Evaluation of Model Comparison for Delta-Compression in Model Persistence

Markus Scheidgen[1]

Humboldt Universität zu Berlin, Germany
scheidge@informatik.hu-berlin.de

Abstract. Model-based software engineering is applied to more and more complex software systems. As a result, larger and larger models with longer and longer histories have to be maintained and persisted. Already, a lot of research efforts went into model versioning, comparison, and repositories. Existing strategies either record and persist changes (change-based repositories, e.g. EMF-Store) or relay on existing text-based version control systems to persist whole model revisions (state-based repositories). Both approaches have advantages and disadvantages. We suggest a hybrid approach that infers changes via comparison to persist delta-compressed model states. Our hypothesis is that delta-compression requires a trade-off between comparison quality and execution time. Existing model comparison frameworks are tailored for comparison quality and not necessarily execution time performance. Therefore, we evaluate and compare traditional line-based comparison, an existing model comparison framework (EMF-Compare), and our own framework (EMF-Compress). We reverse engineered the Eclipse codebase and it's history with MoDisco to create a large corpus of evolving example models for our experiments.

1 Introduction

In *BigMDE*, we assume that model-based software engineering (MBSE) is applied to increasingly complex software systems and that the size of models that we need to process is getting increasingly larger. But not only the size of models is increasing, they also evolve over longer periods of time, and have longer revision histories. To cope with such evolving models, the MBSE community began to adapt the concepts of existing text-file-based version control systems for models. Respective model repositories allow clients to navigate model history, compare revisions, and merge different branches of development [1]. We can distinguish two basic strategies for model repositories: state-based and change-based [8]. State-based systems store individual revisions of a model and whenever a user needs to compare two revisions the respective models are compared. In a change-based system, only the differences (or the operations causing these differences) are persisted and when a user requires a particular revision the respective model is re-created from prior revisions and respective difference data.

Both strategies have advantages and disadvantages. Change-based repositories, e.g. EMF-Store [8], require to record editing operations to infer and persist

changes. This requires a tight integration of editors and model repository. While this works well for certain editors like MVC-based graphical editors (e.g. GMF), it does not for background-parsing-based textual editors (e.g. Xtext), mixed-, or closed editing environments. For state-based repositories, we can use existing version control systems that were developed for managing text files [1]. Here, models are persisted in their serialized form, e.g. as XMI/XML files. While this is independent from the editing environment and works for all models that can be serialized without information loss, it imposes an unnatural organization scheme onto modeling. Clients have to organize large models in terms of directories and files. To be efficient, the respective files have to be reasonably sized.

We propose a third strategy that uses a change-based strategy internally, and expose the characteristics of a state-based system to the editing environment and the user. Git, for example, allows us to access file revisions as a whole, but stores many revisions of the same file in a delta-compressed *pack-file*. We will persist model histories in terms of changes, but provide and take models as a whole. This requires us to create model differences through model comparison, since we can not rely on difference information provided by the editing environment. Similar to version control systems, this strategy does not force us to use the same comparison algorithms for storing models tightly (also know as delta-compression) and for presenting changes and similarities to users. We can use algorithms with good comparison quality and good performance characteristics internally for frequent delta-compression, and different algorithms with excellent comparison quality (i.e. minimal editing distance) and less performance to occasionally present changes to users.

In this paper, we want to evaluate comparison algorithms for delta-compression. We developed a framework called *EMF-Compress* [12]. We implemented signature-based matching strategies, far simpler than the similarity-based matching of existing frameworks like EMF-Compare. In contrast to existing comparison frameworks, we use a meta-model that allows us to persist differences (i.e. delta-models) between models without referencing the compared models directly. We conducted experiments with reverse engineered MoDisco [3] Java models that were taken from the Eclipse source-code history [13]. In these experiments, we compare compression quality and execution times for different matching algorithms and provide execution times for comparison (compression) and patching (de-compression) to prove the feasibility of our approach.

The paper is organized as follows. The following Section 2 provides a brief introduction into model comparison. Section 3 introduces our delta-compression tailored comparison framework. The evaluation Section 4 describes and discusses our experiments in compressing the reverse-engineered Eclipse code-base. We present related work, future research questions, conclusions in Sections 5 and 6.

