Tool Support for Multi-amalgamated Triple Graph Grammars

Erhan Leblebici(✉), Anthony Anjorin, and Andy Schürr

Technische Universität Darmstadt, Darmstadt, Germany
{erhan.leblebici,anthony.anjorin,andy.schuerr}@es.tu-darmstadt.de

Abstract. We present in this paper our tool support with eMoflon (www.emoflon.org) to incorporate the concept of *multi-amalgamation* into *Triple Graph Grammars* (TGGs). Multi-amalgamation provides a mechanism similar to a *foreach* loop for graph transformation rules by consolidating multiple applications of rules depending on how many rule applications are available at transformation time. TGGs are a well-known technique used to specify bidirectional model transformation, where consistency is described via triple rules that build up source, target, and correspondence models simultaneously. Combining both techniques in eMoflon yields a TGG implementation that can handle bidirectional consistency relations between source and target elements, whose number is unknown at design time and can only be determined at transformation time. Our goal with this extension is to tackle transformation scenarios that are currently beyond the capabilities of classical TGGs.

Keywords: Triple graph grammars · Multi-amalgamation · eMoflon

1 Introduction and Motivation

Triple Graph Grammars (TGGs) [14] are a declarative and rule-based technique to specify bidirectional model transformation, which plays a crucial role in Model-Driven Engineering (MDE). Formalizing models as graphs, a TGG comprises *triple rules* that state how to build up source and target graphs connected via a correspondence graph. Hence, a TGG is a constructive *grammar* for consistent *triples* of *graphs*. From this grammar, forward and backward transformation rules are derived to realize model transformation in the respective direction.

A crucial limitation when specifying consistency with triple rules is the fact that they are graph patterns of fixed size, requiring and creating a constant number of related elements in a single application of the respective rule. This is not always sufficient in practice as the number of involved elements for the desired notion of consistency might depend on concrete models and, therefore, be impossible to determine at specification time. Intuitively, a *foreach* loop-like feature is missing to specify consistency for an arbitrary number of elements.

The expressiveness issue with fixed rule patterns as well as a formal solution to the problem, namely *amalgamation*, have already been explored in classical graph transformation. In [1], amalgamation is introduced as combining the

F. Parisi-Presicce and B. Westfechtel (Eds.): ICGT 2015, LNCS 9151, pp. 257–265, 2015.
DOI: 10.1007/978-3-319-21145-9_16

applications of two rules (called *multi-rules*) over the shared application of an embedded subrule (called *kernel rule*). This is generalized in [15] to combining n multi-rule applications, and is formalized in [5] as *multi-amalgamation*. With multi-amalgamation, transformations are not specified via plain rules but via *interaction schemes* that contain a kernel rule and an arbitrary number of multi-rules that embed the kernel rule. Multi-rules are *consolidated* over the kernel to a *multi-amalgamated rule* at transformation time depending on how many rule applications over the same kernel are available for a concrete model.

To the best of our knowledge, existing TGG implementations neither support multi-amalgamation nor provide a similar means to overcome the limitations of fixed rule patterns. In this paper, we tackle this gap from a practical perspective and report on our tool support for multi-amalgamated TGGs, i.e., TGGs that are specified via interaction schemes. Our goal is to increase the capabilities of our TGG implementation eMoflon (www.emoflon.org) by utilizing our formal results from [11]. The practical challenges here are to (1) extend the visual syntax appropriately for multi-amalgamation, and (2) handle multi-amalgamated rules without a high impact on scalability. Moreover, we provide a quantitative evaluation (runtime measurements) of our implementation and compare it with our choice of another bidirectional tool, namely medini QVT [9]. A demo session with our running example is available via a virtual machine[1] in SHARE [6].

The rest of the paper is structured as follows: After introducing a running example that is beyond the capabilities of classical TGGs in Sect. 2, we introduce in Sect. 3 multi-amalgamated TGGs with eMoflon by solving the running example. In Sect. 4, we discuss related work and evaluate our implementation quantitatively with a tool comparison. Finally, Sect. 5 concludes the paper.

