Electronic Communications of the EASST
Volume 54 (2012)

Proceedings of the 7th International Workshop on Graph Based Tools (GraBaTs 2012)

Visual Modeling and Analysis of EMF Model Transformations Based on Triple Graph Grammars

Claudia Ermel, Frank Hermann, Jürgen Gall and Daniel Binanzer

12 pages

Guest Editors: Christian Krause, Bernhard Westfechtel
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Visual Modeling and Analysis of EMF Model Transformations Based on Triple Graph Grammars

Claudia Ermel[1], **Frank Hermann**[2], **Jürgen Gall**[1] **and Daniel Binanzer**[1]

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
claudia.ermel@tu-berlin.de, juergengall@gmx.de, daniel.binanzer@freenet.de
[2] Interdisciplinary Center for Security, Reliability and Trust, Université du Luxembourg
frank.hermann@uni.lu

Abstract: The tool HENSHIN is an Eclipse plug-in supporting visual modeling and execution of rule-based EMF model transformations. This paper describes the recent extensions of HENSHIN by a visual editor for triple graph grammars (TGGs). The visual editor (called HENSHINTGG) supports a compact visualization of triple rules in an integrated editor panel. Internally, triple graph rules are represented as HENSHIN rules and can be simulated using the HENSHIN EMF model transformation engine. Our extension supports the automatic generation of forward translation rules for transforming source into target models. A converter from HENSHIN TGG rules to the graph transformation analysis tool AGG allows a systematic check for conflicts of forward translation rules in AGG based on critical pair analysis.

Keywords: EMF, model transformation tool, triple graph grammar, Henshin

1 Introduction

Model transformations play an important role in model driven development. In graph transformation based approaches and tools, rules express basic transformation steps. In particular, triple graph grammars (TGGs) [Sch94] are a formal technique to specify and reason about bidirectional model transformations. Using graph triples, the relations of source and target models is specified declaratively, by mapping the elements of a correspondence model to corresponding elements of the source and target model. A TGG describes how consistent graph triples are derived synchronously by applying *triple rules*. From such a TGG, so-called *operational rules* can be derived automatically to perform unidirectional forward or backward model transformations. TGGs have shown to be a suitable formal basis for model transformations and to reason about properties such as correctness, completeness, or functional behaviour [HEGO10, EEHP09].

The paper presents the new, visual TGG modelling and analysis environment HENSHINTGG that makes use of the existing formally founded EMF model transformation engine HENSHIN. In contrast to existing TGG implementations [GHL12, ALPS11, BGH+05], HENSHINTGG does not only specify and perform EMF model transformations by TGGs but generates *forward translation rules* (synthesized from forward and source rules) according to [HEGO10] and offers a converter to translate forward translation rules to the graph transformation analyzer AGG [AGG12] in order to benefit from AGG's critical pair analysis for conflict detection. Fig. 1 shows an overview of the overall workflow using the main tool features of HENSHINTGG.

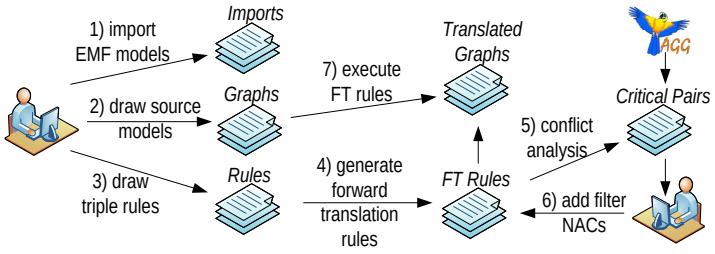

Figure 1: Workflow overview of using HENSHINTGG for EMF model transformation

HENSHIN is an Eclipse plug-in supporting visual modeling and execution of EMF model transformations, i.e., transformations of models conforming to a meta-model given in the EMF `Ecore` format.[1] The transformation approach is based on algebraic graph transformation according to the double pushout (DPO) approach [EEPT06] which are lifted to EMF model transformation by also taking containment relations in meta-models into account [BET12, ABJ+10].

Structure of the Paper: We present our visual TGG editor in Sec. 2 and describe the generation of forward translation rules based on [HEGO10] in Sec. 3. An example for a conflict analysis of forward translation rules converted to AGG based on critical pairs is presented in Sec. 4, while Sec. 5 explains the automatic EMF model translation. In Sec. 6, we compare related approaches and tools to our tool and conclude the paper with an outlook to future work.

2 The Visual TGG Editor

Using triple graph grammars [Sch94], models are defined as pairs of source and target graphs, which are connected via a correspondence graph together with its embeddings into these graphs. In this section, we review main constructions and results of model transformations based on TGGs and introduce our visual TGG editor HENSHINTGG.

A triple graph $G = (G_S \xleftarrow{s_G} G_C \xrightarrow{t_G} G_T)$ consists of three graphs G_S, G_C, and G_T, called source, correspondence, and target graphs, together with two graph morphisms $s_G : G_C \to G_S$ and $t_G : G_C \to G_T$. A triple graph morphism $m = (m_S, m_C, m_T) : G \to H$ between triple graphs G and H consists of three graph morphisms $m_S : G_S \to H_S$, $m_C : G_C \to H_C$ and $m_T : G_T \to H_T$. A typed triple graph G is typed over a triple type graph TG by a triple graph morphism $type_G : G \to TG$.

