Visualizing traceability in model transformation compositions

Visualizing Traceability in Model Transformation Compositions*

M.F. van Amstel, A. Serebrenik, and M.G.J. van den Brand

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{m.f.v.amstel, a.serebrenik, m.g.j.v.d.brand}@tue.nl

Abstract. Analysis and execution of domain-specific models is typically performed by applying model transformations to transform them to formalisms suitable for that purpose. In this way, model transformations provide great flexibility. However, it is the transformed models that are analyzed and executed instead of the domain-specific models themselves. Consequently, analysis and execution results have to be traced back to the source models. This raises the issue of traceability in model transformations. This issue becomes even more apparent when compositions of multiple model transformations are considered. Therefore, we propose in this paper to visualize traceability in (compositions of) model transformations. In this visualization, the relation between the transformation functions that comprise a model transformation and the source and target metamodel of that transformation are made explicit.

1 Introduction

Model-driven engineering (MDE) is a software engineering paradigm in which models play a central role throughout the development process [7]. Models are intended to provide abstractions tailored towards the domain in which the software to be developed should be applied. For that purpose, domain-specific languages (DSLs) are used. DSLs typically have a narrow focus and models created using DSLs are mostly not directly useable for different purposes than describing the domain for which they have been developed such as analysis and execution. Reusing domain-specific models for such purposes

* This work has been carried out as part of the FALCON project under the responsibility of the Embedded Systems Institute with Vanderlande Industries as the industrial partner. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Embedded Systems Institute (BSIK03021) program.

is one of the challenges that should be solved by MDE [5]. Typically, model transformations are used to transform domain-specific models into models suitable for different purposes. Applying model transformations provides great flexibility, since enabling the use of a DSL for a different purpose solely requires the implementation of another model transformation. The disadvantage of using model transformations to enable analyses or execution is that analyses are not performed on the domain level, but on the level of the target language [4]. Therefore, results acquired from analyzing the target models of a model transformation have to be related to their corresponding domain-specific source models.

This process of tracing the origin of model elements is called *model traceability* [1]. Sometimes, it is also referred to as *model provenance*. When a model transformation is not performed in a single step, but using a composition of multiple model transformations, a model element may be modified by more than one model transformation. In this case, model traceability becomes even harder and manual traceability is no longer feasible.

In this paper, we propose to apply a visualization technique to facilitate model traceability in compositions of model transformations. In this visualization, the relation between a model transformation and its source and target metamodel is visualized. This enables model traceability by making explicit the way in which target metamodel elements that are generated by a model transformation are related to the source metamodel elements they are generated from. The visualization technique enables visualization of multiple model transformations. In this way, model traceability of compositions of model transformations is enabled. The visualization can also be used in the other direction, i.e., to establish what generated target metamodel elements are based on a particular source metamodel element. Besides tracing source and target metamodel elements, the visualization can also be used to facilitate metamodel coverage analysis [8].

The remainder of this paper is organized as follows. In Section 2, we explain the visualization technique in more detail. Section 3 describes our implementation. In Section 4, we discuss related work. Conclusions and directions for further research are provided in Section 5.

2 Traceability Visualization

2.1 Visualization

A model transformation is defined on the metamodel level, i.e., it transforms models conforming to a source metamodel to models conforming to a target metamodel. In other words, it defines a relation between a source metamodel and a target metamodel. Accordingly, the elements that comprise a model transformation, i.e., the transformation functions, define a relation between elements of a source metamodel and elements of a target metamodel. By making this relation explicit, the traceability problem mentioned in Section 1 can be alleviated. Therefore, we propose to visualize this relation as shown in Figure 1. Figure 1 shows part of a screen shot from the tool TraceVis [10]. The model transformation visualized here is an ATL model transformation.