2 About Matches, Differences, Merges

In this paper, we only consider two-way comparison. We always compare a left (original) model with a right (revised) model. Algorithms for comparing two

text files (i.e. two list of comparable items) exist for a very long time. These algorithms are searching for the minimal editing distance, that is the smallest possible set of changes (adding lines, removing lines, changing lines) necessary to modify one file and get the other. One particular algorithm is *Meyer's* [11] algorithm that compares two list in $\mathcal{O}(N * D)$ where N is the sum of both sizes and D the number of changed lines.

Models are not list of items; they are graphs of items. But, each model element contains lists of items, i.e. their attribute and reference values. Assuming that the order of values does not change arbitrary or volatile between revisions, we could apply existing algorithms, like Meyer's, to each value-set in each model element. But in order to use this approach, we need to know which values (primitive values as well as other model elements) are supposed to be equal and which are not.

Therefore, existing model comparison frameworks use two steps. First, model elements are matched. Frameworks use different matching algorithms to establish pairs of matching model elements from both compared models. In a second step, these matches are used to find differences, comparing element by element, feature by feature. In such comparison processes, the matching strategy influences the comparison quality. Do we yield a minimal set of differences or a large set of differences only depends on how good the matches are.

We can distinguish different types of matching-strategies. Among others: signature-based and similarity-based matching. Signature-based algorithms only consider local model element properties like its meta-class, name, parameters, and its position in the containment hierarchy (i.e. its parent). Similarity-based algorithms consider the larger neighborhood of an element to assess its shape in order to establish matching pairs. Finding and comparing signatures is a straightforward, linear process. Similarity-base algorithms on the other hand can be considerably different based on the used neighborhood, considered characteristics, used heuristics, etc.

Existing frameworks, like EMF-Compare, are tailored for presenting differences to users and for merging models. They use similarity-based matching for a maximum comparison quality and represent matches and differences in a comparison model. These comparison models reference both compared models. This is fine when the goal of a merge is to create a new merged model from two existing compared models. But in delta-compression, we want to re-create one model from the other and a respective difference model. This is also known as *patching* and the difference model is than called a *patch*.

3 Model Comparison for Compression

We developed the comparison framework *EMF-Compress* [12] with the goal to evaluate signature-based comparison for delta-compression. This means two things. First, we can perform matching and differentiating in one single step. In a signature-based matching algorithm, we consider the position of an element (i.e. its parent) to be part of its signature. As a consequence, a moved model

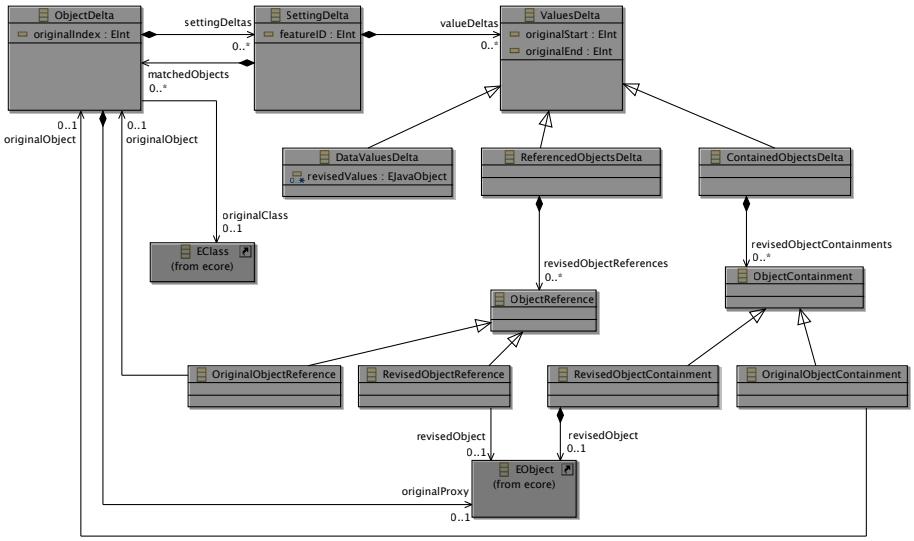

Fig. 1. Combined meta-model for matches and differences.

element (i.e. a model element that changes its parent) is not matched with its original. This is obviously bad for comparison quality, but we also only need to establish matches locally within individual value-sets. Therefore, we can match elements *on the fly* during differentiation. We start by assuming that the root elements match. We then apply Meyer's algorithm to all features. While Meyer's algorithm is applied, we establish matches among the values of each individual value-set. We recursively proceed for all matches in all containment references throughout the containment hierarchy.