Example 1 (Triple Type Graph) *Fig. 2 shows the type graph TG of the triple graph grammar TGG for our example model transformation from class diagrams to database models. The source component TG_S defines the structure of class diagrams while in the target component the structure of relational database models is specified. Classes correspond to tables, attributes to columns, and associations to foreign keys. Morphisms starting at a correspondence part are indicated by dashed arrows.*

The HENSHINTGG editor uses EMF models as type graphs and EMF instance models con-

[1] Note that we use the terms *meta-model* and *model* in this paper, which are called *EMF model* and *model instance* in the EMF documentation, respectively.

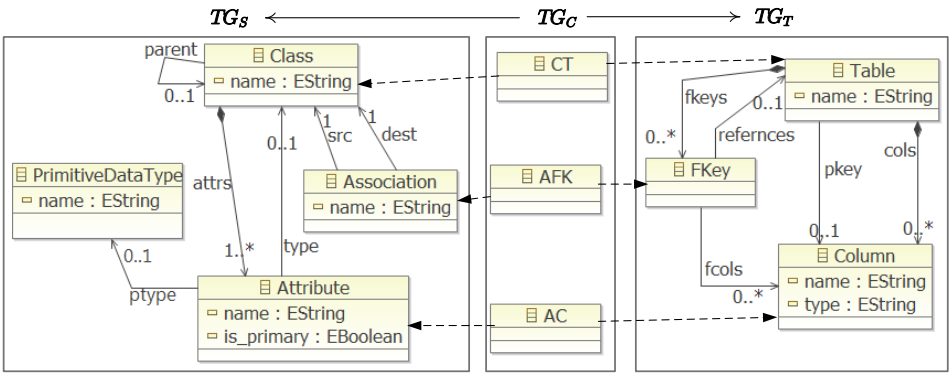

Figure 2: Triple type graph for *CD2RDBM* as triple EMF model

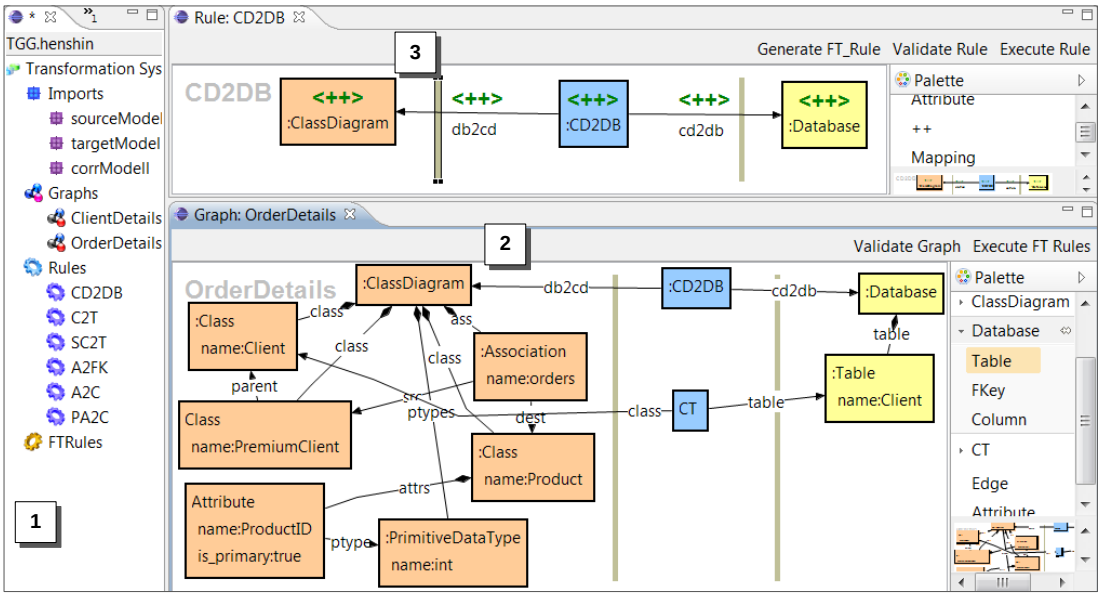

Figure 3: Graphical user interface of the visual TGG editor

forming to the respective EMF models as typed (attributed) graphs[2]. The three EMF models in Fig. 2 have been edited outside the visual TGG editor using the graphical GMF editor for EMF, but any other EMF model editor or generator can be used as well. The morphisms are implemented as references between the types of the three different EMF models. EMF models are imported into the visual TGG editor which enables the use of previously produced EMF models. The names of the three imported EMF models *source, correspondence* and *target* that comprise the triple type graph, are shown in the top compartment *Imports* of the tree view 1 in Fig. 3.