2 Running Example

As our running example, we use an excerpt of a transformation between class diagrams and their HTML-like documentations (e.g., Javadoc). In particular, we focus on transforming inheritance links in class diagrams to hyperlinks in documents (and vice versa). The most important requirement with respect to our contribution is that hyperlinks must be created in the documents for direct super classes (as from now referred to as direct hyperlinks) as well as for the transitive closure of all super classes (referred to as transitive hyperlinks). For simplicity, we allow multiple inheritance but forbid repeated inheritance, i.e., we assume that a transitive hyperlink is not induced over multiple ways. Figure 1 shows a class diagram and its consistent documentation.

Obviously, the number of transitive hyperlinks to be created when transforming an inheritance link depends on the concrete class diagram. Consider the inheritance link between Employee and Person in Fig. 1 and assume all other parts of the class diagram is documented consistently. Besides creating a direct and a transitive hyperlink from the subclass document (Employee) to the super

[1] Direct link to the virtual machine: http://is.ieis.tue.nl/staff/pvgorp/share/?page=ConfigureNewSession\&vdi=XP-TUe_TGG-Comparison_eMoflonEMF.vdi.

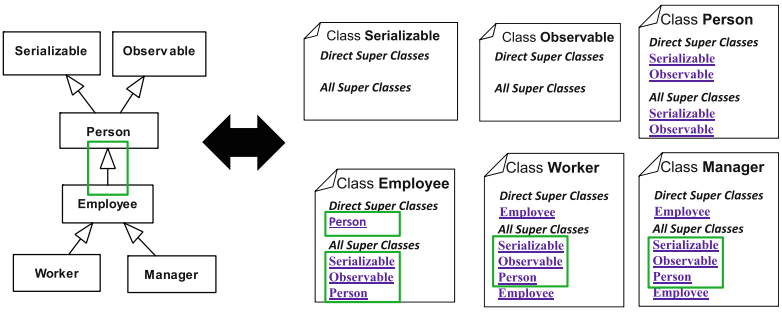


Fig. 1. A class diagram and its corresponding documentation

class document (Person), three additional steps must be repeated to create additional transitive hyperlinks: (S.1) from the subclass document to all transitive super class documents (from Employee to Serializable and Observable), (S.2) from all transitive subclass documents to the super class document (from Worker and Manager to Person), and (S.3) from all transitive subclass documents to all transitive super class documents (from Worker and Manager to Serializable and Observable). While creating ten hyperlinks in total for our concrete case, this number ranges in general between two and arbitrarily many depending on the class diagram, making a consistency specification with fixed patterns impossible.

3 Multi-amalgamated TGGs with eMoflon

In this section, we specify a TGG using multi-amalgamation with eMoflon, that is indeed able to describe the consistency relation required for our example. All diagrams except the last one in this section are specified with eMoflon's frontend.

To the left of Fig. 2, a triple of metamodels for our running example is depicted. The source metamodel describes class diagrams consisting of classes with inheritance links (specified via the reference super). Accordingly, the target metamodel describes hyperlinked documents. We distinguish between directLinks representing hyperlinks for a direct inheritance relation and allLinks representing

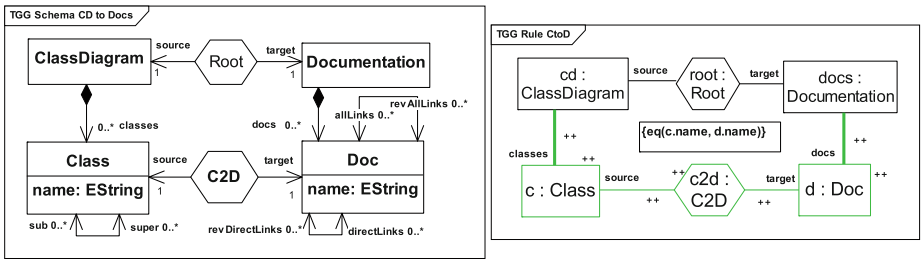
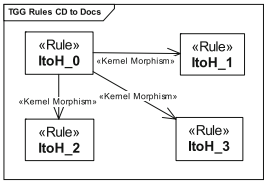


Fig. 2. The metamodel triple and a triple rule for the running example

hyperlinks for the transitive closure of inheritance relations. Finally, the hexagon-shaped correspondences relate class diagrams with their documentation.

