[THM+25] B. Taghavi, R. Heinrich, A. Marin, B. Rumpe, S. Stüber, S. Weber:
Semantic Validation for Slingshot Simulator Using MontiArc.
In: Softwaretechnik-Trends Band 45, Heft 1, Gesellschaft für Informatik e.V., Feb. 2025.

Semantic Validation for Slingshot Simulator Using MontiArc

Bahareh Taghavi[1], Robert Heinrich[1], Adrian Marin[2], Bernhard Rumpe[2], Sebastian Stüber[2], and Sebastian Weber[3]

[1] Karlsruhe Institute of Technology, Germany
[2] Software Engineering, RWTH Aachen University, Germany, https://se-rwth.de
[3] FZI Research Center for Information Technology, Germany

Abstract

As software systems become increasingly complex, the demand for effective analysis has grown substantially. To meet this demand, various analysis techniques and components need to be integrated that cover the domain comprehensively. However, it is crucial to ensure that these components work together in a coherent and semantically meaningful manner. The purpose of this paper is to investigate validating the semantic correctness of the Slingshot simulator, which illustrates the composition of three distinct components of analysis. Semantic correctness in system analysis ensures that interactions and data exchanges between components accurately reflect the intended behavior and properties of the system. We use the MontiArc framework to enhance component models with constraints. It demonstrates the behavior of components using automata and verifies that these constraints are valid within automata transitions, ensuring that the automata do not produce invalid results.

1 Introduction

As software systems have grown larger and more complex, the need for tools and techniques to analyze their properties has increased significantly. By leveraging modular analysis components, developers can reuse and combine different techniques and components to address unique challenges posed by complex software architectures, as discussed in [7] for the composition of metamodels. Furthermore, composing these components into a cohesive framework facilitates comprehensive evaluation, ensuring different aspects of a system are considered thoroughly. However, composing modular analysis components can present challenges. These include ensuring compatibility between different components and maintaining semantic integrity.

Slingshot [8] exemplifies this kind of composition by integrating three different analysis components based on the Palladio Component Model [6]. While it provides the syntactic interfaces necessary for proper communication between components, it does not validate the semantic correctness of their composition. Given that Slingshot is an extensible framework, any changes to one of these components or the addition of new components must also be semantically validated to ensure the correct behavior of the overall system.

MontiArc [10] is a textual modeling language designed to describe component and connector systems. It can verify and validate semantic constraints specified at the ports of components that are connected through connectors. A previous study [9] proposed adding constraint checking to MontiArc by using Slingshot as a running example. MontiArc also supports modeling internal behavior of components using the MontiArcAutomaton modeling language. With this capability, we can model each component's behavior by automata and ensure that modified components work correctly within the current composition.

Contribution In this paper, we explore modeling the simplified behavior of each component of Slingshot while specifying constraints at the ports of the components so we can validate the semantics of their composition. We can use it to verify that these constraints are valid in the automata and ensure that the automata do not produce invalid outputs as a result.

2 Slingshot Simulator: An Extensible Event-Driven Simulator

Palladio [6] is an approach to model, simulate, and analyze software architectures, keeping a focus on performance, maintainability, reliability, and other quality properties. Slingshot [8] is the latest simulator for Palladio based on the event-driven architecture. Slingshot consists of three analysis components: `UsageSimulation`, `SystemSimulation`, and `ResourceSimulation`. As shown in Figure 1, each component has incoming and outgoing ports, with specific constraints applied to them. The simulation begins with `UsageSimulation`, which uses usage scenarios to model individual use cases of the system. Each `UsageScenario` includes a workload that describes usage intensity. Then, the `UsageSimulation` simulates user requests as calls to the system. The `SystemSimulation` simulates the system behavior and then sends the demand for resources to `ResourceSimulation` to simulate the resource environment. The communication continues

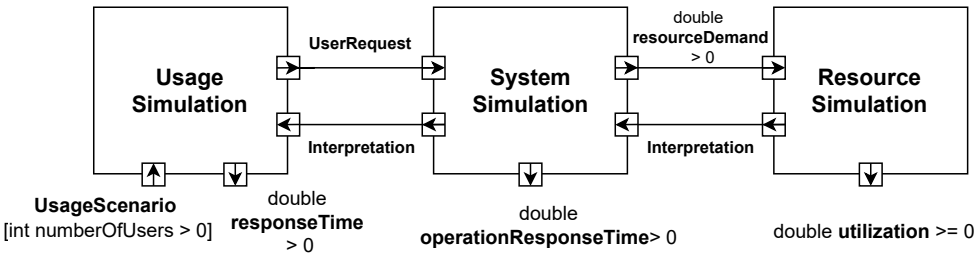

Figure 1: Slingshot MontiArc Model.

back and forth between the components until performance metrics such as `responseTime` and hardware `utilization` can be simulated.