Once a triple type graph is available (i.e., the three EMF models have been imported), triple graphs typed over this type graph may be edited, e.g. for modifying inputs and intermediate

[2] For more details on the formal correspondence of typed attributed graphs and EMF models see [BET12].

states when testing model transformations[3]. The visual TGG editor supports editing of triple graph nodes and edges by offering the available types in the palette of the triple graph panel 2. Only triple graphs conforming to the triple type graph can be created. Moreover, only source triple graph elements (colored in red) can be created and modified in the left-hand part of the editor, correspondence graph elements (blue) in the center, and target graph elements (yellow) in the right part. The separators between the different triple panels can be moved using the mouse. Morphisms from correspondence to source and target elements are drawn as edges across the separators. Fig. 3 shows a sample triple graph *OrderDetails* containing a complete source part (the class diagram) but incomplete corresponding target and correspondence graphs.

$$\begin{array}{ccccc} L = (L_S & \xleftarrow{s_L} & L_C & \xrightarrow{t_L} & L_T) \\ {\scriptstyle tr}\downarrow & {\scriptstyle tr_S}\downarrow & & {\scriptstyle tr_C}\downarrow & {\scriptstyle tr_T}\downarrow \\ R = (R_S & \xleftarrow[s_R]{} & R_C & \xrightarrow[t_R]{} & R_T) \end{array} \qquad \begin{array}{ccc} L & \xrightarrow{tr} & R \\ {\scriptstyle m}\downarrow & (PO) & \downarrow{\scriptstyle n} \\ G & \xrightarrow[t]{} & H \end{array}$$

Figure 4: Triple rule (left) and triple transformation step (right)

Triple graphs can be generated by applying *triple rules* to the start graph. Triple rules synchronously build up their source, target and correspondence graphs, i.e., they are non-deleting. A triple rule tr (left part of Fig. 4) is an injective triple graph morphism $tr = (tr_S, tr_C, tr_T) : L \to R$ and w.l.o.g. we assume tr to be an inclusion. Given a triple graph morphism $m : L \to G$, a triple graph transformation (TGT) step $G \overset{tr,m}{\Longrightarrow} H$ (right part of Fig. 4) from G to a triple graph H is given by a pushout of triple graphs. A grammar $TGG = (TG, S, TR)$ consists of a triple type graph TG, a triple start graph $S = \varnothing$ and a set TR of triple rules.

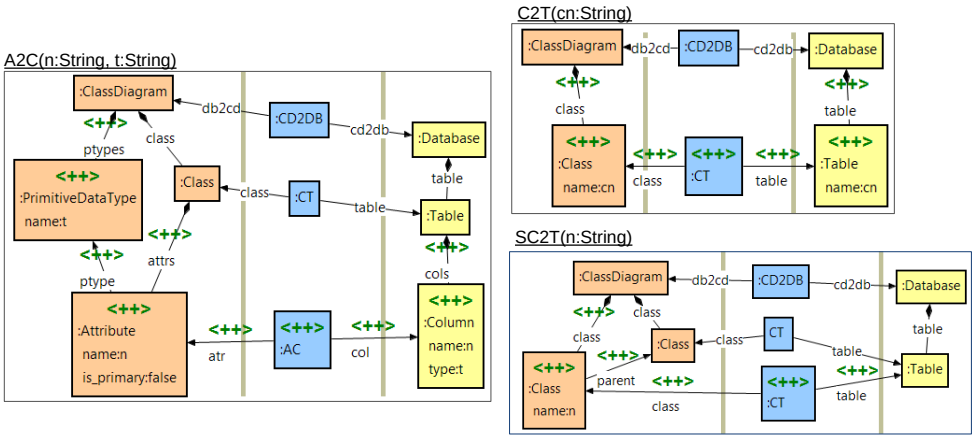

Figure 5: Some rules for the model transformation *CD2RDBM* (HENSHINTGG screenshots)

Example 2 (Triple Rules) *The triple rules shown in Fig. 5 are part of the rules of the grammar TGG for the model transformation CD2RDBM. In* HENSHINTGG*, triple rules are drawn in*

[3] Moreover, the triple graph editor can be used for resolving inconsistencies within a future extension of the tool to model synchronization.

short notation, i.e. left and right hand side of a rule are depicted in one triple graph. Elements which are created by the rule are labeled by "++". Rule CD2DB *(see* 1 *in Fig. 3) synchronously creates a class diagram together with the corresponding database. Analogously, rule* C2T *creates a class with name "n" together with the corresponding table in the relational database. A subclass is connected to the table of its superclass by rule* SC2T*. Attributes with type "t" are created together with their corresponding columns in the database component via rule* A2C*.*

The visual HENSHINTGG editor for triple rules consists of three panel parts like the visual triple graph editor (see 3 in Fig. 3). But in addition to the triple graph editor, the rule editor palette offers a green "++" to mark elements as *created* (and to unmark marked elements if necessary). Note that HENSHINTGG checks triple rules for consistency at editing time, i.e. if a node is "++"-marked, all incident edges are marked automatically, as well.

