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 as 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 the quality concerns of maintainability and performance. We then introduce three model transformation benchmarks and discuss how they are affected by these challenges. 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. This specification is automatically refined to a running software system using *model transformations*, thus enabling a reduced implementation effort. A large variety of modeling and model transformation languages has emerged to address the heterogeneity in the involved software domains and scenarios.

As MDE is increasingly applied in industrial large-scale scenarios, various limitations of the existing techniques and tools have been revealed. One of the key problem areas concerns the scalability of model transformations. 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 of millions of model elements)"* [1]. To this end, a number of performance optimizations and new execution modes, such as incremental [2,3] or concurrent [4] model transformations, have been provided to the MDE community.

Inspired by our experience of developing model transformations for 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, input models *and* the transformation specification may be affected by scalability issues at the same time, a combination that leads 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. Techniques and tools in this domain 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: The size of these artifacts can reflect the *essential complexity* of the intended transformation; model transformations are a particular kind of software, the development of which is generally complicated by the complexity of the imposed requirements. In addition, multiple factors contribute to *accidental* complexity in model transformations: The size of individual rules might reflect the size of the involved meta-models. For instance, the UML metamodel is infamous for its size and complexity that leads to complicated rules even when expressing transformations that are simple on a conceptual level [5]. Furthermore, a common situation involves *families of rules*, i.e. sets of rules that exhibit a high degree of commonalities. While some transformation languages offer built-in concepts to manage families of rules 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 of rule sets affected by these issues. We have used some of these scenarios as an evaluation basis in our recent work [6,7,8,9]. 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. We provide the benchmark set together with usage instructions at GitHub: `https://github.com/dstrueber/bigtrafo`

The rest of this paper is structured as follows. In Sect. 2, we outline the scalability challenges in more detail. Sect. 3 introduces two benchmark kinds to assess related quality aspects. We present our benchmark set in Sect. 4, discussing its limitations in Sect. 5. In Sects. 6 and 7, we discuss related work and conclude.

2 Scalability Challenges

In this section, we discuss scalability challenges related to large transformations as observed in our own work and in the literature.

Maintainability is one of the main quality goals during the development of a model transformation [10,11,12]. 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 of 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, large individual rules lead to large diagrams that may not scale to the cognitive capacities of the human mind. A detrimental effect of large diagram size on understandability has been found for the scope of class diagrams [13].

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 quality aspect of model transformations [10,11,12]. 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 [2,3] and parallel transformations [4], the standard mode remains *batch transformation* [2]. 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. [14] report on a case where a transformation engine was unable to execute a transformation system with 250 rules. As language-level support for such issues is widely unavailable [15], the size of the rule set if often reduced using ad-hoc solutions.

3 Benchmark Kinds

Based on these challenges, we identify two different benchmark kinds for model transformation approaches (see Fig. 1). The distinction of these benchmark kinds allows us to study maintainability and performance disjoint from each other. A single approach to address both issues might not necessarily be most effective: Another approach is to maintain rules in a form that maximizes their maintainability and to apply a performance optimization before executing the rules [6].

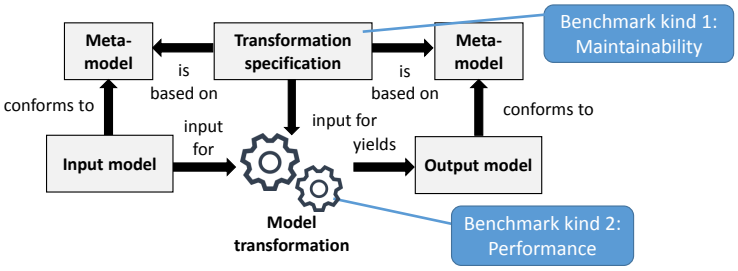

Fig. 1. Benchmark kinds: Overview

Benchmark kind 1: Benchmarking for Maintainability (M).

- **Goal.** Specifying a transformation with beneficial 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, number of rules, element-per-rule ratio required to specify the entire transformation.

Benchmark kind 2: Benchmarking for Performance (P).

- **Goal.** Specifying a transformation with beneficial 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.

Usage. To use the benchmark set, a transformation tool designer needs to follow three steps: First, implement a new solution to one of the provided benchmarks. Second, validate the solution. Third, measure the mentioned quality attributes.

