MONDO-SAM: A Framework to Systematically Assess MDE Scalability

Benedek Izsó, Gábor Szárnyas, István Ráth and Dániel Varró
Fault Tolerant Systems Research Group
Department of Measurement and Information Systems
Budapest University of Technology and Economics
H-1117, Magyar Tudósok krt. 2.
Budapest, Hungary
{izso, szarnyas, rath, varro}@mit.bme.hu *

ABSTRACT

Processing models efficiently is an important productivity factor in Model-Driven Engineering (MDE) processes. In order to optimize a toolchain to meet scalability requirements of complex MDE scenarios, reliable performance measures of different tools are key enablers that can help selecting the best tool for a given workload. To enable systematic and reproducible benchmarking across different domains, scenarios and workloads, we propose MONDO-SAM, an extensible *MDE benchmarking framework*. Beyond providing easily reusable features for common benchmarking tasks that are based on best practices, our framework puts special emphasis on metrics, which enables scalability analysis along different problem characteristics. To illustrate the practical applicability of our proposal, we demonstrate how different variants of a model validation benchmark featuring several MDE tools from various technological domains have been integrated into the system.

1. INTRODUCTION

As Model-Driven Engineering (MDE) has gained mainstream momentum in complex system development domains over the past decade, scalability issues associated to MDE tools and technologies are nowadays well known [6]. To address these challenges, the community has responded with a multitude of benchmarks.

The majority of these efforts have been created by tool providers for the purpose to measure performance developments of specific engines [8, 2]. As a notable exception, the Transformation Tool Contest (TTC) [1] attempts cross-technology comparison by proposing multiple cases which are solved by the authors of (mainly EMF based) MDE tools. TTC cases focus on measuring query and transformation execution time against instance models of increasing size. TTC promotes reproducibility by providing pre-configured virtual machines on which individual tools can be executed; however, the very nature of this environment and the limited resources make precise comparison difficult.

Benchmarks are also used outside of the MDE community. The *SP^{2}Bench* [7] and *Berlin SPARQL Benchmark (BSBM)* [3] are SPARQL benchmarks over semantic databases (triple stores). The first uses RDF models based on the real world DBLP bibliography database, while the latter is centered around an e-commerce case study. Both benchmarks scale up in the size of models (up to 25M and 150B elements), however SP^{2}Bench does not consider model modifications, and BSBM does not detail query and instance model complexity. SPLODGE [4] is another similar approach, where SPARQL queries were generated systematically, based on metrics for a predefined dataset. Queries are scaled up to three navigations (joins), but other metrics as the complexity of the instance model were not investigated. The common technological characteristics of these benchmarks is that they are frequently run on very large computer systems that are not accessible to most users, or rely on commercial software components that are hard to obtain.

To summarize, currently available graph based benchmarks are affected by two main issues: (i) technologically, they are frequently built on virtualized architectures or have exotic dependencies, making measurements hard to reproduce independently; and (ii) conceptually, they typically only analyze measurement results against a limited view of the problem: the execution time of a fixed task scaled against increasing model size. As a result, the relative complexity of current benchmarks can not be precisely quantified, which makes them difficult to compare them to each other.

In previous work [5], we have found that other metrics (such as various query complexity measures, instance model characteristics, and the combination of these) can affect results very significantly. Building on these results, in this paper we propose the extensible MONDO-SAM framework that is integrated into the official MONDO benchmark open repository[1]. MONDO-SAM provides *reusable benchmark-*

*This work was partially supported by the CERTIMOT (ERC_HU-09-01-2010-0003) and MONDO (EU ICT-611125) projects partly during the fourth author's sabbatical.

BigMDE'14 July 24, 2014. York, UK.

[1]`http://opensourceprojects.eu/p/mondo/d31-transformation-benchmarks/`

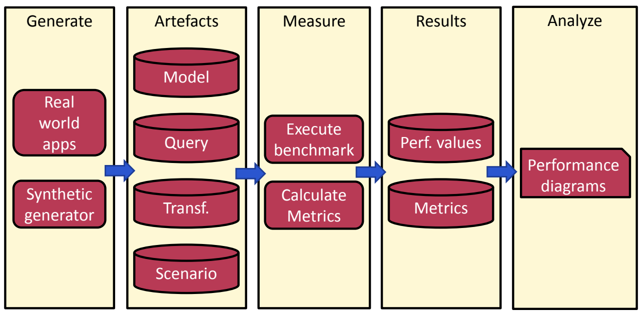


Figure 1: Benchmarking process.