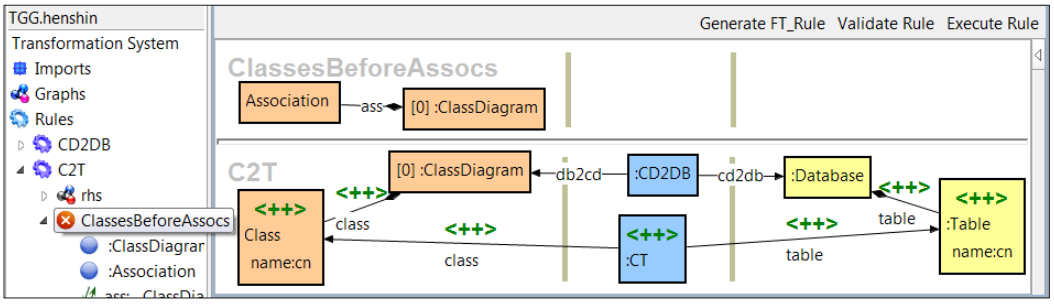

Figure 6: Triple rule *C2T* with NAC *ClassesBeforeAssocs*

HENSHINTGG supports negative application conditions for triple rules that forbid the presence of certain structures when applying a rule [EEHP09, GEH11]. A visual NAC editor can be opened via the tree view and consists of a three-panel triple graph editor again. A rule may have several NACs, the one to be shown in the visual NAC editor has to be selected in the tree view. Fig. 6 shows rule *C2T* with an additional NAC that forbids the synchronous creation of a class and a table if there are associations in the class diagram. The morphism from the rule to one of its NACs is indicated by equal numbers for mapped nodes (in Fig. 6, the *ClassDiagram* node is mapped to the NAC). Edges are mapped accordingly automatically. The rule palette entry *Mapping* supports the definition of a mapping from the triple rule to a NAC. Note that only unmarked elements (without "++") can be mapped to NAC elements, a consistency property which is also checked automatically by the editor.

A triple rule can be applied by clicking the button *Execute Rule* in the rule's tool bar (the upper right corner in Fig. 6), and selecting the graph the rule should be applied to. The result is shown in the view of the selected graph.

3 Generation of Forward Translation Rules

From each triple rule *tr*, so-called operational rules can be automatically derived [Sch94] for parsing a model of the source or target language (source and target rules) and for model transformations from source to target or backwards (forward and backward rules), as depicted in Fig. 7.

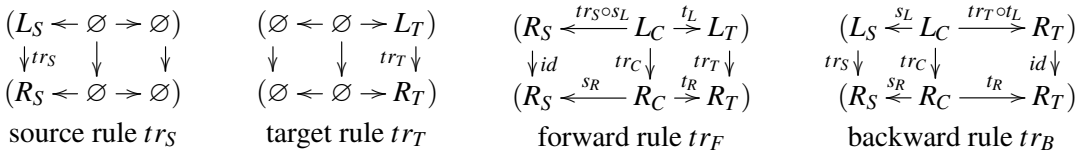

Figure 7: Derived operational rules of a TGG

According to [HEGO10], the extension of forward rules to *forward translation rules* is based on additional Boolean attributes for all elements in the source component, called *translation attributes* that control the translation process by keeping track of the elements which have been translated so far. This ensures that each element in the source graph is translated at most once.

The algorithm for constructing forward translation rules from triple rules is as follows (see [HEGO10] for its formal definition): For each triple rule tr, initialize the forward translation rule $tr_{FT} = tr_F$ by the forward rule tr_F. Add an additional Boolean attribute *isTranslated* to each source element (node, edge or attribute) of tr_{FT}. In the left-hand side of tr_{FT}, for each source element, the value of the *isTranslated* attribute is set to **false** if the element is generated by the source rule tr_S of tr, otherwise it is set to **true**. In the right-hand side of tr_{FT}, the value of all *isTranslated* attributes is set to **true**. For all source elements in NACs, the attribute *isTranslated* is set to **true** as well.

Note that in contrast to forward translation rules, pure forward rules need additional control conditions, such as the source consistency condition in [EEHP09], to ensure correct executions. In HENSHINTGG, forward translation rules are computed automatically. The translation attributes for nodes and edges [4] are kept separately as an external pointer structure in order to keep the source model unchanged. In the source graph editor panel of a forward translation rule, all elements that are still to be translated are marked by a "<tr>" tag.

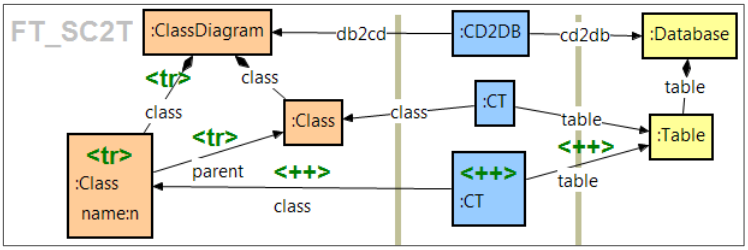

Figure 8: Forward translation rule *FT_SC2T* generated from triple rule *SC2T*

Example 3 (Forward translation rule) *Fig. 8 shows the forward translation rule* FT_SC2T *generated from triple rule* SC2T. *The* Class *node and its incident edge are marked by a "<tr>" tag as to be translated, since these model elements correspond to the model elements generated by the source rule of triple rule* SC2T.

