Towards Continuous Integration of Performance Models for Lua-Based Sensor Applications

Manar Mazkatli
manar.mazkatli@kit.de

Martin Armbruster
martin.armbruster@kit.edu

Anne Koziolek
koziolek@kit.de

Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany

Abstract

Architecture-level performance models (aPMs) provide valuable insights for design decisions with performance predictions. Due to source code changes and system adaptations, it remains challenging to keep aPMs up-to-date and ensure their consistency during agile software development. The Continuous Integration of Performance Models (CIPM) approach focuses on maintaining the consistency between aPMs and software artifacts. After each significant change during the software development and operation, CIPM automatically updates aPMs.

However, the current implementation of CIPM is limited to Java- and microservice-based applications.

In this paper, we evaluate whether conceptual changes are required if CIPM is applied to the Lua programming language and industrial sensor applications. Our evaluation is based on a real Lua-based sensor application from the SICK AppSpace ecosystem and an artificial one. The findings demonstrate the feasibility of CIPM as no significant conceptual changes were required, but rather technical ones.

1 Introduction

Software systems are becoming more complex due to frequent changes in the source code at development time (Dev-time) and system composition at operation time (Ops-time). Architecture-level Performance Models (aPMs) [1, 5] model the software architecture to decrease the complexity and understand the software system. Besides, aPMs enable efficient assessments of performance impacted by design decisions.

Agile development complicates the task of keeping aPMs up-to-date with software artifacts because of the frequent changes and short or no design phases. Existing approaches for the consistency management lack a comprehensive handling of all changes affecting an aPM leading to concerns about the accuracy of aPMs and related performance predictions.

To address these challenges, the Continuous Integration of architectural Performance Models (CIPM) approach was proposed [6]. CIPM automatically updates a parameterized aPM after source code changes at Dev-time and changes at Ops-time (e.g., deployment and workload). This supports the comprehension of software systems and cost-effective proactive identification of performance issues.

However, the current prototypical implementation of CIPM is limited to Java- and microsevice-based applications. This paper aims to evaluate the applicability of CIPM for another programming language and technology. In particular, we aim to answer the following research question: *Are there conceptual changes required if the approach is implemented for another programming language or technology?* Hence, we adapt and evaluate the CIPM approach for Lua-based sensor applications. Our key contributions are:

- A metamodel and parser for Lua (based on the Xtext grammar from Melange [4])
- Consistency Preservation Rules (CPRs) keeping Lua models, an aPM and Instrumentaion Model (`IM`) consistent to apply the CIPM approach

We evaluate our extensions with one real-world and one artificial application from the SICK AppSpace[1]. The evaluation indicates the applicability of the CIPM approach to SICK AppSpace apps so that no conceptual changes are required. However, additional technical changes are still necessary.

In the following section (Section 2), we introduce the required background. Then, the approach in Section 3 is followed by its evaluation in Section 4. At last, the paper is concluded with related work and a conclusion in Section 5 and Section 6, respectively.

2 Foundation

The *CIPM approach* updates an aPM incrementally enabling architecture-based performance predictions [6]. It also ensures consistency between the aPM and software artifacts (source code and measurements). Thus, CIPM employs VITRUVIUS which is a model-based consistency preservation platform [7]. VITRUVIUS allows the definition of CPRs specifying how to preserve consistency between related elements for a given specific change.

During Dev-time, CIPM detects changes in a Git repository and applies them in VITRUVIUS so that CPRs are executed and the related parts in the aPM

[1]See `https://www.sick.com`.

and an IM are updated [6]. The IM contains instrumentation points for code parts that have been changed by the recent commit. Consequently, the adaptive instrumentation of CIPM generates instrumented code to monitor the instrumented parts.

CIPM monitors the changed code of the application adaptively (i.e., monitoring is deactivated after an accurate calibration) to also reduce the monitoring overhead [6]. The resulting monitoring data is used to incrementally calibrate the aPM's parameters considering their parametric dependencies and enabling performance predictions.

CIPM's current implementation uses a Java code model [8], the Palladio Component Model (PCM) as aPM and measurements from the Kieker monitoring tool [2]. The PCM is a framework to model and analyze component-based architecture including the component's abstract behavior expressed as Service Effect Specifications (SEFF) and their contained actions [5].

