Karlsruhe Reports in Informatics 2011,32
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Palladio Days 2011

Proceedings

17–18 November 2011
FZI Forschungszentrum Informatik, Karlsruhe, Germany

Steffen Becker, Jens Happe, Ralf Reussner (Editors)

2011

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Please note:
This Report has been published on the Internet under the following Creative Commons License:
http://creativecommons.org/licenses/by-nc-nd/3.0/de.

Palladio Days 2011

Proceedings

17–18 November 2011
FZI Forschungszentrum Informatik, Karlsruhe, Germany

Steffen Becker, Jens Happe, Ralf Reussner (Editors)

Preface

The prediction of software quality (e.g. performance, reliability and maintainability) based on software architectures is useful in many software development scenarios, such as support for design decisions, resource dimensioning or scalability analysis.

The open source tool Palladio can be seen as an "software architecture simulator". Palladio includes a metamodel for specifying software architectures (Palladio Component Model, PCM), a simulator (SimuCom), and a set of analytical solvers to gather simulation data on different software quality attributes. By its flexible design, extensive documentation, and high number of industrial case studies, Palladio is a mature platform to be utilised by other developers and scientists to explore further possibilities of modelling and simulating architectures. There are several dimensions of building on Palladio: extending Palladio for specific application domains, such as embedded systems, adding analyses for additional quality metrics (such as maintainability) or using Palladio for non-software architectures (e.g., production plants or logistics).

Therefore, the Palladio Days 2011 have the goal to bring together practitioners using Palladio and researchers who intend to work on Palladio as well as those who drive the Palladio project.

The focus topic of this year's Palladio Days is *Performance Prediction of Software on Virtualised Resources.*

Programme Committee Chairs

- Steffen Becker, University of Paderborn
- Jens Happe, SAP Research Karlsruhe
- Ralf Reussner, Karlsruhe Institute of Technology/FZI Forschungszentrum Informatik

Programme Committee

- Samuel Kounev, Karlsruhe Institute of Technology
- Heiko Koziolek, ABB Research, Zurich
- Klaus Krogmann, FZI Forschungszentrum Informatik
- Mircea Trifu, FZI Forschungszentrum Informatik

Organizers

- Steffen Becker, University of Paderborn
- Erik Burger, Karlsruhe Insitute of Technology
- Benjamin Klatt, FZI Forschungszentrum Informatik
- Ralf Reussner, Karlsruhe Institute of Technology/FZI Forschungszentrum Informatik

Programme

Thursday 17 November 2011

9:00 PCM Coding Session
incl. presentation of the new issue tracking and code management
12:00 Lunch Break
13:00 Opening and Welcome (Organizers)
13:15 **Keynote**
Raffaela Mirandola, Politechnico di Milano
"A journey across three decades of software performance engineering approaches"
14:00 Break
14:15 **Palladio Days Paper Session**
Sebastian Lehrig and Thomas Zolynski
Performance Prototyping with ProtoCom in a Virtualised Environment: A Case Study
Philipp Merkle and Jörg Henß
EVENTSIM – An Event-driven Palladio Software Architecture Simulator
15:45 Break
16:00 **PCM: Active projects and visions (PDAYS PC)**
Ralf Reussner and Jörg Henß
Palladio in the Model-Driven Simulation Lifecycle
Samuel Kounev, Fabian Brosig, Nikolaus Huber
From offline to online component models for run-time resource management
17:00 Break
17:15 Round Table Talk: "Future development of the PCM, Identification of Cooperation opportunities"
19:00 Dinner and Socializing at *Hoepfner Burghof*

Friday 18 November 2011

9:00 **Invited reports on recent trends and developments in PCM**
André van Hoorn
Utilizing PCM for Online Capacity Management of Component-Based Software Systems
Michael Hauck and Benjamin Klatt
Non-Invasive Palladio Development
10:00 Break
10:30 PCM Organisation and Publicity
11:30 Closing Discussion: Summary, Next Steps, Next Palladio Days, etc.
12:00 Lunch
13:00 Closing and Farewell

Contents

Performance Prototyping with ProtoCom in a Virtualised Environment: A Case Study

Sebastian Lehrig, Thomas Zolynski

***Abstract*—Performance prototyping is an often used technique to assess the performance of software architectures early in the development process without relying on models of the system under study. ProtoCom is a prototype generator for the PCM realised as model-2-text transformation for which no experience report in a larger, virtualised setting exists. In this paper, we report on four case studies performed with an improved version of ProtoCom and report on the results gained with respect to analysis accuracy and usability. Our results demonstrate that the new version is much easier to use than previous versions and that results gained in our virtualised execution environment help in early assessments of performance under realistic conditions.**

***Index Terms*—D.2.11 Software architectures; D.2.10.h Quality analysis and evaluation; D.2.2 Design tools and techniques.**

I. Introduction

Recent research is directed towards early predictions of software quality attributes like performance already at design time [1], [2]. In case of performance predictions we create architectural models showing the system's structure, behaviour, and allocation. These models are subsequently transformed into analytic or simulation based performance models. The predictions gained from these models provide insights on the expected performance of the system under study and thus avoids cost-intensive redesigns of the system.

While the use of performance prediction models is state of the art, these models still underly several assumptions. Some of these assumptions hold in realistic cases others do not. An approach to deal with the latter is to implement performance *prototypes*, i.e., small test programs which developers can deploy on realistic hardware environments and which simulate resource demands on processing resources, e.g., CPUs, Discs, passive resources, e.g., memory, or communication links, e.g., passing of messages. However, the benefit gained by such performance prototypes in contrast to pure model-based approaches lacks experience reports, especially in recent execution environments like virtualised servers. Research in these virtualised environments is of particular relevance due to the increasing interest in cloud-computing.

Several approaches for performance prototyping have been presented in the past (e.g., [3]–[5]). Among these approaches, also a model-driven approach named ProtoCom has been presented for the Palladio Component Model [6]. ProtoCom transforms PCM components into components implemented in an industrial component model like EJB. For behaviours specified in RDSEFFs ProtoCom generates code to create artificial resource demands. Deployers then allocate these components on their middleware servers. Earlier work on ProtoCom reported its technical background, a more detailed case study involving more complex systems with multiple resource types in advanced environments like virtual servers is still lacking.

