Differencing of Model Transformation Rules: Towards Versioning Support in the Development and Maintenance of Model Transformations

Timo Kehrer[1], Christopher Pietsch[2], Daniel Strüber[3]

timo.kehrer@informatik.hu-berlin.de, cpietsch@informatik.uni-siegen.de, strueber@uni-koblenz.de,
[1] Humboldt-Universität zu Berlin, Germany,
[2] Universität Siegen, Germany,
[3] Universität Koblenz-Landau, Germany

Abstract. With model transformations arising as primary development artifacts in Model-driven Engineering, dedicated tools supporting the development and maintenance of model transformations are strongly required. Calculating differences between versions of transformations is an essential service for effectively managing their evolution. In this tool demonstration paper, we present support for this task which is tailored for the differencing of graph-based model transformation rules. Our implementation is based on the model differencing framework SiLift which we adapt to the Henshin model transformation language. We demonstrate the usefulness of this feature using a running example.

1 Introduction

Model-driven engineering (MDE) emphasizes model transformation as a central activity in all stages of software construction. Many model transformation approaches have emerged to address a multitude of transformation scenarios [1]. However, less effort has been spent in providing dedicated tools supporting transformation developers in a variety of development and maintenance tasks. With model transformations arising as primary development artifacts, such tools are strongly required to fully turn the MDE vision into reality. Tools supporting some development tasks, such as clone detection for model transformations [2], have been addressed by the model transformation research community recently. In general, however, developers are not nearly as broadly supported as they are used to from traditional development environments, which is still a major obstacle to a more widespread adoption of MDE [3].

In this tool demonstration paper, we address the problem of managing transformations evolving into several versions, which must be supported by a differencing service for comparing two versions of a transformation. Providing a high-quality differencing service is challenging in the case of visual model transformation languages, particularly when the language's abstract syntax substantially differs from the graphical syntax tool users are familiar with. This is the case for

graphical transformation rules developed in Henshin [4,5], a transformation language and framework based on graph transformation concepts targeting models defined in the Eclipse Modeling Framework (EMF). Here, differences obtained from off-the-shelf comparison tools are often too low-level and hard to understand for tool users, since they report fine-grained changes based on Henshin's abstract syntax maintained in the background of the Henshin diagram editor. To mitigate this problem, modern model differencing frameworks such as EMF Compare [6] or SiLift [7] can be adapted to a given (visual) language. In this paper, we present our adaptation of SiLift to the Henshin model transformation language and environment. This way, we obtain more user-friendly descriptions of changes reflecting the way how transformation rules are edited in the Henshin diagram editor. We demonstrate the usefulness of this feature using an example. Further information, including a tool demo as well as download and installation instructions, are provided at `http://pi.informatik.uni-siegen.de/projects/SiLift/icmt2017.php`.

2 Motivating Example

Consider the transformation rules *startColumn* and *extendColumn* on the right-hand side of Fig. 3, which are part of a Henshin transformation constructing a *sparse grid* as described in the "Comb benchmark" by Varró et al. [8]. Both rules are shown in the graphical syntax of the Henshin diagram editor, where left- and right-hand sides (LHS, RHS) of a rule are combined into a single graph with annotations, e.g. «create» indicating RHS graph elements without a LHS counterpart. The syntactic sugar `@Grid` specifies the presence of a *Grid* node to be used as container for all created nodes.

Assume that we obtained the rule *extendColumn* from the rule *startColumn* using the Henshin diagram editor. This can be achieved in a seemingly simple way in six edit steps: (1) renaming the rule from *"startColumn"* to *"extendColumn"*, (2) changing the kind of the rule parameter *next* from *out* to *inout*, (3) setting the identifier of the upper left node to *"next"*, (4) converting the upper left node into a node to be preserved, (5) doing the same conversion for the

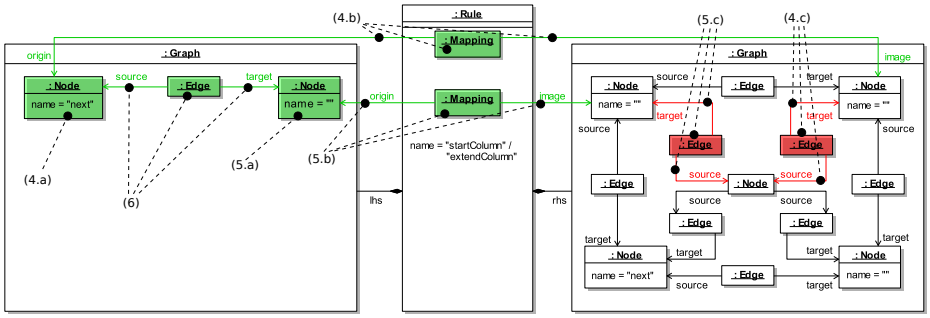

Fig. 1: ASG changes caused by editing the rule *startColumn* to become the revised rule *extendColumn*.

upper right node, and (6) converting the edge connecting these nodes into an edge which is to be preserved.