3 Approach

This paper presents an extension to the CIPM approach to support Lua-based sensor applications from the SICK AppSpace. Therefore, we extend CIPM with a metamodel for Lua (subsection 3.1) and CPRs to update the PCM and IM (subsection 3.2). The implementation and evaluation of this extension were carried out by Burgey during his master thesis [9].

3.1 Modeling Lua Code

As the CPRs in VITRUVIUS are defined at the metamodel level, a metamodel for Lua is required. Therefore, we extend an existing Xtext-based[2] Lua grammar from the Melange project [4] and generate the metamodel from it. An excerpt of the metamodel is depicted in Figure 1. The generated parser creates a model from a single code file with the root element `Chunk` representing the content of the Lua file. The content itself consists of a `Block` acting as the container for `Statements`. Because CIPM requires one model for the entire source code encompassing all individual models, they are combined into one model (the `Application`). Additionally, every `Chunk` is encapsulated in a `SourceFile` representing the file from which the `Chunk` was parsed, and several `SourceFiles` are contained within an `App` element which describes a SICK AppSpace app allowing their modeling.

3.2 Consistency Preservation Rules

The CPRs are mainly focused on updating the PCM if the code changes. In this context, every `App` is mapped to a PCM component so that a component is created or deleted if an `App` is created or deleted. Additionally, apps expose functions as their API with `serveFunction` function calls. Therefore, we generate

[2]Xtext is a development framework for languages and provides the generation of metamodels, parsers, serializers and more. See `https://eclipse.dev/Xtext/`.

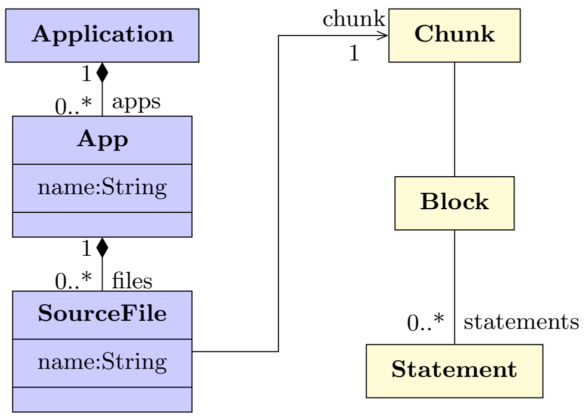

Figure 1: Excerpt from the Lua metamodel [9]. Blue classes are extensions.

an interface for every component in which the served functions are contained as the component's services. Due to the dynamic typing of Lua, we currently use one data type to represent any type in the Lua model.

To model the abstract behavior of the apps' exposed functions, we introduced CPRs on the statement level which update the SEFFs fine-grained. If a statement is affected by a change, contained function calls are extracted and classified regarding the targeted component (the same or a different one). Based on this classification, the statement is assigned to an existing or newly created SEFF action: existing if the targeted component is the same and there is a suitable action for internal calculations and new if there is no such action or the targeted component is a different one. In contrast, if statements are removed, corresponding actions are only removed if the statement was the last one related to the action.

We also define CPRs that create instrumentation points in IM for each new action in PCM to later calibrate it. Since new statements can be assigned to an existing SEFF action, a CPR ensures that the updated SEFF actions also have corresponding activated instrumentation points.

4 Evaluation

In our evaluation, we aim to apply the CIPM approach to Lua-based sensor applications. This raises the question: How accurately can the CIPM approach update the models of the aforementioned applications? Additionally, we explore the extent to which the defined CRPs can decrease the necessary monitoring overhead. Thus, we took the Git history of two sensor applications and put the commits into CIPM to simulate the development of the applications. For every commit, we then check that the models (Lua Model, PCM and IM) are accurately updated by calculating two metrics [9]: the Jaccard Coefficient for comparing updated Lua and PCM models with reference ones to measure their similarity, and the F-Score to match IM elements with the recently updated PCM elements, ensuring IM has all required instrumentation points.

