Traceability Visualization in Model Transformations with TraceVis*

Marcel F. van Amstel, Mark G.J. van den Brand, and Alexander Serebrenik

Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
{M.F.v.Amstel,M.G.J.v.d.Brand,A.Serebrenik}@tue.nl

Abstract. Model transformations are commonly used to transform models suited for one purpose (e.g., describing a solution in a particular domain) to models suited for a related but different purpose (e.g., simulation or execution). The disadvantage of a transformational approach, however, is that feedback acquired from analyzing transformed models is not reported on the level of the problem domain but on the level of the transformed model. Expressing the feedback on the level of the problem domain requires improving traceability in model transformations.

We propose to visualize traceability links in (chains of) model transformations, thus making traceability amenable for analysis.

1 Introduction

Domain-specific languages (DSLs) play a pivotal role in the model-driven engineering (MDE) paradigm. DSLs describe the domain knowledge [7] and offer expressive power focused on that domain through appropriate notations and abstractions [9]. A typical application of a DSL such as [1,6], however, goes beyond *describing* the domain knowledge, and includes, for example, execution or simulation (cf. the discussion of executability of DSLs in [15]). Hence, application of a DSL requires adequate tool support for execution and simulation. In MDE, typically the transformational approach is adopted to this end [18].

The main advantage of the transformational approach is the flexibility it provides by reusing existing formalisms. Adapting the DSL for a different purpose, such as simulation or execution, solely requires the implementation of another model transformation. However, the disadvantage of a transformational approach, is that analyses are not performed on the domain level, but on the level of the target language of a model transformation [14]. This raises the issue of *traceability*, i.e., results acquired from analyzing the target models of a model transformations have to be related to the source models.

Traceability plays an essential role in a number of typical model development scenarios such as debugging and change impact analysis. When debugging models, it is important to understand whether the erroneous part of the target model

* 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.

Z. Hu and J. de Lara (Eds.): ICMT 2012, LNCS 7307, pp. 152–159, 2012.

results from the source model or from the transformation itself, and to pinpoint the corresponding parts of the source model and/or transformation functions. Moreover, when source models are about to change, one should determine the effect of proposed changes on the target model.

In this paper, we propose to apply a visualization technique to facilitate the analysis of the relation between source models, model transformations, and target models. The proposed approach is also applicable to chains of model transformations in which the target model(s) of a preceding model transformation serve as source model(s) for the subsequent one. In this way, the approach enables traceability of model transformation compositions.

On the model level, our visualization makes explicit what source model elements are the origin of a target model element and what transformation elements are involved in creating a target model element. The model developer can therefore identify source model element(s) and transformation element(s) responsible for producing (erroneous) target model element(s), as well as generated target model element(s) based on a particular source model element (cf. Section 3.4).

The remainder of this paper is structured as follows. We start by stating the requirements has to satisfy in Section 2. Next we discuss the visualization in Section 3. In Section 4, we review the related work. Section 5 concludes the paper and provides directions for further research.

2 Requirements

We start by identifying requirements the visualization approach should satisfy. Our first requirements follow from the definition of traceability: the visualization should be able to represent *structure of* (**Req1**) and *relations between* (**Req2**) the source (meta)model, the target (meta)model and the model transformation. The visualization should further be able to present model transformations with *multiple models* serving as input and as output (**Req3**).

The next group of requirements is related to our intention to apply the visualization to chains of model transformation. The visualization, hence, should allow the user to inspect *traceability across multiple transformation steps* (**Req4**), e.g., to identify all source model elements that indirectly are responsible for producing an (erroneous) target element. Moreover, the user should be able to *ignore (some of) the intermediate transformation steps* (**Req5**), if desired.

Scrutinizing these requirements we observe similarity with visualizing traceability in traditional software development: models correspond to software development artifacts such as requirements specifications, design documents, and source code. Relating test cases to requirements is essentially the same as relating source models to target models through multiple transformation steps. This realization led to the decision to use a traceability visualization tool originally developed for traditional software development process. The tool, called TraceVis [17], turned out to be perfectly suitable for our purpose.

