Scalability of Model Transformations: Position Paper and Benchmark Set

Daniel Strüber[1], Timo Kehrer[2], Thorsten Arendt[1,3], Christopher Pietsch[4], Dennis Reuling[4]

[1] Philipps-Universität Marburg, Germany
[2] Politecnico di Milano, Italy
[3] GFFT Innovationsförderung GmbH, Bad Vilbel, Germany
[4] Universität Siegen, Germany

Abstract. As model transformations are often considered the "heart and soul" of Model-Driven Engineering (MDE), the scalability of model transformations is vital for the scalability of MDE approaches as a whole. The existing research on scalable transformations has largely focused on performance behavior if the involved input models grow in size. In this work, we address a second key dimension for the practical scalability of model transformations: The effect as the transformation specification itself grows larger. We outline a number of challenges related to large model transformations, specifically affecting the quality concerns maintainability and performance. We introduce three model transformation benchmarks and discuss how they are affected by these challenges. The transformation rule sets in these benchmarks have acted as an evaluation basis in our previous work. Our objective is to establish a community benchmark set to compare model transformation approaches with respect to the aforementioned quality concerns.

1 Introduction

Over the recent years, *Model-Driven Engineering* (MDE) has started to grow into a mature software engineering discipline. To facilitate the development of complex software systems, MDE envisions the use of abstraction and automation principles: Models are used to provide an abstract specification of a software system, thereby enabling to tame complexity during software design. This specification is automatically refined to a running software systems using *model transformations*, thus enabling to reduce implementation effort. A large variety of modeling and model transformation languages has emerged to address the heterogeneity in the involved software domains and transformation scenarios.

As MDE is increasingly applied in industrial large-scale scenarios, a number of limitations of the existing MDE techniques and tools has been revealed. One of the key problem areas concerns the scalability of model transformation tools. In this area, existing research has mostly focused on scalability as the input models of transformations grow in size. Kolovos et al. summarize the goal of the existing works as *"advancing the state of the art in model querying and transformation* *tools so that they can cope with large models (in the order millions of model elements)"* [1]. To tackle this goal, a number of performance optimizations as well as new execution modes, such as incremental [2] or concurrent [3] model transformations, have been provided to the community.

Inspired by our experience of developing model transformations within several research projects, in this paper we argue that a second key dimension of scalability has been neglected so far: The scalability of the transformation specification. In particular, we point out that the performance of a model transformation is likely to deteriorate with the size and complexity of its specification. To make matters worse, the models in a particular transformation scenario *and* the transformation specification may be affected by scalability issues at the same time, in combination leading to a more drastic scalability challenge. Furthermore, the size of a specification might also challenge its maintainability, even to the point that it becomes unusable when viewed and edited with the default tools.

We focus on the technical scope of rule-based model transformations. In this domain, available techniques and tools are affected by the size of individual rules as well as the size of the overall rule set. Multiple factors contribute to the emergence of large rules and rule sets: (i) The size of these artifacts can reflect the complexity of the intended transformations. Model transformations are a particular kind of software, the development of which is generally complicated by the complexity of the imposed requirements. (ii) The size of individual rules might grow due to technical reasons. For instance, the UML meta-model is infamous for its size and complexity that leads to complicated rules even when expressing transformations that are rather simple on a conceptual level [4]. (iii) A common situation involves the management of *families of rules*, i.e. sets of rules that exhibit a high degree of commonalities.While the built-in concepts provided by the available transformation languages can help to deal with this issue to some extent, we found these concepts insufficient in several cases as described in this paper.

The contribution of this work is twofold. First, to illustrate our position, we explore the scalability issues encountered during our past experiences and relate them to existing experiences from the literature. Second, we provide a set of benchmark scenarios affected by these issues. We have used some of these scenarios as an evaluation basis in our recent work [5, 6]. Our aim is to make them available for other researchers. By providing the rule sets with a systematic description, as part of a publicly available repository, our goal is to facilitate the comparison of model transformation approaches with respect to their scalability.

