Towards Scalability Analysis of State-based Model Comparison

Martin Armbruster
martin.armbruster@kit.edu
Karlsruhe Institute of Technology,
Karlsruhe, Germany

Manar Mazkatli
manar.mazkatli@kit.edu
Karlsruhe Institute of Technology,
Karlsruhe, Germany

Alp Toraç Genç
ufyzm@student.kit.edu
Karlsruhe Institute of Technology,
Karlsruhe, Germany

Anne Koziolek
koziolek@kit.edu
Karlsruhe Institute of Technology,
Karlsruhe, Germany

Abstract

State-based model comparison is crucial in Model-Driven Engineering (MDE) for identifying differences between model versions. These differences support various tasks such as version control and the synchronization of related models to reflect the latest version.

The performance and scalability of state-based model comparison impact overall system efficiency and are particularly important in continuous integration environments with frequent automated comparisons.

This study empirically evaluates state-based model comparison using a Java code model, examining how the performance behaves. The results provide insights to enhance comparison techniques for large and evolving models in MDE workflows.

1 Introduction

Model differences are important for various tasks such as synchronization, impact analysis, and automated evolution of software models. State-based comparison detects differences between two versions of a model by *matching* similar elements and *differencing* their changes [1]. It identifies additions, deletions, and modifications directly from the model states, without requiring access to an explicit change history. This approach suits practical model-driven engineering (MDE) scenarios where tools often do not record fine-grained edits, preventing the use of change-based or operation-based methods that depend on recorded histories of changes [3]. Therefore, tools such as EMF Compare [5] are widely used for various state-based comparisons in MDE, enabling the implementation of different state-based matching strategies, such as name-based or structural ones, to suit diverse modeling needs [2].

The performance and scalability of state-based model comparisons directly impact the efficiency of software systems that rely on them, especially affecting the performance of continuous integration (CI) workflows, including automated tests using model comparison. Frequent comparisons of large, evolving models can introduce significant computational overhead and reduce development responsiveness. One concrete example is the Continuous Integration of Performance Models (CIPM) approach [10], which aims to keep architectural performance models up-to-date during agile development. After each code commit, CIPM computes differences between the previous and the current version of the source model. These differences are derived using a state-based comparison strategy and then propagated to related performance models using model-based consistency management platforms [7]. As noted, updating the architectural performance model (aPM) and automated tests in CIPM are affected by state-based comparison performance on large-scale models. Hence, this paper presents a first step towards answering the question: "*How well do state-based model comparison approaches scale with large-scale models evolving in continuous integration workflows?*"

To address this question, the paper empirically investigates the performance of state-based model comparisons in CIPM, using the CIPM prototype to assess how the comparison behaves. The results aim to improve the performance of comparison strategies in large-scale MDE.

2 Background

Continuous Integration of architectural Performance Models (CIPM) is an approach that keeps the aPM up to date throughout development and operation. During development, CIPM automatically updates a source code model to extract source code changes. Based on these changes, CIPM updates the aPM repository model for each commit in the CI pipeline [10], using state-based model comparison before and after the commit. CIPM currently supports Java source code and uses a custom language-specific algorithm [2] for these comparisons to improve the matching of models by leveraging language-specific structural and semantic features, thus capturing the hierarchy and dependencies more accurately than generic similarity-based approaches [2].

CIPM also calibrates performance parameters of the aPM (e.g., resource demands) using measurement data from tests or runtime environments [6].

By continuously updating and calibrating the aPM, CIPM enables accurate architecture-based performance prediction throughout the software lifecycle. While the scalability of CIPM during operation has been evaluated based on increasing measurement volumes [10], scalability of state-based model comparison during development still needs to be addressed.

3 Evaluation

This section presents a first evaluation of the state-based model comparison in CIPM by executing CIPM's code for the model comparison and measuring its execution time.

3.1 Process

To evaluate the state-based model comparison, repository-based evaluation tests were introduced. A repository-based evaluation test RPT considers a subset of commits $C = \{c_1, \ldots, c_M\}$ from a Git repository R by comparing a subset of all possible commit pairs $CP \subseteq C^2$ with a model comparison algorithm MC and an algorithm $ExpMC$, providing automatically calculated, approximated expected results for model comparisons.

