Poster Abstract: The OMPCM Simulator for Model-Based Software Performance Prediction

Jörg Henss
Karlsruhe Institute of Technology
Karlsruhe, Germany
henss@kit.edu

Philipp Merkle
Karlsruhe Institute of Technology
Karlsruhe, Germany
merkle@kit.edu

Ralf H. Reussner
Karlsruhe Institute of Technology
Karlsruhe, Germany
reussner@kit.edu

ABSTRACT

Software performance models play an important role in early stage quality evaluations. Performance models in particular allow for comparing architectural alternatives before unfavourable design decisions are made that need to be revised in a costly procedure. The Palladio component model (PCM) is a modelling language for component-based software architectures. Instances of the PCM can already be analysed for their performance using analytical or simulative approaches. It is, however, difficult to obtain accurate performance predictions for network-intensive distributed systems. This is mainly due to the simplistic network model used so far. In this paper, we present the OMPCM simulator, which integrates OMNeT++ network simulation with architecture-level software performance prediction. OMPCM models can be automatically created from PCM models using a chain of model transformations.

Categories and Subject Descriptors

I.6.3 [**Simulation and Modeling**]: Applications

Keywords

Palladio Component Model, Simulation, Component Based Software

1. INTRODUCTION

Component-based software engineering (CBSE) [12] is increasingly used to facilitate large-scale reuse. In the CBSE development process, component software is assembled from components, each specifying what services are provided and what services are required. Assembled components interact not only on a functional level, but also affect each other's quality attributes; a fast component which consumes a slow required service will appear slow to its own consumers because quality attributes propagate from required to provided interfaces [2]. Interfaces should therefore specify not only operation signatures, but also the corresponding quality of service (QoS). Specifically, capturing QoS properties in interfaces enables reasoning on the overall quality of a component-based system.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
OMNeT Workshop 2013, March 05-07
Copyright © 2013 ICST 978-1-936968-76-3
DOI 10.4108/icst.simutools.2013.251704

The Palladio Component Model (PCM) [1] is a modelling language to express QoS-level component specifications. It captures software systems from different view points to account for different roles being involved in the CBSE development process [5]. These views specifically include component behaviour abstractions, the assembly of components and their deployment. Various analytical and simulative solvers are delivered with the PCM to allow for predicting the quality of a system represented by a PCM model. The Palladio approach has been shown to be well applicable [1], especially for performance predictions.

Modelling and simulation of network communication is, however, currently limited with the PCM. This weakens not only the prediction accuracy for network-intensive systems, but also misses the opportunity to simulate different network configurations before implementing them. Extensive network communication arises especially within distributed systems, where components deployed on different hardware nodes work together toward a common goal.

This paper contributes the OMPCM simulator for model-based software performance prediction. It combines the expressiveness of the PCM with detailed network simulation capabilities. OMPCM transforms PCM models to OMNeT++ simulation models, for which we use state-of-the-art model transformation techniques, including QVT-O [7] and XText [13]. The high degree of automation establishes a tight integration with existing Palladio tools.

In Sec. 2, we provide an overview of the Palladio component model. Sec. 3 motivates and describes the SimCore intermediate model used. The OMPCM simulator and the integration process are presented in Sec. 4. Sec. 5 discusses validation results. The paper concludes with a summary and future work in Sec. 7.

2. PALLADIO COMPONENT MODEL

The Palladio component model (PCM) ([1, 10]) is a modelling language for component-based software architectures and a central part of the Palladio approach. Given the abstract description of a software system as a model, quality predictions can be automatically performed for software performance and reliability. This way, architectural alternatives can be evaluated from early development stages on; no actual implementation is needed. A number of analytical solvers and a discrete-event simulator named SimuCom are provided for quality predictions. SimuCom, however, simulates network communication on a high level of abstraction, which does not allow for comparing different network topologies, for example.

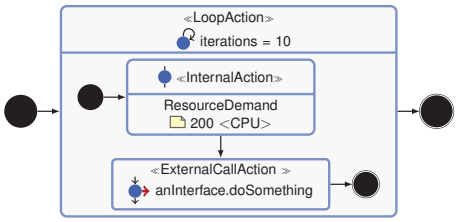


Figure 1: Example of an RD-SEFF

A PCM model consists of five partial models, each of which is created and maintained by a specific developer role. Component developers specify and implement components. Software architects assemble the system from existing components or request additional ones from component developers- System deployers allocate components to their execution environment; prior to the deployment, they specify the execution environment in terms of so-called resource containers and interconnections between them. Domain experts describe the typical system usage by providing usage scenarios.