The rest of this paper is structured as follows. In Sect. 2, we outline the above mentioned scalability challenges in more detail. Sect. 3 introduces two benchmark kinds to assess the improvement of related quality aspects. In Sect. 4, we present and discuss our benchmark set. In Sect. 5 we discuss related work, and Sect. 6 concludes the paper.

2 Scalability Challenges

Maintainability is one of the main quality goals during the development of a model transformation [7–9]. It refers to the capability to modify the transformation after its initial deployment when it has to be adapted, e.g., to address changing requirements or to remove defects.

A necessary prerequisite for maintainability is understandability: If the specification is difficult to understand, the time required to perform a change as well as the risk creating defects while doing so increases. Intuitively, the understandability of a transformation system is related to two separate dimensions of size: the number of rules and the size of individual rules. In a large set of rules, the difficulty lies in pinpointing the target location for an intended change. In turn, a large size of individual rules may stretch the limitations of visual representations – large diagrams may not scale. This intuition has been confirmed experimentally for the scope of class diagrams, where a detrimental effect of larger diagram sizes on understandability was determined by Störrle [10].

Another obstacle to maintainability concerns the scalability of the model transformation editor. The existing transformation editors were often designed and tested for transformation systems of small to medium scale. In a rule set with hundreds of rules and hundreds of elements per rule, the response time during loading, navigating, and changing rules might be substantial, thus prohibiting the transformation to be edited in an efficient manner. This obstacle can be addressed in two ways: Either by improving the scalability of the editor or by providing a more compact representation of the overall rule set.

Performance is another key concern of model transformation systems [7–9]. In particular, in the case of graph-based model transformation languages, the execution of a transformation entails the NP-complete sub-graph isomorphism problem, leading to substantial execution times as input models and rules grow.

In practice, the performance of a model transformation depends on its execution mode. Despite considerable progress on particular execution modes such as incremental [2] and parallel transformations [3], the standard mode remains *batch transformation*. In batch mode, all rules in the rule set are applied as long as one of them is applicable. This mode is often found in translation, simulation, and refactoring scenarios. In a batch transformation, each rule increases the computational effort of the transformation system; the larger the rule set becomes, the harder it is to keep it tractable. For instance, Blouin et al. [11] report on a case where a transformation engine was unable to execute a transformation system with 250 rules. Such issues often lead to ad-hoc solutions to reduce the size of the rule set as dedicated language-level support is widely unavailable [12].

3 Benchmark Kinds

Based on the identified scalability areas, we identify two different benchmark kinds for model transformation approaches (see Fig. 1). The distinction of these benchmark kinds allows to study maintainability and performance disjoint from

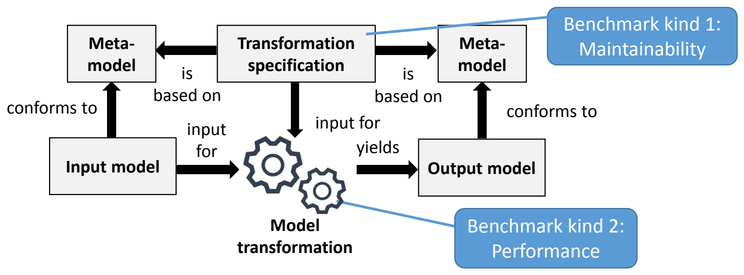

Fig. 1. Benchmark kinds: Overview

each other. While large specifications may be generally affected by both issues, a single approach to address both issues might not necessarily be most effective: Another approach is to refactor the rules for improved maintainability and to apply a background performance optimization before executing the rule set [5].

Benchmark kind 1: Benchmarking for Maintainability (M).

- **Goal.** Deriving a rule set with improved maintainability properties.
- **Scope.** We consider the maintainability property *compactness*.
- **Rationale.** Smaller rule sets and rules may be easier to read, entail less individual edits to perform a single change, and show better performance behavior when viewed and edited in standard editors.
- **Metrics.** Accumulative number of rule elements required to specify the entire transformation.

Benchmark kind 2: Benchmarking for Performance (P).

- **Goal.** Deriving a rule set with improved performance properties.
- **Scope.** We consider the performance property *execution time*.
- **Rationale.** Lower execution times of the included model transformation may lead to general improvements in the existing MDE approaches.
- **Metrics.** Accumulative execution time on a set of given input models.

