not be a correspondence-source or correspondence-target edge, since then $c$ would be included in the match because of criterion (f). Moreover, it can not be a target node, since it has been created with an edge from $c$ because of criterion (d) and each target node must be matched together with an edge from exactly one correspondence node because of criterion (b). Thus, either it matches $c$ or it does not match at all. It cannot be a target edge either, because at least one of its ends is a created target node because of criterion (e). This target node was created with an edge from $c$ because of criterion (d) which must be matched together with an edge from exactly one correspondence node because of criterion (b). Thus, either it matches $c$ or it does not match at all. This is a contradiction with our assumption that the new match would not include the correspondence node $c$. Concluding, it is correct that we only need to search for new matches including some previously created correspondence node $c$ (see update actions on $P$ in (1) and (3) of Definition 6). Thereby, it is enough to check only those rules for new matches that contain in their LHS a correspondence node of the type of $c$ because of criterion (a) and since otherwise this rule would not be able to match $c$ anyway (see construction of $U$ in (2) and (4), rule application via rule in $U$ as given in (3) of Definition 6).

Note that, in general, a new match for a rule $r^{\mathrm{IF}}$ can also arise if another rule $r'^{\mathrm{IF}}$ deletes something forbidden by $r^{\mathrm{IF}}$. This might happen if bookkeeping edges are deleted because the corresponding elements were translated. Consequently, another rule might be able to match these translated elements. In this case, it would forbid bookkeeping edges to these elements and we have found a trigger dependency of the type delete-forbid. However, since in this case rule $r'^{\mathrm{IF}}$ would create exactly one correspondence node and $r^{\mathrm{IF}}$ would expect to use it. This means that we have taken into account this new match creation already above, because delete-forbid dependencies coincide in this sense with produce-use-dependencies.

Since each active correspondence node $c$ that was created by some previous transformation step may be deleted from the active correspondence set as soon as all matches at that time have been found (see (5) in Definition 6), we need to argue that anyhow no matches can be missed. Suppose that after having deleted $c$, in one of the next transformation steps a match for a rule in $IMP$ including $c$ exists. Because of the above argumentation, this can only be the case if some other correspondence node $c'$ has been created that has led to a new match. This match would include $c$ and $c'$ and would be found via the active correspondence node $c'$ which is the last correspondence node created to enable this match.

Having these arguments down pat, we can argue by induction over the number of rule applications via $\rightarrow_{\mathrm{IMP}}$ that $(B^N_a SC_a T_a, P_a, \perp, \emptyset) \rightsquigarrow^+_{\mathrm{IMP}} (B^N_i SC_i T_i, P, c, U)$ if $B^N_a SC_a T_a \rightarrow^+_{\mathrm{IMP}} B^N_i SC_i T_i$: First we prove that $(B^N_a SC_a T_a, P_a, \perp, \emptyset) \rightsquigarrow^+_{\mathrm{IMP}} (B^N_1 SC_1 T_1, P_1, c_1, U_1)$ if $B^N_a SC_a T_a \rightarrow_{\mathrm{IMP}} B^N_1 SC_1 T_1$ via some rule $r^{\mathrm{IF}}_1$. Since rule $r^{\mathrm{IF}}_1$ is applicable to $B^N_a SC_a T_a$ the axiom rule $a^{\mathrm{IF}}$ must have created a correspondence node $c_1$ enabling this application. Therefore using (2) and (3) in Definition 6, we obtain $(B^N_a SC_a T_a, P_a, \perp, \emptyset) \rightsquigarrow_{\mathrm{IMP}} (B^N_a SC_a T_a, P_a, c_1, U_1) \rightsquigarrow_{\mathrm{IMP}} (B^N_1 SC_1 T_1, P_1, c_1, U_1)$, where $r^{\mathrm{IF}}_1$ belongs to $U_1$ and $P_1$ consists of $P_a$ together with the correspondence nodes created in this last step.