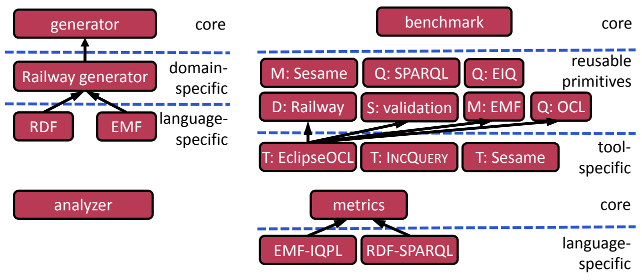


Figure 2: Benchmark framework architecture.

ing primitives (like metrics evaluation, time measurement, result storage) that can be flexibly organized into *benchmarking workflows* that are specific to a given case study. MONDO-SAM also provides an API so that tehnologically different tools can be integrated into the framework in a uniform way. A unique emphasis of the framework is built-in support for metrics calculation that enables characterization of the benchmarking problems as published in [5]. The built-in reporting facility allows to investigate the scalability of MDE tools along different metrics in diagrams. Finally, the entire framework and integrated case studies can be compiled and run using the Maven build system, making deployment and reproducible execution in a standard, Java-enabled computing environment feasible.

2. OVERVIEW OF THE FRAMEWORK

2.1 A process model for MDE benchmarks

The benchmarking process for MDD applications is depicted in Fig. 1. Inputs of the benchmark are the instance model, queries run on the instance model, the transformation rules or modification logics and a scenario definition (or workflow) describing execution sequences. In this case, scenario can describe MDD use cases (like model validation, model transformation, incremental code generation), including warmup and teardown operations, if required. Inputs can also be derived from real-world applications, or are synthetically generated providing complete control over the benchmark. Complexity of the input is characterized by metrics, while scenario execution implementations are instrumented to measure resource consumption (wall-clock times, memory and I/O usage). Finally, these measured values and calculated metrics are visualized on diagrams automatically to find the fastest tool, or to identify performance improvements of a specific tool.

2.2 Architecture

The benchmark framework consisting of four components is depicted in Fig. 2. The *generator component* allows synthetic generation of benchmark inputs. The core module handles configuration, domain-specific modules describe generation method of input data (like generation of instance models, queries), and language-specific modules serialize generated logical artifacts into files (like EMF models or OCL queries). The selected domain constrains languages, as domain description concepts must be supported. For example transitivity or multi-level metamodeling is not supported by EMF, but the latter is required by the e-commerce case study of BSBM. Generated models should be semantically equivalent, however, it is a question whether structural equality should be preserved. E.g. in certain cases EMF models must have a dedicated container object with containment relations to all objects which is not required in RDF.

2.3 Core features

Benchmark component. The *benchmark component* (in Fig. 2) measures performance of different tools for given cases. A case can be defined as a quintuple of $(\mathcal{D}, \mathcal{S}, \mathcal{M}, \mathcal{Q}, \mathcal{T})$, where $\mathcal{D}$ defines the domain, $\mathcal{S}$ the scenario, $\mathcal{M}$ the modification and $\mathcal{Q}$ the query. The $\mathcal{T}$ modules implement tool specific glue code and select $\mathcal{D}, \mathcal{S}, \mathcal{M}, \mathcal{Q}$. All modules reuse common functions of the core, like configuration (with default values and tool-specific extensions), wall-clock time measurement which is done with highest (nanosecond) precision (that does not mean same accuracy), and momentary memory consumption, which are recorded in a central place. At runtime, language-specific modifications (transformations), queries, and instances of the selected domain must be available.

Model instantiator. A common aspect of the *generator and the benchmark module* is reproducibility. In tool-specific scenario implementations boundaries are well separated by the scenario interfaces, and where generation or execution is randomized, a pseudo-random generator is used with the random seed set to a predefined value. However, nondeterministic operations (like choosing an element from a set) and tool implementations can disperse results between runs.

Metrics evaluator. To describe benchmark input with quantitative values, they are characterized by metrics which are evaluated by the *metrics component.* Language specific implementations analyze model-query pairs, and store calculated metric values centrally gathered by the core which are analyzed later together with the measured values.

Result reporting and analysis. When measurement and metrics data become available, the *analyzer component* (implemented in R) automatically creates HTML report with diagrams. To show scalability according to different measures, on the x axis metrics can be selected, while the y axis represents resource consumption. Raw data can be postprocessed, i.e. dimensions can be changed (e.g. to change time to ms dimension to reflect its accuracy), and derived values can be calculated (e.g. the median of incremental recheck steps, or total processing time).

2.4 Best Practices to Minimize Validity Threats

During the execution of the cases, noise coming from the environment should be kept at minimum. Possible sources of noise include the caching mechanisms of various components (e.g. file system and the database management system), warm-up effect of the runtime environment (e.g. the Java Virtual Machine), scheduled tasks (e.g. cron) and swapping. For large heaps, the Garbage Collector of the JVM can block the run for minutes, so minimizing its call is advised which is achieved by setting minimal and maximal heap size to an equal value, thus eliminating GC calls at memory expansions.