These components interact through well-defined interfaces; however, they do not inherently support the validation of their composition. In order to ensure all components function as intended within the larger system, a rigorous validation approach is required.

3 Methodology

MontiArc [10] is a modeling language and framework for developing component and connector architectures. It provides a structured way to design, analyze, and implement complex systems by defining components and their interactions. Components exchange information through unidirectional connectors that link directed ports of compatible data types. The definition of inter-component communication and MontiArc models simulation are based on Focus [1].

MontiArc was also extended to provide component I/O behavior using automata. The automata are defined by a finite number of states, with at least one state designated as the initial state. Each transition is characterized by a source and a target state.

The constraint checking from [9] is a black-box testing method that verifies system functionality without requiring knowledge of the internal implementation. In order to understand the internal behavior of components, a black-box view is not sufficient, since it only tests the interface behavior of components. By incorporating automata into each component, we gain a white-box perspective, enabling us to model and observe the component's internal behavior.

To enable verification capabilities using MontiArc, we first create a simplified model of Slingshot's components and their interfaces in the Component&Connector paradigm, modeling the communication between components as connections between ports, as shown in Figure 1. We add constraints to these ports that specify how the inputs and outputs of the components must be structured. Figure 1 shows parts of these constraints, such as the restriction of the number of users to a positive number in the usage scenario, which serves as the input for the `UsageSimulation` component. The outputs of the simulation are also constrained. For instance, `utilization` (an output result of `ResourceSimulation`) cannot be less than zero. According to Palladio's definition of utilization [6], it represents the percentage of time that the active resource was in a processing state during a fixed time period.

We illustrate the behavior of each component using state charts. As an example, Figure 2, shows a simplified `ResourceSimulation` component with only ActiveResources (e.g., CPU) depicted. The `Initial` state is triggered by a resource demand request, indicating the need to obtain a resource. If an active resource is idle, the `JobInitiated` transition moves the system from the `Idle` state to the job `Processing` state. Depending on the resource management policies, additional jobs can be initiated on the resource. We define a `counter` in the internal state that increments each time a job is initiated, indicating the number of jobs running on this resource. The `counter` then decreases whenever a job is finished, updating the number of running jobs. The `utilization` result is derived from this `counter` value.

Since component behavior is modeled with automata, inaccuracies or oversimplifications in the models can lead to incorrect validation results, as defective automata may not accurately reflect the real behavior of the components.

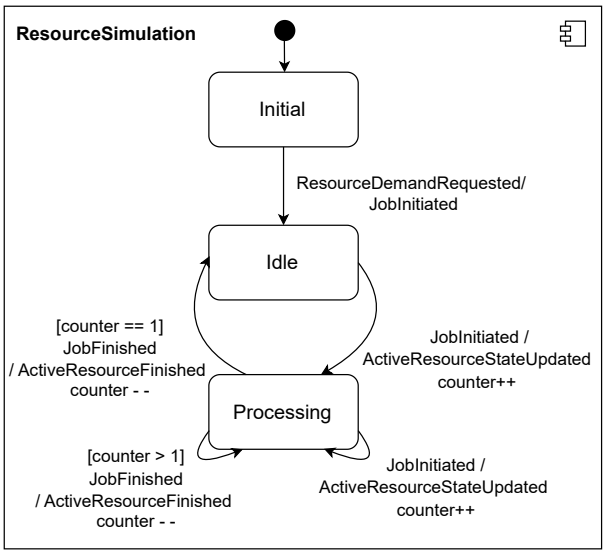

Figure 2: `ResourceSimulation` behavior.
Fulfills constraint from Figure 1: `counter` $\geq 0 \implies$ `utilization` ≥ 0.