Component specifications intended to be used in performance predictions carry for each provided service a so-called resource-demanding service effect specification (RD-SEFF). An RD-SEFF captures performance-relevant component behaviour by an action chain. The actions in an RD-SEFF mainly include calls to required services, the resource demands issued in between, and various control flow constructs. Resource demands can be issued to processing resources (e.g. processors or storage devices) or passive resources (e.g. a semaphore). Control flow constructs include branches and loops to express conditional or repeated behaviour. Parallel control flows can be described using forks and joins. The graphical syntax of an RD-SEFF is closely aligned with UML activity diagrams as illustrated in Fig. 1.

Creating a PCM model requires the modellers to estimate model parameters like the interarrival time of users, which are often not known precisely or underly a probability distribution. To account for randomness and the modeller's uncertainty, most values in a PCM model can be specified by so-called stochastic expressions (StoEx). Beyond probability functions, a StoEx may contain constants, variables and arithmetic operations (see [10] for details). If a StoEx contains one or more variables, its value depends on the context in which the expression is evaluated.

3. SIMCORE MODEL

While the PCM relies on RD-SEFFs for modelling component behaviour, we aimed OMCPM to be conceptually independent from the PCM and adaptable to other performance models that also employ a control-flow oriented modelling style such as UML activity or BPMN models. Therefore we developed a new, more simplistic intermediate model, called SimCore, for the description of simulated control flow.

SimCore was designed to comprise a small number of basic modelling elements. Each element should be easy to understand and have a clear simulation semantics. This is a difference to the PCM, where the execution semantics of RD-SEFF elements is often not evident before consulting the specification. Especially the scope of stochastic variables, i.e. how they are evaluated, is not precisely specified, but rather encoded implicitly in the simulator implementation. Furthermore, many RD-SEFF modelling elements overlap in

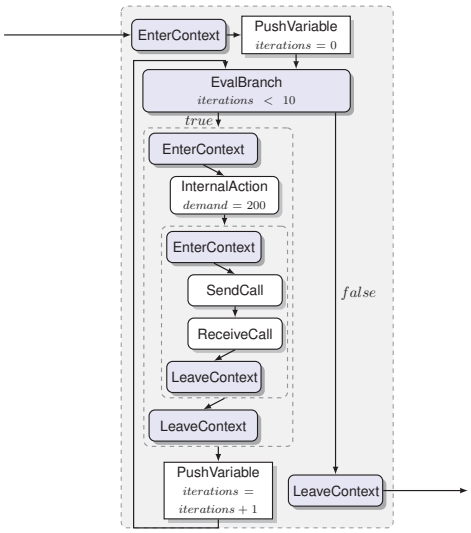


Figure 2: SimCore model of the RD-SEFF shown Fig. 1

their behaviour, leading to duplicated code in corresponding model transformations or interpreters. There are, for instance, multiple types of loop and branch actions, all of which share a common core behaviour.

To avoid these redundancies, SimCore breaks up RD-SEFF actions into basic SimCore elements in such a way that each RD-SEFF action is described by one or more SimCore primitives. Thus, SimCore actions can be combined to resemble a certain RD-SEFF action. This makes specification of semantics more easy and precise and prevents the repeated implementation of shared simulation behaviour.

This approach is similar to the principles of reduced instruction set architectures (RISC) [11]. By using a reduced set of less specialised elements, the development of the simulation is simplified. When compared to RD-SEFFs, a SimCore model, however, requires more elements to express the same behaviour. As an example, Fig. 2 shows the corresponding SimCore model for the RD-SEFF from Fig. 1. Yet we do not consider the growth in model size a problem, because the SimCore model is an internal intermediate model, which is not meant to be exposed to the system modeller. An overview on the SimCore elements can be found on the OMPCM website [8].

4. OMPCM SIMULATOR

A simulation study with OMPCM involves several steps as shown in Fig. 3. Initially, the various developer roles jointly create a PCM model of the software to be analysed. Besides performance-relevant behaviour of components, their assembly and deployment on servers, the PCM model also includes typical usage scenarios.

In the next step, a QVT-O model-to-model (M2M) transformation automatically transforms the PCM model to a corresponding network definition (NED) model. As target for the QVT-O transformation, we use a metamodel that was derived from an adapted version of the NED grammar using the Xtext framework. Internally, the M2M-transformation first creates a SimCore representation for each RD-SEFF. Informations on the architecture, deployment, workload and

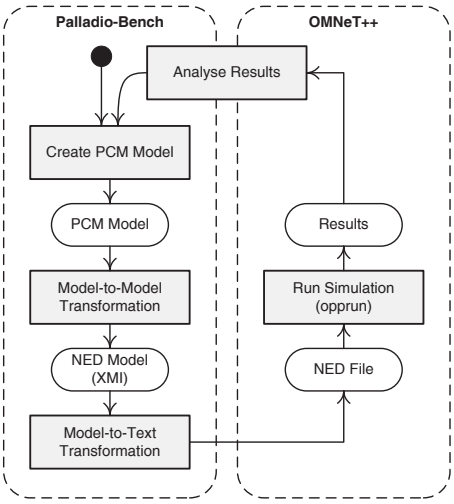


Figure 3: Activities and flow of artefacts in OMPCM

execution environment are then combined to obtain an integrated representation aligned with OMNeT++'s NED file format.