Following the basic MDE principle "everything is a model" [9], Henshin transformation rules are represented as models, thus model differencing tools can be used for comparing two versions of a rule. However, available tools deliver change descriptions on the abstract syntax level which are often hard to understand. For example, the generic differencing facility of EMF Compare [6], without being adapted to the Henshin transformation language, describes the effect of edit step (4) in terms of three low-level changes: (4.a) the creation of a LHS node of type *Node*, (4.b) the creation of a *Mapping* associating that node with an already existing one in the RHS, and (4.c) the deletion of a containment edge connecting the RHS node with the container node of type *Grid*. Note the syntactic sugar `@Grid`, which causes change (4.c). Analogously, three low-level changes are reported for edit step (5), in the sequel referred to as (5.a), (5.b) and (5.c). Finally, step (6) is reported as introducing a new edge in the RHS instead of a conversion of the existing edge into an edge to be preserved. Fig. 1 illustrates these low-level ASG changes using the well-known UML object diagram notation (some containment edges are represented by nesting nodes); ASG elements colored in red are deleted, while elements colored in green are created.

3 Differencing of Transformation Rules Using SiLift

Structural differencing tools basically calculate a difference between two model versions, say v_1 and v_2, in two steps: First, a model-matching algorithm [10] identifies corresponding nodes and edges in both ASGs. Subsequently, a difference is derived from a matching: Elements of v_1 which do not have a corresponding element in v_2 are considered to be deleted, while elements of v_2 not having a corresponding element in v_1 are considered to be created. In Sect. 2, we have seen that using such primitive graph operations on the ASG typically impairs the readability of the resulting differences.

The model differencing framework SiLift [7] is a suitable basis for semantically lifting low-level changes to the level of edit operations, based on the approach presented in [11]. SiLift needs to be configured for a specific modeling language by specifying the edit operations available in this language. These edit operations

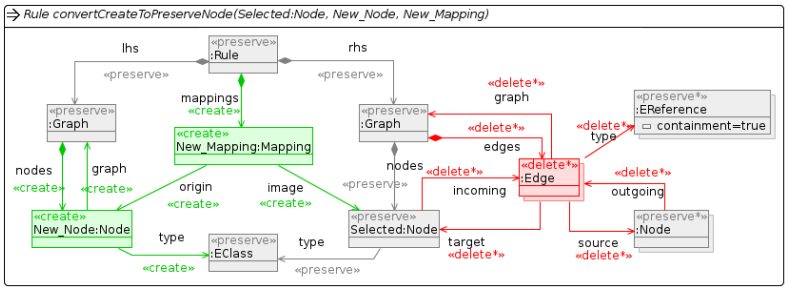

Fig. 2: Edit operation *convertCreateNodeToPreserveNode* specified in Henshin.

need to be provided as Henshin rules. To support the differencing of Henshin transformations, we produced two sets of edit operations. First, we generated a complete set of *elementary* edit operations from the Henshin meta-model using the approach and supporting tool presented in [12,13]. In addition, we manually specified 15 complex edit operations available in the Henshin diagram editor, including the operations *convertCreateNodeToPreserveNode* covering edit steps (3) and (4) as well as *convertCreateEdgeToPreserveEdge* covering edit step (6) of our example. The Henshin specification of edit operation *convertCreateNodeToPreserveNode* is shown in Fig. 2, aggregating all change actions causing the low-level changes (4.a),(4.b),(4.c) and (5.a),(5.b),(5.c), respectively. Note that the multi-rule parts, indicated by the * symbol, are used to indicate optional rule actions; they are not applied when a root node of an EMF model is converted.

To present calculated differences in a user-friendly way, we integrated the Henshin diagram editor with SiLift's difference presentation UI. Fig. 3 shows how the difference obtained from comparing the rules *startColumn* and *extendColumn* is presented. The control window on the left displays the edit steps; selecting an edit step causes that its arguments are highlighted in the Henshin diagram(s) on the right.

