[WHT+24] S. Weber, J. Henß, B. Taghavi, T. Weber, S. Stüber, A. Marin, B. Rumpe, R. Heinrich:
Semantics Enhancing Model Transformation for Automated Constraint Validation
of Palladio Software Architecture to MontiArc Models.
In: Software Architecture. ECSA 2024 Tracks and Workshops, A. Ampatzoglou, J. Pérez, B. Buhnova, V. Lenarduzzi,
C. C. Venters, U. Zdun, K. Drira, L. Rebelo, D. Di Pompeo, M. Tucci, E. Y. Nakagawa, E. Navarro (Eds.),
Volume 14937, pp. 30-38, LNCS, Springer Nature Switzerland, Cham, Sep. 2024.

Semantics Enhancing Model Transformation for Automated Constraint Validation of Palladio Software Architecture to MontiArc Models

Sebastian Weber[1(✉)], Jörg Henß[1], Bahareh Taghavi[2], Thomas Weber[2], Sebastian Stüber[3], Adrian Marin[3], Bernhard Rumpe[3], and Robert Heinrich[2]

[1] FZI Research Center for Information Technology, Karlsruhe, Germany
{sebastian.weber,henss}@fzi.de
[2] Karlsruhe Institute of Technology, Karlsruhe, Germany
{bahareh.taghavi,thomas.weber,robert.heinrich}@kit.edu
[3] Software Engineering, RWTH Aachen University, Aachen, Germany
{stueber,marin,rumpe}@se-rwth.de

Abstract. Component-based software architecture allows software architects to design systems by composing components with syntactically defined interfaces. These models can be used for the analysis and prediction of the functional and non-functional properties of the system. While tools for the modeling and analysis of such systems, e.g., the Palladio approach, support the syntactic validation of the composition, they lack the capability to validate the semantic composition. If, e.g., one component requires and one provides an integer value, they can be composed, independently of whether this composition is actually semantically sound. To support software architects in the semantic validation of their system models, we propose a model transformation tool, that allows to transform system models from Palladio models to MontiArc models, enrich them with semantic constraints and validate these constraints with the MontiArc workbench. We present exemplary results of this transformation and validation applied to a simplified model of a component-based simulator of the Palladio approach.

Keywords: Semantic Constraint Validation · Software Architecture · Model Transformation · Palladio · MontiArc

1 Introduction

Software plays an ever more important role in society and economy. To ensure it fulfils the requested functionality and quality properties, such as performance and security, a multitude of different analysis techniques are available. Because these analysis techniques are designed to evaluate specific questions, they use different modeling formalisms tailored to these questions. The Palladio simulators

A. Ampatzoglou et al. (Eds.): ECSA 2024, LNCS 14937, pp. 30–38, 2024.
https://doi.org/10.1007/978-3-031-71246-3_4

[15] and the Palladio Component Model (PCM) [15] are examples and while they enable the architectural modeling and analysis of component-based software systems, they do not support the validation of the composition of such components beyond a syntactic level, i.e., as discussed in [6]. As long as the syntactic interfaces between the components match, they can be composed, but it is crucial to also verify that all the components can work together and communicate properly by semantic validation. Because Palladio aims at analyzing quality properties of systems, we do not introduce support for semantic constraints directly in Palladio, but instead we introduce a tool that allows for the transformation from a PCM model to a MontiArc [4] model. MontiArc supports the textual modeling of component&connector systems and allows the validation of semantic constraint specified at the ports of the components, which are connected through the connectors. These constraints are specified in an additional model to avoid adding more complexity to the PCM and enrich the MontiArc model generated from the PCM. Our tool is publicly available at Github (https://github.com/FeCoMASS/Model-Transformation-for-Automated-Constraint-Validation) and an explanatory video is available at https://fecomass.github.io/fecomass/videos/.