Subsequently, a model-to-text transformation generates from the XMI-based NED model a textual NED file, understood by OMNeT++. Both transformation steps introduced so far are seamlessly integrated in the Palladio-Bench.

The resulting NED file can then be used to run an OMNeT++ simulation of the system. Once the simulation finishes, the results can be analysed and visualised either in the OMNeT++ IDE or can be imported back into the Palladio-Bench for comparison with results from other Palladio model solvers.

The simulation code for OMPCM was implemented in a modular fashion as shown in Fig. 4. Where possible, we tried to reuse existing OMNeT++ concepts and implementations and to extend them with the required functionality.

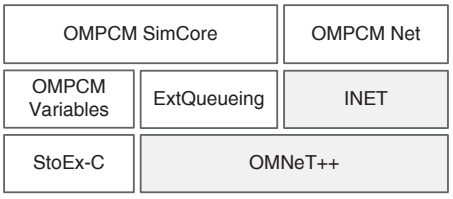


Figure 4: Technology stack of OMPCM

The *StoEx-C* module implements the stochastic expression (StoEx) language introduced in Sec. 2. NED expressions, though similar to a StoEx, cannot be used for representing stochastic expressions due to limited expressiveness. The *OMPCM Variables* module contains methods for evaluating context-dependent StoEx expressions. The *ExtQueueing* module is an extension to the queuing network model shipped with OMNeT++. It provides an implementation of parametric queues, used in OMPCM to describe hardware resources. Currently, first-come, first-served (FCFS) and processor sharing (PS) are supported as scheduling policies for processing resources. The *OMPCM SimCore* module contains the implementation for the control flow elements and provides means to manage the variable definitions valid throughout a simulated request. The context of a request is encapsulated in a message that is passed along the control flow path. The position of the message represents a request's current instruction. Besides the SimCore elements, two workload drivers were implemented to generate load on a simulated system. The *OMPCM Net* module bridges between the INET framework and the OMPCM SimCore module. It provides component proxies, which act as surrogates for remotely deployed components. Calling a component service on a surrogate causes the proxy to use a network connection to the remote component. Hence, the response time of a service may also be influenced by network induced delays.

5. VALIDATION

While validation in simulation studies usually determines if the simulation accurately represents the modelled system [6], we assessed the credibility of OMPCM by comparing it to Palladio's reference simulator SimuCom. SimuCom has been shown to yield accurate results in a number of case studies (e.g., [1]). Repeating these case studies with OMPCM would be tedious while at the same time delivering few additional insights. As network simulation capabilities of SimuCom are fairly limited we excluded network influences and leave their validation for future work.

We consider OMPCM valid if it produces results consistent with SimuCom's simulation results when both are run with the same model. Consistent means that we do not call for identical results but accept differences induced by different implementations of the pseudo-random number generators. Likewise, different strategies to manage events in the future event list (FEL) may cause differences. Given these results, we compare end-to-end response times and utilisation of processing resources.

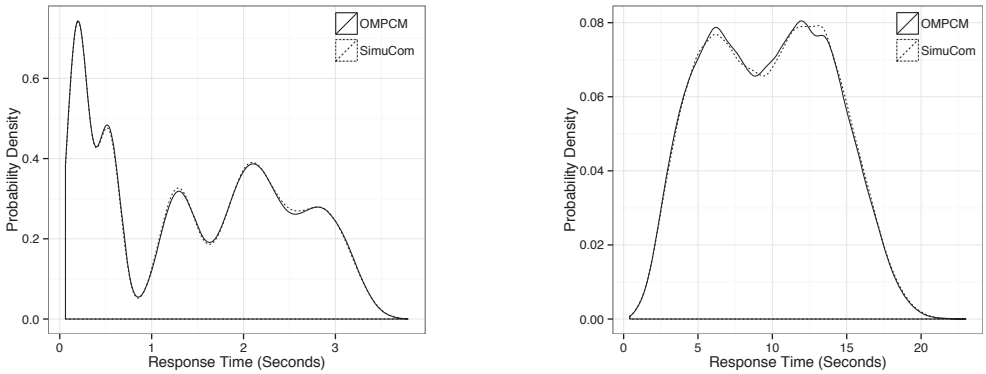
As input to the simulators under comparison we used the MediaStore case study described in [1]. We simulated 150,000 seconds of operation both with OMPCM and SimuCom and used a closed workload with a population of one (Fig. 5a) and ten (Fig. 5b), respectively.

