Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 279 (2) (2011) 33–41

www.elsevier.com/locate/entcs

Performance Certification of Software Components

Erik Burger [1] Ralf Reussner [2]

Software Design and Quality
Institute for Program Structures and Data Organization, Faculty of Informatics
Karlsruhe Institute of Technology, Germany

Abstract

Non-functional properties of software should be specified early in the development process. In a distributed process of software development, this means that quality requirements must be made explicit in the specification, and the developing party of a commissioned component needs to deliver not only the implemented component, but also a description of its non-functional properties. Based on these artefacts, a conformance check guarantees that the implemented component fulfills the performance requirements.
We extend the notion of model refinement to non-functional properties of software and propose a refinement calculus for conformance checking between abstract performance descriptions of components. The calculus is based on a refinement notion that covers the performance-relevant aspects of components. The approach is applied to the Palladio Component Model as a description language for performance properties of components.

Keywords: model refinement, certification, performance modeling

1 Introduction

During the design of component-based systems, it is useful to model non-functional properties of a system, like performance, already in early stages of the development process. Developers often see quality of service as a property of software that is checked and corrected once the product is completed. This "fix-it-later" practice is, however, a reason for quality problems in software development. Just like testing is an integral part of the implementation process that should be integrated from the beginning, performance modelling enables the developer of a system to make design decision based on analyses and simulations.

⋆ This work is granted by the GlobaliSE project, a research contract of the state of Baden-Württemberg which is funded by the Landesstiftung Baden-Württemberg gGmbH.

[1] E-Mail: burger@kit.edu

[2] E-Mail: reussner@kit.edu

doi:10.1016/j.entcs.2011.11.010

Abstract performance models can, however, also be used to express performance requirements in the specification phase of componenent-based software development. As the development procedes, additional performance models are created to describe the properties of the design, and eventually the implemented component. In order to prove that the performance requirements are met in all these stages, a notion of *refinement* for performance is needed. By using performance refinement in the development process, the developer can check at any time if the requirements are still met and which properties may be violated.

Even if the commissioned component is delivered without a performance specification, it can be reconstructed by reverse engineering methods such as static code analysis and analyses of monitored execution traces [8]. However, as such a reconstructed performance description can differ from a manually specified one, the refinement calculus still is needed to show the compliance.

Since the performance of a component is influenced by many factors, the refinement calculus should take this into account by offering several levels of refinement. In this paper, we propose a refinement calculus that is based on component properties like external call sequences and usage of resources. The aspects of this refinement method make use of formal methods like finite automata and the resource demand calculus presented in [4], which make it possible to prove valid performance refinement on an abstract level.

The contribution of this paper is firstly a model for parameterised component performance specifications and secondly a calculus of refinement. The proposed language for component performance specifications is based on the performance prediction model used in the *Palladio Component Model* [1], a metamodel for the description of component-based software architectures. The PCM has been used in several industrial case studies and offers methods for the prediction of quality of service attributes, especially performance and reliability, as well as tool support for modelling and prediction. We use the PCM as a description language for performance properties of components since it offers parametric dependencies between various aspects of a component-based system, like deployment, assembly and usage profile.

This paper is structured as follows: In Section 2, we give a brief introduction into the Palladio Component Model. In Section 3, the scenario for software performance certification and the refinement calculus are presented. The assumptions and limitations of the approach are discussed in Section 4. Related work is mentioned in Section 5 before the paper concludes with Section 6.

2 Foundations

The *Palladio Component Model (PCM)* [1] is a meta-model for the description of component-based software architectures. The model is designed with a special focus on the prediction of Quality-of-Service attributes, especially performance. Service Effect Specifications (SEFF) describe the relationship between provided and required services of a component. In particular, the PCM SEFFs are the first calculus which

takes all influencing contextual factors of component performance into account explicitly.

In the PCM metamodel, they are defined in the form of *Resource Demanding Service Effect Specifications (RDSEFF)*, which are used for performance prediction and contain a probabilistic abstraction of the control flow. RDSEFFs use a notation stemming from UML activity diagrams, i.e. activities are denoted by nodes. For each RDSEFF, a *resource demand* can be specified as well as dependencies of transition probabilities and resource demands on the formal parameters of the service. RDSEFFs can be annotated to each provided service of a component. They describe

- how the service uses hardware/software resources;
- how the service calls the component's required services.

