Electronic Communications of the EASST
Volume 54 (2012)


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

## EMorF - A tool for model transformations

Lilija Klassen and Robert Wagner

6 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

# EMorF - A tool for model transformations

**Lilija Klassen**[1] **and Robert Wagner**[2]

[1] klassen@solunar.de, [2] wagner@solunar.de, http://www.solunar.de/
Solunar GmbH, Germany

**Abstract:**

In this paper, we present EMorF - a model transformation tool for EMF. EMorF supports the specification and execution of in-place model transformations as well as model-to-model transformations. The graphical though formal specification is based on (triple-) graph grammars, which are executed by an interpreter system. In this paper, we focus on the provided tool support for the development and execution of model transformations.

**Keywords:** triple graph grammars, graph transformation, model transformation, EMF, EMorF

## 1 Introduction

Model transformations are an integral part of the model-driven software development approach. The requirements and expectations regarding model transformation techniques are very demanding. A practical solution should be formal and easy to use. It should allow specifying and executing in-place model transformations in order to support model refinement, model refactoring for software maintenance and renewal as well as model inconsistency checking with automated inconsistency resolution. In addition, a practical formalism should allow specifying model-to-model transformations which can be used for the evolution of models, for the migration of data, and for model exchange between different tools.

EMorF[1] is an open source model transformation tool for the Eclipse Modeling Framework (EMF). It supports the specification of in-place model transformations as well as model-to-model transformations. In-place model transformations are specified graphically by graph rewriting rules and model-to-model transformations are specified by triple graph grammars [Sch94]. The main advantage of triple graph grammars is that they allow executing a model-to-model transformation in both directions. Moreover, they can be used to check the correspondence between two models and to synchronize models incrementally [GW09]. Because of a strong graph and category theoretic background, graph grammars have precise formal semantics.

In order to be more expressive, EMorF integrates the Object Constraint Language (OCL) for the specification of constraints and application conditions. In addition to that, EMorF provides a simple but powerful application programming interface to the EMorF rules and the EMorF interpreter system.

There are several graph grammar based model transformation tools, e.g. AGG [AGG12], Henshin [Ecl12], eMoflon [MOF12], Fujaba [FUJ12] or TGG-Interpreter [TGG12]. But to the best

[1] http://www.emorf.org

of our knowledge only the TGG-Interpreter is based on EMF models and supports model-to-model transformations specified with triple graph grammars, which are executed by an interpreter system. But in contrast to EMorF the TGG-Interpreter does not support in-place model transformations.

The remainder of this paper is organized as follows. In the next section we give a brief and informal introduction to the basic concepts of our tool. In Section 3, we present the tool support for the specification of in-place model transformation. The tool support for model-to-model transformations is described in Section 4. The paper closes with a conclusion and an outlook on future work in Section 5.

## 2 EMF (Meta-) Model

EMorF is a tool for the transformation of models that are based on the Eclipse Modeling Framework [EMF12]. The reason for using EMF is that it is a well-established technology for building tools operating on (data) models with well-defined meta-models. For this purpose, EMF provides tool support for meta-modeling, code generation and dynamic model instantiation. In addition, EMF is supported by many other standards, frameworks and tools. This ensures a wide application area for EMorF.

An EMF meta-model consists of classes with attributes, associations with multiplicity annotations (including composition and reference relationships), as well as generalizations. EMorF relies on EMF meta-models and (triple) graph grammars. For this purpose, the concepts are mapped to each other where classes and associations correspond to nodes and edges in a typed graph. As a consequence, models based on a meta-model are interpreted as graphs where objects correspond to nodes and links correspond to edges.

## 3 In-place Model Transformation

In-place model transformations are specified using EMorF graph rewriting rules. For this purpose, EMorF provides a graphical editor with a palette, a property view and an outline view. Figure 1 shows the EMorF editor and its corresponding views.

The graphical editor in Figure 1 shows the *PullUpAttribute* refactoring rule. This rule removes one attribute from a subclass and inserts it into its superclass. However, the refactoring can only be applied, if the refactoring does not introduce any attribute naming conflicts, i.e. if none of the subclasses of the superclass already uses the same attribute name as the attribute to be moved.