Correctness check. In both benchmark kinds, a highly desirable feature would be a correctness check: The considered rule set should show the same behavior before and after the restructuring. A pragmatic approach to ensure correctness is to apply both transformations to a set of test models, expecting equal results. This approach provides no sufficient, but at least a necessary correctness criterion. The development of this check is ongoing work.

4 Benchmarks

Our benchmark set, summarized in Table 1, comprises three benchmarks, called *Edit Rules*, *Recognition Rules*, and *Translation Rules*, respectively. These benchmarks will be described in more detail in the remainder of this section.

Benchmark	**Kind**	**Scenario**	**#Rules**	**#N**	**#E**	**#A**	**#Models**
Edit Rules	M	FM	53	708	831	243	n.a.
	M	UML	1404	6865	2721	1433	n.a.
Recognition Rules	P	FM	53	1773	2853	717	125
	P	UML	1404	26162	30777	14816	19
Translation Rules	M+P	OCL2NGC	54	2259	2389	1142	10

Table 1. Overview of the benchmark set. For each benchmark, the columns give the benchmark name, benchmark kind, scenario, number of rules, nodes, edges, attributes, and provided input models.

4.1 Edit Rules

Context and objectives Edit commands as offered by visual editors and modern model refactoring tools are typical forms of edit operations on models. Many tools of an MDE tool suite must be customized to the way how models can be edited, model versioning [13, 14], refactoring tools [15] as well as model mutation [16] being examples of this. Therefore, explicit specifications of the available edit operations are required. Rule-based in-place model transformations are well-suited for that purpose [17–19]. However, developing and maintaining a set of *edit rules* is challenging. Firstly, edit rules can become large in case of complex model restructuring operations. Secondly, the set of edit which required to specify every possible model modification becomes huge in case of comprehensive languages such as the UML. Thus, the specification of edit operations is an adequate benchmark addressing the maintainability of model transformation rules. We selected the following two scenarios in which suitable edit operations have been specified:

FM. The first scenario refers to the editing of feature models [20], a widely used variability modeling approach in software product line (SPL) engineering. The selected edit operations have been defined in [21]. The main objective of this work is to define edit operations which can be classified w.r.t. their semantic impact on the set of valid feature combinations. A *refactoring* leaves this set unchanged, a generalization (specialization) enlarges (shrinks) the set of valid feature combinations. Such edit operations are often complex restructurings defining several non-trivial application conditions.

UML. The second scenario addresses the editing of UML models. Due to the complexity of the UML meta-model, edit operations on UML models turn out to be rather complex when being specified based on the abstract syntax [4]. Restricting the navigability of an association to one end is an example of this [18]. In this benchmark, the selected UML edit operations refer to those parts of the UML which are used in [22], a case study on using a UML-based approach for modeling the Barbados Crash Management System (bCMS).

Meta-Models In the FM scenario, the selected edit rules are specified over the meta-model defined in [21]. Concerning the UML, the selected edit rules are defined over the UML standard meta-model defined by the OMG [23]. The subset being relevant in this benchmark is given by the UML models of the bCMS case study which uses class diagrams, sequence diagrams and statecharts. In sum, the relevant subset includes 83 meta-classes, together with their various relations, out of the 243 meta-classes of the standard UML meta-model.

Rules The feature model edit rule set comprises the 53 complex restructuring operations on feature models defined in [21]. The largest rule in this rule set comprises 74 nodes, 146 edges, and 14 attributes.

The UML edit rule set comprises 1404 edit rules, all of them specifying edit operations which can be considered elementary from a user's point of view in the sense that they cannot be split into smaller edit operations being applicable to a model in a meaningful way. Consequently, the individual rules in this rule set are considerably smaller than the feature model rules, the largest one comprising 9 nodes, 11 edges, and 14 attributes.