The basic outline of the TraceVis visualization is shown in Fig. 1. It represents three hierarchies (left, middle, right) connected by means of traceability links.

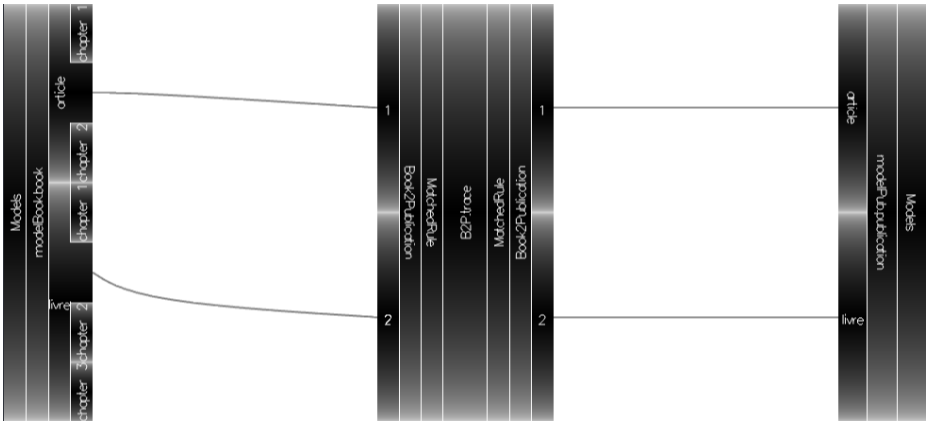

Fig. 1. TraceVis shows the relations between different hierarchies

The middle hierarchy is mirrored to allow the relations with both the left and right hierarchies to be represented. When the user selects elements from one of the hierarchies, connected elements from other hierarchies and the connecting traceability links become highlighted (cf. **Req4**). Moreover, TraceVis allows the user to hide a hierarchy (cf. **Req5**). Finally, it is known that while visualization of relations as straight lines is well-suited for smaller number of relations, it does not scale up well [12]. Therefore, TraceVis supports hierarchical edge bundling [12] specially designed for scalability.

3 Traceability on the Model Level

3.1 Basic Visualization

Fig. 1 shows part of a screen shot from the TraceVis tool applied to a "Book 2 Publication" model transformation [10]. The source model, the target model and the model transformation are represented as hierarchies: the left one, the right one and the middle one, respectively.

In hierarchies representing source and target models, the outermost columns represent the roots of the hierarchies. These are artificial nodes created for grouping (multiple) model(s) that serve as input and output of a model transformation (cf. **Req3**). The next level of the hierarchies, represented by the second leftmost and the second rightmost columns, shows the filenames of the input, and output models, respectively. The remaining levels of the hierarchy correspond to the elements of the input and output models. Model elements in these columns are shown *on top* of the elements they contain. In this way, the containment hierarchy in the models is visualized as a hierarchy.

The middle hierarchy represents a transformation. This hierarchy is mirrored to allow both the relations with the input and output models to be represented. The middle column serves for grouping all the modules of the transformation, it

