Scalability of Consistency Preservation with Vitruvius

Benedikt Jutz
benedikt.jutz@kit.edu

Thomas Weber
thomas.weber@kit.edu

Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract

The Vitruvius tool allows users to develop software-intensive systems by interacting with the underlying models through views suitable to users' tasks. When users change the system by applying model deltas, Vitruvius automatically preserves model consistency. However, it is currently unknown how well this consistency preservation scales in practice. Therefore, our goal is a first estimation of the scalability of consistency preservation in Vitruvius with regard to the number of model deltas. To that end, we conduct a performance evaluation by transforming existing UML models into a series of deltas and measure the time consistency preservation required for processing them. While applying model deltas to reconstruct the UML model, consistency preservation constructs a coupled Java code model consistent with the UML model. Through a linear regression analysis, we show that for the use case described above, consistency preservation scales nearly linearly with the number of deltas ($\rho = 0.97$). However, our evaluation is limited by the choice of Vitruvius metamodels, existing UML models, and only one scenario for consistency preservation. Therefore, we aim to conduct an extended performance evaluation in the future.

Index terms— Vitruvius, scalability, performance, consistency preservation

1 Introduction

Systems of reasonable complexity, e.g. cyber-physical systems, are developed by a heterogeneous team of experts from different domains. Orthographic Software Modeling (OSM) [4] is a development approach for such systems where experts interact with it through suitable views. These provide access to the Single Underlying Model (SUM) describing the entire system.

In the base OSM approach, the SUM is constructed from scratch. However, this is far less efficient and feasible than integrating existing parts of the system that experts already work with. To reflect this, Vitruvius [8], a practical implementation of the OSM approach for the Eclipse platform[1], allows the reuse of existing models by combining them into a Virtual SUM (V-SUM). To preserve consistency between those models, Consistency Preservation Rules (CPRs) are defined. CPRs are transformations to specify how to transform one model in response to a change in another model. Vitruvius applies the CPRs automatically whenever a change occurs.

However, it is currently unknown how the consistency preservation mechanism in Vitruvius scales with the number of model deltas. Scalability matters especially for the industry use of Vitruvius, where models get huge. The problem is that consistency preservation may require exponential time in terms of the number of CPRs [8]. Previous work on Vitruvius did not investigate the performance in practice. In [9], Klare et al. assert that performance depends on the number of deltas and, therefore, would not be a problem. In [10], Klare and Gleitze assert that the upper runtime bound for restoring consistency will probably not be reached in practice.

In our work, we conduct a performance evaluation to gain early insights into how consistency preservation scales in Vitruvius regarding the number of model deltas. For that, we transform existing UML models into a sequence of model deltas and apply them against an empty V-SUM that combines UML and Java models. We measure the number of deltas and the overall time to restore consistency, i.e., restore the UML model.

The remainder of this paper is structured as follows. Section 2 explains how Vitruvius restores consistency in response to user changes. Section 3 describes how we planned our performance evaluation, and Section 4 presents the evaluation results. We discuss these results and the shortcomings of our evaluation in Section 5 and conclude with Section 6.

2 Background

Here we explain how Vitruvius preserves consistency in response to model deltas applied by users. We take the UML-Java case study of Vitruvius as our example [2].

Example Consider a software project where software architects define the system structure with UML class diagrams, and programmers implement that system in Java. Both types of developers want the structure and implementation to be consistent. For example, an architect defines a new interface in the class diagram,

[1]`https://eclipseide.org`

[2]`https://github.com/vitruv-tools/Vitruv-CaseStudies/tree/main/bundles/tools.vitruv.applications.umljava`

extracts multiple occurrences of the same method definition across different classes, and has those classes implement the new interface. Then, the program code should contain this new interface, and programmers should be able to work with it.

To support such scenarios with Vitruvius, we first define a V-SUM metamodel (V-SUMM) from two metamodels, one for UML diagrams, and one for Java code, e.g. from JaMoPP [3]. Then, we add so-called *view types* [9] to project views from UML and Java model elements. For example, these views are class diagram editors and code editors enhanced to link to their counterparts. Vitruvius also applies changes occurring on views on the underlying models. These changes or *model deltas* can be either atomic or composed of other model deltas [8]. Atomic deltas include adding or removing model elements, changing the values of their attributes, and setting references to other model elements [6].

Finally, to preserve consistency, we define preservation rules in the `Reactions` language, consisting of `Reactions` and `Routines` [9]. A `Reaction` calls one or more `Routines` whenever a model delta with a given delta type and model element type occurs. For example, one `Reaction` handles the creation of a new interface in the class diagram. The called `Routine` first checks that a given condition holds on the target model and then transforms the target model with additional model deltas, and optionally, by calling other `Routines` in turn. For example, our sample routine first checks that no Java interface corresponding to the newly added UML interface exists in the code base. It then creates a new addition delta for that interface in the Java code base and calls other `Routines` to set its visibility, name, and other properties.