4.2 Edit Operation Recognition Rules

Context and objectives To optimally support continuous model evolution, sophisticated tool support for model version management is needed, the calculation of a difference between two versions of a model being one of the most essential tool functions. Instead of reporting model differences to modelers element-wise, their grouping into semantically associated change sets helps in understanding model differences [18]. Edit operations as used in our first benchmark are the concept of choice to group such change sets. [18] presents an approach which automatically translates specifications of edit operations into so called *recognition rules* being used by an algorithm which recognizes edit operations in a low-level difference of two model versions. Essentially, this algorithm searches for all matches of each recognition rule in a given difference. This is a computational expensive pattern matching problem. Thus, the optimization of a recognition rule set is a primary concern and an adequate performance benchmark.

Meta-Models and Models Concerning the modeling languages comprised by this benchmark, we selected the same languages along with the corresponding meta-models as in our first benchmark.

In general, suitable low-level model differences on which to apply the edit operation recognition rules are obtained as follows: Given (i) a pair of input models which can be considered to be revisions of each other, and (ii) a model matcher which determines the corresponding elements in both versions, the differencing engine presented in [18] creates a low-level difference which can be synthesized as a "difference model". It basically defines five types of changes which can be observed in a low-level difference, namely the insertion/deletion of objects/references as well as attribute value changes. For this benchmark, we selected the following pairs of models and matchers.

FM. For feature models, we use the synthetically created differencing scenarios presented in [21]. In sum, we have 125 pairs of feature models of different characteristics and varying sizes, ranging from 100 up to 500 features per model. To calculate the 125 low-level differences, a dedicated matcher for pairs of feature models [21] is used to determine corresponding elements.

UML. In the UML scenario, we selected the models provided by the bCMS case study. In fact, bCMS defines not only a single model but is designed as SPL. We used 20 models representing valid instances of this SPL, one core model and 19 additional variants. By differencing each variant with the core, we produced 19 low-level differences. To determine corresponding elements, we used a dedicated matcher that exploits persistent unique identifiers attached to elements.

Rules As described in [18], recognition rules can be automatically generated from their corresponding edit rules. Thus, it was a natural choice to generate the rules of this benchmark from the edit rules of our first benchmark. Consequently, we have 53 recognition rules for the FM and 1404 for the UML scenario. As seen in Table 1, recognition rules are generally larger than their edit rule counterparts. The largest recognition rule comprises 77 nodes, 154 edges, and 20 attributes in the FM case and 73 nodes, 81 edges, and 40 attributes in the UML case.

4.3 Constraint Translation Rules

Context and objectives OCL is the standard technology to restrict the set of models that can be created from a given meta-model. A meta-model and its OCL constraints together provide a declarative definition of a modeling language. Yet in some situations, a constructive definition of the language may be required, e.g., to systematically enumerate all possible models for verification purposes. Such a constructive approach is provided by graph grammars [24]. To integrate these metamodel- and grammar-based approaches to language definition, [25] describes a migration technique that transforms the OCL constraints contained in a meta-model to application conditions in grammar rules, using nested graph constraints (NGCs) as an intermediate form.

Models and Meta-Models As input models for our benchmark, we used an assortment of ten OCL constraints designed for a large coverage of applicable rules. The size of the input models, comprising meta-models with embedded constraints as well as the OCL standard library, containing operators and literals referenced by the constraints, ranges from 1832 to 1854 model elements

OCL2NGC is an exogenous model transformation. Its implementation involves three meta-models: The OCL pivot meta-model[1] acts as source meta-model. The NGC meta-model, provided as part of the benchmark, acts as target meta-model. The trace meta-model[2] acts as a language-agnostic auxiliary meta-model to manage the correspondence between source and target model elements.

[1] https://wiki.eclipse.org/MDT/OCL/Pivot_Model

[2] https://wiki.eclipse.org/Henshin_Trace_Model

Rules Our implementation of the transformation described in [25] comprises a model transformation system with 54 rules. In addition, a control flow to guide the rule execution is specified, using the Henshin concept of transformation units. The main performance bottleneck, which we call bottleneck rule subset (BRS), is a subset of 36 rules that are applied in batch mode, i.e., as long as one of them can be matched.