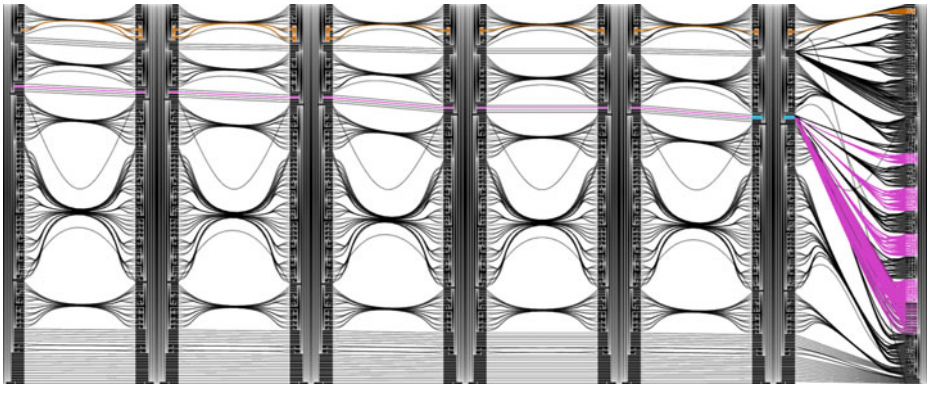

Fig. 2. TraceVis visualizes a chain of six model transformations

is labeled with the filename of the trace model that is visualized (cf. Section 3.3). The columns adjacent to the middle column are used for grouping the different kinds of transformation elements available in the model transformation language: e.g., in ATL we group helpers, matched rules, lazy matched rules, unique lazy matched rules, and called rules. Proceeding from the middle column outwards we see the column showing the actual transformation elements, e.g., ATL transformation rules. Finally, the outermost columns visualize the instances of the transformation rules, i.e., run-time applications of a rule to model elements.

3.2 Visualizing Transformation Chains

Fig. 2 shows a chain of six endogenous model transformations. The DSL on which the transformations are defined is aimed at modeling systems consisting of objects that operate in parallel and communicate with each other [1]. The transformations perform a stepwise refinement of the source model and bring it closer to the implementation by replacing, e.g., synchronous communication with asynchronous one, and lossless channels with lossy ones.

In Fig. 2 the transformation hierarchies are hidden, i.e., only the input and output models of the transformations are visualized. In all transformations, the model is copied and "slightly" modified. This is why there are many "straight" lines in the visualization. Since the model transformations refine the model, changes to the models are all local. For the first transformation, a selection (orange) is made that shows such a local change. While this change might appear as being hard to detect, it becomes apparent when zooming in on a part of the visualization (cf. Fig. 3). Only the last model transformation changes the model drastically. In the visualization, a single model element (pink) is selected in the penultimate model. This single model element gives rise to many model elements in the target model. This particular model transformation extends the model with a protocol implementation consisting of many model elements.

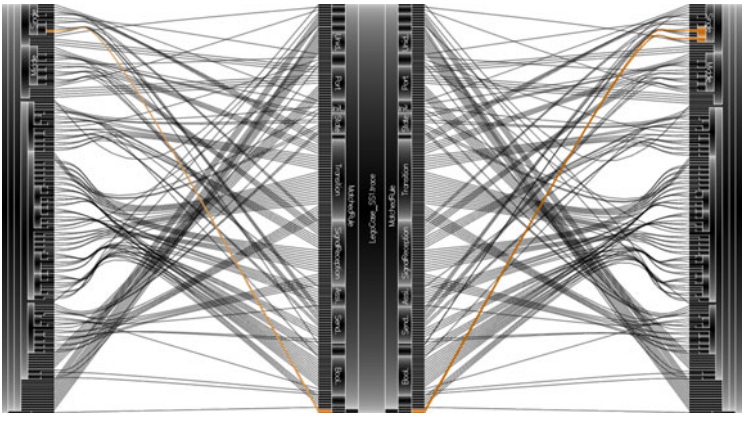

Fig. 3. Zooming in reveals that two lazy matched rules modify the model

3.3 From Model Transformations to TraceVis

Since TraceVis has originally not been designed for model transformations, we had to implement a complementary tool for automatically generating TraceVis input from model transformations. To make the distinction between different applications of the same transformation element, we have to analyze the trace model. To obtain the trace model the transformation should be executed, since without executing the transformation there is no target model and, hence, no trace model. Various approaches can be chosen to acquire a trace model from a model transformation execution. One can adapt the transformation engine such that it generates a trace model. In spirit of MDE we have opted, however, for a model transformation rather than transformation engine adaptation. The entire tool chain for generating trace models is presented in Fig. 4.

First, a higher-order model transformation, called *tracer adder* and implemented in ATL, takes as an input the model transformation being visualized, say T, and augments T such that the trace model is generated as an additional output model of T. This transformation is based on the one described by Jouault [13]. Next, the augmented transformation is applied to input models, resulting in output models and a trace model that contains links both to the input models and to the output models. Finally, the input models, the output models and the trace model are transformed to an XML file that serves as input for the TraceVis tool. This transformation has been implemented in Java and can be reused if other model transformation languages are considered.