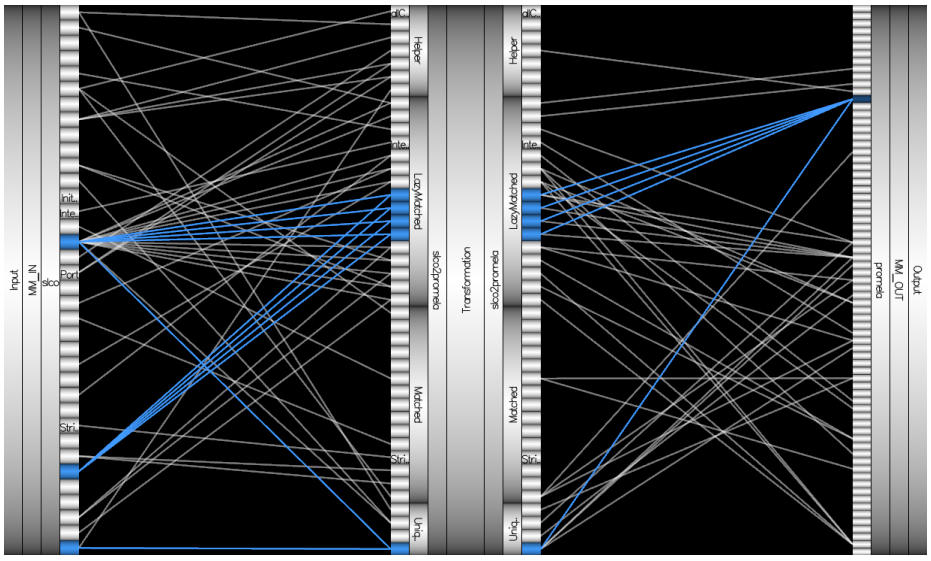

Fig. 1. Visualization of the relation between the source metamodel and target metamodel of a model transformation (1)

On the left hand side of Figure 1, a source metamodel is shown. On the right hand side of the figure, a target metamodel is shown. The outermost columns are for grouping, they are labeled *Input* and

Output respectively. The second column shows the names of the metamodels as they are referred to in the transformation. In this case, there is only one input metamodel, labeled *MM_IN*, and only one output metamodel, labeled *MM_OUT*. The third column shows the packages that are present in the metamodels, in this case both the input and the output metamodel contain one package only, viz., *slco* and *promela* respectively. The fourth row shows the metaclasses that are present in the metamodels. The zoom factor required for making the figure makes that the names of only a few metaclasses are visible in the screen shot.

The seven columns in the middle of Figure 1 show a model transformation. The three columns to the left of the center column are the same as the three columns to the right of it. The middle column is for grouping all the modules of the transformation, it is labeled *Transformation*. The second column from the center (both to the left and to the right) shows the modules that comprise the transformation. In this case, there is only one module, named *slco2promela*. The third column is used for grouping the different kinds of transformation elements. In this case, an ATL model is visualized. The transformation elements that are available in ATL are helpers, matched rules, lazy matched rules, unique lazy matched rules, and called rules. These are shown in the third column. The fourth column shows the actual transformation elements, in this case transformation rules and helpers.

There is a line between a metamodel element and a transformation element, if the metamodel element is *covered* by the transformation element. An input metamodel element is covered if it serves as input for a transformation function in the transformation. An output metamodel element is covered if it is generated as output by a transformation function in the transformation.

2.2 Applications

The first application of the visualization is facilitating model traceability, i.e., tracing the transformation elements and the source model elements that together are responsible for the generation of a target model element. By following the links in the visualization, it can be derived which transformation elements are responsible for generating

a particular target metamodel element. When an error manifests in a target model, e.g., a run-time error in an executable target model, this may be the result of an erroneous model transformation. The visualization can be consulted to establish the transformation elements that should be checked for correctness. By further following the links, viz., from the transformation elements to the source metamodel elements, it can be derived which source metamodel elements are possible origins of a target metamodel element. This information can be used to determine the source model elements that a target model element is generated from. In this way, the visualization facilitates providing feedback of results from analyzing a target model to a source model. Recall that the source model of a (composition of) model transformation(s) is typically a domain-specific model. Application of this visualization technique, therefore, makes implementing tools for a DSL, such as a type checker or a simulator, less urgent, since the results of dedicated tools for these purposes can easily be traced back to the source model. Note, however, that this is merely a palliative and that tools specific for the source language are always preferred.

The TraceVis tool provides features to assist in tracing source metamodel elements from target metamodel elements. First, there is the possibility to select elements and highlight all elements related to it. For example, in Figure 1, a single target metamodel element near the top is selected. Consequently, all transformation elements that cover that element are highlighted as well. Moreover, all source metamodel elements that are covered by the highlighted transformation elements are highlighted as well. In this way, it is possible to focus on a particular part of the model transformation or the involved metamodels. Second, it is possible to hide the transformation layer. The effect of this is shown in Figure 2. The transformation that is visualized in this figure is the same one as the one visualized in Figure 1. When hiding the transformation layer, the relation between source and target metamodel elements is visualized directly. This may assist model traceability even more.