Secondly, EMF-Compress uses a comparison meta-model that represent differences under the assumption that the right model is not available and that it needs to be re-created from the left model. This means the comparison model (depicted in Fig. 1) represents all differences as changes (i.e. deltas). An `ObjectDelta` represents all changes between two matched elements without referencing these elements directly. A `SettingDelta` is a container for differences (and matches) in a feature (via `featureID`). A `SettingDelta` can contain matching elements (`matchedObjects`) identified via index (`originalIndex`) and *ValueDeltas* that described changed ranges of values with the feature. Ranges are also identified via indices. There are different classes used to represent differences depending on the value type (primitive, contained object, cross-referenced object) and whether the value is part of the left (original) model or is a new value that is added to the right (revised) model. All object values that are added have to be contained in the comparison model, since it is supposed to be independent of the right (revised) model. All object values that are removed or moved

within the model are referenced based the *ObjectDelta* instance that represent the object in the comparison model.

Therefore, the comparison model does not reference any of the compared model directly. The information in a comparison model is sufficient to recreate (i.e. patch) a right (revised) model from an left (original) model.

4 Evaluation

4.1 Setup

The subject for our experiments is a corpus that comprises a subset of the Eclipse Foundation's source code repositories. The Eclipse Foundation maintains code (and other artifacts, like documentation, web-pages, etc.) in over 600 Git repositories. We took the largest 200 of those repositories that actually contained Java code. These 200 repositories contain about 600 thousand revisions with a total of over 3 million *compilation unit* (CU) revisions that contain about 400 million SLOC (lines of code without empty lines and comments). The Git repositories take 6.6 GB of disk space. The generated MoDisco model representation of these repositories comprises over 4 billion objects and is persisted in EMF-Fragments [14] and a binary serialization format with about 230 GB of diskspace.

We run all experiments on a server computer with four 6-core Intel Xeon X74600 2.6 GHz processors and 128 GB of main memory. However, *EMF-Compress* operates mostly in a single thread with the exception of JVM and Eclipse maintenance tasks. *EMF-Compress* runs on Eclipse Mars' versions of EMF in a Java 8 virtual machine. All operations were run with a maximum of 12 GB of heap memory. The system runs on GNU/Linux with a 3.16 kernel. *EMF-Compress* operations are long running computations over thousands of revisions and CUs and respective algorithms are invoked over and over again. As usual for such macro-benchmark measures, we are ignoring JIT warm-ups and other micro-benchmark related issues [16].

The use-case behind these experiments is a repository wide application of reverse-engineering (with MoDisco) for the purpose of analysis [13]. The goal of this model-based mining of software repositories (MSR) [4] approach, is an AST-level model of a large-scale software repository (e.g. Eclipse) that allows clients to derive metrics, dependencies, design structure matrices, and other data for statistical analysis over the history of many software projects. Not only would we benefit from lesser space requirements of the resulting models, delta-compression and comparison models would also allows us to omit unnecessary processing of unchanged model parts during model analysis.

4.2 Signature- v Similarity- v Line-based Comparison

As a first experiment, we want to measure and compare signature-based, similarity-based, and line-based comparison both by comparison quality and execution

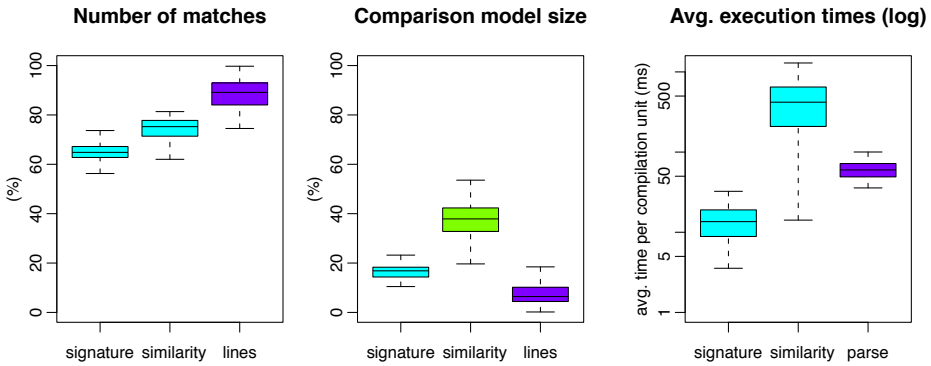

Fig. 2. Signature-based comparison with *EMF-Compress*, similarity-based comparison with EMF-Compare, and line-based comparison on the original source-code files.