Resource demands in RDSEFFs abstractly specify the consumption of resources by the service's algorithms, e.g., in terms of CPU units needed or bytes read or written to a hard disk. Resource demands as well as calls to required services are included in an abstract control flow specification, which captures call probabilities, sequences, branches, loops and forks.

RDSEFFs abstractly model the externally visible behaviour of a service with resource demands and calls to required services. They present a grey box view of the component, which is necessary for performance predictions, because black box specifications (e.g., interfaces with signatures) do not contain sufficient information. RDSEFFs are not white box component specifications, because they abstract from the service's concrete algorithms and do not expose the component developer's intellectual property. Component developers specify RDSEFFs during or after component development and thus enable performance predictions by third parties.

3 Certification of Software Component Specifications

3.1 Certification Scenario

The proposed refinement calculus can be applied in a scenario of certification described in [4]. This scenario is depicted in Figure 1. In the proposed component-based software development process, a specifications document for components is created and enriched by non-functional requirements concerning the performance of a component (depicted as "Performance Requirements" on the left hand side). These requirements have to be expressed formally in the specifications document, using an abstract performance description language.

The performance requirements serve as a contract which has to be fulfilled by the implementing party. However, performance descriptions can not only be used in the specification of a software system, but also to describe an actual implementation of this system.

Based on the specifications document, the implementation of the component is created, usually by a third party supplier. The resulting component is shipped with a description of its performance properties (depicted as "Performance Description" on the right hand side). This description can be determined by the developer in

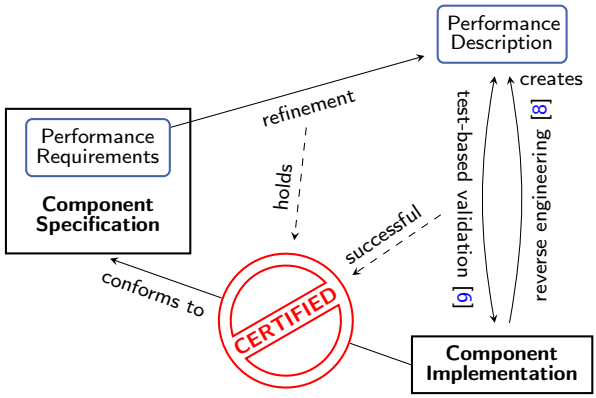


Figure 1. Certification Scenario

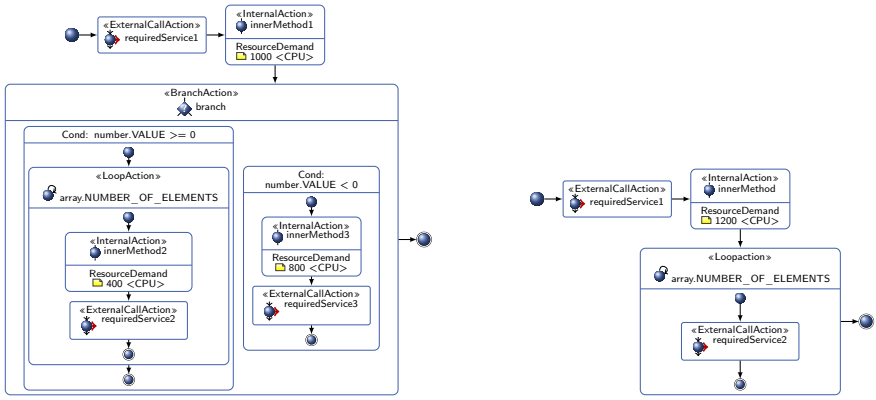
two ways: In the first case, the developer of the component creates the performance description manually. The conformance of this description to the actual implementation is validated by the methods described in [6]. In the second case, the reverse engineering techniques discussed in [8] are used to create the performance descriptions a posteriori from the implemented component. In this case either the component delivering party or the component commissioning party can perform the reverse engineering. Assuming the correctness of these reverse engineering techniques, the resulting performance description can be used for a comparison with the requirements.