We have shown that the visualization technique we propose enables tracing target metamodel elements back to the source metamodel elements from which they are generated. Obviously, the visualization can also be used for analyses in the other direction, i.e., to

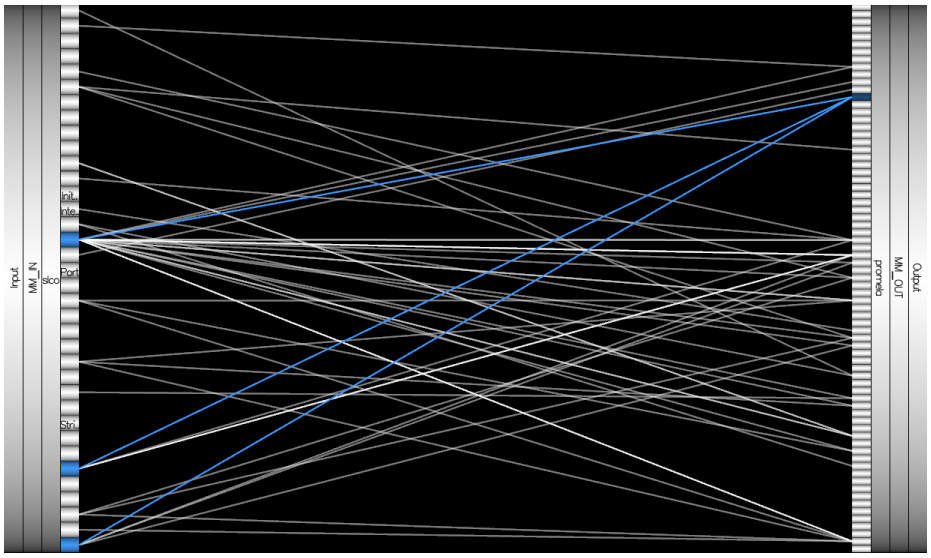

Fig. 2. Visualization of the relation between the source metamodel and target metamodel of a model transformation (2)

establish which generated target metamodel elements are based on a particular source metamodel element. In this way, the effect of a change in a source model on a target model can be determined. This is referred to as impact analysis [2]. Impact analysis has a number of useful applications. When a source model needs to be refactored for some reason, multiple alternative refactorings may be applicable to achieve the same goal. Impact analysis can be applied to determine which of the alternatives is least invasive for the target model. In this case, impact analysis can also be used to facilitate determining the parts of the target model that have to be retested to ensure its correctness.

Besides tracing source and target metamodel elements, the visualization can also be used for metamodel coverage analysis. Metamodel coverage analysis can be used to analyze the completeness of a transformation. Using the visualization, it can be observed whether all the metamodel elements from the source model that should be transformed are in fact covered by the transformation. Conversely, it can be observed whether all metamodel elements from the target metamodel that should be generated are generated by the transformation. The coverage visualization can, of course, also be used to

detect metamodel elements that should *not* be covered by a transformation. Coverage analysis can also be used to facilitate the construction of test sets for model transformations. It can be used to determine the model elements the test set should focus on. Test sets do not have to contain model elements that are not covered by a model transformation, since they will not be transformed anyway. Another application of coverage analysis is to assist in the process of co-evolution of metamodel and model transformation. It can be used to determine whether the changes to a metamodel affect the transformation, i.e., whether the changes are breaking or non-breaking [3]. If the meta-classes that are covered by a transformation do not change, there is no problem and the transformation is still usable with the evolved metamodel. However, if there are meta-classes that are covered by a transformation that do change or are removed, then the transformation will have to evolve as well. The visualization can be consulted to determine which transformation functions are affected by changed or removed meta-classes. We presented a visualization technique for coverage analysis in [8]. The visualization presented here improves on that one by providing a more intuitive layout.

2.3 Compositions of Model Transformations

In Section 2.2, we addressed traceability visualization of a single model transformation. TraceVis enables visualization of compositions of model transformations as well. Figure 3 shows again part of a screen shot from the tool TraceVis. In this case, however, not one, but a composition of seven model transformations is visualized. In order not to clutter the image too much, only the source and target metamodels of the model transformations are shown, i.e., the transformation layers are hidden.

In Figure 3, again, one target metamodel element has been selected, the topmost one in this case. Consequently, the TraceVis tools calculates a transitive closure and highlights all elements and relations in this closure. In this way, traceability analysis of a composition of model transformations can be performed.