The first application we used is a tiny artificial one built from sample apps and covers 7 commits, with overall 862 added and 283 removed lines affecting one to four files. The second one is a real-world app for detecting and sorting objects from images based on colors, whose history comprises 12 commits with 6651 added and 2663 removed lines affecting one to 13 files.

The results show accurate updates in all models based on commits from both applications, except for 6 commits within the second application. These 6 commits led to accurate updates in nearly all elements of the Lua model (at least 94.7%), with discrepancies arising only from interchanged statements. The algorithm to match Lua elements to detect changes struggles in this case in handling the element order. This affected the subsequent PCM update, resulting in a minimum accuracy of 94% for those 6 commits. For the `IM`, it was accurately updated by the 6 commits except one, where just one instrumentation point was not activated as intended, resulting in a 99% accuracy.

To conclude, the limitation in the implemented Lua matching algorithm is due to the implementation itself, not conceptual constraints. In the future, we rectify this issue for exact Lua model updates.

Concerning the required monitoring overhead based on the updated `IM`, the results indicate a reduction of up to 77.1% for the first application and 73.5% for the second one in the number of instrumentation points, compared to the potential required number without applying CIPM's adaptive instrumentation.

To conclude, our observation reveals nearly accurate model updates, barring minor technical issues in model matching implementation, and confirms overhead reduction without conceptual changes to CIPM.

5 Related Work

Lua Analysis in Rascal (AiR) by Klint et al. [3] is a framework to statically analyze Lua programs that customize the behavior of games. Implemented in Rascal, Lua AiR parses the Lua files and performs different checks including a type check by considering the interfaces between the Lua code and the game engine. Compared to Lua AiR, our extensions to CIPM use an Xtext-based grammar with a generated metamodel instead of Rascal. Besides, types and interfaces between Lua and its execution environment are currently not considered in CIPM while we target aPMs.

6 Conclusion

The CIPM approach enables the automatic update of aPMs to analyze software systems. Its current implementation is limited to Java and microservice-based applications. In this paper, we investigated if conceptual changes are required to support another programming language or technology, particularly Lua-based sensor applications from the SICK AppSpace.

CIPM is extended with a metamodel and parser for Lua and CPRs to update the PCM and `IM`. Evaluation with two sensor applications indicates that CIPM can effectively update models for Lua-based sensor applications with reductions in required monitoring overhead. Furthermore, the absence of conceptual changes in the CIPM approach highlights its potential suitability for model-based performance prediction across diverse application domains that use different programming languages and technologies.

In future work, we plan to complete the support for Lua-based sensor applications by implementing the adaptive instrumentation for Lua. Afterwards, we want to evaluate the full CIPM approach with more real-world Lua applications and other languages.

Acknowledgment

This publication is based on the research project SofDCar (19S21002) that the German Federal Ministry for Economic Affairs and Climate Action funds. The first two authors contribute equal to this work.

References

[1] D. A. Menascé, V. A. Almeida, and L. W. Dowdy. *Performance by Design: Computer Capacity Planning by Example.* Prentice Hall, 2004.

[2] A. van Hoorn, J. Waller, and W. Hasselbring. “Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis”. In: *Proceedings of the 3rd ACM/SPEC Intl. Conference on Performance Engineering.* 2012.

[3] P. Klint, L. Roosendaal, and R. van Rozen. “Game Developers Need Lua AiR”. In: *Entertainment Computing - ICEC.* Springer, 2012.

[4] T. Degueule et al. “Melange: A Meta-Language for Modular and Reusable Development of DSLs”. In: *Proceedings of the 2015 ACM SIGPLAN Intl. Conference on Software Language Engineering.* SLE 2015. ACM, 2015, pp. 25–36.

[5] R. H. Reussner et al. *Modeling and Simulating Software Architectures – The Palladio Approach.* Cambridge, MA: MIT Press, Oct. 2016. 408 pp.

[6] M. Mazkatli et al. “Incremental Calibration of Architectural Performance Models with Parametric Dependencies”. In: *IEEE Intl. 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] M. Armbruster. *Parsing and Printing Java 7-15 by Extending an Existing Metamodel.* Tech. rep. July 28, 2022.

[9] L. Burgey. “Continuous Integration of Performance Models for Lua-Based Sensor Applications”. Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology (KIT), 2023.