Because of this, this paper gives a detailed experience report on case studies performed with an improved version of ProtoCom. We present our implemented improvements taking ProtoCom from the proof-of-concept phase to a readily useable tool. We focus on new concepts and usability requirements which became necessary for the execution of larger case studies.

We validated the new version of ProtoCom on a set of four standard PCM case studies in the course of preparing measurements for a larger research project. The study has been performed on virtualised Blade servers connected by a virtualised network. The results we gained during the process show good applicability of the new version of ProtoCom as well as a good accuracy of the results while still capturing performance relevant factors of the prototype's execution infrastructure.

The contribution of this paper is an extended report on experiences gained while doing four case studies with an improved version of the ProtoCom performance prototype generator. We present measurements taken on a virtualised Blade server environment using the improved ProtoCom generator. We discuss the results and highlight new application areas for ProtoCom and future research directions for performance prototyping.

This paper is structured as follows. Section II briefly revises ProtoCom and its underlying model-2-text transformation and highlights extensions of the improved version. Section III explains our case studies, their virtualised measurement environment, and the gained results. We discuss these results and their impact on further research directions in Section IV. After relating our work to other Palladio Component Model literature and non-PCM works in Section V, we conclude our paper.

II. Foundation

In this section we briefly describe the mapping between PCM models and the corresponding prototypes generated by ProtoCom. We will focus on improvements done to the previous version of ProtoCom as described by Becker [7].

S. Lehrig is student at University of Paderborn, Zukunftsmeile 1, 33102 Paderborn, Germany. E-mail: lehrig@mail.upb.de

T. Zolynski is student at University of Paderborn, Zukunftsmeile 1, 33102 Paderborn, Germany. E-mail: zolynski@mail.upb.de

A. Overview

ProtoCom is a prototype generator for the PCM. It is realised as model-2-text transformation using the template language Xpand. The previous version of ProtoCom mapped the PCM models to the EJB component model. The usage of EJB led to several usability issues, e.g., need for manual adjustments in the generated source code and an inefficient deployment process. To overcome these issues, we focused on better usability for the improved version. One design decision to achieve this objective was to built directly on Java's remote procedure call mechanism called RMI instead of using JavaEE/EJB. This removes dependencies on application servers like GlassFish and allows to deploy the generated prototypes on JavaSE hosts. Since JavaEE and other middleware platforms also use RMI internally, this modification does not alter the network behaviour significantly. In addition, ProtoCom does not rely on any other feature provided by JavaEE and hence its usage can easily be replaced.

For the transformation from PCM models to ProtoCom prototypes the mapping of four aspects has to be considered: static structure, dynamics of the system, component allocation, and system usage [6]. Due to the change from EJB component model to POJO with RMI, some of the existing mappings had to be modified.

The *static structure* mapping defines how PCM components and their required and provided interfaces are realised as Java classes. Most of the class-based component model concept has been retained unchanged: interfaces are mapped to Java interfaces, Basic Components to classes with simulated RDSEFFs, and Composite Components to facade classes. However, Composite Components now instantiate their subcomponents automatically. Also the establishing of component connections - the Assembly Connectors - now uses different techniques depending on the component type. For Composite Components their inner components are now created and connected directly by a facade class [8] on instantiation, since they are always deployed on the same hardware unit according to the semantics of the PCM. Composed Structures whose inner components can be allocated on different hardware units, e.g., System, use the RMI registry to build up their Assembly Connectors.

Fig. 1 shows how the initialisation, assembly, and communication between components is realised with RMI. When started, a basic component registers itself with the GUID at the RMI registry. If a Composed Structure has inner components on different hardware units, a lookup is necessary when initialising inner components. In this case the component is retrieved using the RMI registry instead of being directly initialised by the Composed Structure. After the assembly of inner components is completed, the Composed Structure registers itself at the RMI registry. Afterwards, Usage Scenarios use the same RMI registry for accessing systems to perform entry level system calls.

The *dynamic behaviour* of a component is defined by RDSEFFs. These are mapped to Java code emulating resource demands. For active resources, e.g., CPU and Discs, the abstract, hardware independent resource demands are translated into hardware dependent ones. To ensure that the execution time of resource demands is consistent on different hardware units, these have to be calibrated using load generating algorithms [6]. In the case of passive resources, RDSEFFs are mapped to semaphores from the `java.util.concurrent` package.

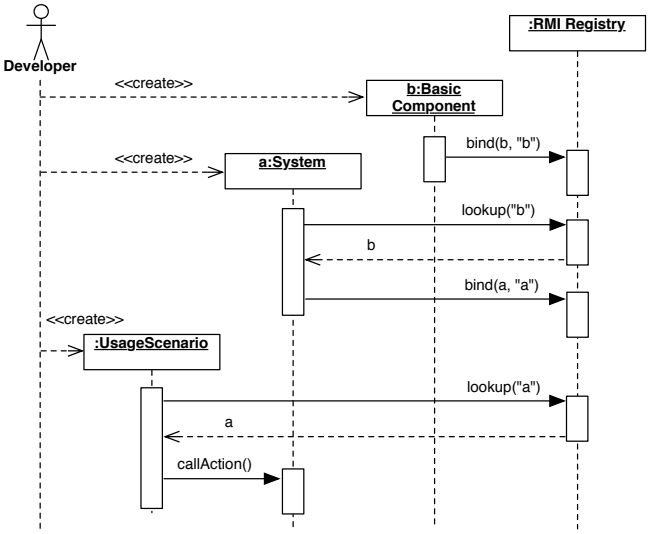

Fig. 1. Sequence Diagram for initialisation and assembly using RMI

The *Allocation* stores the association between components and resource containers. There is no mapping between resource containers in the PCM and any element in ProtoCom, since the prototypes are executed on real hardware units. Nevertheless, we use the Allocation Context to determine a map of hardware nodes which is linked to a list of all components allocated on it. This allows us to start all components of one hardware node in one step.

The final mapping is the *system usage*. PCM Usage Models are used to generate Workload Driver. These simulate the users' behaviour as specified in the model. The implementation uses Java threads to simulate call actions performed on the system.