Suppose that $(B^N_a SC_a T_a, P_a, \perp, \emptyset) \rightsquigarrow^+_{\mathrm{IMP}} (B^N S_{i-1} C T_{i-1} \; i-1, P', c', U')$ if $B^N_a SC_a T_a \rightarrow^{i-1}_{\mathrm{IMP}} B^N_{i-1} SC_{i-1} T_{i-1}$ where after applying (3) in Definition 6 to obtain $(B^N_{i-1} SC_{i-1} T_{i-1}, P', c', U')$ no updates on the active correspondence node (set) and active rule set have took place. We prove that $(B^N_a SC_a T_a, P_a, \perp, \emptyset) \rightsquigarrow^+_{\mathrm{IMP}} (B^N_{i-1} SC_{i-1} T_{i-1}, P', c', U') \rightsquigarrow^+_{\mathrm{IMP}} (B^N_i SC_i T_i, P, c, U)$ if $B^N_a SC_a T_a \rightarrow^{i-1}_{\mathrm{IMP}} B^N_{i-1} SC_{i-1} T_{i-1} \rightarrow_{\mathrm{IMP}} B^N_i SC_i T_i$. We know that the $i-1$th rule application via some rule $r'^{\mathrm{IF}}$ created $B^N_{i-1} SC_{i-1} T_{i-1}$ by matching correspondence node $c'$. Thereby, $r'^{\mathrm{IF}}$ belongs to $U'$ and $c'$ belongs to $P'$. Moreover, we know that some rule $r^{\mathrm{IF}}$ is applicable to $B^N_{i-1} SC_{i-1} T_{i-1}$. Now we have the following cases: Either $r^{\mathrm{IF}}$ matches $c'$ or it does not match $c'$. In the first case, we can either apply directly (3) of Definition 6 obtaining $(B^N_i SC_i T_i, P, c, U)$. Otherwise, we need to switch the active correspondence node by (4) of Definition 6 obtaining $(B^N_{i-1} SC_{i-1} T_{i-1}, P', c, U)$, since the application of $r^{\mathrm{IF}}$ must have been enabled by some correspondence node $c$ contained in $P'$ and different from $c'$. Then, we can apply rule $r^{\mathrm{IF}}$ via (3) of Definition 6 obtaining $(B^N_i SC_i T_i, P, c, U)$.

□

Theorem 5

*Proof* Let $S_N$ be the set of all nodes in $S$. We need to show that $(B^N_{\mathrm{init}} S, \emptyset, \perp, \emptyset) \rightsquigarrow_{a^{\mathrm{IF}}} (B^N_a SC_a T_a, P_a, \perp, \emptyset) \rightsquigarrow^*_{\mathrm{IMP}} (B^N_i SC_i T_i, \emptyset, \perp, \emptyset)$ where $S_N = Trans^{FN}(B^N_i SC_i T_i)$ iff

$B^N_{\text{init}} S \rightsquigarrow_{a^{\text{IF}}} B^N_a SC_a T_a \rightsquigarrow^*_{\text{IMP}} B^N_i SC_i T_i$ where $S_N = Trans^{FN}(B^N_i SC_i T_i)$. In case that no rule of $IMP$ is applied this follows straightforwardly from Definition 6. In case that at least one rule of $IMP$ is applied we use Lemma 3. Then, we can follow that $(B^N_{\text{init}} S, \emptyset, \perp, \emptyset) \rightsquigarrow_{a^{\text{IF}}} (B^N_a SC_a T_a, P_a, \perp, \emptyset) \rightsquigarrow^+_{\text{IMP}} (B^N_i SC_i T_i, P, c, U)$ iff $B^N_{\text{init}} S \rightarrow_{a^{\text{IF}}} B^N_a SC_a T_a \rightarrow^+_{\text{IMP}} B^N_i SC_i T_i$. It follows from $S_N = Trans^{FN}(B^N_i SC_i T_i)$ and Definition 6 that $(B^N_i SC_i T_i, P, c, U) \rightsquigarrow^+_{\text{IMP}} (B^N_i SC_i T_i, \emptyset, \perp, \emptyset)$. This is because each source node has been translated and no rule in $IMP$ is applicable anymore because it would need at least one non-translated element (termination criterion). Therefore, the rules (2) and (5) for updating the active correspondence node and active rule set, and adapting the active correspondence set as given in Definition 6 can be applied up until both sets are empty and the active correspondence node is undefined. □