Multiple Transformations Keeping more than two models consistent, through multiple CPRs between them, is a non-trivial task. Because `Routines` create other model deltas, they may trigger other `Reactions`, leading to potentially complicated interactions. In the worst case, consistency preservation may fail or take exponential time [8]. This is because firstly, the CPRs in a V-SUM must be compatible, i.e., the consistency relations they preserve can be fulfilled simultaneously and do not contradict each other. Secondly, CPRs must be applied in a suitable order to preserve consistency. Finding such an order is a non-decidable problem, however. Even when it exists, finding it may require exponential time ($\mathcal{O}(2^{|\mathbf{t}|})$, where $\mathbf{t}$ is the set of CPRs) [8].

3 Evaluation Planning

In this section, we describe how we conducted the performance evaluation. We followed the guidelines of **Jain** [1]. A replication package is available [11].

System Under Test We ran measurements with the nightly development version of Vitruvius. Our setup was a laptop with an AMD Ryzen 7 Pro 4750U CPU and 32 GB of RAM, running Ubuntu 24.04 LTS, OpenJDK 21.0.4, and the Eclipse Modeling Tools 2024-06. Our V-SUM metamodel was the UML-Java example in Section 2.

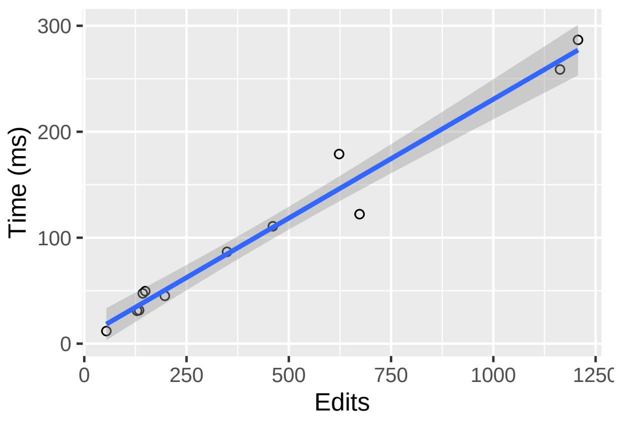

Figure 1: Distribution of the number of deltas in our UML models, and time for consistency preservation. Models are represented by dots.

Workload We used 12 UML models. Eight are from the MediaStore case study system [5], two of them are synthetic, one represents an example UML model[3], and the last one a Java logging application[4]. To exercise consistency preservation, we decided to recreate the UML models in the UML-Java V-SUM. In doing so, Vitruvius transforms the model state into a series of model deltas to reconstruct said state. It then applies those deltas on the initially empty UML model and recreates the corresponding Java model by applying CPRs.

Metrics For each UML model, we measured D as the number of model deltas required for reconstructing it and T as the time in milliseconds required for consistency preservation. We did not consider other properties of the V-SUM itself, like its existing CPRs, or our workload, like single model deltas and their type, or the elements they change.

Measurement We repeated our measurements to reduce measurement errors caused by other programs running concurrently. Following [7, Formula 4.25], we computed the required number of repetitions for all models as $n = 210$, for a relative error bound of $e = 0.02$ and significance value of $\alpha = 0.05$. We estimated the required standard deviation σ_{T_i} and mean μ_{T_i} of each model i in 30 test runs. For later analysis, we used the arithmetic mean of all measurements of T_i.

[3]`https://repository.genmymodel.com/suresh519/MyProject`

[4]`https://github.com/orhanobut/logger`

4 Evaluation Results

Data Description Values for D range between $\min_D = 54$ and $\max_D = 1207$ deltas, with a mean of $\mu_D = 440.2$ and standard deviation of $\sigma_D = 402.13$. Values for T range between $\min_T = 11.87ms$ and $\max_T = 286.68ms$, with $\mu_T = 105.04ms$ and $\sigma_T = 91.60ms$. Figure 1 shows the distribution of D and T.

Statistical Tests To test for a positive correlation of the number of model deltas D with the runtime T for consistency preservation, we computed Spearman's correlation coefficient. We identified a significant, large effect of D on T ($\rho = 0.97, p < 2.2 * 10^{-16}$). D and T do not follow a multivariate normal distribution, as reported by the Henze-Zirkler test ($HZ = 0.98, p < 0.005$), so we did not use Pearson's correlation coefficient. To predict T from D, we used linear regression. This resulted in the linear model $T = (6.2884 + 0.2244D)ms$. This model fits well with $R^2 = 0.967, F = 323$, and $p < 6.1 * 10^{-9}$. Figure 1 shows this approximation as a blue line.