To the right of Fig. 2, an exemplary *triple rule*, namely CtoD (Class to Document), is shown. A triple rule *matches* a pre-condition (depicted in black) and *extends* it to a post-condition by creating new elements (depicted in green with a ++ mark-up). The triple rule CtoD requires the root elements[2], i.e., a related pair of a class diagram and a documentation model, and creates a related pair of a class and a document. The attribute condition eq(c.name,d.name) ensures that the names of the class and the document are equal.



Next, we describe how an inheritance link and the related hyperlinks, whose number depends on a concrete case, are created consistently. The main idea is to specify consistency as an *interaction scheme* of rules instead of plain rules. An interaction scheme, as shown in the diagram to the right, consists of a *kernel rule*, in our case ItoH_0 (Inheritance to Hyperlink), and a set of *multi-rules*, in our case ItoH_1, ItoH_2, and ItoH_3, that include the kernel rule via a so-called *kernel morphism*, and accomplish an additional *remainder*. Figure 3 depicts the internal structures of these rules. Multi-rule nodes originating from the kernel rule can be distinguished via a gray shading. White nodes in a multi-rule, and consequently their incident edges, represent the remainder. While the remainders in Fig. 3 are only in the target domain, remainders over three domains (in the same multi-rule) are possible.

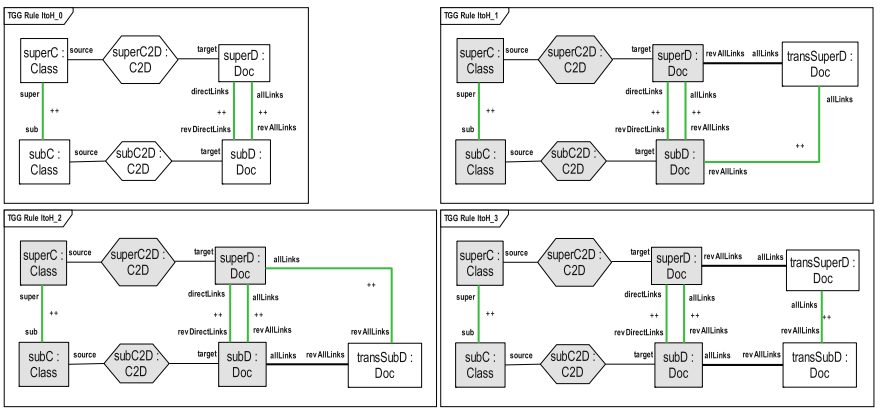


Fig. 3. A kernel rule and three multi-rules

The kernel rule ItoH_0 requires two related pairs of classes and documents, creating an inheritance in the source side and two hyperlinks in the same direction in the target side (a reference directLinks as well as a reference allLinks are

[2] Due to space limitations, we omit the simple rule that creates these root elements.

created). The multi-rules ItoH_1, ItoH_2, and ItoH_3 include the kernel rule and handle the three additional steps S.1, S.2, and S.3, respectively, as discussed in the previous section. Each of them creates one additional transitive hyperlink in the remainder (a reference of type allLinks) between two documents as long as they are indirectly connected by the kernel part. That is, transitive hyperlinks for an arbitrary depth of inheritance relations can be created.