## References


1. Amelunxen, C., Klar, F., Königs, A., Rötschke, T., Schürr, A.: Metamodel-based tool integration with MOFLON. In: ICSE '08: Proceedings of the 30th ICSE, pp. 807–810. ACM, New York (2008)
2. Bapodra, M., Heckel, R.: From graph transformations to differential equations. In: Graph and Model Transformation. EC-EASST (2010)
3. Becker, B., Giese, H.: Incremental verification of inductive invariants for the run-time evolution of self-adaptive software-intensive systems. In: Proceedings of 23rd IEEE/ACM International Conference on Automated Software Engineering—Workshops, pp. 33–40. IEEE Computer Society Press (2008)
4. Biermann, E., Ermel, C., Taentzer, G.: Precise semantics of EMF model transformations by graph transformation. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds) Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems, Toulouse, France, September 28–October 3, 2008. LNCS, pp. 53–67 (2008)
5. Burmester, S., Giese, H., Niere, J., Tichy, M., Wadsack, J.P., Wagner, R., Wendehals, L., Zündorf, A.: Tool integration at the meta-model level within the fujaba tool suite. Int. J. Softw. Tools Technol. Transf. (STTT) **6**(3), 203–218 (2004)
6. de Lara, J., Vangheluwe, H.: AToM3 as a Meta-CASE Environment (DFD to SC). In: Proceedings of the 4th International Conference on Enterprise Information Systems (2002)
7. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of algebraic graph transformation. EATCS Monographs in Theoretical Computer Science. Springer, Berlin (2006)
8. Ehrig, H., Ermel, C., Hermann, F.: On the relationship of model transformations based on triple and plain graph grammars. In: GRaMoT'08: Proceedings of the third International Workshop on Graph and Model Transformations, pp. 9–16. ACM, New York (2008)
9. Ehrig, H., Ermel, C., Hermann, F., Prange, U.: On-the-fly construction, correctness and completeness of model transformations based on triple graph grammars. In: Proceedings Models 2009 Model Driven Engineering Languages and Systems. LNCS, vol. 5795/2009, pp. 241–255. Springer Berlin (2009)
10. Ehrig, H., Prange, U.: Formal analysis of model transformations based on triple graph rules with kernels. In: ICGT'08: Proceedings of the 4th International Conference on Graph Transformation, pp. 178–193. Springer, Berlin (2008)
11. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story Diagrams: a new graph rewrite language based on the unified modeling language and java. In: TAGT'98: Selected papers from the 6th International Workshop on Theory and Application of Graph Transformations. Lecture Notes in Computer Science (LNCS), vol. 1764/2000, pp. 296–309. Springer, London, 16–20 November 2000
12. Giese, H., Glesner, S., Leitner, J., Schäfer, W., Wagner, R.: Towards verified model transformations. In: Hearnden, D., Süß, J., Baudry, B., Rapin, N. (eds.) Proceedings of the 3rd International Workshop on Model Development, Validation and Verification (MoDeVa), Genova, Italy, pp. 78–93. Le Commissariat à l'Energie Atomique—CEA, (2006)
13. Giese, H., Hildebrandt, S.: Incremental model synchronization for multiple updates. In: Proceedings of the 3rd International Workshop on Graph and Model Transformations, May 12, 2008, Leipzig, Germany, volume Proceedings of GraMoT'08, May 12, 2008, Leipzig, Germany. ACM Press (2008)
14. Giese, H., Hildebrandt, S.: Efficient model synchronization of large-scale models. Technical Report 28, Hasso Plattner Institute at the University of Potsdam (2009)
15. Giese, H., Hildebrandt, S., Lambers, L.: Toward bridging the gap between formal semantics and implementation of triple graph grammars. Technical Report 37, Hasso Plattner Institute at the University of Potsdam (2010)
16. Giese, H., Hildebrandt, S., Lambers, L.: Toward bridging the gap between formal semantics and implementation of triple graph grammars. In: Proceedings of MoDeVVa 2010, Models Workshop on Model-Driven Engineering, Verification and Validation, pp. 19–24. IEEE Computer Society (2010)
17. Giese, H., Hildebrandt, S., Neumann, S., Wätzoldt, S.: Industrial Case Study on the Integration of SysML and AUTOSAR with Triple Graph Grammars. Technical Report 57, Hasso Plattner Institute at the University of Potsdam (2012)
18. Giese, H., Hildebrandt, S., Seibel, A.: improved flexibility and scalability by interpreting story diagrams. In: Magaria, T., Padberg, J., Taentzer, G. (eds.) Proceedings of the Eighth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2009), vol. 18. Electronic Communications of the EASST (2009)
19. Giese, H., Neumann, S., Hildebrandt, S.: Model synchronization at work: keeping SysML and AUTOSAR models consistent. In: Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., Westfechtel, B. (eds.) Graph Transformations and Model Driven Enginering—Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday. Lecture Notes in Computer Science, vol. 5765, pp. 555–579. Springer, Berlin (2010)
20. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Softw. Syst. Model. (SoSyM) **8**(1) (2009)
21. Greenyer, J., Kindler, E.: Comparing relational model transformation technologies: implementing query/view/transformation with triple graph grammars. Softw. Syst. Model. **9**(1), 21–46 (2010)
22. Hermann, F., Ehrig, H., Golas, U., Orejas, F.: Efficient analysis and execution of correct and complete model transformations based on triple graph grammars. In: Proceedings of the First International Workshop on Model-Driven Interoperability, MDI '10, pp. 22–31. ACM, New York (2010)
23. Hermann, F., Ehrig, H., Orejas, F., Golas, U.: Formal analysis of functional behaviour for model transformations based on triple graph grammars. In: Ehrig, H., Rensink, A., Rozenberg, G., Schrr, A. (eds.) Graph Transformations. Lecture Notes in Computer Science, vol. 6372, pp. 155–170. Springer Berlin (2010)
24. ITU International Telecommunication Union. ITU-T Recommendation Z.100: Specification and Description Language (SDL) (2002)

25. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Satellite Events at the MoDELS 2005 Conference. LNCS, vol. 3844, pp. 128–138. Springer, Berlin (2006)
26. Kindler, E., Rubin, V., Wagner, R.: An adaptable TGG interpreter for in-memory model transformation. In: Schürr, A., Zündorf, A. (eds.) Proceedings of the 2nd International Fujaba Days 2004, Darmstadt, Germany, vol. tr-ri-04-253 of Technical Report, pp. 35–38. University of Paderborn (2004)
27. Klar, F., Lauder, M., Königs, A., Schürr, A.: Extended triple graph grammars with efficient and compatible graph translators. In: Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., Westfechtel, B. (eds.) Graph Transformations and Model-Driven Engineering. Lecture Notes in Computer Science, vol. 5765, pp. 141–174. Springer, Berlin (2010)
28. Küster, J.: Definition and validation of model transformations. Softw. Syst. Model. **5**(3), 233–259 (2006)
29. Lambers, L., Ehrig, H., Prange, U., Orejas, F.: Embedding and confluence of graph transformations with negative application conditions. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) Proceedings of International Conference on Graph Transformation (ICGT'08), LNCS, vol. 5214, pp. 162–177. Springer, Heidelberg (2008)
30. Lambers, L.: Certifying Rule-based models using graph transformation. PhD thesis, Technische Universität Berlin (2010)
31. Lambers, L., Hildebrandt, S., Giese, H., Orejas, F.: Attribute handling for bidirectional model transformations: the triple graph grammar case. In: Proceedings of First International Workshop on Bidirectional Transformations. EC-EASST, (2012, to appear)
32. Mens, T., Van Der Straeten, R., D'Hondt, M.: Detecting and resolving model inconsistencies using transformation dependency analysis. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) Model Driven Engineering Languages and Systems. Lecture Notes in Computer Science, vol. 4199, pp. 200–214. Springer, Berlin (2006)
33. Object Management Group. MOF 2.0 QVT 1.0 Specification (2008)
34. Orejas, F., Guerra, E., de Lara, J., Ehrig, H.: Correctness, completeness and termination of pattern-based model-to-model transformation. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO. LNCS, vol. 5728, pp. 383–397. Springer, Berlin (2009)
35. Orejas, F., Lambers, L.: Delaying constraint solving in symbolic graph transformation. In: Proceedings of International Conference on Graph Transformation (ICGT'10), vol. 6372, pp. 43–58 (2010)
36. Schäfer, W., Wagner, R., Gausemeier, J., Eckes, R.: An Engineer's Workstation to support integrated development of flexible production control systems. In: Ehrig, H., Damm, W., Desel, J., Grosse-Rhode, M., Reif, W., Schnieder, E., Westkämper, E. (eds.) Integration of Software Specification Techniques for Applications in Engineering. Lecture Notes in Computer Science (LNCS), vol. 3147, pp. 48–68. Springer, Berlin (2004)
37. Schürr, A.: Specification of graph translators with triple graph grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science. LNCS, vol. 903, pp. 151–163. Spinger, Herrsching (1994)
38. Schürr, A., Klar, F.: 15 Years of triple graph grammars: research challenges, new contributions, open problems. In: 4th International Conference of Graph Transformation, ICGT 2008, Leicester, United Kingdom, September 7–13, 2008. LNCS, vol. 5214, pp. 411–425. Springer, Berlin (2008)
39. Sendall, S., Kozaczynski, W.: Model transformation: the heart and soul of model-driven software development. IEEE Software, pp. 42–45 (2003)
40. Stevens, P.: Bidirectional model transformations in QVT: semantic issues and open questions. Softw. Syst. Model. **9**(1), 7–20 (2010)
41. Taentzer, G., Ermel, C., Rudolf, M.: The AGG-approach: language and tool environment. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) Handbook of graph grammars and computing by graph transformation, vol. 2: Applications, Languages and Tools, pp. 551–603. World Scientific, Singapore (1999)
42. Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Incremental model synchronization for efficient run-time monitoring. In: Ghosh, S. (ed.) Models in Software Engineering, Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4–9, 2009, Reports and Revised Selected Papers. LNCS, vol. 6002, pp. 124–139. Springer, Berlin (2010)