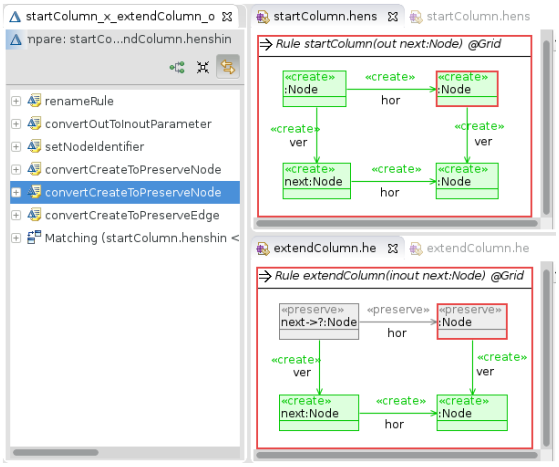

Fig. 3: Difference obtained from comparing Henshin transformation rules *startColumn* and *extendColumn* using SiLift.

Currently, a technical limitation when working with Henshin is that structural elements of a transformation rule must be equipped with UUIDs. These are used to (i) establish correspondences between elements in two versions of a rule and (ii) to identify graphical elements in Henshin diagrams.

4 Related Work

Many model differencing techniques were proposed in the last 15 years. The approaches surveyed in [10] focus on the matching step of the difference calculation. All of them suffer from the problem that only low-level changes are derived from a matching. More recently, the customization of the generic differencing engine has been addressed in EMF Compare [6], and adaptable techniques for lifting low-level changes to the level of edit operations have been proposed [14,15]. However, to the best of our knowledge, none of these approaches has yet been adapted to a graphical rule-based model transformation language. Moreover, edit operations for Henshin transformation rules can be quite complex, and their specification

needs a sophisticated language. Instead of defining a custom configuration language, SiLift uses Henshin for this purpose, a dedicated transformation language supporting in-place transformations in an expressive way. Finally, the set of available complex edit operations can be easily extended by Henshin tool users since they do not need to learn an additional configuration language.

5 Conclusion and Future Work

In this paper, we presented an adaptation of the SiLift model differencing framework to the Henshin model transformation language, and demonstrated its benefits compared to off-the-shelf model differencing tools producing rather low-level results. However, the provided high-level differencing support only serves as an initial step towards an effective management of Henshin transformations evolving into several versions and variants. In future work, we want to address further essential yet highly challenging versioning services on top of differencing, particularly the patching and updating of Henshin transformations.

References

1. T. Mens and P. Van Gorp, "A taxonomy of model transformation," *ENTCS*, vol. 152, 2006.
2. D. Strüber, J. Plöger, and V. Acreţoaie, "Clone detection for graph-based model transformation languages," in *ICMT*, 2016.
3. J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal, "Industrial adoption of model-driven engineering: Are the tools really the problem?" in *MoDELS*, 2013.
4. T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: advanced concepts and tools for in-place emf model transformations," in *MoDELS*, 2010.
5. D. Strüber, K. Born, K. D. Gill, R. Groner, T. Kehrer, M. Ohrndorf, and M. Tichy, "Henshin: A usability-focused framework for emf model transformation development," in *ICGT*, 2017.
6. C. Brun and A. Pierantonio, "Model differences in the eclipse modeling framework," *UPGRADE*, vol. 9, no. 2, 2008.
7. T. Kehrer, U. Kelter, M. Ohrndorf, and T. Sollbach, "Understanding model evolution through semantically lifting model differences with SiLift," in *ICSM*, 2012.
8. G. Varró, A. Schurr, and D. Varró, "Benchmarking for graph transformation," in *Symposion on Visual Languages and Human-Centric Computing*, 2005.
9. J. Bézivin, "On the unification power of models," *SoSym*, vol. 4, no. 2, 2005.
10. D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige, "Different models for model matching: An analysis of approaches to support model differencing," in *CVSM@ICSE*, 2009.
11. T. Kehrer, U. Kelter, and G. Taentzer, "A rule-based approach to the semantic lifting of model differences in the context of model versioning," in *ASE*, 2011.
12. T. Kehrer, G. Taentzer, M. Rindt, and U. Kelter, "Automatically deriving the specification of model editing operations from meta-models," in *ICMT*, 2016.
13. M. Rindt, T. Kehrer, and U. Kelter, "Automatic generation of consistency-preserving edit operations for MDE tools," in *Demos@MoDELS*, 2014.
14. P. Langer, M. Wimmer, P. Brosch, M. Herrmannsdörfer, M. Seidl, K. Wieland, and G. Kappel, "A posteriori operation detection in evolving software models," *Journal of Systems and Software*, vol. 86, no. 2, 2013.
15. D. E. Khelladi, R. Hebig, R. Bendraou, J. Robin, and M.-P. Gervais, "Detecting complex changes and refactorings during (meta) model evolution," *Information Systems*, vol. 62, 2016.