We developed a DSL for specifying systems consisting of concurrent, communicating objects [9]. Models developed with this DSL are not directly executable. Therefore, we implemented a chain (com-

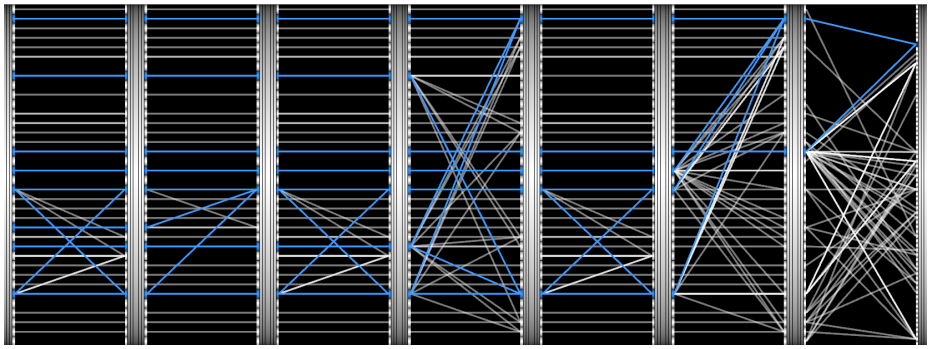

Fig. 3. Visualization of a composition of model transformations

position) of model transformations that transforms domain-specific models to executable models. This implies that run-time errors wil manifest on the level of the target language and not on the level of the DSL. Also, since we did not implement a type checker for our DSL, possible type errors will manifest when compiling an executable target model. The chain of model transformations can also be used to generate simulation models and verification models. Again, results of simulations and verification processes are not reported on the level of the source model, but on the level of the respective analysis model. In a similar way as presented in Section 2.2, the visualization facilitates traceability analysis that can be used to discover the origin of possible type errors or run-time errors and also to relate analysis results to source models. Of course, the visualization can also be used to facilitate impact analysis of chains of model transformations. The visualization is particularly useful for facilitating traceability analysis and impact analysis of compositions of model transformations since manual analysis is typically infeasible.

3 Implementation

We have implemented a tool that automatically generates a TraceVis input file from a collection of model transformations and accompanying metamodels. Currently, we implemented the visualization technique for ATL only, but the architecture of the tool allows easy extension for different model transformation formalisms. Figure 4 depicts the extensible architecture of the toolset.

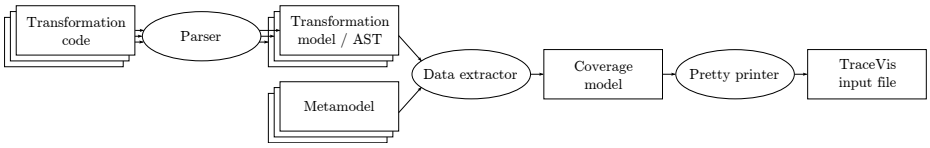

Fig. 4. Tool architecture

In the first step, the code of the model transformations under study is parsed, resulting in models that represent the abstract syntax trees (ASTs) of these transformations. These models can be represented in different formats depending on the transformation formalism they are implemented with. In the second step, the coverage data is extracted from the models representing the ASTs of the transformations and the source and target metamodels of the transformations and represented as another model. For ATL, this extraction is implemented as an ATL model transformation. Note that extending the tool with another model transformation formalism, solely requires the implementation of this second step for that formalism. In the third and last step, the coverage model is transformed to an XML file that can be used as input for the TraceVis tool. We used a Java program to perform the pretty printing.

4 Related Work

Von Pilgrim et al. present a technique for visualizing chains of model transformations as well [11]. In their visualization, diagrammatic representations of models are shown on two-dimensional planes in a three-dimensional space. Lines between these planes connect source model elements to target model elements. The main difference with our visualization is that ours is based on the relation between transformation and metamodels, whereas theirs focuses on relations between models based on a transformation. They do not visualize the model transformation themselves. If we apply our visualization to models instead of metamodels, we can perform similar traceability analyses. Since their visualization provides diagrammatic representations of models, this may lead to scalability issues when huge models or long transformation chains need to be analyzed.