Contributions: Our first contribution is the transformation of a PCM model to a MontiArc. The second contribution is the incorporation of constraints within a MontiArc model and the last contribution is the addition of constraint checking to MontiArc. Figure 1 shows the transformation and enrichment process our tool applies to check the semantic validity of the composition of the system modeled in the PCM.

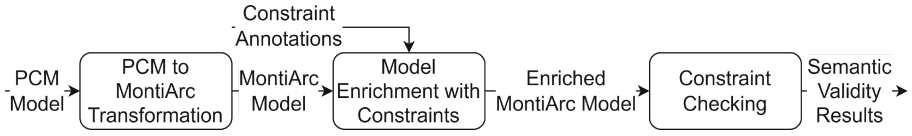

Fig. 1. Transformation Process and Artifacts

The following Sect. 2 introduces the Palladio approach, its simulators and the modeling formalism PCM. In addition, the model of a simplified exemplary system is presented. Section 3 introduces MontiArc and shows how the exemplary system is modeled there. The next Sect. 4 explains the automatic transformation between PCM and MontiArc, followed by Sect. 5 introducing how the constraints in the MontiArc model are checked to validate the modeled system. Section 6 concludes the paper with a discussion about the current state of the tool and an outlook on future work planned to expand it.

2 Palladio and Running Example

Palladio [15] is an approach for simulating software architecture, aiming to analyze and predict performance, among other quality properties. The tooling that

implements the Palladio approach is known as Palladio-Bench [7]. The Palladio Component Model (PCM) is a domain-specific modeling language and is composed of multiple sub-models, each targeting a particular developer role. Component developers contribute by specifying behavioral aspects of their components and interfaces in the repository model. Subsequently, system architects leverage these repositories to assemble concrete component-based software systems in the assembly model. Meanwhile, system deployers focus on modeling the resource environment and allocating components across different resources. Business domain experts are also responsible for providing usage models that describe critical usage scenarios and outline user behavior. Palladio facilitates model evaluation through simulation, enabling the prediction of performance metrics like response times and hardware utilization under specified workloads.

Running Example: We use a simplified model of Slingshot [9], the latest simulator for Palladio based on an event-driven architecture, to showcase the PCM models our tool requires as depicted in Fig. 2. Slingshot currently comprises three simulation components: the *UsageSimulation*, the *SystemSimulation*, and the *ResourceSimulation* component. In order to start a simulation, the *UsageSimulation* component looks up the workload from a usage scenario and begins interpretation based on its parameters. The connection between the *UsageSimulation* and *SystemSimulation* components occurs via user requests, which model service calls within the system. In addition, the *SystemSimulation* component and the *ResourceSimulation* component are connected by requesting resource demands.

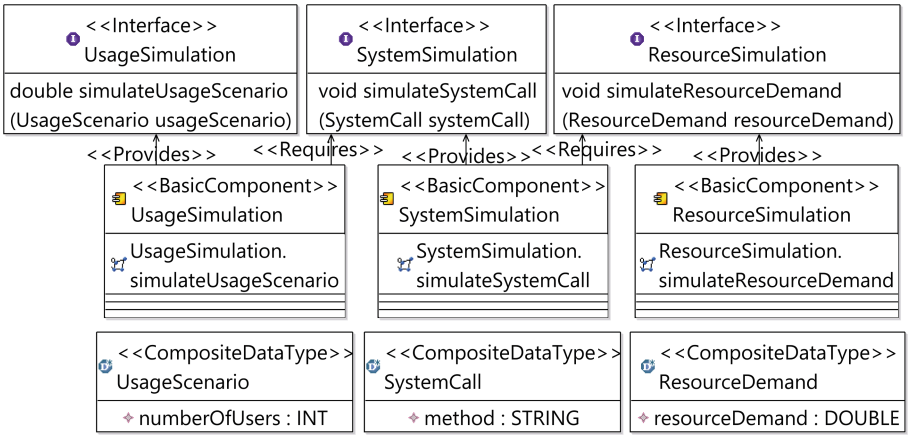

Fig. 2. Repository Model of Slingshot

The components of the repository are instantiated in the assembly model shown in Fig. 3. These assembly contexts are connected through directed connectors to either delegate from a system role to an assembly context role or between provided and required assembly context roles.