The availability of both the performance requirements and the performance description is a necessary precondition for the approach proposed in this work. If both artefacts are present, it is to be determined if the implementation description is a *refinement* of the performance requirements. For this purpose, a formal refinement definition is specified that allows both parties to check the conformance of implementation to specification regarding the performance properties, on the level of the abstract descriptions. With the help of a checking tool, which could be provided by a trusted certification authority, it is then checked if a refinement relation between the two artefacts holds, and if positive, the certificate can be issued. In case this performance description has been created manually, a validation has to be performed, which is indicated by "test-based validation" in Figure 1. If the refinement relation holds and the test-based validation is successful, this means that the implementation complies with the performance requirements.

3.2 Hierarchical Refinement

For the refinement of performance, we propose a refinement calculus, which will be explained in detail in the following. With this calculus, different aspects of refinement are expressed. The conformance of external call protocols is checked first, since this conformance is a necessary preconditition for components to be compared for refinement. Then, resource demands of active resources like CPU, memory and hard disk are considered.

For the definition of the refinement calculus, we use the Palladio Component



(a) Specification RDSEFF R_1 (adapted from [1], p.7)

(b) Implementation RDSEFF R_2

Figure 2. RDSEFF example

Model [1] as a base for our component-based performance models. In the Palladio Component Model, the performance properties of a component are described using the formalism of *Resource Demanding Service Effect Specifications (RDSEFF)*. These specifications contain different types of actions for the modeling of control flow, acquiring/releasing of resources and resource demands for several types of resources, such as CPU, HDD, Network and so on.

As a running example, we will use the RDSEFFs depicted in Figure 2(a) (R_1) as performance specification and the RDSEFF in Figure 2(b) (R_2) as implementation performance description. The specification RDSEFF R_1 includes calls to required services, which are expressed as `ExternalCallAction` elements. Computations within the component are abstracted as `InternalAction` elements. The control flow is only modelled between calls to external services; control flow within external action is abstracted. In the example RDSEFF R_1, there are dependencies on input variables: the `BranchAction` is parameterised by the input variable `number`, while the number of loop iterations in the `LoopAction` depends on the size of the input variable `array`.

In the following sections, we will use this example to illustrate the different parts of the refinement calculus.

3.3 Refinement of External Calls

External Calls describe how a component interacts with other components. They model the calls of a component to required services of another component to which it is connected. Since an actual component instance can be connected to arbitrary components that offer compatible interfaces, no statements can be made about the performance behaviour of these external calls. This is why the compliance of external call sequences is checked first.

We describe the sequence of external calls as a non-deterministic finite-state automaton according to the approach presented in [12] to express the calls to external services including the parameters as transitions in finite automata. We use the

substitutibility notion defined there as the criterion for refinement.

3.4 Refinement of Resource Demands

Apart from passive resources, components can also consume active resources such as CPU, memory, or network. In [4], we presented a rule-based approach for the refinement of performance properties based on resource demands.

Since we have already dealt with external calls in the preceding section, they are not regarded here. For the refinement of resource demands, we do not regard the external calls from the RDSEFFs and only deal with the resource demands of internal actions. Using the refinement calculus from [4], we can match the actions of two different RDSEFFs and check for refinement.

If we take the RDSEFF from Figure 2(a) (R_1) and check for refinement from it to the RDSEFF of Figure 2(b) (R_2), we can see that the resource demand of R_2 is 1200 CPU cycles in `innerMethod`. In R_1, we have a branch action, so we will have to take into account all possible execution sequences to check for refinement. In the (simple) example here, there are two possibilities for CPU resource demands:

(i) $1000 + 400 \cdot \#l$, where $\#l$ is the number of loop iterations

(ii) 1800 in the second branch

Note that we do not take the branch condition or probabilities into account here; the refinement rule states that if the resource demands of R_2 are lower or equal than those of R_1 for all possible traces, then the refinement relation holds. In our case here, this is true if the number of loop iterations is greater than zero, meaning that refinement holds if the input variable `array` is not empty. This illustrates that the refinement relation is dependent on the usage context; in this case, one can easily relate the value of the variable `array` to the refinement relation. In more complex cases, it may not be possible to solve such a dependency analytically (see next section). Thus, it can only be determined whether the refinement relation holds if the usage profile is known. For example, if we know from the usage profile that arrays always have at least size 1, then the refinement relation from R_1 to R_2 holds in this case.