[4] An extension to mark attributes of nodes separately is under way.

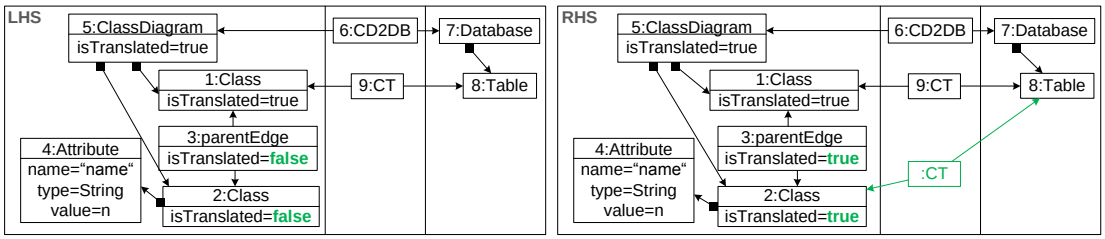

Figure 9: Rule *FT_SC2T* in abstract HENSHIN syntax

Forward translation rules can be edited in a restricted visual triple rule edtior which allows for a manual extension of additional NACs. All other rule editor operations are blocked because forward translation rules are generated automatically and should not be changed manually. Fig. 9 shows the abstract syntax of the forward translation rule *FT_SC2T* from Fig. 8, as it is represented in HENSHIN, where left-hand and right-hand sides of a rule are kept separately, with morphisms inbetween. We can see how the translation attributes of source elements are switched from **false** to **true**.

For matching, we internally keep two tables (hashmaps) "TranslatedNodes" und "TranslatedEdges" based on the IDs of the elements of an EMF instance model. These tables are constructed and updated dynamically during transformation execution. A match is valid if for each matched element we have one of the following cases: 1) its translation attribute is **true** and its ID is present in the corresponding table of translated elements, or 2) its translation attribute is **false** and its ID is not present in the corresponding table of translated elements.

4 Conflict Analysis Based on AGG

According to [HEGO10], a forward translation sequence $G_0 \overset{tr^*_{FT}}{\Longrightarrow} G_n$ is called *complete* if G_n is *completely translated*, i.e. all translation attributes of G_n are set to true. A model transformation based on forward translation rules with NACs (consisting of a source graph G_S, a target graph G_T, and a complete forward translation sequence $G_0 \overset{tr^*_{FT}}{\Longrightarrow} G_n$) is terminating if each forward translation rule changes at least one translation attribute from **false** to **true**; it is *correct* if each forward translation results in a triple graph that can be generated by triple rules, and it is *complete* if for each source graph there is a forward translation sequence that results in a triple graph that can be generated by triple rules.

However, not all terminating forward translation sequences are complete. A counter example is a forward translation rule sequence applied to the triple graph *TwoClasses* consisting of a parent class named *Client* and a subclass named *PremiumClient* connected to class *Client* by a *parent* edge (see source graph in Fig. 10).

The incomplete forward translation sequence is as follows: *FT_CD2DB*; *FT_C2T* (applied to class *PremiumClient*); *FT_C2T* (applied to class *Client*). The sequence is terminating (no forward translation rule can be applied any more), but the result after applying this sequence is a triple graph where not all translation attributes are set to **true**, i.e. not all source model elements have been translated: the *parent* edge could not be translated.

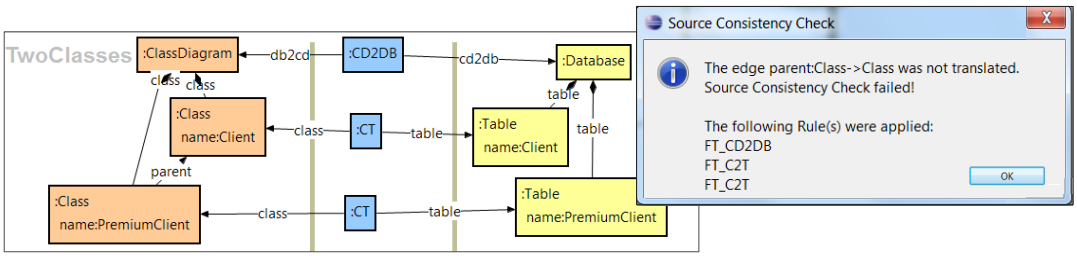

Figure 10: Incomplete forward translation sequence: *parent* edge could not be translated

In HENSHINTGG, elements that could not be translated are reported as error message in the triple graph panel showing the (partial) translation result (see Fig. 10). This allows the user to reason about possible conflicts between rule applications. The reason why the *parent* edge was not translated by the given forward translation sequence is a conflict between rule *FT_C2T* (applied to class *PremiumClient*) and *FT_SC2T* which could not be applied to class *PremiumClient* after the application of rule *FT_C2T*.