time. Please note, line-based comparison always refers to comparing the original source files and not any serialized representation of the models. The resulting line-based numbers are not really comparable, but provide a frame of reference. Due to the low runtime performance of similarity-based comparison, we had to limit our measurements to the first 1000 revisions of the 100 biggest (by Git-size) Eclipse projects. For signature-based comparison, we used our own framework *EMF-Compress*. The used signatures comprise parent, meta-class, and where applicable names. For similarity-based comparison, we used EMF-compare in its default configuration. For line-based comparison, we used the data provided by Git.

Fig 2 show the results. Different colors suggest data that is not directly comparable and that has to be read very carefully. The left chart depicts the comparison quality by means of matched elements. For models this is the number of actual matched elements. For lines this is the number of *matched* lines, i.e. lines that were not added, removed, nor changed. The chart shows the fraction of matched elements relative to the overall number of elements without the uncompressed initial revisions. The data is accumulated for each analyzed repository. As expected, similarity-based comparison produces good comparison quality, since it uses far more data to establish matches than signature-based matching. The signature-based matching quality is worse, but still of a similar magnitude. We cannot draw any sensible conclusions from comparing these model-based numbers with the number of matched lines, because lines span different numbers of model elements and shorter lines match more likely than longer lines.

The middle chart compares the size of the resulting comparison models in relation to the size of the original models. This data also does not include the uncompressed initial revisions. For models, we used the approximated size of a binary serialized representation of all models. For lines, we used the average model size per line for added lines and two bytes for removed lines to approximate sizes for a corresponding comparison model. Measuring the size of comparison mod-

els is not very informative, since the comparison models of EMF-Compare and *EMF-Compress* comprise different data. EMF-Compare expresses changes by referencing to the differentiating model elements, while *EMF-Compress* stores changed features, indices, and includes all added values. Our line comparison model approximation does not include any overhead for organizing the differences.

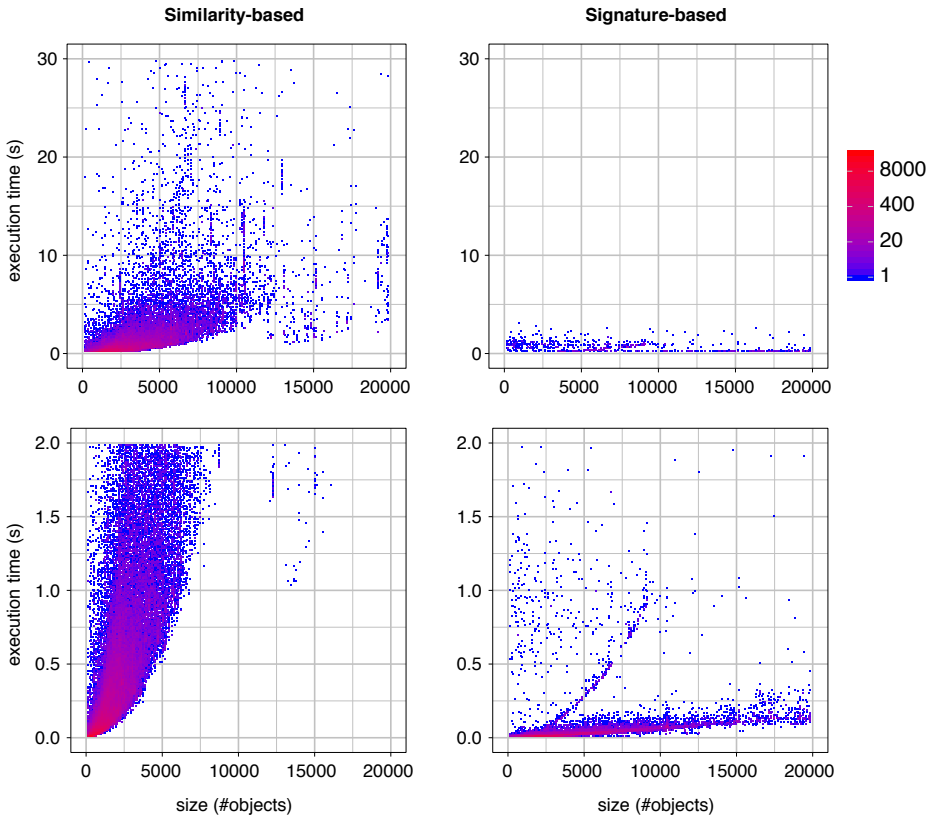

Fig. 3. Comparison execution time depending on model size for similarity- and signature-based matching. The plots show data measured for 300k compilation units taken from different Eclipse projects. Top row depicts all measurements, the lower row magnifies a smaller range.