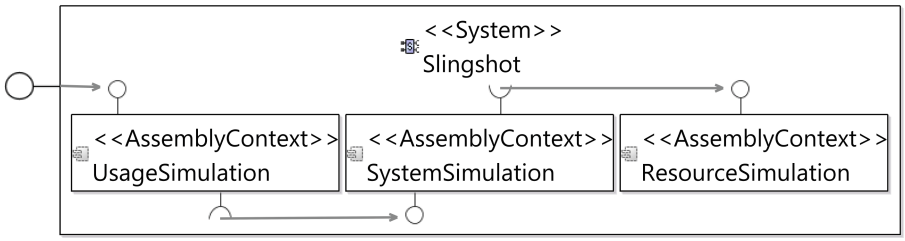

Fig. 3. Assembly Model of Slingshot

3 MontiArc

MontiArc (https://github.com/MontiCore/montiarc) [4] is a textual modeling language to describe component&connector systems. The components receive input messages and sent output messages via typed and directed ports. Only over these explicitly defined ports, communications is possible. This reduces hidden links.

MontiArc has a precisely defined semantic foundation in FOCUS [2]. The textual MontiArc models can be mapped into the mathematical FOCUS space to understand the semantics of the models [5]. This enables formal verification of MontiArc models at design time. The formal proofs are performed using MontiBelle [11,12] and the interactive theorem prover Isabelle [14]. In [10] this approach is demonstrated in cooperation with Airbus to verify properties over an uplink feed of an avionic system.

To leverage of MontiArc's simulation and verification capabilities we model Slingshot's component repository model as seen in Fig. 2 in MontiArc. Listing 1.1 shows an excerpt of the generated MontiArc model with constraints, restricting the number of users in a usage scenario to a positive number. Slingshot is modeled as the system component which is further decomposed into atomic subcomponents representing simulation components. Each sub-component instantiates their respective component definitions, which in turn define their ports in accordance to the event-driven data-flow paradigm employed in Slingshot/Palladio. The sub-components are instantiated and communicate through connections of their ports.

```
component Slingshot {
  port <<condition = "x.numberOfUsers > 0
                   && x.numberOfUsers < 100">>
       in UsageScenario usageScenario;
  component UsageSimulation {
    port <<condition = "x.numberOfUsers > 0">>
         in UsageScenario usageScenario;
  }
  UsageSimulation usageSimulation;
  usageScenario -> usageSimulation.usageScenario;
}
```

Listing 1.1. Simplified Excerpt of the MontiArc Model with Constraints

4 Transforming PCM to MontiArc

Our transformation uses the repository and assembly model of the PCM as inputs. The first step of the transformation is the extraction of complex data types from the PCM repository model. Primitive data types like `int` or `char` can be translated directly while complex data types require their own definition in a MontiArc class diagram model. In addition to complex data types defined in the PCM model, method signatures with two or more parameters are also transformed to MontiArc data types, to be able to specify constraints for them. The result of this step is the MontiArc class diagram containing all necessary data types for the description of the PCM system model as a MontiArc model.

In the next step, the PCM system model is transformed. First, the system, which is the root element of an assembly model, is transformed to the core component of the MontiArc model. Afterwards, the assembly contexts, which are instances of components specified in the repository, are transformed to sub components. The MontiArc component a PCM assembly context is transformed to have MontiArc ports according to the PCM roles of the PCM component this PCM assembly context encapsulates. The last part of this step is the transformation of PCM connectors, which connect PCM assembly contexts according to the roles from the encapsulated component, to MontiArc connectors. As last step, the generated MontiArc model is enriched with the constraints specified as Ecore annotations in the additional input model. Each annotation refers to a directed connector in the PCM assembly model and has two key value pairs, one describing the guarantee of the component the connector comes from and the other one describing the assumption of the component the connector goes to. These constraints are then mapped onto the corresponding MontiArc ports.