From Figure 5, it can be seen that both simulators yield consistent predictions of the end-to-end response time. Taking into account the probabilistic nature of both simulators, we suppose that both simulators would yield identical results when using same FEL implementation and aligning the random numbers. Deviations in the predicted utilisation (not depicted) are likewise negligible. For the MediaStore model, OMPCM predicts an average CPU utilisation of 33.59% while SimuCom reports 33.54% utilisation on average for the application server. The validation results indicate that OMPCM yields accurate performance predictions.

6. RELATED WORK

A comprehensive comparison of approaches for performance prediction of component-based software systems can be found in [4]. In the following, we present two approaches where OMNeT++ was used for simulation of system behaviour.

Hennig et al. [3] propose a performance simulation approach for JEE-based applications. The simulated applications are modelled with a UML CASE tool, where sequence diagrams are used to describe the behaviour. The UML model is later on converted and linked to a predefined OMNeT++ model of a JBoss application server. The presented



(a) Closed workload population = 1

(b) Closed workload population = 10

Figure 5: Comparison of predicted response times

approach is limited to JEE software and no detailed information on the simulation of resource demands is provided.

Pustina et al. [9] describe a method for the performance evaluation of communication systems. Their approach also uses an annotated UML model which is converted into a queuing network model. The performance model is then evaluated using an implementation of queueing networks based on OMNeT++.

7. CONCLUSIONS

In this paper, we presented a novel simulator for the Palladio component model that brings together architecture-centric software performance prediction with network-centric performance prediction. Our approach of using model-driven techniques eased the development of OMPCM and allowed us to enable a seamless integration with the existing Palladio tooling. Especially the usage of Xtext helped us to extract a metamodel for the NED language, which made it ready for model-driven development. This approach is applicable to other domain models as well and can speed up conversion of existing models to OMNeT++.

The validation indicates that the OMPCM simulator, including SimCore, yields correct results in comparison to the Palladio reference simulator. The mapping of component behaviour to the SimCore model is suited for the proper description of control flows. An evaluation showed that OMPCM is also on a par with the reference simulator concerning the execution speed of simulation. When adding network support to the simulation, speed is considerable decreased by 90% since the number of simulated events skyrockets. Nevertheless, the fine-grained network model allows us to perform more detailed research on network influences and the mutual impact of systems and network.

The OMPCM simulator is publicly available for download. This makes not only available OMNeT++'s rich network simulation capabilities in architectural analyses, but also lowers the initial hurdle to integrate the Palladio approach into network simulations. In future, we plan to further assess OMPCM in a more rigorous case study, where predicted performance metrics will be compared to measurements gathered from a real system.

8. ACKNOWLEDGMENTS

The authors would like to thank Steffen Becker for fruitful discussions on the design of OMPCM.

9. REFERENCES

[1] S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. *J. Syst. Softw.*, 82:3–22, 2009.

[2] D. Hamlet, D. Mason, and D. Woit. *Component-Based Software Development: Case Studies*, chapter Properties of Software Systems Synthesized from Components, pages 129–159. World Scientific Publishing Company, March 2004.

[3] A. Hennig, D. Revill, and M. Pönitsch. From UML to performance measures—simulative performance predictions of IT-systems using the JBoss application server with OMNeT++, 2003.

[4] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634 – 658, 2010.

[5] H. Koziolek and J. Happe. A QoS Driven Development Process Model for Component-Based Software Systems. In *Proc. 9th Int. Symposium on Component-Based Software Engineering*, pages 336–343. Springer-Verlag Berlin Heidelberg, 2006.

[6] A. M. Law and W. D. Kelton. *Simulation modeling and analysis*. McGraw-Hill series in industrial engineering and management science. Boston, 2000.

[7] Object Management Group. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*, Jan. 2011.

[8] OMPCM. http://sdqweb.ipd.kit.edu/wiki/OMPCM.

[9] L. Pustina, S. Schwarzer, M. Gerharz, P. Martini, and V. Deichmann. A practical approach for performance-driven UML modelling of handheld devices - a case study. *J. Syst. Softw.*, 82(1):75–88, 2009.

[10] R. Reussner, S. Becker, E. Burger, J. Happe, M. Hauck, A. Koziolek, H. Koziolek, K. Krogmann, and M. Kuperberg. The Palladio Component Model. Technical report, Karlsruhe, 2011.

[11] A. H. J. Sale. The RISC style of architecture. *Australian Computer Journal*, 21(3):97–99, 1989.

[12] C. Szyperski. *Component software : beyond object-oriented programming*. ACM Press books. Addison-Wesley, Harlow, 1 edition, 1999.

[13] Xtext. http://www.eclipse.org/Xtext/.