B. ProtoCom Extensions

Recently ProtoCom has been improved in both, its usability and the adaption of features previously only available in SimuCom.

One major improvement in usability is the possibility to use generated prototypes right 'out of the box', i.e., the developers do not have to manually adjust the generated code as in the previous version. The generated prototypes can directly be deployed on any target hardware unit hosting JavaSE. Performance tests are started by either writing a configuration file or by using a newly included menu. The necessary calibration for active resources is automatically done before the first usage scenario is started.

Further extensions to ProtoCom include the support of additional resource types. Passive resources are implemented as

semaphores from the `java.util.concurrent` package. The delay resource demands utilise the sleep mechanism of Java threads.

III. Measurement Results on Blade Server

This section describes the measurement on the virtual environment running on Blade servers. Section III-A describes the case studies under consideration. Section III-B continues with the description of the environment and the process for taking measurements. We discuss issues about the CPU and HDD calibration in Section III-C. Finally, Section III-D briefly presents the results of the measurements.

A. Case Studies

We applied ProtoCom in four case studies, each modelling distributed, component-based systems. Fig. III-A shows the models of those case studies [9]. They describe different domains ranging from enterprise to industrial process control systems. In particular, they differ in properties like their resource usage, kind of workload (open or closed), their allocations to hardware devices, and their assembled system's complexity.

In order to reason about the performance of each modelled system, the following list gives a brief description for each model with respect to properties that may have an influence on its performance.

- **Media Store** models a part of an online media store that allows to up- and download media like music files [10]. The system stores the files within a database and watermarks them on download to provide a copy protection. Therefore, the resource container `Application Server` provides the platform for presentation and application layer, and the resource container `MySQL Database Server` for the data layer. The `Digital Watermarking` component has the main impact on CPU usage (when watermarking), and the `Media Database` on the HDD usage (when files are written or read from it). The `Database Cache` caches files to provide a faster access to frequently requested files. The usage scenario for Media Store is a closed workload with one user. The user uploads files with a probability of 20% and downloads files with a probability of80%.
- **SPECjEnterprise2010** models a standardised J2EE benchmark for measuring the scalability and performance of J2EE servers and containers [11]. It models the system of an automobile manufacturer. The customers of the manufacturer, typically automobile dealers, use a web interface to access the manufacturer's product catalogue, purchase cars, etc. The system consists of two resource containers: `WLS` (Web Logic Server) for the presentation and application layer, and `Oracle Database Server` for the data layer. The model does not consider HDD demands but the application layer has high CPU demands, e.g., for scheduling a session. The usage scenario describes an open workload where users arrive with an exponentially distributed inter-arrival time with a mean of $1/45$ seconds. Hereby, each user registers an order to the system that needs to be scheduled.
- **Process Control System** describes an industrial distributed control and automation system [2]. The respective component names and semantics are obscured to protect confidential information. The model consists of the three resource containers `Server 1` to `3` which represent the obfuscated server-side part of the system. The allocated components may communicate over communication paths if they are deployed on different devices. Furthermore, the model uses composite components in order to describe a more complex system. Most of the components have CPU demands. Additionally, the composite component `C12` has HDD demands. The model features one closed and three open workloads as usage scenario. The closed workload models an alarm event that sometimes occurs within the system. The other workloads model reaction to alarm events, received sensor data, and calculations within the system.
- **Business Reporting System** is a model of a management information system [12]. The system is capable of managing users that want to receive live data or statistical analysis. It consists of four devices: `Server 1` enables users to access the system by a Tomcat web server, `Server 2` and `4` realise the application layer, and `Server 3` the data layer. Besides the network demands, the model only specifies CPU demands. `Server 2` has a CPU with three cores whereas all other servers (and models) have only one core per CPU. The usage scenario for the Business Reporting System specifies one open workload in which a user executes several actions within the system and finally commits a maintenance request.

B. Setting

For every model, we used a PCM-to-ProtoCom transformation and exported the complete source code (without further changes) including its external libraries to a JAR file. We used the Eclipse export feature for creating runnable JAR files. Afterwards, we uploaded the JAR files onto our virtualised environment via SCP.

The virtualised environment utilised runs on a HP ProLiant BL460C G6 Blade server with the following specification: Intel Xeon X5650 2.67GHz (2 physical processors each with 6 physical cores, i.e., $2 \cdot 2 \cdot 6 = 24$ logical processors in the OS), 96GB RAM, ESX4.1 operating system. The virtual environment consists of four virtual machines, each having the following specification: 3 vCPUs (mounted as real CPUs, not as kernels, i.e., hyper-threading disabled), 8192MB RAM, two 12GB HDDs, openSUSE 11.3 (x86_64), kernel version 2.6.34.7-0.5-desktop, /tmp directory mounted to a SAN (Storage Area Network; HP LeftHand P4000 with MDL series drive[1]).