Remark: Although a multi-rule includes the kernel rule completely according to its formal definition [1,5], we allow a compact syntax by reducing the kernel part of a multi-rule to a minimal interface that is sufficient to specify the remainder. In Fig. 4, we depict ItoH_1, ItoH_2, and ItoH_3 in this way by repeating only the target side of the kernel rule rather than its entire pattern. This is useful for maintainability as refactorings in the kernel rule do not break the compliance of the multi-rules as long as the changes do not concern the remainder.

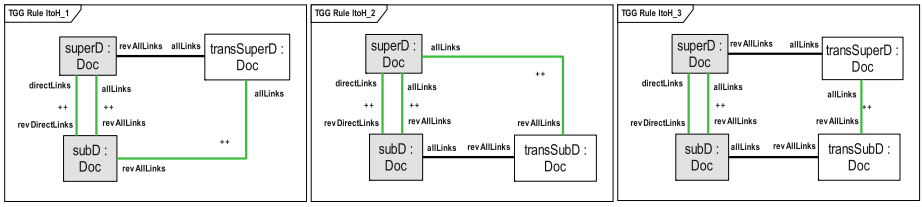


Fig. 4. Compact syntax for multi-rules

At transformation time, the multi-rules of an interaction scheme are consolidated to a *multi-amalgamated rule*. The size of this consolidation depends on how many applications of the multi-rules are available that agree on the same kernel match. If there does not exist any multi-rule applications but only a kernel rule application, the multi-amalgamated rule is identical to the kernel rule. Considering again the repeated steps for the inheritance link between the classes Person and Emplyoee in Fig. 1, the following multi-rule applications are available for our interaction scheme: (S.1) ItoH_1 twice while matching Serializable and Observable in the remainder, (S.2) ItoH_2 twice while matching Worker and Manager in the remainder, and (S.3) ItoH_3 four times while matching all four possible combinations of Serializable and Observable with Worker and Manager in the remainder. Figure 5 depicts the resulting multi-amalgamated rule. For presentation purposes, nodes matching the same element are merged to one node.

Note that such a (possibly very large) multi-amalgamated rule is not specified explicitly by the transformation designer but accomplished by eMoflon at transformation time given an interaction scheme and a model. From a consistency point of view, the multi-amalgamated rule in Fig. 5 relates one inheritance link to ten hyperlinks (nine allLinks and one directLinks). In the forward direction, therefore, it translates one inheritance link to ten hyperlinks. Analogously, ten hyperlinks are translated to one inheritance link in the backward direction.

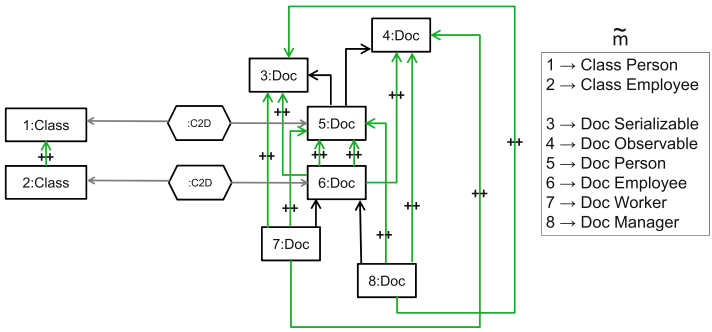


Fig. 5. The induced multi-amalgamated rule for our example

Technically, eMoflon *compiles* interaction schemes to programmed graph transformation in order to realize transformations with multi-amalgamated rules. This compilation is transparent to the user and enables the utilization of control flow structures to find *all* occurrences of multi-rule matches at transformation time. That is, the semantics of multi-amalgamated steps in eMoflon are defined via *maximal matchings* comparable to a *foreach* loop, introduced on a formal level in [11]. In each atomic transformation step, our governing *control algorithm* applies a kernel rule and complements the remainders of all available multi-rule applications. A plain triple rule without any interaction scheme forms therefore the special case where there is no remainders to be complemented.

4 Related Work and Evaluation with Comparison

In this section, we discuss existing TGG implementations and other bidirectional transformation tools with a focus on their support for a *foreach* construct. We also provide a quantitative evaluation (runtime measurements) of our implementation and a comparison with one representative from the latter group.