The flow of an RPT is as follows: RPT clones R and parses a Java code model JCM_i for each $c_i \in C$. Then, RPT computes the expected model comparison results $ExpMCRes_{i,j} = ExpMC(c_i, c_j) \in \{similar, notSimilar\}$ for each $(c_i, c_j) \in CP$. $ExpMC$ analyzes the Git diff between c_i and c_j for effective code changes (i.e., code that is actually changed in the diff patch). Upon finding any effective code change, the expected result is $notSimilar$. Otherwise, it is $similar$. Afterward, RPT computes the actual model comparison results $ActMCRes_{i,j} = MC(JCM_i, JCM_j) \in \{similar, notSimilar\}$ by executing the model comparison, whose execution time is measured. Then, RPT checks if $ActMCRes_{i,j} \stackrel{?}{=} ExpMCRes_{i,j}$ for each $(c_i, c_j) \in CP$. While this check is not directly relevant for the measurements, it ensures that the results $ActMRes_{i,j}$ are correct according to $ExpMRes_{i,j}$. To accelerate future test runs, RPT saves all $ExpMCRes_{i,j}$ and JCM_i on disk and reuses them in subsequent runs.

3.2 Setup

The evaluation here consists of 2 RPTs, one for TEAMMATES [8] and one for the Corona-Warn-App Server (CWA-Server) [9]. TEAMMATES [8] is a cloud-based tool with a web-based frontend and Java-based backend. It is a real-world application, used for feedback management for students and instructors. CWA-Server [9] is the server part of the Corona-Warn-App,[1] which enables proximity communication for exposure notification. This evaluation considers TEAMMATES and CWA-Server, as both are real-world Java applications and subjects of a previous evaluation [10].

[1] `https://github.com/corona-warn-app`

Group	Minimum	Maximum
TEAMMATES (different)	8.1 s (sd = 313.7 ms, $f33d0b - 83f518$)	8.4 s (sd = 323.1 ms, $83f518 - 48b67b$)
TEAMMATES (identical)	3.7 s (sd = 203.4 ms, 648425)	3.7 s (sd = 157.7 ms, $ce4463$)
CWA-Server (different)	4.2 s (sd = 173.5 ms, $c22f93 - 6e9702$)	4.5 s (sd = 229.7 ms, $9323b8 - 206e8c$)
CWA-Server (identical)	2.1 s (sd = 84.3 ms, $7e1b61$)	2.2 s (sd = 124.0 ms, $206e8c$)

Table 1: Minimum and maximum average execution time within each identified group. The compared commits are contained in every cell.

Similar to previous experiments [10], for TEAMMATES, we choose the commit sequence $\langle 648425, 48b67b, 83f518, f33d0b, ce4463 \rangle$. For CWA-Server, we choose $\langle 7e1b61, 6e9702, c22f93, 33d1c9, 9323b8, 206e8c, 3977e6, 94bca6a \rangle$. For each pair (c_i, c_j) of commits within these sequences, we compared their corresponding Java models in consecutive ($(c_i, c_j) \in CP$) and reverse order ($(c_j, c_i) \in CP$), resulting in $2 * (N - 1)$ comparisons, where N is the number of commits in a sequence. Additionally, every Java model was compared to itself ($(c_i, c_i) \in CP$, N comparisons). In total, this leads to $1 + 3 * (N - 1)$ comparisons per sequence (i.e., 13 for TEAMMATES and 22 for CWA-Server).

The model sizes for TEAMMATES range between 406,180 and 412,826 model elements and 585,925 and 596,336 references.

For CWA-Server, the model sizes range between 218,079 and 224,292 model elements and 303,642 and 312,560 references. We executed the measurements on a laptop with an Intel Core i7-7700HQx8 CPU, 16 GB RAM, Fedora Linux 42, and Java 11. Every comparison was repeated 100 times.

3.3 Results

Figure 1 shows the measurements of all comparisons. We identified four distinct groups: one for the comparison of different models in TEAMMATES (red circles), one for the comparison of identical models in TEAMMATES (grey pluses), one for the comparison of different models in CWA-Server (blue Xs), and one for the comparison of identical models in CWA-Server (green minuses).

For the analysis of single measurement series, we calculated the average execution time and standard deviation, displayed in Table 1. In addition, we checked if they are independent and identically distributed [4]. For 26 of 35 measurement series, this was not the case so that we did not further analyze them.