In order to ensure completeness in the general case, the execution of model transformations may require backtracking (not implemented in HENSHINTGG). However, as shown in [HEOG10], backtracking is not necessary, if the significant critical pairs between transformation rules are strictly confluent and the system is terminating, i.e., a system satisfying this condition does not have to be confluent in the general sense. HENSHINTGG implements a converter from triple rules in HENSHIN to the graph transformation analysis tool AGG, which provides a critical pair analysis engine. A critical pair is a conflict between two rules in minimal context and it is significant, if the overlapping graph can be embedded in a possible intermediate state of a model transformation sequence. In particular, it is not significant if a fragment in the source component cannot be embedded into a valid source model due to language constraints.

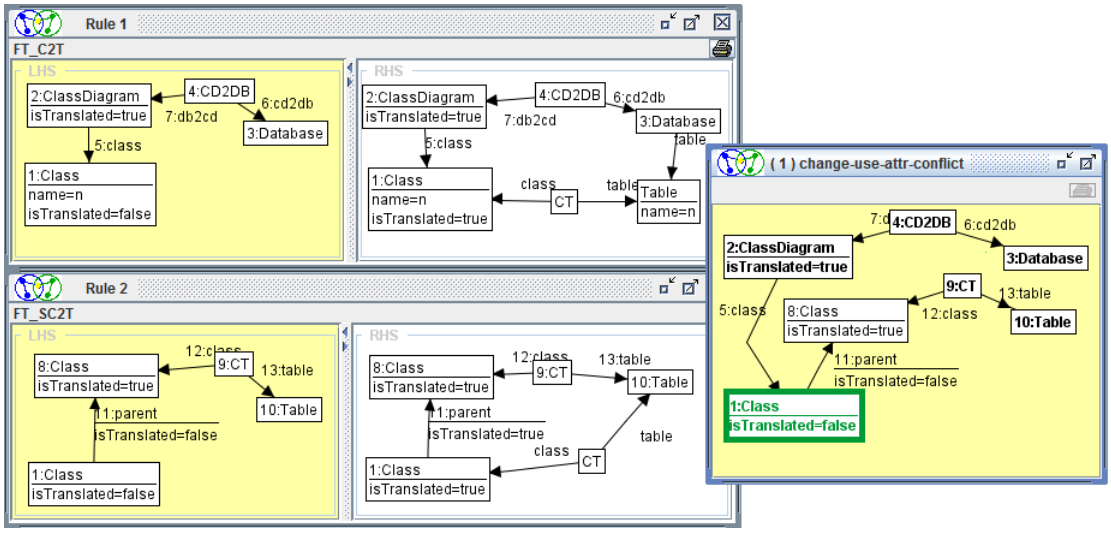

Figure 11: Critical pair between rules *FT_C2T* and *FT_SC2T* computed by AGG

Fig. 11 shows the (only) critical pair between the rules *FT_C2T* and *FT_SC2T* as depicted by the AGG critical pair analyzer. In the window to the right, the critical overlapping graph of both rules' left-hand sides is shown, and it is indicated that we have a *change-use-attr* conflict, since both rules want to access and change the *isTranslated* attribute of the subclass.

In order to avoid the conflict shown in Fig. 11, the easiest way is to add a NAC to rule *FT_C2T* that forbids its application to classes which have a parent class. According to [HEGO10], such additional conflict-avoiding NACs are called *filter NACs* and can be generated automatically. Note, however, that the generation of filter NACs is not yet supported by HENSHINTGG.

5 Performing Model Transformation in HENSHINTGG

Using a set of confluent forward-translation rules, we can be sure to always get a complete forward translation sequence, i.e., all elements are translated and the result is unique. The upper part of Fig. 12 shows rule *C2T*, now extended by a filter NAC. With this extension, the set of forward-translation rules now is confluent, since there are no critical pairs any more.

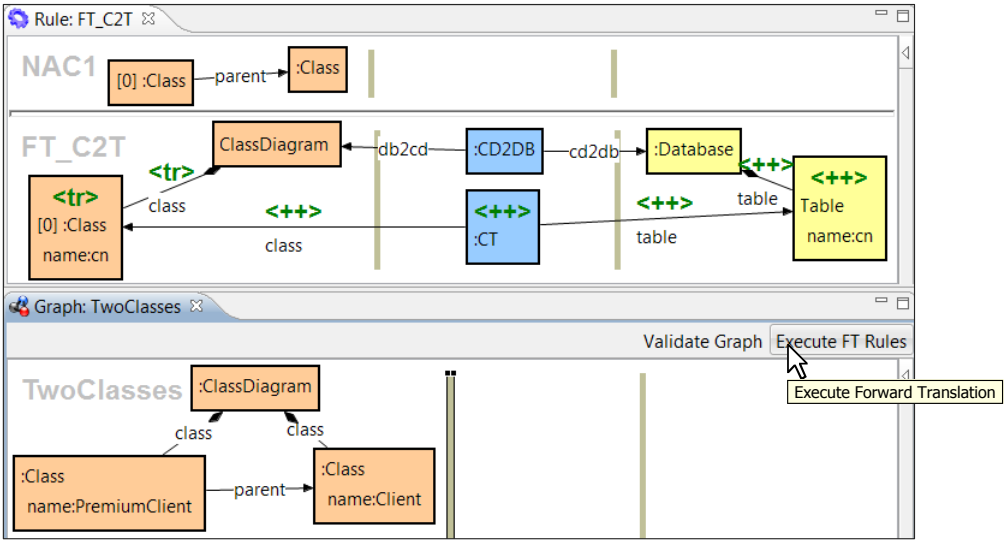

Figure 12: Rule *FT_C2T* with filter NAC (top) and input graph *TwoClasses* (bottom)