For the specification of such a graph rewriting rule, EMorF uses the short-hand FUJABA-notation where the left- and right-hand sides of a rule are specified in a single area. To distinguish between patterns that have to be bound and patterns that have to be created or deleted, the corresponding objects and links are marked with the appropriate annotations. For example, in the *PullUpAttribute* rule shown in Figure 1, the objects *father*, *son* and *attr* as well as the link *inheritance* have been marked with the ≪bind≫ annotation, whereas the link *oldAttrLink* is marked with a ≪delete≫ and the link *newAttrLink* with a ≪create≫ annotation.

The objects and links are created in the editor by simply dragging them from the palette onto the drawing area. For this purpose, the palette offers all types that are contained in the corre-

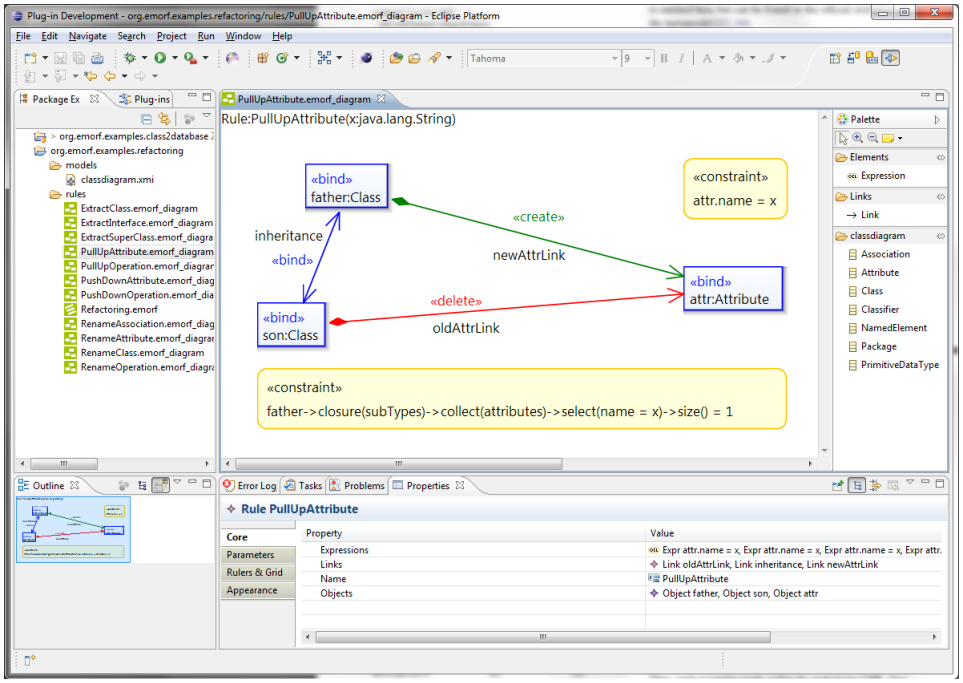


Figure 1: EMorF graphical editor showing refactoring rule *PullUpAttribute*

sponding meta-model. However, in many cases it is not sufficient to specify a pattern consisting of objects and links only. Often, additional constraints have to be specified as well. Furthermore, it is necessary to provide ways and means for the specification of attribute value modification within a rule. For this purpose, we have integrated OCL support into EMorF.

The OCL support in EMorF comes in the form of expression nodes. The expression nodes are also created using the palette. The OCL expressions can be edited easily within the graphical editor due to in-place editing support with built-in content assist functionality.

The OCL was designed originally as a constraint and object query language without any side effects. Meanwhile, there are extensions that allow using OCL also for assignments. A prominent example is the QVT language. Therefore, we decided to use OCL expressions both for constraints and assignments as well. However, we did not introduce new language concepts but annotate the expression either by ≪constraint≫ or ≪assignment≫ annotations. In the case of a constraint, a live validation checks if the expression is of type boolean. In the case of an assignment expression, it is checked whether the result is of the same type as the attribute. If the validation fails, the user is informed about the malformed OCL expression in the standard problem view of the Eclipse IDE.

Figure 1 contains two OCL constraints. The first constraint expression `'attr.name = x'` requires that the name of the attribute that should be pulled up has to be equal to the rule parameter `'x'`. The second constraint `'father->closure(subTypes)->...'` ensures that the modification described by the rule is only applied if no attribute naming conflicts exist.