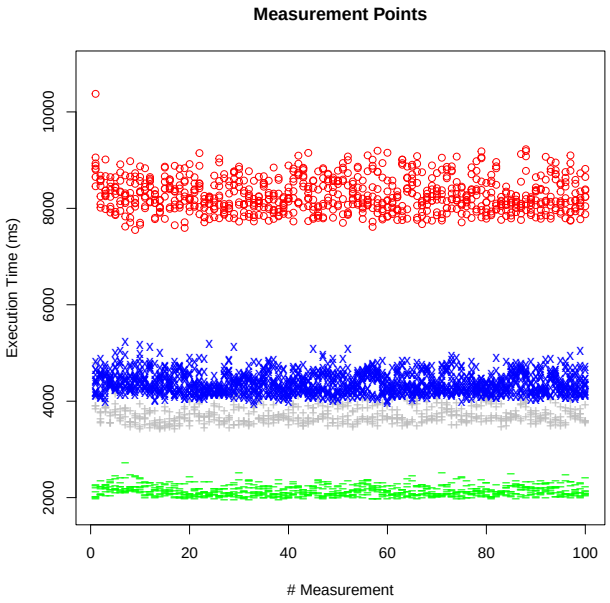

Figure 1: Results for all measurement series. Four groups can be identified: group 1 (red circles) for different models in TEAMMATES, group 2 (grey pluses) for identical models in TEAMMATES, group 3 (blue Xs) for different models in CWA-Server, and group 4 (green minuses) for identical models in CWA-Server.

3.4 Discussion

As Figure 1 outlines, there is a difference in the execution times between TEAMMATES and CWA-Server. They roughly correspond to the model sizes of the TEAMMATES and CWA-Server commits so that the difference could be largely attributed to the differences in model sizes.

However, the different model sizes cannot explain the time difference between the comparisons of identical and different models. As a consequence, there are further factors influencing the performance and scalability of the state-based comparisons. In particular, one factor can be the number of single comparisons for matching model elements. We hypothesize that this number is larger for different models, which require more comparisons for unequal elements. Another factor can be the model structure of the Java models. While we only observed the number of model elements and references, each Java model exhibits non-trivial structures, which can have an influence on the comparison. A third potential factor can be the comparison of single model elements for matching them, as they are language-specific and depend, therefore, on the concrete elements.

4 Conclusion

In this paper, we present a first evaluation of the performance of the state-based model comparison in CIPM for selected projects. This is a first step towards investigating the scalability of state-based comparisons. Within the measurements, we could identify four different groups, which are not only influenced by the model sizes. In future work, we plan to further analyze the measurements, investigate the factors influencing the performance, and assess the scalability of the state-based comparison.

All code and data of this paper are available online.[2]

Acknowledgments

This work was supported by the DFG (German Research Foundation) with the Collaborative Research Center "Convide" — SFB 1608 — 501798263 and funded by the topic Engineering Secure Systems, KASTEL Security Research Labs by the Helmholtz Association (HGF).

References

[1] C. Brun and A. Pierantonio. "Model differences in the eclipse modeling framework". In: *UPGRADE, The European Journal for the Informatics Professional* 9.2 (2008), pp. 29–34.

[2] D. S. Kolovos et al. "Different models for model matching: An analysis of approaches to support model differencing". In: *2009 ICSE Workshop on Comparison and Versioning of Software Models*. 2009, pp. 1–6.

[3] M. Koegel et al. "Operation-based conflict detection". In: *Proceedings of the 1st International Workshop on Model Comparison in Practice*. IWMCP '10. Malaga, Spain: Association for Computing Machinery, 2010, pp. 21–30.

[4] J.-Y. Le Boudec. *Performance Evaluation of Computer and Communication Systems*. 1st. Lausanne, Switzerland: EPFL Press, 2011.

[5] P. Langer. *EMF Compare*. 2019.

[6] M. Mazkatli et al. "Incremental Calibration of Architectural Performance Models with Parametric Dependencies". In: *IEEE International Conference on Software Architecture (ICSA 2020)*. Salvador, Brazil, 2020, pp. 23–34.

[7] H. Klare et al. "Enabling consistency in view-based system development – The Vitruvius approach". In: *Journal of Systems and Software* 171 (2021).

[8] *TEAMMATES Developer Web Site*. June 29, 2022.

[9] *Corona-Warn-App Server*. May 10, 2023. URL: `https://github.com/corona-warn-app/cwa-server` (visited on 08/05/2025).

[10] M. Mazkatli et al. "Continuous integration of architectural performance models with parametric dependencies – the CIPM approach". In: *Automated Software Engineering* 32.2 (May 29, 2025).

[2] `https://doi.org/10.5281/zenodo.17348850`