In the implementation of framework components, only the minimal amount of libraries should be loaded. On one hand, proper organization of the dependencies is the responsibility of the developer. On the other hand it is enforced by the framework architecture, as tool-specific implementations are independent, and functions as entry points calling the framework that uses inversion of control (IoC) without the usage of additional execution environments, such as OSGi.

To alleviate random disturbances, each test case is run several times (e.g. ten times) by the framework and aggregated by the analyzer.

3. INTEGRATED CASE STUDIES

The usability of the framework is demonstrated by four examples. Three variations of the previously published Train Benchmark, and a new, soon to be released model comprehension benchmark are integrated into the framework.

3.1 Basic Train Benchmark

The first version of the Train Benchmark [9] compares the performance of EMF-IncQuery with Eclipse OCL and its incremental version, the OCL Impact Analyzer in an incremental model validation use case. Instance models are generated from a railway domain, and four hand-written queries (with different complexity) perform model validation tasks. The scenario starts with a model loading phase, where the instance is read from a file, followed by a check phase, where a model validation query is executed (returning constraint violating elements). Afterwards (to simulate a user in front of an editor), multiple (100) edits and rechecks performed. In this case batch, incremental validation time and memory consumption was measured.

One kind of diagrams display execution times as the function of model and query metrics. Fig. 3 shows total execution time for a specific query and scenario in a logarithmic diagram for different tools. On the x axis model size (the number of nodes and edges) is displayed, together with the number of results, and the number of changes in the result set. Although model size is the most influencing performance factor during the load phase, in the check phase, especially for incremental tools other metrics come into the picture as most influencing factors, like the result set size, or the number of variables in a query [5].

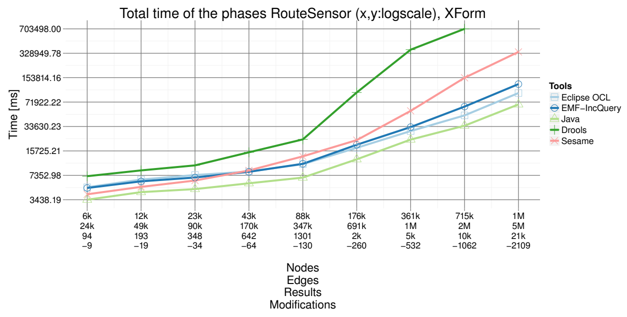


Figure 3: Required time to perform a task.

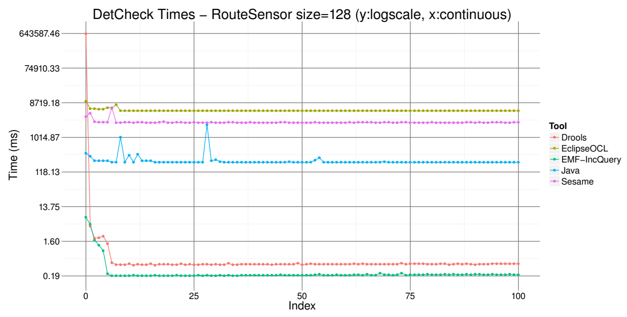


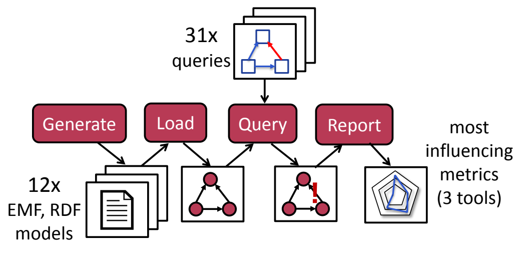
Figure 4: Check time during revalidations.

3.2 Extended Train Benchmark

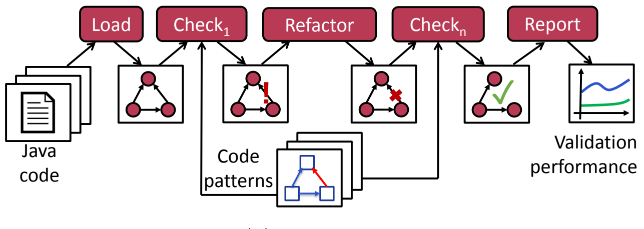
The extended version is available online[2] which introduces new languages: in addition to EMF, RDF and GraphML model formats were added. New tools (Drools, Sesame, 4store and Neo4j) were added, and queries were translated to each tool's native language. From now not all tools have in-memory implementation, some use hard disk as storage, so to lower disk overhead, memory filesystems were used for storage. Also it should be noted that some databases compiled as JARs next to the benchmark code, some database use native server daemons that are also handled by the benchmark execution framework. In this case a new scenario variation is defined, where after the batch validation, larger modification is performed in one edit phase (to simulates automatic model correction), and finally recheck is executed.