The right chart of Fig. 4 shows the average execution time spend on comparison in all repositories compared to the time that was necessary to create models for the original Java code. Similarity-based comparison is magnitudes slower than signature-based comparison. Fig. 3 provides a closer look on the execution times. This chart plots execution times spend on individual CUs in relation to their sizes in number of model elements. While signature-based comparison

scales linearly for most CUs, similarity-based comparison is above linear. Additionally, the comparison times seem to depend on CU structure and not only CU size, since the measured times are particularly large for some CUs.

4.3 Top-Level Comparison v Deep Comparision

For a second experiment, we want to compare different levels of comparison. The hypothesis is that if we only try to match the first levels in a containment hierarchy and only use equality to compare the rest of the models, we can save execution time and still yield sufficient comparison quality. We devised two different matching algorithms. The first only uses top-level named elements and their signature for matching (e.g. Java packages, classes, methods). All other elements (i.e. everything contained in method bodies) only matches if they are equal to each other. The second algorithm tries to match all model elements based on their signature. We also provide line-based matching data for reference.

This time, we measure comparison for all 200 Eclipse projects, all revisions, all CUs. Fig. 4 shows the results. The top row of charts compares artifact sizes for some example Eclipse projects. The lower left shows the average size of uncompressed initial revisions and respective comparison model sizes relative to the overall size of the uncompressed models. The execution times are given as an average per revision for both compression algorithms and the times necessary to decompress (i.e. reconstruct a revision from its predecessor and the comparison model).

The comparison quality with only matching the named elements is significantly worse than with matching all elements. But more surprisingly, the execution time performance is also much worse. Even though, we do not have to match all model elements, we still have to compare them for equality.

5 Related Work

EMF-Store is an example for a change-based model version control system (Koegel and Helming [8]). An example for a model repository based on existing source-code version control systems is presented by Altmanniger et al. [1].

Different frameworks for model comparison exist: for example EMF-Compare by Brun and Pierantonio [2] or SI-Diff by Kehrer et al. [5]). Stephan and Cordy present a feature-based comparison of existing approaches and frameworks [15]. Kolovos et al. did the same for matching-algorithms [10].

In this paper, we only assumed meta-model agnostic matching-algorithms. But it is assumed that comparison quality can be increased, when matching is implemented depending on the meta-model of compared models. Kolovos et al. [9] and Kehrer et al. [5] present approaches for customization of matching algorithms.

In [6,7] Kehrer et al. research the posibilities to infer high-level editing operations from low-level comparison models. This can be valuable for presenting differences to users in a better way and to represent model differences/changes more efficiently in smaller delta-models.

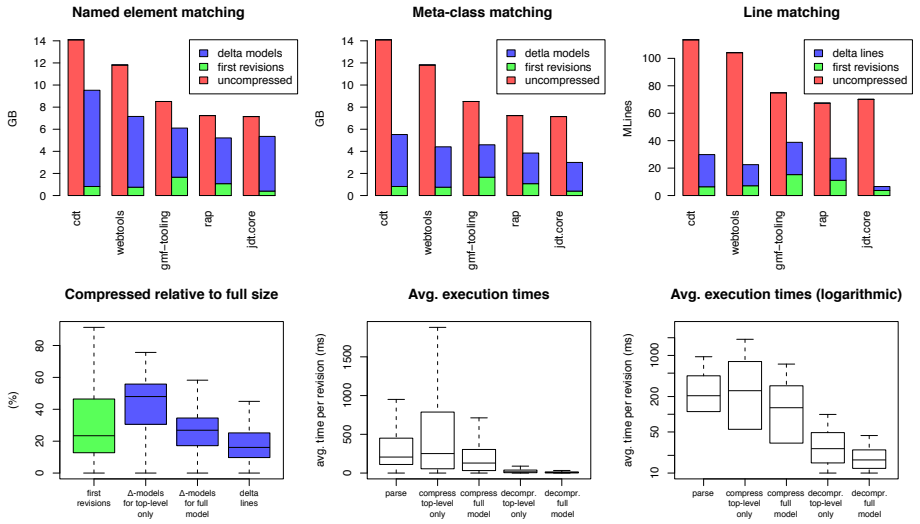

Fig. 4. Compression quality by percentage of full size and percentage of matched elements compared to line-based compression.