Benchmark Repository

We provide the benchmark set together with an evaluation framework at GitHub: `https://github.com/dstrueber/bigtrafo`

Using the benchmark set. For each scenario, we provide a skeleton demonstrating the application of our evaluation framework to the contained rule set. For example, a class called `OclBenchmark`[3] contains the skeleton for the constraint translation scenario. Users can modify this skeleton to customize it to their own or another model transformation approach.

Contributing to the benchmark set. We encourage the submission of additional benchmarks and scenarios by other researchers, ideally in a form that follows the basic directory and file structure of the available benchmarks. Such submissions are conveniently supported by GitHub's pull request feature.

5 Related Work

A number of benchmark sets has been introduced in the literature. Benelallam et al. [26] propose a benchmark set focusing on scalability of queries and transformations to large input models. A seminal benchmark paper for graph transformation languages has been contributed by Varró et al. [27]. Bergmann et al. [28] propose a benchmark set for incremental transformations. None of these benchmarks addresses the scalability of transformations specifications.

Izsó et al. [29] propose a MDE benchmark framework targeting a large variety of use cases, such as model validation, transformation, and code generation. The main idea is to define benchmarks in a systematic way by assembling them from reusable benchmark primitives, such as metrics evaluation. The authors also provide some emerging results, some of them pointing in the direction of the stance maintained in our paper. An integration of our benchmark set with this framework is a feasible direction for future work.

6 Conclusion

In this work, we address the scalability of model transformation specifications, focusing on the quality goals *performance* and *maintainability*. We provide a publicly available set of benchmark scenarios from our recent work, encouraging

[3] https://github.com/dstrueber/bigtrafo/blob/master/de.bigtrafo.benchmark/src/de/bigtrafo/benchmark/ocl/OclBenchmark.java

other researchers to compare their transformation tools and contribute additional benchmarks. In the future, we aim to explore the relationship between large models and transformation specifications further to investigate the following research question: *when do performance optimizations complement each other and when does one optimization dominate the other*?

References

1. D. S. Kolovos, L. M. Rose, N. Matragkas, R. F. Paige, E. Guerra, J. S. Cuadrado, J. De Lara, I. Ráth, D. Varró, M. Tisi, and J. Cabot, "A research roadmap towards achieving scalability in model driven engineering," in *BigMDE Workshop on Scalability in Model Driven Engineering.* ACM, 2013, p. 2.
2. F. Jouault and M. Tisi, "Towards incremental execution of atl transformations," in *Proc. of Int. Conference on Model Transformations.* Springer, 2010, pp. 123–137.
3. L. Burgueño, J. Troya, M. Wimmer, and A. Vallecillo, "On the concurrent execution of model transformations with linda," in *BigMDE Workshop on Scalability in Model Driven Engineering.* ACM, 2013, p. 3.
4. V. Acretoaie, H. Störrle, and D. Strüber, "Transparent model transformation: Turning your favourite model editor into a transformation tool," in *Int. Conference on Model Transformations.* Springer, 2015, pp. 283–298.
5. D. Strüber, J. Rubin, T. Arendt, M. Chechik, G. Taentzer, and J. Plöger, "RuleMerger: Automatic Construction of Variability-Based Model Transformation Rules," in *Int. Conference on Fundamental Approaches to Software Engineering.* Springer, 2016, pp. 122–140.
6. D. Strüber, J. Rubin, M. Chechik, and G. Taentzer, "A Variability-Based Approach to Reusable and Efficient Model Transformations," in *Int. Conference on Fundamental Approaches to Software Engineering.* Springer, 2015, pp. 283–298.
7. E. Syriani and J. Gray, "Challenges for addressing quality factors in model transformation," in *Int. Conference on Software Testing, Verification and Validation.* IEEE, 2012, pp. 929–937.
8. M. Wimmer, S. M. Perez, F. Jouault, and J. Cabot, "A catalogue of refactorings for model-to-model transformations." *Journal of Object Technology*, vol. 11, no. 2, pp. 1–40, 2012.
9. C. M. Gerpheide, R. R. Schiffelers, and A. Serebrenik, "A bottom-up quality model for qvto," in *Int. Conference on the Quality of Information and Communications Technology.* IEEE, 2014, pp. 85–94.
10. H. Störrle, "On the Impact of Layout Quality to Understanding UML Diagrams: Size Matters," in *Int. Conference on Model Driven Engineering Languages and Systems.* Springer, 2014, pp. 518–534.
11. D. Blouin, A. Plantec, P. Dissaux, F. Singhoff, and J.-P. Diguet, "Synchronization of models of rich languages with triple graph grammars: An experience report," in *Int. Conference on Model Transformation.* Springer, 2014, pp. 106–121.
12. A. Kusel, J. Schönböck, M. Wimmer, G. Kappel, W. Retschitzegger, and W. Schwinger, "Reuse in model-to-model transformation languages: are we there yet?" *Software & Systems Modeling*, vol. 14, no. 2, pp. 537–572, 2013.
13. T. Kehrer, U. Kelter, M. Ohrndorf, and T. Sollbach, "Understanding model evolution through semantically lifting model differences with silift," in *Int. Conference on Software Maintenance.* IEEE, 2012, pp. 638–641.