## Author Biographies

**Holger Giese** is a full professor at the Hasso Plattner Institute for Software Systems Engineering at the University of Potsdam. Beforehand he was assistant professor for object-oriented specification of distributed systems in the Software Engineering Group of the University of Paderborn since 2001. He studied technical computer science at the University of Siegen and received his engineering degree in October 1995. He received a doctorate in Computer Science at the Institute of Computer Science at the University of Münster in February 2001. His research focus is the model-driven development of software-intensive systems covering the specification of dynamic and flexible systems by services, collaborations, patterns, and components, approaches to analyze and formally verify such models, and approaches for model synthesis. The main focus are systems that are typically distributed systems, embedded real-time systems as well as systems that are capable to adapt and coordinate themselves. Furthermore, he does research on model transformation, concepts for generating source code for structure and behavior, and the general problem of model integration during the process of model-driven development. He is member of the Association for Computing Machinery, the IEEE Computer Society, and the German Informatics Society.

**Stephan Hildebrandt** studied Software Systems Engineering at the Hasso Plattner Institute at the University of Potsdam. He received his Bachelor degree in 2006 and his Master degree in 2008. Currently, he is working as a research assistant in the System Analysis and Modeling Group. His main area of research is model-driven engineering, in particular model transformation and model synchronization.

**Leen Lambers** is a postdoctoral researcher working on the DFG-project CorMoranT (Correct Model Transformations) in the group of Prof. Holger Giese at the Hasso Plattner Institute for Software Systems Engineering at the University of Potsdam since January 2010. She received her PhD for her dissertation "Certifying Rule-Based Models using Graph Transformation" at the Technical University of Berlin in December 2009, where she has been a scientific assistant in the group of Prof. Hartmut Ehrig from October 2003 until December 2009. Her main research focus is formal modeling and analysis in software engineering, in particular, using graph transformation. She has spent research periods in the group of Prof. Mauro Pezzé at the University of Milano Bicocca and in the group of Prof. Fernando Orejas at the Technical University of Catalonia. She is currently the PC co-chair of the 11th International Workshop on Graph Transformation and Visual Modeling Techniques. She is serving as a PC member for the 5th International Conference on Model Transformation and the 6th International Conference on Graph Transformation.