[1] http://h18006.www1.hp.com/products/quickspecs/13254_div/↩13254_div.pdf

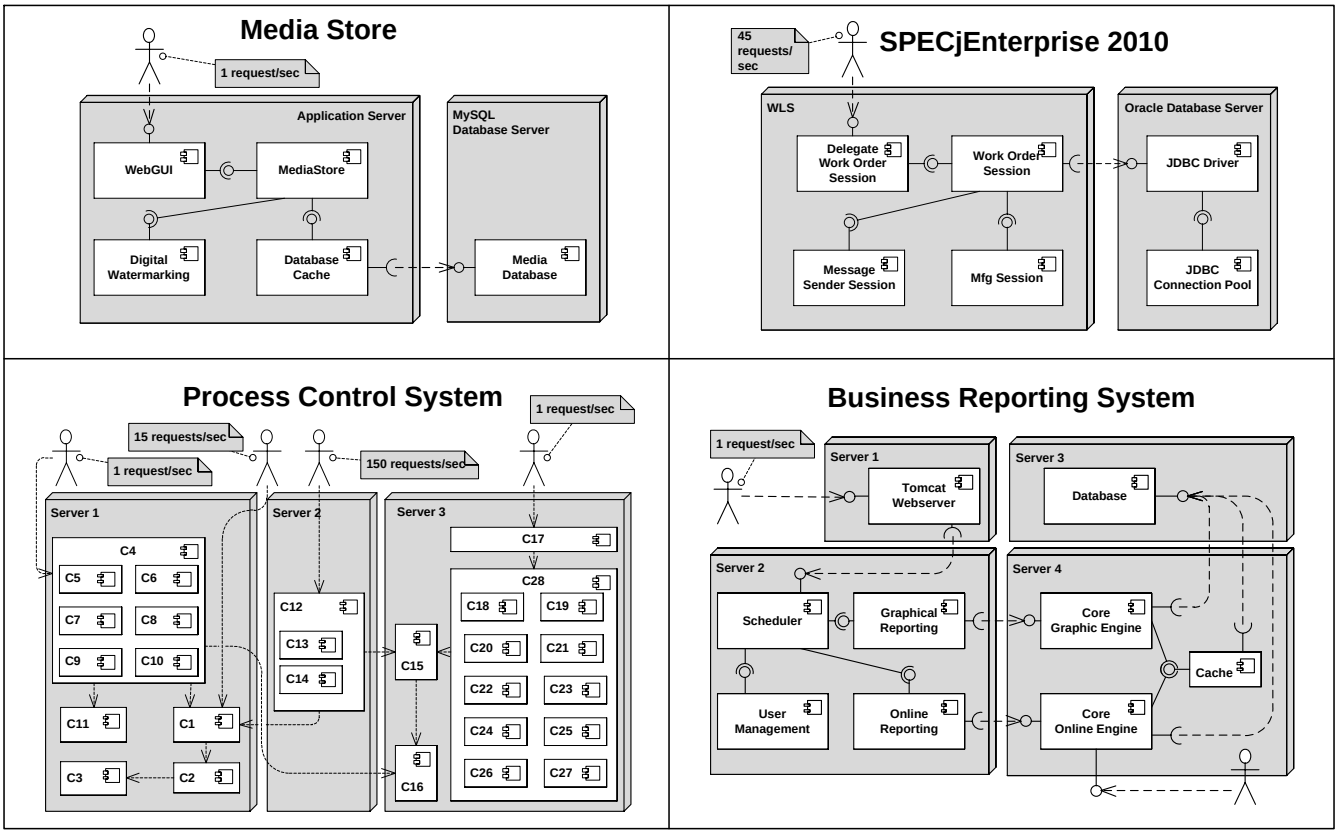

Fig. 2. Case study systems [9]

We ran the measurements via SSH. For every resource container specified by the PCM instance, we used a dedicated virtual machine and deployed the corresponding container on it. For each model, we started the RMI registry, the system, and the usage scenarios on the first virtual machine. This generates no overhead on the first machine since only resource containers create load. To overcome the fact that each virtual machine had three CPUs but the models specify mostly one CPU, we pinned via `taskset` the execution to the first processor of the system. For `Server 2` of the Business Reporting System we started the simulation without `taskset` since the model specifies three CPUs for this case. For measuring the CPU utilization, we used the command `mpstat` once per second and calculated the mean value for the idle time of the measurement in percentages. We calculated the CPU utilisation by simply substracting the mean idle time from 100%. Finally, we downloaded the measurement results via SCP and loaded them into our PCM workspace. The overall process took us around 15 minutes per case study when the time for executing the prototype and its calibration are not considered.

C. Calibration

We calibrated CPU and HDD for each virtual machine with the `Fibonacci` and `Large Chunk` calibration strategies, respectively. The `Fibonacci` strategy simulates CPU intensive tasks with minimised RAM access. For the HDD calibration, large files are loaded repetitively. The calibration took around two hours. Nonetheless, since the calibration files are saved after calibration, this was only necessary once.

At first we encountered unexpected values for the Media Store case study. We were able to identify the reason for this: the HDD calibration went wrong. The cause for this was the high amount of RAM available within the virtual machine. After all files were read for the first time, the operating system had stored all files in RAM. This had the consequence that instead of reading the files from HDD, the calibration read them directly from the RAM. Therefore, we implemented the HDD calibration in a way that the overall size of calibration files takes 110% of the RAM size forcing the operating system to swap.

We used a simple model to evaluate that the result was a better and more realistic calibration. The model had one hardware device, a HDD processing rate of 1000, one service which demands 1000 HDD units, and a closed workload usage scenario in which one user requests the service over and over again. Therefore, the mean response time is expected to be around 1.0. The red, left peak of Fig. 3 shows the result of the measurement before we increased the overall size. With a mean value of 0.59, the response times were clearly too fast. The blue, right peaks show the result after we improved the calibration. In this case the results have a mean value of

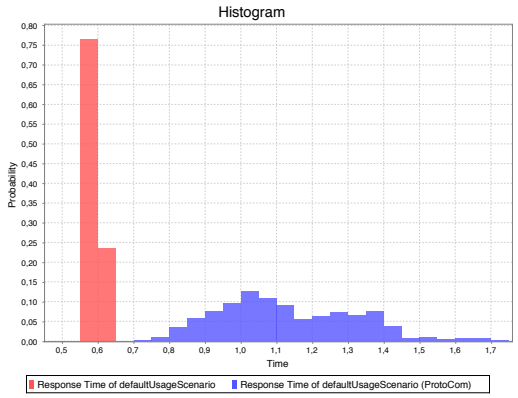

Fig. 3. Results for the simple model for testing HDD calibration

1.13 and lie almost normally distributed around the 1.0 mark. The results show a realistic behaviour when reading from the HDD. For instance, the response times depend on the sectors where the files are stored and on the time slices given by the operating system. The calibration takes this into account: the response times are distributed and the two higher blue peaks correspond to one time slice difference.

D. Results

In order to evaluate the quality of our measurements, we compare them to measurements created with SimuCom. For SimuCom we used the default configuration with 10000 measurements. Fig. 4 shows comparisons of response times for each case study as histogram and cumulative distribution function, respectively. The first row shows the results for Media Store, the second for SPECjEnterprise2010, the third for the sensor data receive usage scenario of the Process Control System, and the last row for the Business Reporting System. Red values correspond to measurements of ProtoCom and blue values to measurements of SimuCom. For the Business Reporting System case study, we additionally took ProtoCom measurements with the RAM intensive `SortArray` CPU calibration strategy (green values) instead of using the `Fibonacci` strategy. This way we model scenarios where algorithms heavily depend on RAM usage. SimuCom does not support simulation of such behaviour.