5 Constraint Checking

We present a solution to automatically provide consistency analysis to the source PCM models by checking the constraints produced during the transformation to

```
(declare-sort UsageScenario)
(declare-fun numberOfUsers (UsageScenario) (Int))
(declare-fun x () UsageScenario)
(assert (=> (and (< (numberOfUsers x) 100)
                 (> (numberOfUsers x) 0))
            (> (numberOfUsers x) 0)))
```

Listing 1.2. Translated Constraints in SMT-Lib

MontiArc. Constraints are embedded into stereotypes over ports of C&C architectural components. These employ a reduced form of Assumption/Guarantee formalism [2] implemented through an assertion paradigm similar to the programming language Eiffel [13]. The restriction in question is that the constraints are not expressed through formulas handling potentially infinite port flow histories but only instant values present on ports. This is first done by parsing the generated model and processing all constraint pairs defined with the *condition* stereotype. Processing constraints starts by parsing the constraints into the MontiCore expression framework [8] and performing a translation from these to SMT-Lib [1] formulas. Our implementation reuses an OCL to SMT translator restricted on the set of MontiCore expressions. The variant of the OCL accepted by the translator is described in [16]. Our restriction of the OCL language is introduced to facilitate a simple and decidable set of constraints that can be reliably solved by conventional SMT solvers, in addition to only handling quantifier-free constraints. Furthermore, complex objects transmitted on ports also require their datatypes to be represented and translated to SMT. Our approach uses implementation focused class diagrams and objects diagrams as presented in [16]. Microsoft's Z3 SMT solver [3] checks the SMT-Lib formulas. To formulate checkable constraints we build implications over the assumptions and the guarantee obligations. As port datatypes are instances of classes we introduce a SMT sort for each class. Listing 1.2 introduces the declaration of the *UsageScenario* sort and corresponding function *numberOfUsers* introduced for its attribute its attribute, all in accordance to the class diagram generated for the PCM model in Fig. 2. The return type of the function is represented by the SMT-Lib built-in `Int` sort.

The translation of the constraints takes place by introducing a variable x and and declaring it in the respective sort of the port's type, i.e., the sort generated for the class, here *UsageScenario*. The implications and logical operators used in constraints are then 1-to-1 mapped to SMT-Lib. The result of the checker is then constructed with a classic approach, i.e., to check validity we negate the implication and check for unsatisfiability. The result is then produced to the standard output with a statement about the validity of the implication and the the attributed ports. Listing 1.3 presents an output of the constraint checker for the valid Slingshot model, referencing the connected ports in question. If the negated implication is satisfiable, then the implication is not valid. Listing 1.4 presents the output of the constraint checker if we modify the target constraint

```
Constraint of Port usageScenario in Component Slingshot
    guarantees constraint of Port usageScenario in
    Component UsageSimulation
```

Listing 1.3. Output of the Constraint Checker for the Constraint-Enriched Model

```
[ERROR] Found error in port-constraint. Constraint of Port
     usageScenario in Component UsageSimulation does not
    follow from constraint of usageScenario in Component
    Slingshot.
Counterexample: objectdiagram x {
  usageScenario_0:UsageScenario {
    int numberOfUsers=1;
  };
}
```

Listing 1.4. Output of the Constraint Checker for a Negative Result in the Model

in listing 1.1 to require a positive number of users. The tool errors but not before processing all implications. The tool presents a counterexample in the form of an object diagram, performing a retranslation of the SMT-Lib model to the object-oriented representation.