In EMorF, the specified rules are executed in an interpreted way. In general, there are different

ways how an interpreter executes rules: (1) an interpreter can execute a rule directly (like in TGG-Interpreter [TGG12]), (2) an interpreter can translate a rule into an intermediate representation and execute this representation immediately (like in Henshin [Ecl12]), or (3) an interpreter executes a precompiled intermediate representation which has been created by a corresponding generator. In the case of EMorF, the rules are precompiled into an intermediate representation. The EMorF generator, which is part of the interpreter system, uses a minimum branching graph algorithm to produce intermediate representations. These intermediate representations serve as search plans that are executed if the user invokes the interpreter.

There are three different ways to invoke rule execution in EMorF. The first possibility is to use a context sensitive pop-up menu on the selected object of the model that has to be transformed. The pop-up menu offers potentially applicable rules, i.e. rules containing objects of the same type as the selected object. The user can choose to apply the rule only once or to apply the rule on all possible matches. In addition, if rule parameters are specified, the user is asked to enter necessary parameter values. The usage of the pop-up menu is quite simple and allows for an interactive rule execution. It is quite comfortable especially during the development of graph rewriting rules.

The second possibility to invoke a rule is to use the standard Eclipse run configuration dialog. In this dialog, all necessary information are collected and stored. Therefore, this way of rule invocation is quite suitable if the rule has to be applied over and over again.

The third and last possibility for rule invocation is made available by the EMorF API. The provided interface to the rules and the interpreter system allows to implement control structures and to inspect matching results of applied rules and their references to objects and links programmatically. In addition, the graph rewriting rules can be split up and applied in a match-only mode, i.e. modifications are not executed at all or in a later step. This allows for fine grained control which might be helpful in implementing more complex algorithms used for instance for simulation or model refactoring.

## 4 Model-to-Model Transformation

For model-to-model transformation the same graphical editor as for in-place model transformations is used. However, in this case we employ the visual and bi-directional transformation technique of triple graph grammars [Sch94]. A triple graph grammar is a declarative definition of a model transformation and consists of a set of transformation rules. In Figure 2, a triple graph grammar rule is shown.

The rule specifies a consistent correspondence mapping between the objects of the source and the target model and is part of a transformation between a class diagram and a relational database schema. In particular, the presented rule defines a mapping between a class and a table with a column and a primary key and uses some additional assignments and constraints.

A triple graph grammar rule is specified in the editor in the same way as an in-place model transformation. However, as shown in Figure 2, the graphical editor is separated into three different areas representing the source, the correspondence and the target domain. Since each domain can have its own meta-model, the palette is also subdivided into different areas to hold the elements of the involved meta-models.

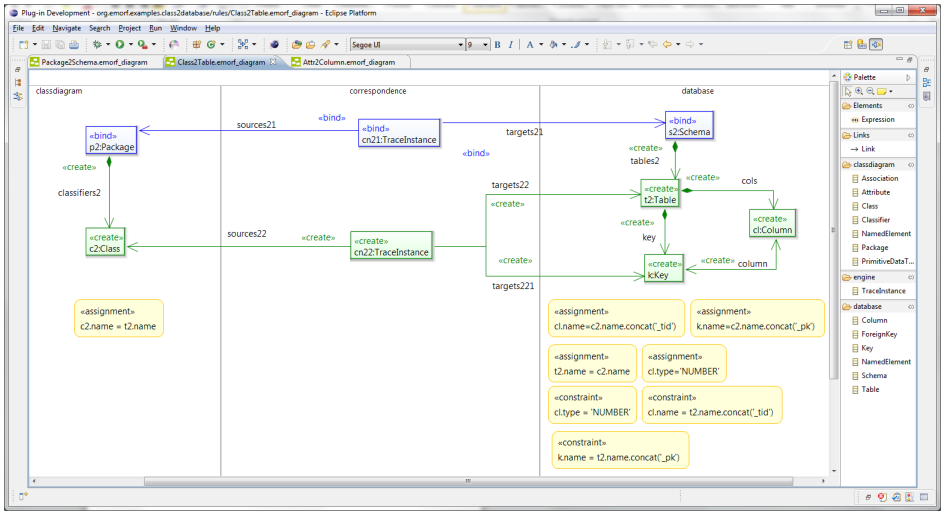


Figure 2: A triple graph grammar rule

Model-to-model transformation rules can be executed in source-to-target (forward) or in target-to-source (reverse) direction. In addition, the interpreter is able to check the correspondence mapping and to create the traceability objects of the correspondence domain (cf. Figure 2). The result of a transformation depends on the execution mode and is a file that contains either a model corresponding to the source or the target domain or a trace model holding the correspondence mappings. The correspondences can be inspected using the trace viewer shown in Figure 3.