The values of SimuCom and ProtoCom mostly coincide. As the cumulative distribution functions show, the measured time values differ from each other by a maximum of around 0.250 sec. for Media Store, 0.008 sec. for SPECjEnterprise2010, 0.030 sec. for the Process Control System, and 15.000 sec. for the Business Reporting System case study. These values correspond to a maximum relative differences of around 50%, 132%, 400%, and 50%, respectively. As the last two case studies show, the response times of ProtoCom can exceed the values of SimuCom, as well as the other way round. Also, overlaps as in the Media Store case study are possible. The measurements taken with the `SortArray` calibration show a different behaviour: the values are widely spread compared to the other results.

Table 5 shows the measured values for mean response time, throughput, and CPU utilisation for the case studies. The columns depict the absolute reference values of SimuCom, the absolute values of ProtoCom, and its relative difference compared to SimuCom, respectively.

	SimuCom (reference)	ProtoCom	ProtoCom (relDiff)
Media Store			
RT(Mean)	1.332	1.028	-22.8%
TP	0.751	0.972	29.4%
U(AppServer_CPU)	34.1%	51.1%	49.9%
U(DBServer_CPU)	1.4%	48.9%	3392.9%
SPECjEnterprise			
RT(Mean)	0.043	0.048	11.6%
TP	44.679	41.667	-6.7%
U(Oracle_CPU)	19.6%	43.6%	122.4%
U(WLS_CPU)	62.5%	56.8%	-9.1%
PCS			
RT(Mean)	0.004	0.014	256.3%
TP	149.258	125.000	-16.3%
U(Server1_CPU)	6.5%	32.2%	395.4%
U(Server2_CPU)	1.1%	1.6%	45.5%
U(Server3_CPU)	55.0%	54.3%	-1.3%
BRS			
RT(Mean)	14.765	7.238	-51.0%
TP	0.995	0.990	-1%
U(Server1_CPU)	44.7%	53.1%	18.8%
U(Server2_CPU)	68.4%	73.5%	7.5%
U(Server3_CPU)	73.8%	78.4%	6.2%
U(Server4_CPU)	23.3%	26.6%	14.2%

KEY: RT = Response Time (sec), TP = Throughput (requests / sec), U = Utilization, relDiff = relative difference, PCS = Process Control System, BRS = Business Reporting System

Fig. 5. Measured results for the case studies

IV. Discussion

In this section we discuss our process and results with respect to usability, the techniques we used for taking measurements, the influence of the virtualisation, and the results of the measurements.

Usability We significantly improved the usability of ProtoCom. The improvement mainly results from 1) no manual adjustment of generated code is needed anymore; and 2) the JAR file runs ”out of the box” in a distributed environment and without configuring an application server like GlassFish.

Nonetheless, we still had to do some repetitive and manual work for receiving our results which caused additional 15 minutes per measurement. This was necessary because not all important deployment information for ProtoCom are included in the PCM: where to deploy the RMI registry, the system, containers, and usage scenarios? How to connect these entities? How to access the entities? How to handle hardware- and platform-specific properties that differ from the resource environment? For all those questions we had to find a manual solution, e.g., specifying IP addresses, using SCP and SSH, pinning to processors, manually measuring CPU utilisation, manually exporting to a JAR file, and using the menu for selecting a specific component. Therefore, we want to include these information in a mark model for the PCM-to-ProtoCom transformation and to fully automate the deployment and measurement steps as a future work.

Taking measurement Just like in SimuCom, time measurements of simulated action calls of usage scenarios are automatically taken by sensors. We deployed usage scenarios