(1) Implementation. To guide benchmark users during the implementation of new solutions, each benchmark has three specifications: A textual *high-level specification* provided in the research paper from which the transformation was obtained. A *reference solution* based on the Henshin model transformation language [16]. A *black-box specification* based on a set of input models and the output models produced by applying the Henshin solution to these models.

(2) Validation. The provided black-box specifications can be used to validate the correctness of new solutions. To this end, the new solution needs to be applied to the provided input models. Afterwards, the produced output models can be compared against the ones provided with the benchmark. A solution is correct if the produced output models are isomorphic to the provided ones.

(3) Measurement. To measure compactness, we provide a measurement framework in our repository. This framework is currently customized for Henshin rules;

Benchmark	Kind	Scenario	#Rules	#N	#E	#A	#Models
Edit Rules	M	FM	58	803	945	272	108
	M	UML	1404	6865	2721	1433	1766
Recognition Rules	P	FM	58	3758	6580	1217	26
	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, and attributes.

measurements for further transformation languages can be established by implementing new instantiations. Such new instantiations of the measurement framework can account for custom language features, such as support for modularity. Execution time is measured in terms of the time required to execute the transformation on all input models. Reference times are provided in the repository.

Mailing list. To support communication about our benchmarks, we maintain a mailing list at `http://www.freelists.org/list/bigtrafo`. The purpose of this mailing list is twofold: First, it provides a platform to announce new solutions and benchmark results. Second, this platform can be used to clarify questions about the benchmarks and particular solutions.

4 Benchmarks

Our benchmark set, summarized in Table 1, comprises three benchmarks. The *Edit Rules* and *Recognition Rules* benchmarks inherently focus on the maintainability and performance concerns, respectively, while *Translation Rules* is a benchmark where these quality concerns are both relevant, thus allowing the relationship between both concerns to be studied. The statistics given in the table indicate the size of each transformation in the provided Henshin specification.

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 [17,18], refactoring tools [19] as well as model mutation [20] 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 [21,22,23]. 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 being required to specify every possible model modification becomes huge in case of comprehensive languages such as UML. Thus, the specification of edit operations is an adequate benchmark addressing the maintainability of model transformation rules. We selected two scenarios in which suitable edit operations have been specified:

FM. This scenario refers to the editing of feature models [24], a widely used variability modeling approach in software product line (SPL) engineering. The selected edit operations have been defined in [25]. Groups of edit operations are distinguished 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 with several non-trivial application conditions.

UML. This 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 [5]. Restricting the navigability of an association to one end is an example of this [22]. In this benchmark, the selected UML edit operations refer to those parts of the UML which are used in [26], a case study on using a UML-based approach for modeling the Barbados Crash Management System (bCMS).

Meta-Models and Models. In the FM scenario, the selected edit rules are specified over the meta-model defined in [25]. Concerning the UML scenario, the selected edit rules are defined over the UML standard meta-model defined by the OMG [27]. 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. We derived test models for both rule sets. Deriving application examples from the rule structures yielded at least one test model per rule.

Rules. The FM edit rule set comprises the 58 complex restructuring operations on feature models defined in [25]. 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: these rules cannot be split into smaller edit operations being applicable to a model in a meaningful way. Consequently, the individual rules in the UML rule set are considerably smaller than the FM rules, the largest one comprising 9 nodes, 11 edges, and 14 attributes. To support their step-wise comprehension and re-implementation by benchmark users, both rule sets exhibit an organization into groups of semantically related rules.

4.2 Edit Operation Recognition Rules

Context and objectives. To support continuous model evolution, sophisticated tool support for model version management is needed. One of the most essential tool functions is the calculation the difference between two versions of a model. Instead of reporting model differences to modelers element-wise, their grouping into semantically related change sets helps in understanding model differences [22]. Edit operations as used in our first benchmark are the concept of choice to group such change sets. [22] presents an approach which automatically translates specifications of edit operations into *recognition rules*. These rules are 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 computationally 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 [22] 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 [25]. We have 26 revision pairs of feature models of different characteristics and varying sizes, ranging from 100 up to 500 features per model. To calculate the low-level differences, we used dedicated matcher for pairs of feature models [25] 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 an entire 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 element identifiers.