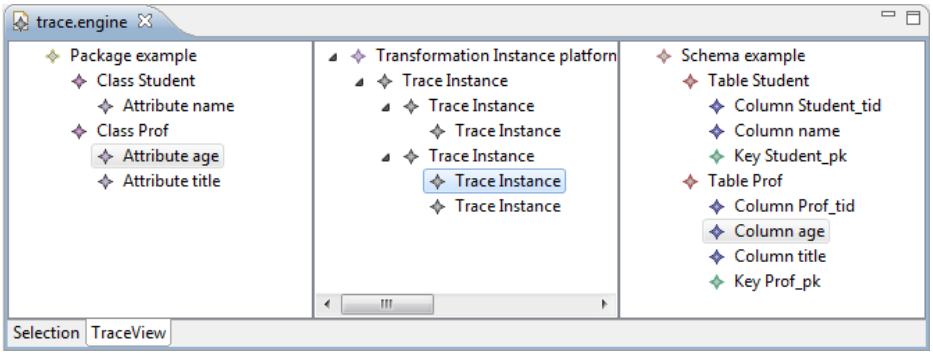


Figure 3: EMorF trace view showing the correspondence mappings after a transformation

A model-to-model transformation is executed in EMorF with the help of a launch configuration, where paths to an input, an output and a trace model are specified. After the direction mode (forward, mapping, or reverse) is configured, the model-transformation can be executed either in standard or in debug mode.

## 5 Conclusion and Future Work

In this paper, we presented the model transformation tool EMorF. EMorF operates on EMF models and relies on the visual and formal technique of graph grammars and triple graph grammars. EMorF supports both techniques in one tool by one and the same graphical editor, an interpreter system for rule execution, a debugger and other views. In addition, EMorF integrates the OCL standard and enables this way advanced specifications with negative application conditions, closures and flexible path expressions, whereas the included debugger enables users to detect and diagnose errors by going step by step through their model transformation.

Currently assignments in EMorF model-to-model transformation rules must be specified for each transformation direction seperately. This is different to the specification in [Sch94], where the assignments are derived from declarative attribute constraints. In order to be compatible with the approach from [Sch94], we plan to introduce such declarative attribute constraints in EMorF.

Another short-term target of our future work is to improve the graphical user interface, for instance to support the specification of OCL expressions in an integrated text editor, to complete the EMorF API and to analyse the performance compared to other tools. As a long-term target, we plan to integrate our incremental model synchronization algorithms into EMorF. Then we want to investigate how EMorF can be used for inconsistency management and other use cases.

## Bibliography

[AGG12] TFS-Group. AGG. Technical University of Berlin, 2012.
http://tfs.cs.tu-berlin.de/agg/

[Ecl12] The Eclipse Foundation. Henshin. 2012.
http://www.eclipse.org/modeling/emft/henshin/

[EMF12] The Eclipse Foundation. Eclipse Modeling Framework (EMF). 2012.
http://www.eclipse.org/modeling/emf

[FUJ12] Software Engineering Group. Fujaba Tool Suite. University of Paderborn, 2012.
http://www.fujaba.de

[GW09] H. Giese, R. Wagner. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling (SoSyM)* 8(1):21–43, February 2009.

[MOF12] Real-Time Systems Lab. eMoflon. Technical University of Darmstadt, 2012.
http://www.moflon.org/

[Sch94] A. Schürr. Specification of graph translators with triple graph grammars. In Mayr et al. (eds.), *Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG '94*. LNCS 903, pp. 151–163. Herrsching, Germany, June 1994.

[TGG12] Software Engineering Group. TGG-Interpreter. University of Paderborn, 2012.
http://www.cs.uni-paderborn.de/index.php?id=12842&L=1