3.5 Completeness of the Approach

With the calculus for performance refinement, all constructs that are available for the description of Resource Demanding Service Effect Specifications in the Palladio Component Model are covered. The first aspect, *External Calls*, covers `ExternalCallAction` elements, but also the control flow elements `BranchAction`, `LoopAction` and `ForkAction`. The second aspect, *Resource Demands*, covers `InternalAction` elements with the annotated `ResourceDemand` descriptions.

If we look at the contexts that a component possesses, the refinement calculus presented is independent from the *assembly context* of a component, meaning that the component on which refinement is applied can be composed arbitrarily with other components without losing the refinement property. Also, since the third

refinement step is only on abstract resource demands like CPU cycles or memory, the approach is also independent from the *deployment context* of the component.

An RDSEFF element that has not been regarded in the description of the refinement calculus is `SetVariableAction`. We neglect variables on purpose, following the paradigm of [12] that parameter values should not be regarded in the description of component interfaces. Furthermore, the elements `AcquireAction` and `ReleaseAction` are not included in the current approach; the handling of passive resources is left to future work.

4 Assumptions/Limitations

4.1 Usage Profile

The refinement approach presented in this paper is currently only valid under the assumption of a fixed usage profile. This means that in every `ResourceDemand` element, the stochastic expressions are computable without dependencies on input parameters. This limits the expressivity of the refinement calculus, since refinement cannot be expressed fully independently from all component contexts. In the refinement scenario presented in Subsection 3.1, this limitation means that the usage profile for which the certificate is to be issued has to be defined before the certification process, and the certificate would than be limited to the specified profiles.

4.2 Formal Semantics of the Palladio Component Model

The formal refinement check is only correct under the assumptions that the refinement rules that are used are also correct. The preservation of resource demands or the fulfillment of performance requirements is not checked directly, but is encoded in the refinement rules: if there is a valid application of rules, then the refinement relation holds. The rules themselves are not formally proven to be correct in this paper. This could be achieved using a formal description of the Palladio Component Model, e.g. using the transformation to Queueing Petri Nets (QPN) in [7, chapter 4.4]. The problem is however that the notion of performance also has to be defined in the formalism that is target of such a transformation. Based on this, a transformation can be used to prove that a "QPN refinement" relation exists, and from this fact, the existence of refinement between the RDSEFFs can be proven.

5 Related Work

For the analyis of performance properties of component-based software, many (academic) component models exist, which are mostly targeted on analysis of existing systems. If a software system is designed from scratch, the process should be supported by a development environment that also offers modeling techniques for creating new systems. As an extension to UML, the UML MARTE profile [11] can be used for the modelling of real-time and embedded systems. From the SPE community, several metamodels of the performance domain are available, most notably

CB-SPE [3] and KLAPER [5].

Abstract performance models of software component can be created in early stages of development as well as for existing software. In order to obtain performance models from black-box components, Krogmann et al. [8] have developed a reverse engineering approach that uses genetic algorithms, static and dynamic analysis, and benchmarking. The approach has been validated for Java-based systems. The reverse engineering approach is part of the certification approach shown in Figure 1. If an existing component is to be certified, the performance description of the implemented component must be created first. Since it cannot be assumed that sources of the software are available for the purpose of certification, the black-box approach is used to gain the performance properties.

Performance modelling and analysis is often based on simulations and testing. Formal approaches are rare and can best be found in the field of probabilistic model checking, for example the PRISM tool [9], which combines conventional correctness checks with stochastic processes to reason about reliability and performance [2], [10]. However, the approach is lacking the possibility to model the systems parametrically with respect to usage profiles and execution environment.

6 Conclusion and Future Work

In this paper, we present a refinement calculus which checks whether an implemented component conforms to an abstract performance specification. Together with reverse-engineering methods and test-based performance validation, this calculus can be used in a certification scenario to provide for a complete chain of conformance relations from abstract specifications to source code with respect to performance properties. Expressing refinement on an abstract level protects intellectual property such as internal implementation details and source code, while still providing a certification statement that is based on formal methods rather than just meeting standards in a development process.

The calculus uses the parametric modelling features of the Palladio Component Model, so that the refinement is independent from the execution environment of the component, which comprises deployment on hardware and assembly with other components. Independence from the usage profile is planned in a future version of the refinement calculus, but not included at the moment due to the unsolved problem of comparing stochastical functions with respect to performance properties. Since parametric modelling of user behaviour is one of the key advantages of Palladio, including it into the refinement calculus should be a main objective of future work. Furthermore, the handling of passive resources is not included in the current approach.