14. T. Kehrer, U. Kelter, and D. Reuling, "Workspace updates of visual models," in *Int. Conference on Automated Software Engineering.* ACM, 2014, pp. 827–830.
15. T. Arendt and G. Taentzer, "A tool environment for quality assurance based on the eclipse modeling framework," *Automated Software Engineering*, vol. 20, no. 2, pp. 141–184, 2013.
16. D. Reuling, J. Bürdek, S. Rotärmel, M. Lochau, and U. Kelter, "Fault-based Product-line Testing: Effective Sample Generation Based on Feature-diagram Mutation," in *International Software Product Line Conference.* ACM, 2015, pp. 131–140.
17. D. S. Kolovos, R. F. Paige, F. A. Polack, and L. M. Rose, "Update transformations in the small with the epsilon wizard language," *Journal of Object Technology*, 2003.
18. 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 *Int. Conference on Automated Software Engineering.* IEEE, 2011, pp. 163–172.
19. T. Mens, "On the use of graph transformations for model refactoring," in *Generative and transformational techniques in software engineering.* Springer, 2006, pp. 219–257.
20. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and S. A. Peterson, "Feature Oriented Domain Analysis (FODA)," CMU, Tech. Rep., 1990.
21. J. Bürdek, T. Kehrer, M. Lochau, D. Reuling, U. Kelter, and A. Schürr, "Reasoning about product-line evolution using complex feature model differences," *Journal of Automated Software Engineering*, pp. 1–47, 2015.
22. A. Capozucca, B. Cheng, N. Guelfi, and P. Istoan, "OO-SPL modelling of the focused case study," in *International Workshop on Comparing Modeling Approaches*, 2011. [Online]. Available: https://orbilu.uni.lu/bitstream/10993/12572/1/bCMS-SPL-complete-submit.pdf
23. Object Management Group, "Uml 2.4.1 superstructure specification," OMG Document Number: formal/2011-08-06, 2011.
24. H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation: Volume 2: Applications, Languages and Tools.* world Scientific, 1999.
25. T. Arendt, A. Habel, H. Radke, and G. Taentzer, "From Core OCL Invariants to Nested Graph Constraints." in *Int. Conference on Graph Transformation.* Springer, 2014, pp. 97–112.
26. A. Benelallam, M. Tisi, I. Ráth, B. Izso, and D. Kolovos, "Towards an open set of real-world benchmarks for model queries and transformations," in *BigMDE Workshop on Scalability in Model Driven Engineering*, 2014.
27. G. Varró, A. Schürr, and D. Varró, "Benchmarking for graph transformation," in *IEEE Symposium on Visual Languages and Human-Centric Computing.* IEEE, 2005, pp. 79–88.
28. G. Bergmann, Á. Horváth, I. Ráth, and D. Varró, "A benchmark evaluation of incremental pattern matching in graph transformation," in *Int. Conference on Graph Transformation.* Springer, 2008, pp. 396–410.
29. B. Izsó, G. Szárnyas, I. Ráth, and D. Varró, "Mondo-sam: A framework to systematically assess mde scalability." in *BigMDE Workshop on Scalability in Model Driven Engineering*, 2014, pp. 40–43.