HENSHINTGG supports the automatic forward translation of a given source model by offering a button *Execute Forward Translation* in the tool bar of the EMF source model to be translated (see the bottom part of Fig. 12). Having pressed the button, the forward translation rules are executed in arbitrary order; confluence of the transformation system guarantees a unique result. The resulting target triple graph is shown in the same window as the source model since the translation is performed *in-place*. Fig. 13 shows the target triple which is the result of translating the source model in Fig. 12. In addition, the sequence of applied forward translation rules is shown to the modeller in the message window. For debugging purposes, also single forward translation rule applications can be executed, analogously as for triple rules.

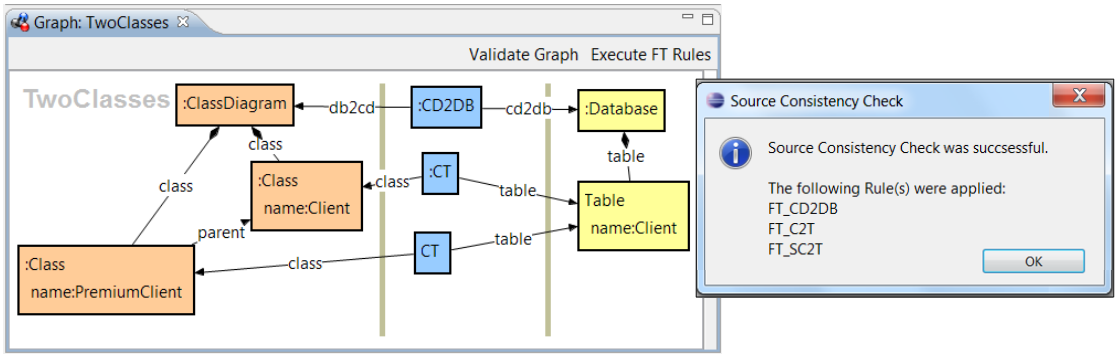

Figure 13: Result of the Forward Translation of Source Model *TwoClasses*

6 Related Work and Conclusion

General model transformation tools such as ATL [JABK08] and MOMENT2-MT [MOM12] are usually used to perform in-place model transformations and do not restrict the structure of transformation rules. Thus, they do not ensure TGG-specific properties like preservation of source models [Sch94] and syntactical correctness and completeness [EEHP09]. Moreover, the forward and backward transformations are manually specified and not generated from a single specification. While ATL and MOMENT2-MT use textual specification techniques, graph transformation tools like HENSHIN (in-place) [ABJ+10] and FUJABA [Fuj12] offer the visual specification of transformation rules, i.e., a form of visual programming interface.

In addition to HENSHINTGG, further TGG tools based on EMF are available. The TGG interpreter [GK10] provides a feature to define OCL expressions as rule conditions, while formal application conditions cannot be specified. However, the formal results concerning correctness and completeness [EEHP09] are not available for systems with OCL conditions. The TGG tools MOTE (model transformation engine) [GW09] and eMoflon [ALPS11] perform a compilation to the FUJABA tool suite [BGH+05, Fuj12] for the execution of model transformations. While eMoflon supports the specification of TGGs with negative application conditions (NACs), this is not the case for MOTE. MOTE offers certain optimization strategies concerning efficiency. Since correctness cannot be ensured for optimizations, the tool executes dynamic run-time checks to validate that a model transformation sequence was executed correctly [GHL12]. Moreover, MOTE uses a relaxed notion of correspondences for triple graphs, where correspondence nodes may link an arbitrary numer of source and target nodes [GHL12].

In order to improve efficiency of TGG tools, suitable static and dynamic conditions have been studied that allow to completely avoid backtracking. Klar et al. use a restricted class of TGGs for which they describe explicit dynamic conditions based on pre-checking contextual edges when translating a node [KLKS10]. Lauder et al. leverage these restrictions on TGGs and introduce the notion of precedence TGGs, where rules are required to form a partial order concerning the execution [LAVS12]. However, these conditions are not checked statically. Giese et al. present efficiency conditions for a restricted class of TGGs [GHL12], where, e.g., each forward rule has to translate at least one source node and may not be in conflict with another rule via a critical pair. The first condition excludes examples where the translation of a single edge or attribute is

handled separately by one rule [HEEO12], and the second condition excludes the well-studied case study on the object relational mapping [EEHP09] used in this article. The tool was extended by a prototypical export [GHL12] of so-called bookkeeping rules to AGG for conflict analysis, but it does not provide re-import and evaluation.