Planas et al. present a metamodel coverage technique for ATL [6]. Their technique is aimed at determining full coverage of a metamodel by a set of ATL matched rules. They state that a set of ATL rules is source-covering when all elements of the source metamodel may be navigated through the execution of these rules. Similarly, they state that a set of ATL rules is target-covering when all elements of the target metamodel may be created and initialized through the execution of these rules. The result of their analysis is a yes or no answer stating whether all source and target model elements are covered. The visualization presented here, enables an in-depth analysis of the coverage relation between model transformation and metamodel.

5 Conclusions and Future Work

In this paper, we presented an approach for visualizing traceability in (compositions of) model transformations. We also presented a number of applications of the visualization technique. First, it facilitates debugging of (domain-specific) models by enabling tracing the results of analyses performed on generated models back to the models they were generated from. Second, it enables studying the effects of the evolution of a source model, or a modification in a model transformation, i.e., it facilitates impact analysis. Finally, it can be applied for coverage analysis. Coverage analysis can be employed to assess the completeness of a transformation, to derive test sets for model transformations, or to assist in the process of co-evolution of metamodel and model transformation.

The approach we presented here, is focused at visualizing metamodels and model transformations for facilitating traceability analysis. However, there are some drawbacks to the approach. When multiple transformation functions emit model elements of type t, it is unclear what transformation function was responsible for generating a particular model element of type t. This can be solved by performing the analysis on the model level instead of the metamodel level. Currently, the analysis can be performed statically, i.e., without running the transformation. In case traceability analysis is required for a single model on the model level, the visualization has to be created dynamically, i.e., from run-time transformation data.

We consider enabling the visualization on the model level as a point for further research.

References

1. N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Systems Journal*, 45(3):515–526, 2006.
2. R. S. Arnold. *Software Change Impact Analysis*. IEEE Computer Society, 1996.
3. A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating Co-evolution in Model-Driven Engineering. In *Proceedings of the Twelfth International IEEE Enterprise Distributed Object Computing Conference (EDOC 2008)*, pages 222–231, Munich, Germany, September 2008. IEEE Computer Society.
4. R. Mannadiar and H. Vangheluwe. Debugging in Domain-Specific Modelling. In B. A. Malloy, S. Staab, and M. G. J. van den Brand, editors, *Proceedings of the Third International Conference on Software Language Engineering (SLE 2010)*, volume 6563 of *Lecture Notes in Computer Science*, pages 276–285, Eindhoven, The Netherlands, October 2010. Springer.
5. P. Mohagheghi and J. Aagedal. Evaluating Quality in Model-Driven Engineering. In *Proceedings of the International Workshop on Modeling in Software Engineering (MISE 2007)*, pages 1–6, Minneapolis, MN, USA, May 2007. IEEE Computer Society.
6. E. Planas, J. Cabot, and C. Gómez. Two Basic Correctness Properties for ATL Transformations: Executability and Coverage. In I. Kurtev, M. Tisi, and D. Wagelaar, editors, *Proceedings of the Third International Workshop on Model Transformation with ATL (MtATL 2011)*, volume 742 of *CEUR Workshop Proceedings*, pages 1–9, Zurich, Switzerland, July 2011. CEUR-WS.org.
7. D. C. Schmidt. Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006.
8. M. F. van Amstel and M. G. J. van den Brand. Model Transformation Analysis: Staying Ahead of the Maintenance Nightmare. In E. Visser and J. Cabot, editors, *Theory and Practice of Model Transformations: Proceedings of the Fourth International Conference on Model Transformation (ICMT 2011)*, volume 6707 of *Lecture Notes in Computer Science*, pages 108–122, Zurich, Switzerland, June 2011. Springer.
9. M. F. van Amstel, M. G. J. van den Brand, and L. J. P. Engelen. Using a DSL and Fine-grained Model Transformations to Explore the Boundaries of Model Verification. In *Proceedings of the Third Workshop on Model-Based Verification & Validation From Research to Practice (MVV 2011) (co-located with SSIRI 2011)*, pages 120–127, Jeju Island, Korea, June 2011. IEEE Computer Society.
10. W. J. P. van Ravensteijn. Visual Traceability across Dynamic Ordered Hierarchies. Master's thesis, Eindhoven University of Technology, August 2011.
11. J. von Pilgrim, B. Vanhooff, I. Schulz-Gerlach, and Y. Berbers. Constructing and Visualizing Transformation Chains. In I. Schieferdecker and A. Hartman, editors, *Proceedings of the Fourth European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA 2008)*, volume 5095 of *Lecture Notes in Computer Science*, pages 17–32, Berlin, Germany, June 2008. Springer.