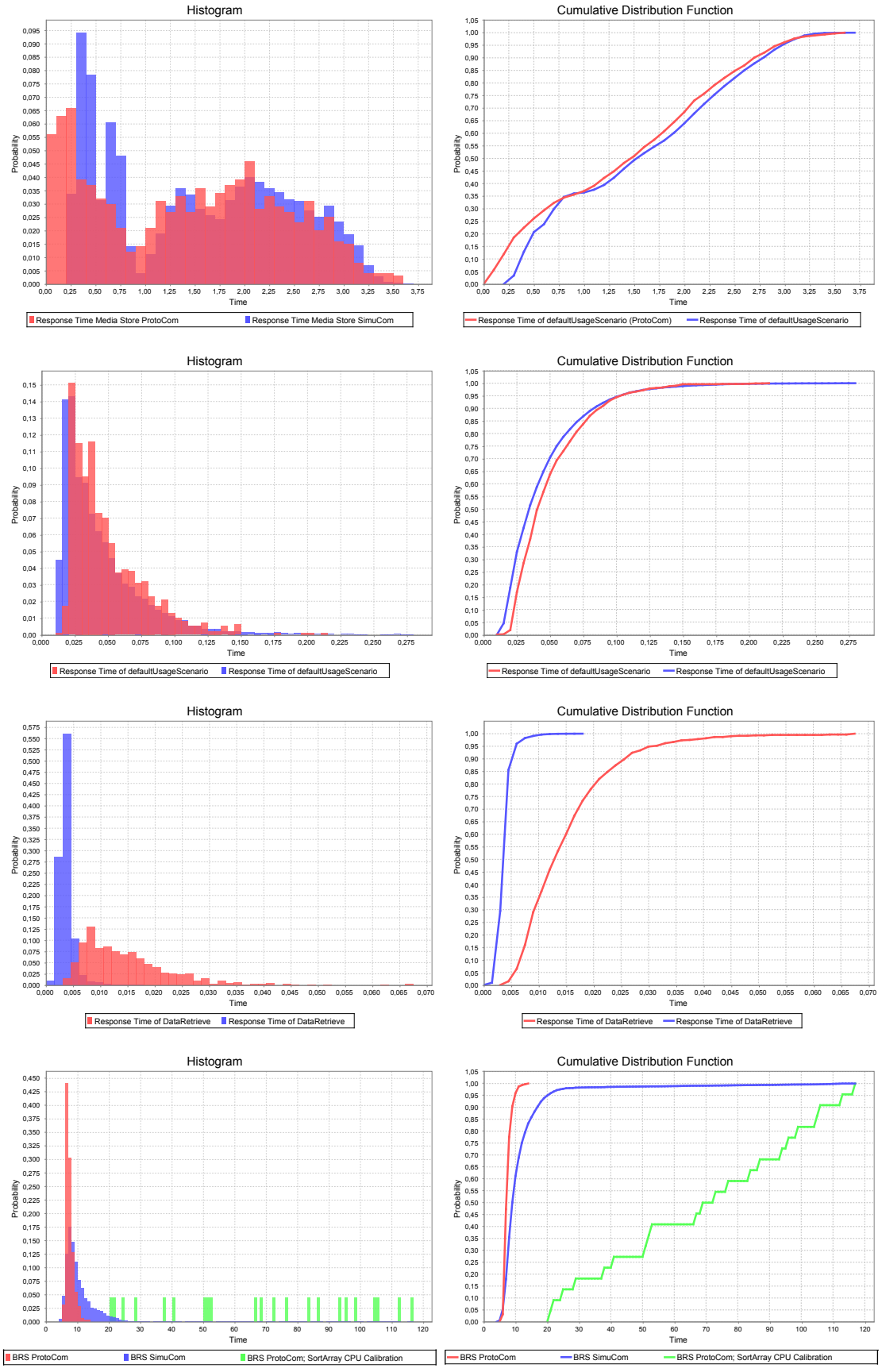

Fig. 4. Results for the case studies

on a hardware unit different from the system components, such that the measurements always include network latency and overhead generated by RMI calls. The measured latency (around 0.21ms) was then also included into the PCM models and hence considered in our SimuCom runs.

We calculated the CPU utilisation manually by using values measured by `mpstat` and a spreadsheet application. This leads to two sources of inaccuracy. First, `mpstat` measures the CPU load at most once a second. The shortest measurement took about eight seconds, such that the amount of taken samples is rather small. The second problem is the CPU load generated by other processes and `mpstat` itself. We only have limited influence on CPU load generated by system processes, yet it was minimised by us.

Virtualisation Virtualised parts of the hardware environment have different impact on the measurements than their physical counterparts. The network latency is lower than in a real network, since the hardware nodes are virtualised on one physical Blade server. However, the overhead created by RMI calls outweighs the latency of the communication medium, such that the effect can be neglected in our case studies. Instead of a physical HDD we used a SAN. Access times of the SAN behave comparable to a physical HDD; the mean access time is the same but the standard deviation is greater. For other parts, e.g., CPU, no discrepancy between virtual and physical resources has been experienced.

Results In general, the results reported in Section III-D show a very good correlation between predictions made by using SimuCom and ProtoCom measurements. We conclude for the first two models we studied, the generated code and its calibration reflect the models well.

The third model has small resource demands leading to short response times. Influencing factors not considered by SimuCom, e.g., parts of middleware, OS, or network, have a high impact on the results. In such cases ProtoCom's results are closer to reality, and thus show the importance of ProtoCom.

Following the argumentation from above, the fourth model where ProtoCom is faster than the SimuCom prediction seems to be suspicious. We assume that such cases relate to variations in the calibration or unidentified OS scheduler features. Further measurements need to be collected to narrow down the cause of this effect. The results of the `SortArray` calibration show that SimuCom's results only hold for the assumption that the modeled system is CPU intense. ProtoCom allows considering cases where RAM is used more heavily.

In addition, for our measurements we used the `Fibonacci` CPU load generator strategy which is known to reflect the predictions of SimuCom well. For measurements taken with different CPU strategies, especially memory-bound strategies like array sorting, ProtoCom shows much larger deviations highlighting missing performance relevant factors abstracted in SimuCom.

V. Related Work

This work extends earlier work [6] on model-driven generation of performance prototypes which has been developed as part of the model-driven quality analysis methods for component-based software systems presented in [7] in the context of the PCM. ProtoCom's main assumption relies on the observation by Hu and Gorton [4], that performance models make simplifying assumptions and thus, require additional validation for specific settings.

Related work can be classified in approaches based on the PCM and approaches not relying on the PCM. In the PCM context, ProtoCom and particularly its workload generators have been applied in validating the Design Space Exploration (DSE) approach by A. Koziolek [13], the modelling of operating system schedulers by Happe [14], the Ginpex approach by M. Hauck [15], or the Software Performance Cockpit by D. Westermann [16]. They showed good calibration results in these works as they did in our case studies. We generalised the generators for this paper and automated the setup of processing rates per resource container. However, none of these approaches focused on reporting a case study with ProtoCom itself - they rather used parts of it in different contexts.

Works not using the PCM have been surveyed and discussed in [6]. In the following, we review the types of results reported by the papers most closely related to ProtoCom. Among those approaches was Woodside and Schramm's performance prototype generator based on LQNs [3]. In their paper, they use an illustrative case study to validate their approach in multi-server settings including network demands. Hu and Gorton [4] illustrate their approach only on an example from the graphics processing domain. They evaluated the number of workers and their relation to the system's performance. In more recent work, Zhou, Gorton, and Lui [5] presented prototype generation for JavaEE applications. They give an example using a limited number of different client workload profiles. However, for none of the reported approaches, we identified a larger case study or experience report. It seems they either remained in a research prototype phase or further applications have not been reported.

VI. Conclusions

This paper presents an experience report in using an improved version of the ProtoCom performance prototype generator in a series of case studies on a virtualised Blade server environment. On the one hand, our report presents the realised improvements on ProtoCom with respect to new concepts and usability. On the other hand, we present and discuss our measurements on virtual Blade servers.

This report helps software architects to judge the applicability and usefulness of model-driven performance prototype generation based on the Palladio Component Model. It gives an impression on the expected accuracy and the lessons one can learn from a prototype in addition to model-based performance analyses methods.