6 Conclusions

We evaluated the potential use of model comparison for delta-compressing state-based model repositories. We could show that signature-based matching algorithms reach sufficient comparison quality with significantly shorter execution times than the existing similarity-based model comparison in frameworks like EMF-Compare. Furthermore, we presented a meta-model for comparison models that does not reference the compared models, can be persisted independently, and allows to re-create one of the compared models from the other. Therefore, we could prove the principle feasibility of time efficient delta-compression for models.

However, there are many unanswered questions. We only evaluated a single algorithm for similarity-based matching. Different algorithms, e.g. meta-model specific algorithms could yield better results. Further, we only considered model representations of source-code following a single meta-model. Different kinds of models could yield different results. What factors influence the performance of model comparison in what way? Delta-compression allows to decompress sequences of model revisions efficiently, but does not allow a random access of particular revisions. In a practical application, we need to consider to persist some intermediate revision as full model to counter that. What factors influence the right amount these intermediate models? The models that we used for evaluation have a convenient size. How can we combine delta-compression with existing approaches in fragmenting and persisting very large models [14]?

References

1. Altmanninger, K., Seidl, M., Wimmer, M.: A survey on model versioning approaches. International Journal of Web Information Systems 5(3), 271–304 (2009)
2. Brun, C., Pierantonio, A.: Model differences in the eclipse modeling framework. UPGRADE, The European Journal for the Informatics Professional 9(2), 29–34 (2008)
3. Bruneliere, H., Cabot, J., Jouault, F., Madiot, F.: Modisco: A generic and extensible framework for model driven reverse engineering. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. pp. 173–174. ASE '10, ACM (2010)
4. Kagdi, H., Collard, M.L., Maletic, J.I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. Journal of Software Maintenance and Evolution: Research and Practice 19(2), 77–131 (2007)
5. Kehrer, T., Kelter, U., Pietsch, P., Schmidt, M.: Adaptability of model comparison tools. Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering - ASE 2012 p. 306 (2012)
6. Kehrer, T., Kelter, U., Taentzer, G.: A rule-based approach to the semantic lifting of model differences in the context of model versioning. 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings pp. 163–172 (2011)
7. Kehrer, T., Kelter, U., Taentzer, G.: Consistency-preserving edit scripts in model versioning. 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings pp. 191–201 (2013)
8. Koegel, M., Helming, J.: EMFStore: a model repository for EMF models. 2010 ACM/IEEE 32nd International Conference on Software Engineering 2, 307–308 (2010)
9. Kolovos, D.S.: Establishing correspondences between models with the epsilon comparison language. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 5562 LNCS, 146–157 (2009)
10. Kolovos, D.S., Di Ruscio, D., Pierantonio, A., Paige, R.F.: Different models for model matching: An analysis of approaches to support model differencing. Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models pp. 1–6 (2009)
11. Myers, E.W.: AnO (ND) difference algorithm and its variations. Algorithmica 1(1-4), 251–266 (1986)
12. Scheidgen, M.: EMF-Compress. https://github.com/markus1978/emf-compress (2016)
13. Scheidgen, M., Fischer, J.: Model-based mining of source code repositories. In: Amyot, D., Fonseca i Casas, P., Mussbacher, G. (eds.) System Analysis and Modeling: Models and Reusability, Lecture Notes in Computer Science, vol. 8769, pp. 239–254. Springer International Publishing (2014)
14. Scheidgen, M., Zubow, A., Fischer, J., Kolbe, T.H.: Automated and transparent model fragmentation for persisting large models. In: Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems (MODELS). LNCS, vol. 7590, pp. 102–118. Springer, Innsbruck, Austria (2012)
15. Stephan, M., Cordy, J.R.: A Survey of Methods and Applications of Model Comparison (June) (2012)
16. Wilson, S., Kesselman, J.: Java Platform Performance: Strategies and Tactics. Addison-Wesley, Boston, MA (2000)