6 Conclusion

We presented a tool for the automatic translation of software architecture models from the Palladio approach and the PCM to the architecture description language of MontiArc to support the validation of semantic constraints specified as annotations of model elements from the PCM. We believe this tool benefits the software architecture community by bridging the gap between both the PCM and MontiArc modeling and analysis approaches to support thorough system analysis without the need for repeated modeling of the same system in different formalisms. While the basic concepts shown in this paper can already be transformed from PCM to MontiArc, our tool currently does not support the full expressiveness of concepts that could be transformed. Furthermore, our tool does not support a retranslation of the result from MontiArc into the PCM. We currently only support a single method per interface, because we specify constraints only referencing connectors corresponding to an interface and not a method. In addition to supporting this, we plan to expand our tool to not only allow for the transformation and validation of PCM models but also the PCM metamodel and the simulator presented as an example in this paper. This allows for the validation of composition on the three different levels of model instance, metamodel, and analysis based on these models.

Acknowledgments. This work was funded by the DFG (German Research Foundation) – project number 499241390 (FeCoMASS), supported by the Collaborative Research Center "Convide" - SFB 1608 - 501798263 and supported by funding from the topic Engineering Secure Systems, KASTEL Security Research Labs funded by the Helmholtz Association (HGF).

References

1. Barrett, C., Stump, A., Tinelli, C., et al.: The SMT-lib standard: version 2.0. In: Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK), vol. 13, p. 14 (2010)
2. Broy, M., Stølen, K.: Specification and Development of Interactive Systems. Focus on Streams, Interfaces and Refinement. Springer, Heidelberg (2001). https://doi.org/10.1007/978-1-4613-0091-5
3. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
4. Haber, A.: MontiArc - Architectural Modeling and Simulation of Interactive Distributed Systems. Aachener Informatik-Berichte, Software Engineering, Band 24, Shaker Verlag (2016)
5. Harel, D., Rumpe, B.: Meaningful modeling: what's the semantics of "semantics"? IEEE Comput. J. **37**(10), 64–72 (2004)
6. Heinrich, R., Strittmatter, M., Reussner, R.H.: A layered reference architecture for metamodels to tailor quality modeling and analysis. IEEE Trans. Software Eng. **47**(4), 775–800 (2021)
7. Heinrich, R., et al.: The palladio-bench for modeling and simulating software architectures. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, pp. 37–40 (2018)
8. Hölldobler, K., Kautz, O., Rumpe, B.: MontiCore Language Workbench and Library Handbook: Edition 2021. Aachener Informatik-Berichte, Software Engineering, Band 48, Shaker Verlag (2021)
9. Katić, J., Klinaku, F., Becker, S.: The slingshot simulator: an extensible event-driven PCM simulator (poster) (2021)
10. Kausch, H., Pfeiffer, M., Raco, D., Rath, A., Rumpe, B., Schweiger, A.: A theory for event-driven specifications using focus and MontiArc on the example of a data link uplink feed system. In: Groher, I., Vogel, T. (eds.) Software Engineering 2023 Workshops, pp. 169–188. Gesellschaft für Informatik e.V. (2023)
11. Kausch, H., Pfeiffer, M., Raco, D., Rumpe, B.: An approach for logic-based knowledge representation and automated reasoning over underspecification and refinement in safety-critical cyber-physical systems. In: Hebig, R., Heinrich, R. (eds.) Combined Proceedings of the Workshops at Software Engineering 2020, vol. 2581. CEUR Workshop Proceedings (2020)
12. Kausch, H., Pfeiffer, M., Raco, D., Rumpe, B.: MontiBelle - toolbox for a model-based development and verification of distributed critical systems for compliance with functional safety. In: AIAA Scitech 2020 Forum. American Institute of Aeronautics and Astronautics (2020)
13. Meyer, B.: Lessons from the design of the eiffel libraries. Commun. ACM **33**(9), 68–88 (1990)

14. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Lecture Notes in Artificial Intelligence, vol. 2283. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9
15. Reussner, R.H., Becker, S., Happe, J., Heinrich, R., Koziolek, A.: Modeling and Simulating Software Architectures: The Palladio Approach. MIT Press, Cambridge (2016)
16. Rumpe, B.: Modeling with UML: Language, Concepts, Methods. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33933-7