TGG Tools: None of the TGG tools we are aware of support multi-amalgamation or a similar means to overcome the limitations posed by fixed rule patterns. Their different strategies when deriving forward and backward transformations have arguably a strong impact on how multi-amalgamation can be realized with these tools. Similar to eMoflon, MoTE [4] compiles TGGs to programmed graph transformation but handles only plain rules without interaction schemes. EMorf [10] and TGG-Interpreter [7] do not compile triple rules but *interpret* them directly at transformation time. A possible support for multi-amalgamation, therefore, requires the interpretation of multi-amalgamated rules constructed at transformation time. HenshinTGG [3], moreover, seems to be a promising TGG tool with regard to our goals, as the underlying graph transformation engine (Henshin) supports multi-amalgamation. This diverse tool support, beside the shared formal foundation, helped TGGs gain acceptance in the context of MDE. Our aim

is to ensure that TGGs remain competitive compared to other bidirectional languages that do not necessarily suffer from the same expressiveness issues.

Bidirectional Tools with Support for *Foreach*: GroundTram [8], an example for tools based on *bidirectional programming*, features bidirectionally interpreted queries that are inherently not restricted to a constant number of elements. The QVT (Query, View, Transformation) standard [13], in particular QVT-R (QVT-Relations), features constructs such as *forall*, *closure*, or recursive invocations to address relating arbitrarily many elements. Echo [12] and JTL [2] support the QVT-R syntax and employ model finding techniques to explore consistent pairs of models. These tools exhibit powerful expressiveness but face the usual scalability problems of model finding. A scalable QVT-R implementation is provided by medini QVT [9]. We managed to solve[3] our example using medini QVT with acceptable execution times and use this solution for a quantitative comparison, evaluating the scalability of eMoflon in the process.

Runtime Measurements: In order to achieve realistic inputs for our runtime measurements, we extracted class diagram models from the following packages in our Eclipse installation: org.antlr, eMoflon tool suite, org.apache, OpenJDK, and org.eclipse. We transformed our models in the forward and backward direction with eMoflon and medini QVT, each time with a fresh Java Virtual Machine with 8 GB memory on an Intel i5@3.30 GHz. In Fig. 6, the median of 15 repetitions is plotted for each case. The y-axis shows the time in seconds in a logarithmic scale while the x-axis lists our models with their sizes. Note that the numbers of inheritance links and hyperlinks represent the portion that is transformed via a multi-amalgamated step in the forward and backward direction, respectively.

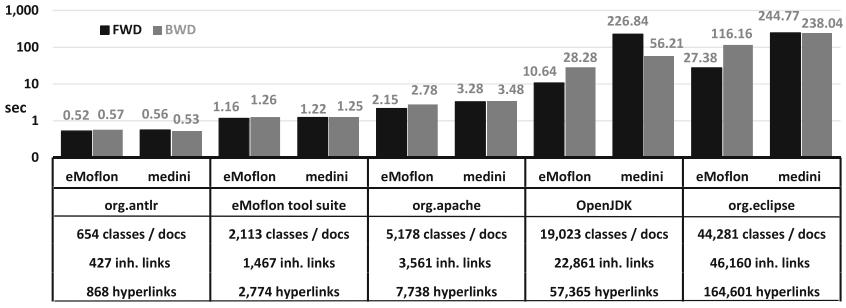


Fig. 6. Runtime measurement results with eMoflon and medini QVT

While both tools exhibit similar execution times for small and mid-sized models, eMoflon outperforms medini QVT in big-sized models with a factor of up to 20. In all cases, eMoflon's backward transformations are slower than its forward transformations. A factor of about 4 (2 min / 0.5 min) is observed for

[3] Our medini QVT solution is also available in the virtual machine in SHARE.

org.eclipse. This is explained by the greater number of elements (hyperlinks) to be matched in the target model compared to the source model. By contrast, medini QVT is faster in the backward direction than in the forward direction in case of big-sized models (again, a factor of about 4 for OpenJDK). Apparently, enforcing consistent hyperlinks is a more difficult task for the QVT-R engine than only checking them. This contrast stresses the conceptual differences between the two approaches. As a final remark, we believe to have closed an expressiveness gap of TGGs with arguably good scalability. It remains to be seen via established benchmarks how eMoflon is seated in a broader circle of bidirectional tools.