Translation to SMT Problem We translate the constraints into an SMT (Satisfiability Modulo Theories) problem and solve this problem using the SMT-Solver Z3 [2]. Z3 has been successfully applied in various projects from both industry and academia [4, 5]. The solver then attempts to instantiate the variables of the SMT problem encoded into Boolean formulas and delivers either a *Satisfiable* judgement and a model or an *Unsatisfiable* judgement. The input of the SMT-Solver is standardized with SMT-Lib [3], which leverages a human-readable form of displaying SMT problems.

We aim to translate the constraints on the inputs and outputs of the components and prove that the automaton fulfills these constraints. We iterate over each transition of the automata to find a counterexample. This counterexample must fulfill the input-constraints and transition-definition, but not the output-constraints:

$$\forall s, s\_next \in State, i \in Input, o \in Output : \\ constraint_{input}(i) \wedge transition(s, i, s\_next, o) \\ \rightarrow constraint_{output}(o)$$

Each transition of the automaton displayed in Figure 2 processes an input event and produces an output event. The `counter` variable is never negative. When receiving a `Job` event, each action fulfills the output constraint, as the `utilization` is computed as a fraction of the counter [6]. This can only be guaranteed by the input assumption that the `ResourceDemand` is greater than 0 implying that the counter is by default initialized with 0. In the case the `ResourceDemand` is not greater than 0, we cannot conclude that the counter was initialized with 0, making a counterexample for the implication easy to find through both transitions incrementing the counter.

4 Conclusion

In this paper, we analyze the semantic validation of an event-based simulator called Slingshot. MontiArc is used to model the Slingshot components by specifying their ports and connections. This approach allows us to validate component compatibility by checking constraints over the ports. Additionally, we model the behavior of the components using automata, which provides a concrete understanding of the system's internal behavior. This method also enables us to verify that the outputs are meaningful and correct within the context of the automata. In future work, we will focus on automating SMT-translation and developing a tool to facilitate this process. We plan to evaluate the tool with a large use case. In addition, we intend to apply conformance checking [11] to ensure semantic consistency within compositions, particularly in scenarios where component replacements occur.

Acknowledgements

This work was funded by the DFG (German Research Foundation) – project number 499241390 (FeCoMASS), KASTEL Security Research Labs and "Kerninformatik am KIT (KiKIT)" funded by the Helmholtz Association (HGF).

References

[1] M. Broy and K. Stølen. *Specification and Development of Interactive Systems. Focus on Streams, Interfaces and Refinement.* Springer Verlag Heidelberg, 2001.

[2] L. De Moura and N. Bjørner. "Z3: An efficient SMT solver". In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer. 2008, pp. 337–340.

[3] C. Barrett, A. Stump, C. Tinelli, et al. "The smt-lib standard: Version 2.0". In: *Proc. 8th Int. workshop on satisfiability modulo theories.* Vol. 13. 2010, p. 14.

[4] N. Bjørner and K. Jayaraman. "Checking cloud contracts in Microsoft Azure". In: *Distributed Computing and Internet Technology: 11th International Conference, ICDCIT 2015. Proceedings 11.* Springer. 2015, pp. 21–32.

[5] R. Mukherjee, D. Kroening, and T. Melham. "Hardware verification using software analyzers". In: *2015 IEEE Computer Society Annual Symposium on VLSI.* IEEE. 2015, pp. 7–12.

[6] R. H. Reussner et al. *Modeling and simulating software architectures: The Palladio approach.* MIT Press, 2016.

[7] R. Heinrich, M. Strittmatter, and R. Reussner. "A layered reference architecture for metamodels to tailor quality modeling and analysis". In: *IEEE Transactions on Software Engineering* 47.4 (2019), pp. 775–800.

[8] J. Katić, F. Klinaku, and S. Becker. "The Slingshot Simulator: An Extensible Event-Driven PCM Simulator (Poster)". In: (2021).

[9] S. Weber et al. "Semantics Enhancing Model Transformation for Automated Constraint Validation of Palladio Software Architecture to MontiArc Models". In: *IEEE European Conference on Software Architecture (ECSA).* 2024.

[10] A. Haber. *MontiArc - Architectural Modeling and Simulation of Interactive Distributed Systems.* Aachener Informatik-Berichte, Software Engineering, Band 24. Shaker Verlag, 2016.

[11] M. Konersmann et al. "Towards a Semantically Useful Definition of Conformance with a Reference Model". In: *Journal of Object Technology (JOT)* 23.3 (July 2024), pp. 1–14.