5 Discussion

For the given V-SUM metamodel and UML models, consistency preservation runtime scaled linearly to the number of edits. However, we note that our evaluation has multiple threats to validity, as classified by Jedlitschka, Ciolkowski, and Pfahl [2]:

External Validity We did not have the time to construct We chose a simple V-SUM metamodel with only two underlying metamodels (UML and Java) and small UML models. Both parts were already part of the Vitruvius case studies. However, realistic V-SUMs are built from tens of thousands of model elements and deltas and include more metamodels. Keeping all these models consistent requires more CPRs, leading to more complex interactions and more time for consistency preservation. To ensure external validity, we must repeat our evaluation with larger V-SUMs and underlying V-SUM metamodels.

Construct Validity We chose to evaluate scalability for one application scenario only: reconstructing another model in a V-SUM through consistency preservation when loading an existing base model. This choice was because we did not have access to any delta sequences for V-SUMs in Vitruvius. However, consistency preservation mainly occurs when editing the V-SUM. In that scenario, the applied composite deltas, e.g. adding a single model element, have fewer atomic model deltas. Additionally, V-SUM users may delete model elements or change their attributes repeatedly. Our model reconstruction scenario does not include such edits. In both cases, these types of edits might have different runtimes. To ensure construct validity, we could try to obtain existing delta sequences from Vitruvius users or synthetically construct them, like Kegel et al. [12].

6 Conclusion

We started to investigate how consistency preservation scales in Vitruvius. We conducted a performance evaluation in which we derived UML-Java V-SUMs by transforming existing UML models into a sequence of model deltas and applying these to an empty V-SUM. We recorded the number of deltas and the time consistency preservation required to reconstruct the Java side. Our statistical analyses identified a linear correlation between these two variables. As future work, we will extend this evaluation with larger, more realistic V-SUMs and more realistic delta sequences. We will further consider other metrics, like CPU and RAM consumption.

Acknowledgements

Author B. Jutz is funded and T. Weber is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1608 - 501798263.

References

[1] R. Jain. *The art of computer systems performance analysis*. Wiley professional computing. New York: Wiley, 1991.

[2] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. "Reporting Experiments in Software Engineering". In: *Guide to Advanced Empirical Software Engineering*. Ed. by F. Shull, J. Singer, and D. I. K. Sjøberg. London: Springer London, 2008, pp. 201–228.

[3] F. Heidenreich et al. *JaMoPP: The Java Model Parser and Printer*. Tech. rep. Technical University Dresden, Sept. 2009.

[4] C. Atkinson, D. Stoll, and P. Bostan. "Orthographic Software Modeling: A Practical Approach to View-Based Development". In: *Evaluation of Novel Approaches to Software Engineering*. Ed. by L. A. Maciaszek, C. González-Pérez, and S. Jablonski. Berlin, Heidelberg: Springer, 2010, pp. 206–219.

[5] M. Strittmatter and A. Kechaou. *The Media Store 3 Case Study System*. Karlsruhe Institute of Technology, 2016.

[6] M. E. Kramer. "Specification Languages for Preserving Consistency between Models of Different Languages". PhD thesis. Karlsruhe Institute of Technology, 2017.

[7] S. Kounev, K.-D. Lange, and J. von Kistowski. *Systems Benchmarking*. 1st ed. Cham: Springer, 2020, pp. XXVII, 426.

[8] H. Klare. "Building Transformation Networks for Consistent Evolution of Interrelated Models". PhD thesis. Karlsruhe Institute of Technology, 2021.

[9] H. Klare et al. "Enabling consistency in view-based system development — The Vitruvius approach". In: *Journal of Systems and Software* 171 (Jan. 2021), p. 35.

[10] H. Klare and J. Gleitze. "Termination and Expressiveness of Execution Strategies for Networks of Bidirectional Model Transformations". In: *Form. Asp. Comput.* 35.3 (Sept. 2023).

[11] B. Jutz and T. Weber. *Scalability of Consistency Preservation in Vitruvius*. Version 1.0.0. URL: `https://doi.org/10.5281/zenodo.13629227`. Sept. 2024.

[12] K. Kegel et al. "A Variance-Based Drift Metric for Inconsistency Estimation in Model Variant Sets". In: *Journal of Object Technology* 23.3 (July 2024). The 20th European Conference on Modelling Foundations and Applications (ECMFA 2024), pp. 1–14.