3.4 Applications

Debugging. In the introduction, we observed that domain-specific models are typically analyzed by transforming them to a formalism suitable for analysis. The disadvantage of this approach is that feedback from the analysis model is

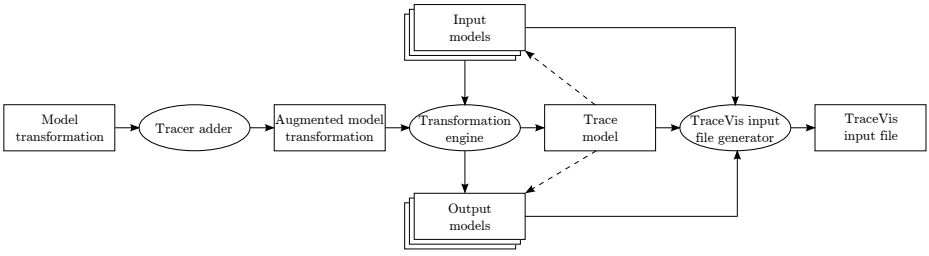

Fig. 4. Tool architecture

not reported in terms of the DSL, but in terms of the generated model. Suppose an error manifests in an executable target model. To fix this error, it has to be related to the elements in the domain-specific source model it is generated from. Using TraceVis, the visualized trace links can be followed from the model elements that caused the error to the model elements in the source model to establish the origin of the error. This is referred to as origin tracking [8].

It may be the case that the manifested error is not caused by an erroneous source model, but by an erroneous model transformation. Since the visualized trace links also show the relations between target model elements and transformation elements, errors in model transformations can easily be found. Also, when obsolete target model elements are generated, the visualization can be consulted to identify the responsible transformation element.

Impact Analysis. In addition to debugging, the visualization can also be used to facilitate change impact analysis. Change impact analysis is the process of determining the effect of proposed changes [3]. Evaluating this effect is considered to be one of the most expensive activities in the software maintenance process [4]. Most maintenance efforts include nowadays means of identifying impacts prior to making extensive software changes [5]. Using our visualization one can determine which target model elements are affected by changed source model elements.

By means of impact analysis, one can determine what part of a target model will change based on a change in the source model it was generated from. For instance, for the transformation chain described in [1], one can determine which part of the target Promela model will be affected by the change in a source SLCO model. Subsequently, this information can be used to perform incremental model checking of the Promela model (cf. [19]). Hence, conducting change impact analysis as a preliminary step reduces the verification effort. A similar argument can be made for, e.g., reducing the simulation effort.

4 Related Work

A traceability framework for model transformations was implemented in the model-oriented language Kermeta [11]. Similarly to [11], we support designers in gathering information on the transformation behavior, and make use of a transformation trace model. However, as opposed to [11], our approach focuses on

visualization. Moreover, while the approach of [11] focuses on model transformations written in Kermeta, our approach is, in principle, language independent although the current implementation is restricted to ATL.

To visualize chains of model transformations, Von Pilgrim et al. place model representations on two-dimensional planes in a three-dimensional space [16]. Lines between these planes connect source model elements to target model elements. As opposed to our work, the model transformation itself is not visualized. Moreover, the choice of two-dimensional planes in a three-dimensional space leads to scalability issues when long transformation chains are considered.

In our previous work [2] we have applied TraceVis to visualize model transformation on the *metamodel* level. While traceability visualization at the metamodel level is similar to traceability visualization at the model level, the information extraction is different from the process described in Section 3.3. The reason for this is that the relation between a model transformation and the elements of its source and target metamodel can be derived from its source code directly without running the transformation. Moreover, the intended applications of the current work differ from those of [2].

5 Conclusions and Future Work

In this paper, we have presented a novel approach to visualization of traceability information in model transformations. Our approach explicates relations on a model level, and is applicable not only to single transformations but also to transformation chains. Applications of the proposed approach range from debugging and coverage analysis to change impact analysis.

The approach consists of two phases. First, the hierarchical structure of models and model transformation(s) as well as relations between these structures are extracted from the model and model transformation files. Second, the hierarchies are extracted and the relations between them are visualized using TraceVis. By visual inspection of the TraceVis visualization, model designers can answer such questions as what source model element(s) and transformation element(s) are responsible for producing what (erroneous) target model element(s), or what generated target model element(s) are based on a particular source model element? Finally, model designers can use the TraceVis visualization as a starting point of a change impact analysis that aims at understanding how changes to a source model affect the target model and the corresponding transformation.