As the benchmark framework records every check and edit time subsequently calls can be displayed on a diagram to show its changes. Fig. 4 depicts such a case for tools at a given model size and query. It can be observed that the first query time is almost always the highest, probably due to the lazy loading of classes and tool initialization. Another interesting point for the incremental EMF-IncQuery and Drools tools is around the tenth check, where evaluation times are dropped significantly. As the same queries are executed, this may be attributed to the changed model structure, or to the kicked in JIT compiler. This diagram also shows the required warmup time for each tool, and its changing in stages.

[2]`https://incquery.net/publications/trainbenchmark/`



(a) Metrics evaluation



(b) Code model

Figure 5: Different use-cases of the framework

3.3 Model Metrics for Performance Prediction

In the article [5] tools are narrowed down to a basic Java implementation, EMF-IncQuery, and Sesame. However, for a modified metamodel nine new instances were generated (belonging to different edge distributions). The benchmark was extended with 31 queries scaling along 5 query metrics. The goal of this paper was not to compare tool performances, but to identify which metrics influence processing time and memory usage the most. (See Fig. 5a.)

Detailed results are available in the paper, however it can be noted that for the EMF-IncQuery tool the number of matches, for Sesame the number of query variables showed high correlation with the check time, and low correlation of model size metrics that also emphasize considering other aspects than model size.

3.4 ITM Factory

The fourth case (inspired by [10]) integrated into the framework is currently under development, and it took another domain from the field of software comprehension. Input of the benchmark are not serialized models, but Java projects. In the first step, source code is read into a software model, transformations are code edits or complex refactor operations. After software modifications, correctness of the code base is validated (Fig. 5b).

In the code modeling case similar investigations can be done, however processing tools should scale in the lines of code (and not in the number of nodes or edges). This also motivates displaying performance as a function of different metrics.

4. CONCLUSION

In this paper we proposed MONDO-SAM, a framework that provides common functions required for benchmarking, and MDE-specific scenarios, models, queries and transformations as reusable and configurable primitives. As the main focus, integrated benchmark cases can be characterized by metrics, which enables the reporting module to analyze the scalability of tools against various complexity measures. We demonstrated the versatility of the framework is demonstrated by the integration of previous versions of the Train Benchmark [9, 5] and a new benchmark from the code model domain.

The extensible framework including the APIs, core components and documentated samples is available as open source code from the MONDO Git repository[3].

5. REFERENCES

[1] Transformation tool contest. `www.transformation-tool-contest.eu`, 2014.

[2] G. Bergmann, I. Ráth, T. Szabó, P. Torrini, and D. Varró. Incremental pattern matching for the efficient computation of transitive closure. In *Sixth International Conference on Graph Transformation*, volume 7562/2012, pages 386–400, Bremen, Germany, 09/2012 2012. Springer.

[3] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *International Journal On Semantic Web and Information Systems*, 5(2), 2009.

[4] O. Görlitz, M. Thimm, and S. Staab. SPLODGE: Systematic generation of SPARQL benchmark queries for Linked Open Data. In C.-M. et al., editor, *The Semantic Web – ISWC 2012*, volume 7649 of *LNCS*, pages 116–132. Springer Berlin Heidelberg, 2012.

[5] B. Izsó, Z. Szatmári, G. Bergmann, Á. Horváth, and I. Ráth. Towards precise metrics for predicting graph query performance. In *IEEE/ACM 28th International Conference on Automated Software Engineering*, pages 412–431, Silicon Valley, CA, USA, 2013. IEEE.

[6] 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 *Proceedings of the Workshop on Scalability in Model Driven Engineering*, BigMDE '13, pages 2:1–2:10, New York, NY, USA, 2013. ACM.

[7] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP2Bench: A SPARQL performance benchmark. In *Proc. of the 25th International Conference on Data Engineering*, pages 222–233, Shanghai, China, 2009. IEEE.

[8] M. Tichy, C. Krause, and G. Liebel. Detecting performance bad smells for henshin model transformations. In B. Baudry, J. Dingel, L. Lucio, and H. Vangheluwe, editors, *AMT@MoDELS*, volume 1077 of *CEUR Workshop Proceedings*. CEUR, 2013.

[9] Z. Ujhelyi, G. Bergmann, Á. Hegedüs, Á. Horváth, B. Izsó, I. Ráth, Z. Szatmári, and D. Varró. EMF-IncQuery: An Integrated Development Environment for Live Model Queries. *Science of Computer Programming*, 2014. Accepted.

[10] Z. Ujhelyi, Á. Horváth, D. Varró, N. I. Csiszár, G. Szőke, L. Vidács, and R. Ferenc. Anti-pattern detection with model queries: A comparison of approaches. In *IEEE CSMR-WCRE 2014 Software Evolution Week*. IEEE, 02/2014 2014.

[3] `https://opensourceprojects.eu/git/p/mondo/trainbenchmark`