In a formal development process, the conformance of implementation to specification is checked using formally proven methods. The refinement calculus presented in this paper enriches the component-based development process in the direction of formal development. However, for a completely formal definition of refinement, the semantics of the performance abstractions used in this paper have to be defined and

the rules of the refinement calculus have to be proven for correctness. This is future work since the notion of performance refinement is new and there is little related work in this field.

The proposed development process brings together two techniques that are used to ensure the quality of component-based software: performance engineering and software certification. The novelty of this approach is to certify non-functional properties based on formal description languages. Using sophisticated performance descriptions like the RDSEFF formalism of the Palladio Component Model, developers cannot only make performance predictions at early stages of the development process, but also check if the performance requirements are met by the final product. In a distributed component development process, performance certification of components helps the system architect to choose from existing implementations and to guarantee the overall quality of the system.

References

[1] Becker, S., H. Koziolek and R. Reussner, *The Palladio component model for model-driven performance prediction*, Journal of Systems and Software **82** (2009), pp. 3–22.
URL http://dx.doi.org/10.1016/j.jss.2008.03.066

[2] Berczes, T., G. Guta, G. Kusper, W. Schreiner and J. Sztrik, *Analyzing a Proxy Cache Server Performance Model with the Probabilistic Model Checker PRISM*, in: *Proc. 5th International Workshop on Automated Specification and Verification of Web Systems (WWV'09)*, 2009.

[3] Bertolino, A. and R. Mirandola, *CB-SPE Tool: Putting Component-Based Performance Engineering into Practice*, in: I. Crnkovic, J. A. Stafford, H. W. Schmidt and K. Wallnau, editors, *Component-Based Software Engineering, 7th International Symposium, CBSE 2004 Edinburgh, UK, May 24-25, 2004, Proceedings*, 2004, pp. 233–248.

[4] Burger, E., *Towards formal certification of software components*, in: B. Bühnová, R. H. Reussner, C. Szyperski and W. Weck, editors, *Proceedings of the Fifteenth International Workshop on Component-Oriented Programming (WCOP) 2010*, Interne Berichte **2010-14** (2010), pp. 15–22.
URL http://sdqweb.ipd.kit.edu/publications/pdfs/burger2010b.pdf

[5] Grassi, V., R. Mirandola and A. Sabetta, *Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach*, Journal on Systems and Software **80** (2007), pp. 528–558.

[6] Groenda, H., *Certification of software component performance specifications*, in: R. Reussner, C. Szyperski and W. Weck, editors, *Proceedings of the Fourteenth International Workshop on Component-Oriented Programming (WCOP) 2009*, Interner Bericht. Fakultät für Informatik, Universität Karlsruhe **2009-11**, 2009, pp. 13–21.
URL http://digbib.ubka.uni-karlsruhe.de/volltexte/1000012168

[7] Koziolek, H., "Parameter dependencies for reusable performance specifications of software components," Ph.D. thesis, Universität Oldenburg, Uhlhornsweg 49-55, 26129 Oldenburg (2008).

[8] Krogmann, K., M. Kuperberg and R. Reussner, *Using Genetic Search for Reverse Engineering of Parametric Behaviour Models for Performance Prediction*, IEEE Transactions on Software Engineering (2010), accepted for publication, to appear.

[9] Kwiatkowska, M., G. Norman and D. Parker, *Quantitative Analysis With the Probabilistic Model Checker PRISM*, Electronic Notes in Theoretical Computer Science **153** (2006), pp. 5 – 31, proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005).
URL http://www.sciencedirect.com/science/article/B75H1-4K07PMJ-2/2/8e7b43844979890481 8f4a384a1d8def

[10] Kwiatkowska, M., G. Norman and D. Parker, *PRISM: Probabilistic Model Checking for Performance and Reliability Analysis*, ACM SIGMETRICS Performance Evaluation Review **36** (2009), pp. 40–45.

[11] Object Management Group (OMG), *UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP (realtime/05-02-06)* (2006).
URL http://www.omg.org/cgi-bin/doc?realtime/2005-2-6

[12] Wehrheim, H. and R. H. Reussner, *Towards more realistic component protocol modelling with finite state machines*, in: *FACS '06 – International Workshop on Formal Aspects of Component Software.*