In the future, we plan to extend our prototype generator further. Features we plan to realise with respect to functionality or usability include an additional mark model for the PCM-to-ProtoCom transformation to fully automate the deployment and measurement steps. Additionally, ProtoCom will be applied in new application domains to validate performance models. As first domain, we consider using ProtoCom in a self-adapting scenario, where the component structure and allocation is adjusted to its context using graph transformations based on Story Diagrams [17].

ACKNOWLEDGMENTS

We would like to thank Heiko Koziolek for granting permission to reuse some of his figures, Anne Koziolek for her work on ProtoCom and the case study models, and Steffen Becker for his constructive input and feedback to our work. This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing"(SFB 901).

REFERENCES

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

[2] H. Koziolek, B. Schlich, C. Bilich, R. Weiss, S. Becker, K. Krogmann, M. Trifu, R. Mirandola, and A. Koziolek, "An industrial case study on quality impact prediction for evolving service-oriented software," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011), Software Engineering in Practice Track*. ACM, New York, NY, USA, 2011.

[3] C. M. Woodside and C. Schramm, "Scalability and performance experiments using synthetic distributed server systems," *Distributed Systems Engineering*, vol. 3, pp. 2–8, 1996.

[4] L. Hu and I. Gorton, "A performance prototyping approach to designing concurrent software architectures," *Proceedings of the 2nd International Workshop on Software Engineering for Parallel and Distributed Systems*, pp. 270–276, 1997.

[5] L. Zhu, I. Gorton, Y. Liu, and N. B. Bui, "Model Driven Benchmark Generation for Web Services," in *SOSE '06: Proceedings of the 2006 International Workshop on Service-Oriented Software Engineering*. ACM, 2006, pp. 33–39.

[6] S. Becker, T. Dencker, and J. Happe, "Model-Driven Generation of Performance Prototypes," in *Performance Evaluation: Metrics, Models and Benchmarks (SIPEW 2008)*, ser. Lecture Notes in Computer Science, vol. 5119. Springer-Verlag Berlin Heidelberg, 2008, pp. 79–98. [Online]. Available: http://www.springerlink.com/content/62t1277642tt8676/fulltext.pdf

[7] S. Becker, *Coupled Model Transformations for QoS Enabled Component-Based Software Design*, ser. The Karlsruhe Series on Software Design and Quality. Universitätsverlag Karlsruhe, 2008, vol. 1.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, USA, 1995.

[9] S. Becker, S. Kounev, A. Koziolek, H. Koziolek, and P. Meier, "Model transformations for performance prediction of component-based software systems," *Submitted to IEEE Transactions on Software Engineering*, 2011.

[10] H. Koziolek, S. Becker, and J. Happe, "Predicting the Performance of Component-based Software Architectures with different Usage Profiles," in *Proc. 3rd International Conference on the Quality of Software Architectures (QoSA'07)*, ser. Lecture Notes in Computer Science, vol. 4880. Springer-Verlag Berlin Heidelberg, July 2007, pp. 145–163. [Online]. Available: http://sdqweb.ipd.uka.de/publications/pdfs/koziolek2007b.pdf

[11] F. Brosig, "Automated Extraction of Palladio Component Models from Running Enterprise Java Applications," Master's thesis, Universität Karlsruhe (TH), Karlsruhe, Germany, June 2009.

[12] H. Koziolek and R. Reussner, "A Model Transformation from the Palladio Component Model to Layered Queueing Networks," in *Performance Evaluation: Metrics, Models and Benchmarks, SIPEW 2008*, ser. Lecture Notes in Computer Science, vol. 5119. Springer-Verlag Berlin Heidelberg, 2008, pp. 58–78. [Online]. Available: http://www.springerlink.com/content/w14m0g520u675x10/fulltext.pdf

[13] A. Martens, H. Koziolek, S. Becker, and R. H. Reussner, "Automatically improve software models for performance, reliability and cost using genetic algorithms," in *WOSP/SIPEW '10: Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*. New York, NY, USA: ACM, 2010, pp. 105–116. [Online]. Available: http://www.inf.pucrs.br/wosp

[14] J. Happe, "Predicting Software Performance in Symmetric Multi-core and Multiprocessor Environments," Dissertation, University of Oldenburg, Germany, August 2008. [Online]. Available: http://oops.uni-oldenburg.de/volltexte/2009/882/pdf/happre08.pdf

[15] M. Hauck, M. Kuperberg, N. Huber, and R. Reussner, "Ginpex: Deriving Performance-relevant Infrastructure Properties Through Goal-oriented Experiments," in *7th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA 2011)*, Boulder, Colorado, USA, June 20-24 2011.

[16] D. Westermann, J. Happe, M. Hauck, and C. Heupel, "The performance cockpit approach: A framework for systematic performance evaluations," in *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010)*. IEEE Computer Society, 2010, pp. 31–38.

[17] T. Fischer, J. Niere, L. Torunski, and A. Zündorf, "Story diagrams: A new graph rewrite language based on the unified modeling language," in *Proc. of the 6th International Workshop on Theory and Application of Graph Transformation (TAGT), Paderborn, Germany*, 1998.

EVENTSIM – An Event-driven Palladio Software Architecture Simulator

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

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

***Abstract*—Evaluating software quality from early development stages on is considered essential for meeting performance requirements. Being accompanied by a number of analytical and simulative software quality prediction tools, the Palladio component model (PCM) is well suited to early performance assessments, where the focus is on component-based systems. Among these tools, the SimuCom simulator is particularly important since analytical approaches impose restrictions on the PCM models to be evaluated. SimuCom, however, uses the process interaction simulation paradigm, which is known for its inferior performance and scalability. We present EventSim, a novel simulator for PCM models which tackles performance and scalability issues by adhering to the event scheduling simulation paradigm instead. Besides performance and scalability, particular emphasis has been put on extensibility. For example, the simulation logic could be easily augmented or partly replaced—even at simulation runtime. Experiments indicate a speed-up of up to 50 times compared to SimuCom. Moreover, EventSim resolves scalability issues encountered with SimuCom.**

I. INTRODUCTION

Uncovering software performance issues in late development stages often entails substantial architectural changes or even causes whole projects to fail [1]. Evaluating software performance throughout the whole development process is therefore crucially important. This is notably true for early development stages since it is particularly expensive to revise architectural decisions already transformed to source code, for example. Early performance assessments, however, imply a lack of runnable software artefacts whose performance could be measured. Hence, early evaluations are commonly based on performance models, which are surveyed in [2] for classical software performance prediction and in [3] for performance prediction of component-based software systems.