5 Conclusion and Future Work

We presented multi-amalgamated TGGs with eMoflon allowing us to specify consistency of an unbounded number of elements. The achieved extension adheres to the rule-based nature of TGGs and is at the same time scalable.

Our focus for future work is *incremental model synchronization* with multi-amalgamated TGGs. We furthermore plan *consistency checks* via correspondence creation between existing models using multi-amalgamated TGGs. Our ultimate goal is mature tool support for concurrent engineering in an MDE context.

References

1. Boehm, P., Fonio, H.R., Habel, A.: Amalgamation of graph transformations: a synchronization mechanism. JCSS **34**(2–3), 377–408 (1987)
2. Cicchetti, A., Di Ruscio, D., Eramo, R., Pierantonio, A.: JTL: a bidirectional and change propagating transformation language. In: Malloy, B., Staab, S., van den Brand, M. (eds.) SLE 2010. LNCS, vol. 6563, pp. 183–202. Springer, Heidelberg (2011)
3. Ermel, C., Hermann, F., Gall, J., Binanzer, D.: Visual modeling and analysis of EMF model transformations based on triple graph grammars. ECEASST **54**, 1–14 (2012)
4. 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 (2010)
5. Golas, U., Ehrig, H., Habel, A.: Multi-amalgamation in adhesive categories. In: Ehrig, H., Rensink, A., Rozenberg, G., Schürr, A. (eds.) ICGT 2010. LNCS, vol. 6372, pp. 346–361. Springer, Heidelberg (2010)
6. van Gorp, P., Mazanek, S.: SHARE: a web portal for creating and sharing executable research papers. Procedia Comput. Sci. **4**, 589–597 (2011)
7. Greenyer, J., Pook, S., Rieke, J.: Preventing information loss in incremental model synchronization by reusing elements. In: France, R.B., Kuester, J.M., Bordbar, B., Paige, R.F. (eds.) ECMFA 2011. LNCS, vol. 6698, pp. 144–159. Springer, Heidelberg (2011)
8. Hidaka, S., Hu, Z., Inaba, K., Kato, H., Nakano, K.: GRoundTram: an integrated framework for developing well-behaved bidirectional model transformations. In: Alexander, P., Pasarenau, C.S., Hosking, J.G. (eds.) ASE 2011, pp. 480–483 (2011)

9. Ikv++: Medini QVT. http://projects.ikv.de/qvt
10. Klassen, L., Wagner, R.: EMorF - A tool for model transformations. ECEASST **54**, 1–6 (2012)
11. Leblebici, E., Anjorin, A., Schürr, A., Taentzer, G.: Multi-Amalgamated Triple Graph Grammars. In: Parisi-Presicce, F., Westfechtel, B., (eds.) ICGT 2015, LNCS 9151, pp. 87–103. Springer, Heidelberg (2015)
12. Macedo, N., Cunha, A.: Implementing QVT-R bidirectional model transformations using alloy. In: Cortellessa, V., Varró, D. (eds.) FASE 2013 (ETAPS 2013). LNCS, vol. 7793, pp. 297–311. Springer, Heidelberg (2013)
13. OMG: QVT Specification, V1.1 (2011). http://www.omg.org/spec/QVT/1.1/
14. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 151–163. Springer, Heidelberg (1995)
15. Taentzer, G.: Parallel and distributed graph transformation : Formal Description and Application to Communication-Based Systems. Ph.D. thesis (1996)