The current version of the extraction tool supports ATL transformations only. Therefore, we consider application of the approach to other model transformation languages to be an important part of the *future work*.

Acknowledgements. We would like to thank Wiljan van Ravensteijn for providing us with TraceVis, Ivo van der Linden for his help on the implementation of data extraction, as well as Joost Gabriels for his comments on the earlier versions of this manuscript.

References

1. van Amstel, M.F., van den Brand, M.G.J., Engelen, L.J.P.: An Exercise in Iterative Domain-Specific Language Design. In: IWPSE-EVOL, pp. 48–57. ACM (2010)
2. van Amstel, M.F., Serebrenik, A., van den Brand, M.G.J.: Visualizing Traceability in Model Transformation Compositions. In: Pre-proceedings of the First Workshop on Composition and Evolution of Model Transformations (2011)
3. Arnold, R.S., Bohner, S.A.: Impact Analysis – Towards A Framework for Comparison. In: Card, D.N. (ed.) ICSM, pp. 292–301. IEEE CS (September 1993)
4. Barros, S., Bodhuin, T., Escudie, A., Queille, J.P., Voidrot, J.F.: Supporting Impact Analysis: A Semi-Automated Technique and Associated Tool. In: ICSM, pp. 42–51. IEEE CS (1995)
5. Bohner, S.A.: Extending Software Change Impact Analysis into COTS Components. In: Proceedings of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop (SEW-27 2002), pp. 175–182. IEEE CS (2002)
6. van den Brand, M.G.J., van der Meer, A.P., Serebrenik, A., Hofkamp, A.T.: Formally specified type checkers for domain specific languages: experience report. In: LDTA, pp. 12:1–12:7. ACM, New York (2010)
7. Brandic, I., Dustdar, S., Anstett, T., Schumm, D., Leymann, F., Konrad, R.: Compliant Cloud Computing (C3): Architecture and Language Support for User-Driven Compliance Management in Clouds. In: CLOUD, pp. 244–251. IEEE CS (2010)
8. van Deursen, A., Klint, P., Tip, F.: Origin Tracking. Journal of Symbolic Computation 15(5-6), 523–545 (1993)
9. van Deursen, A., Klint, P., Visser, J.: Domain-Specific Languages: An Annotated Bibliography. SIGPLAN Notices 35(6), 26–36 (2000)
10. Eclipse Foundation: ATL Transformations, `http://www.eclipse.org/m2m/atl/atlTransformations/`
11. Falleri, J.R., Huchard, M., Nebut, C.: Towards a traceability framework for model transformations in kermeta. In: ECMDA-TW Workshop, pp. 31–40 (2006)
12. Holten, D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. IEEE Trans. Vis. Comput. Graph. 12(5), 741–748 (2006)
13. Jouault, F.: Loosely Coupled Traceability for ATL. In: ECMDA (2005)
14. Mannadiar, R., Vangheluwe, H.: Debugging in Domain-Specific Modelling. In: Malloy, B., Staab, S., van den Brand, M. (eds.) SLE 2010. LNCS, vol. 6563, pp. 276–285. Springer, Heidelberg (2011)
15. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Computing Surveys 37, 316–344 (2005)
16. von Pilgrim, J., Vanhooff, B., Schulz-Gerlach, I., Berbers, Y.: Constructing and Visualizing Transformation Chains. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 17–32. Springer, Heidelberg (2008)
17. van Ravensteijn, W.J.P.: Visual Traceability across Dynamic Ordered Hierarchies. Master's thesis, Eindhoven Univ. of Technology, The Netherlands (2011)
18. Sendall, S., Kozaczynski, W.: Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Software 20(5), 42–45 (2003)
19. Sokolsky, O., Smolka, S.: Incremental Model Checking in the Modal μ-Calculus. In: Dill, D. (ed.) CAV 1994. LNCS, vol. 818, pp. 351–363. Springer, Heidelberg (1994)