HENSHINTGG is based on the formal definitions for TGGs [Sch94, EEHP09, HEGO10] and supports conflict analysis via the converter to AGG. The explicit marking of edges overcomes the restriction in [GHL12] that rules are required to create at least one node. The implementation of a re-import feature for displaying and evaluating the critical pairs is work in progress. HENSHINTGG allows the user to manually use the analysis and optimizations techniques presented in [HEGO10] in order to improve efficiency. The automated generation of filter NACs [HEGO10] can be implemented as a direct extension and is future work. A further line of future work is the extension of the tool to support also backward transformations, model synchronization [HEEO12], critical pair analysis directly in the HENSHIN GUI, and the import of EMF instances. Moreover, we plan to use HENSHINTGG within an industrial case study for software translation for satellite systems and in further case studies to evaluate the results concerning correctness and efficiency.

References

[ABJ+10] T. Arendt, E. Biermann, S. Jurack, C. Krause, G. Taentzer. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In *Proc. 13th Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS'10)*. LNCS 6394, pp. 121–135. 2010.

[AGG12] TFS-Group, TU Berlin. AGG. 2012. http://tu-berlin.de/tfs/agg.

[ALPS11] A. Anjorin, M. Lauder, S. Patzina, A. Schürr. eMoflon: Leveraging EMF and Professional CASE Tools. In *INFORMATIK 2011*. Lecture Notes in Informatics 192, p. 281. Gesellschaft für Informatik, 2011. Extended abstract.

[BET12] E. Biermann, C. Ermel, G. Taentzer. Formal Foundation of Consistent EMF Model Transformations by Algebraic Graph Transformation. *Software and Systems Modeling (SoSyM)* 11(2):227–250, 2012.

[BGH+05] S. Burmester, H. Giese, M. Hirsch, D. Schilling, M. Tichy. The Fujaba Real-Time Tool Suite: Model-Driven Development of Safety-Critical, Real-Time Systems. In *Proc. 27th Int.. Conf. on Software Engineering (ICSE)*. 2005.

[EEHP09] H. Ehrig, C. Ermel, F. Hermann, U. Prange. On-the-Fly Construction, Correctness and Completeness of Model Transformations based on Triple Graph Grammars. In *Proc. 12th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS'09)*. LNCS 5795, pp. 241–255. Springer, 2009.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theor. Comp. Science. Springer, 2006.

[Fuj12] University of Paderborn. Fujaba Tool Suite. 2012. http://www.fujaba.de/.

[GEH11] U. Golas, H. Ehrig, F. Hermann. Formal Specification of Model Transformations by Triple Graph Grammars with Application Conditions. *ECEASST* 39, 2011. http://journal.ub.tu-berlin.de/index.php/eceasst/issue/archive

[GHL12] H. Giese, S. Hildebrandt, L. Lambers. Bridging the gap between formal semantics and implementation of triple graph grammars. *Software and Systems Modeling*, pp. 1–27, 2012. http://dx.doi.org/10.1007/s10270-012-0247-y.

[GK10] J. Greenyer, E. Kindler. Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars. *Software and Systems Modeling (SoSyM)* 9(1):21–46, 2010.

[GW09] H. Giese, R. Wagner. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling (SoSyM)* 8(1), 3 2009.

[HEEO12] F. Hermann, H. Ehrig, C. Ermel, F. Orejas. Concurrent Model Synchronization with Conflict Resolution Based on Triple Graph Grammar. In *Proc. Intern. Conf. on Fundamental Aspects of Software Engineering (FASE'12)*. LNCS. Springer, 2012.

[HEGO10] F. Hermann, H. Ehrig, U. Golas, F. Orejas. Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars. In *Proc. Int. Workshop on Model Driven Interoperability (MDI'10)*. Pp. 22–31. ACM, 2010. http://doi.acm.org/10.1145/1866272.1866277.

[HEOG10] F. Hermann, H. Ehrig, F. Orejas, U. Golas. Formal Analysis of Functional Behaviour of Model Transformations Based on Triple Graph Grammars. In Ehrig et al. (eds.), *Proc. Int. Conf. on Graph Transformation (ICGT' 10)*. LNCS 6372, pp. 155–170. Springer, 2010.

[JABK08] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev. ATL: A model transformation tool. *Science of Computer Programming* 72(1-2):31 – 39, 2008.

[KLKS10] F. Klar, M. Lauder, A. Königs, A. Schürr. Extended Triple Graph Grammars with Efficient and Compatible Graph Translators. In *Graph Transformations and Model Driven Enginering*. LNCS 5765, pp. 141–174. Springer, 2010.

[LAVS12] M. Lauder, A. Anjorin, G. Varró, A. Schürr. Bidirectional Model Transformation with Precedence Triple Graph Grammars. In *Proc. of the 8th European Conf. on Modelling Foundations and Applications*. LNCS. Springer, 2012. Accepted.

[MOM12] University of Leicester. MOMENT2-MT. 2012. http://www.cs.le.ac.uk/people/aboronat/tools/moment2-gt/.

[Sch94] A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In Tinhofer (ed.), *WG94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science*. LNCS 903, pp. 151–163. Springer, 1994.