Rules. As described in [22], 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 58 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 348 nodes, 769 edges, and 57 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. Modeling languages are usually specified in a declarative manner, using a meta-model with well-formedness constraints expressed in the Object Constraint Language (OCL). Yet in some situations, a constructive definition of a modeling language may be required instead, e.g., to systematically enumerate all possible models for verification purposes. Such a constructive approach is provided by graph grammars [28]. To support the generation of a graph grammar from an existing meta-model, we devised a constraint translator that can transform OCL constraints to nested graph constraints (NGCs) [29], which can then be further translated to application conditions in grammar rules.

Models and Meta-Models. As input models for our benchmark, we used 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, 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.

Rules. Our implementation of the transformation described in [29] comprises a model transformation system with 54 rules. In addition, a control flow to guide the rule execution is specified using Henshin's concept of transformation units. The main performance bottleneck is a subset of 36 rules that are applied in batch mode, i.e., as long as one of them can be matched.

5 Limitations and Further Extensions

Since we address the scalability of model transformations, our maintainability benchmark focuses on *size* – the size of individual rules and entire rule sets. While we discuss the detrimental effect of size to maintainability in Sec. 2, size is by no means the only relevant maintainability aspect. A promising direction for future work is to consider additional maintainability metrics from research on software quality. Moreover, we do not study the effect of rule size on the performance of the transformation editor explicitly. While it is our experience that editor performance improves as the transformation size decreases, a dedicated measurement to quantify this performance gain is needed. Finally, the performance benchmark is limited to execution time. Considering additional properties, such as memory or even energy consumption, might lead to interesting insights.

6 Related Work

A number of benchmark sets for queries and transformations has been introduced in the literature. Benelallam et al. [30] propose a benchmark set focusing on scalability to large input models. A seminal benchmark paper for graph transformation languages has been contributed by Varró et al. [31]. Bergmann et al. [32] propose a benchmark set for incremental transformations. None of these benchmarks addresses the scalability of the transformation specification.

Izsó et al. [33] 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. The authors also provide a number of emerging results, some of them pointing in the direction we explore in this paper. An integration of our benchmark set with this framework is feasible as future work.

[1] `https://wiki.eclipse.org/OCL/Pivot_Model`
[2] `https://wiki.eclipse.org/Henshin_Trace_Model`

7 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 benchmark set, encouraging other researchers to compare their transformation tools and contribute additional benchmarks. In the future, we aim to explore the relationship between performance optimizations between large models and transformation specifications further.

References

1. Kolovos, D.S., Rose, L.M., Matragkas, N., Paige, R.F., Guerra, E., Cuadrado, J.S., De Lara, J., Ráth, I., Varró, D., Tisi, M., Cabot, J.: A research roadmap towards achieving scalability in model driven engineering. In: BigMDE Workshop on Scalability in Model Driven Engineering, ACM (2013) 2
2. Johann, S., Egyed, A.: Instant and incremental transformation of models. In: Int. Conference on Automated Software Engineering, IEEE Computer Society (2004) 362–365
3. Jouault, F., Tisi, M.: Towards incremental execution of ATL transformations. In: Proc. of Int. Conference on Model Transformations. Springer (2010) 123–137
4. Burgueño, L., Troya, J., Wimmer, M., Vallecillo, A.: On the Concurrent Execution of Model Transformations with Linda. In: BigMDE Workshop on Scalability in Model Driven Engineering, ACM (2013) 3
5. Acretoaie, V., Störrle, H., Strüber, D.: Transparent model transformation: Turning your favourite model editor into a transformation tool. In: Int. Conference on Model Transformations, Springer (2015) 283–298
6. Strüber, D., Rubin, J., Arendt, T., Chechik, M., Taentzer, G., Plöger, J.: RuleMerger: Automatic Construction of Variability-Based Model Transformation Rules. In: Int. Conference on Fundamental Approaches to Software Engineering, Springer (2016) 122–140
7. Strüber, D., Rubin, J., Chechik, M., Taentzer, G.: A Variability-Based Approach to Reusable and Efficient Model Transformations. In: Int. Conference on Fundamental Approaches to Software Engineering, Springer (2015) 283–298
8. Strüber, D.: Model-Driven Engineering in the Large: Refactoring Techniques for Models and Model Transformation Systems. PhD thesis, Philipps-Universität Marburg, Germany (2016)
9. Strüber, D., Plöger, J., Acretoaie, V.: Clone detection for graph-based model transformation languages. In: Int. Conference on Theory and Practice of Model Transformations. (2016) 191–206
10. Syriani, E., Gray, J.: Challenges for addressing quality factors in model transformation. In: Int. Conference on Software Testing, Verification and Validation, IEEE (2012) 929–937
11. Wimmer, M., Perez, S.M., Jouault, F., Cabot, J.: A catalogue of refactorings for model-to-model transformations. Journal of Object Technology **11**(2) (2012) 1–40
12. Gerpheide, C.M., Schiffelers, R.R., Serebrenik, A.: A bottom-up quality model for QVTO. In: Int. Conference on the Quality of Information and Communications Technology, IEEE (2014) 85–94
13. Störrle, H.: On the Impact of Layout Quality to Understanding UML Diagrams: Size Matters. In: Int. Conference on Model Driven Engineering Languages and Systems, Springer (2014) 518–534
14. Blouin, D., Plantec, A., Dissaux, P., Singhoff, F., Diguet, J.P.: Synchronization of models of rich languages with triple graph grammars: An experience report. In: Int. Conference on Model Transformation. Springer (2014) 106–121

15. Kusel, A., Schönböck, J., Wimmer, M., Kappel, G., Retschitzegger, W., Schwinger, W.: Reuse in model-to-model transformation languages: are we there yet? Software & Systems Modeling **14**(2) (2013) 537–572
16. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: advanced concepts and tools for in-place EMF model transformations. In: Model Driven Engineering Languages and Systems. Springer (2010) 121–135
17. Kehrer, T., Kelter, U., Ohrndorf, M., Sollbach, T.: Understanding model evolution through semantically lifting model differences with SiLift. In: Int. Conference on Software Maintenance, IEEE (2012) 638–641
18. Kehrer, T., Kelter, U., Reuling, D.: Workspace updates of visual models. In: Int. Conference on Automated Software Engineering, ACM (2014) 827–830
19. Arendt, T., Taentzer, G.: A tool environment for quality assurance based on the Eclipse Modeling Framework. Automated Software Engineering **20**(2) (2013) 141–184
20. Reuling, D., Bürdek, J., Rotärmel, S., Lochau, M., Kelter, U.: Fault-based Product-line Testing: Effective Sample Generation Based on Feature-diagram Mutation. In: Int. Software Product Line Conference, ACM (2015) 131–140
21. Kolovos, D.S., Paige, R.F., Polack, F.A., Rose, L.M.: Update Transformations in the Small with the Epsilon Wizard Language. Journal of Object Technology (2003)
22. Kehrer, T., Kelter, U., Taentzer, G.: 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) 163–172
23. Mens, T.: On the use of graph transformations for model refactoring. In: Generative and transformational techniques in software engineering. Springer (2006) 219–257
24. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, S.A.: Feature Oriented Domain Analysis (FODA). Technical report, CMU (1990)
25. Bürdek, J., Kehrer, T., Lochau, M., Reuling, D., Kelter, U., Schürr, A.: Reasoning about product-line evolution using complex feature model differences. Journal of Automated Software Engineering (2015) 1–47
26. Capozucca, A., Cheng, B., Guelfi, N., Istoan, P.: OO-SPL modelling of the focused case study. In: Int. Workshop on Comparing Modeling Approaches. (2011)
27. Object Management Group: UML 2.4.1 Superstructure Specification. OMG Document Number: formal/2011-08-06 (2011)
28. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformation: Volume 2: Applications, Languages and Tools. world Scientific (1999)
29. Arendt, T., Habel, A., Radke, H., Taentzer, G.: From Core OCL Invariants to Nested Graph Constraints. In: Int. Conference on Graph Transformation, Springer (2014) 97–112
30. Benelallam, A., Tisi, M., Ráth, I., Izso, B., Kolovos, D.: Towards an open set of real-world benchmarks for model queries and transformations. In: BigMDE Workshop on Scalability in Model Driven Engineering. (2014)
31. Varró, G., Schürr, A., Varró, D.: Benchmarking for graph transformation. In: IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE (2005) 79–88
32. Bergmann, G., Horváth, Á., Ráth, I., Varró, D.: A benchmark evaluation of incremental pattern matching in graph transformation. In: Int. Conference on Graph Transformation. Springer (2008) 396–410
33. Izsó, B., Szárnyas, G., Ráth, I., Varró, D.: MONDO-SAM: A Framework to Systematically Assess MDE Scalability. In: BigMDE Workshop on Scalability in Model Driven Engineering. (2014) 40–43