The Palladio component model (PCM) [4] is a domain-specific language (DSL) for component-based software architectures. Instances of the PCM, which are also called PCM models hereafter, describe a system's architecture along with quality annotations capturing performance and reliability, for instance. Using model transformations, various types of performance models can be automatically generated from a PCM instance. These performance models can be divided into analytical and simulative approaches, where simulation models are particularly important for analysing complex systems since analytical models suffer the state-space explosion problem.

However, the SimuCom simulator [5] included with the PCM is based on the process interaction simulation paradigm generally known for its inferior performance [6, p. 567]. In addition, SimuCom being implemented in Java further adds to the performance drawbacks since there is no efficient means to represent simulated processes in Java. Using Java, it is common to represent each simulated process by a native, preemptive thread. These are scheduled by the operating system, which decides on their interleaving, thus resulting in an uncontrolled execution order of simulated processes if the simulator does not take any countermeasures. Therefore, costly synchronisation between threads is commonly used to regain control of the execution order of simulated processes.

Alternatives to the use of threads relieve the need for synchronisation but bring some other issues. Stadler et al. [7] suggest an extension of the Java language by coroutines, which lend themselves well to representing simulated processes. Their approach, however, assumes a modified Java runtime environment. Jacobs and Verbraeck [8], by contrast, propose an interpreter for simulated processes. The processes are still modelled as threads, but are loaded into a single-threaded interpreter which resembles the execution in the intended order. Though the interpreter reduces context switching between threads, a high interpretation overhead has been reported [9].

In this paper, we present a PCM simulator called EventSim that circumvents the addressed performance issues by adhering to the event scheduling simulation approach. In contrast to process interaction, event scheduling treats simulated processes as sequences of events, which means that each process has to be broken down into a number of events. While simulated processes represent a period of simulated time, events do not extend in time. To advance the simulation time, the present event schedules another event to occur in the simulated future. Thus, while multiple processes may be active concurrently, events do not overlap each other in simulation time. This creates a great advantage in that all events can be processed in a single thread—one after another.

The contributions of this paper are as follows. First, we describe an approach to simulate PCM models in a fully event-oriented way. Second, we compare our approach to the existing SimuCom simulator in terms of simulation duration. We begin with an overview of the PCM in Section II, which lays the foundation for the following sections. Before we cover our own work, Section III introduces related work

on software performance simulation based on PCM models. Then, we present the EventSim simulator for PCM models in Section IV, which is followed by a validation in Section V. Section VI deals with a performance comparison of EventSim and SimuCom. Finally, Section VII concludes this paper.

II. Palladio Component Model

The Palladio component model (PCM) [10] is a domain-specific language which includes a meta-model targeted to model component-based software architectures in terms of quality attributes, such as performance and reliability. Instances of the PCM represent software systems by capturing their quality-related characteristics on an architectural level. Based on these system abstractions, performance and further extra-functional attributes of planned or existing software systems can be predicted.

The meta-model is designed to support the component-based software engineering process described by Koziolek and Happe [11]. There, four roles are identified being involved in the development of component software. *Component developers* specify and implement components. *Software architects* assemble the system from existing components or request additional ones from the component developers. *System deployers* specify the execution environment in terms of resource containers and their interconnections before deploying components on the containers. *Domain experts* describe the typical system usage by providing usage scenarios. For each role, there is a PCM partial model focusing on elements that are of interest for the respective role. These partial models yield the overall PCM model. They are presented below.

A. Repository Model

The repository model is created by the component developer. Essentially, it contains type-level component specifications that provide and require interfaces, where interfaces are signature list. Providing an interface means that the component implements a service for each signature listed in the interface. Requiring an interface means that the component may use the services listed in the interface to provide its own services.

Additionally to these structural information, the component developer provides information on the behaviour of provided component services. Service behaviour is described by *resource-demanding service effect specifications* (RD-SEFFs), which capture resource demands induced by the modelled service itself (`InternalActions`) along with calls to other services (`ExternalCallActions`). In this way, RD-SEFFs abstract from actual service implementations.

Similar to UML activity diagrams, an RD-SEFF comprises a set of `AbstractActions` that are interconnected in a predecessor-successor relationship, thus imposing a total order on the actions. Hence, we use the term action chain. Despite of the total order, control flow constructs such as loops and branches can be used. For this, `Loop` and `Branch` actions, for instance, allow for a hierarchical nesting. `Loops` encapsulate exactly one action chain along with a loop iteration count. `Branches` encapsulate an action chain for each transition, where each transition is associated with a condition.

B. System Model

The software architect builds a model of the component software by assembling one or more components yielding the system model. Similar to components, systems provide and require interfaces as well. By contrast, however, the system behaviour is not modelled explicitly, but emerges by the collaborative behaviour of assembled components.

C. Resource Environment Model

The system deployer describes the execution environment of components along with their deployment on servers. These so-called `ResourceContainers` provide resources such as processors and storage devices. Each resource is specified by a `ProcessingResourceSpecification`, which captures the processing rate, the scheduling policy (e.g. first-come, first-served) and the number of instances (e.g. processor cores). `ResourceContainers` communicate over communication links modelled by `LinkingResources`.

D. Allocation Model

After having modelled the execution environment, the system deployer allocates components to resource containers. The deployment relation is constituted by `AllocationContexts` in that they reference the component to be deployed (an `AssemblyContext`) along with the deployment target (a `ResourceContainer`).

E. Usage Model

As the performance of a component-based system depends not only on the system itself, but also on the way it is used [12], the usage behaviour needs to be described for proper performance predictions. Therefore, the domain expert describes the system usage in a `UsageModel`. It comprises one or more `UsageScenarios`, each representing a specific use case, which describes the interactions of a class of users with the system [10]. Each `UsageScenario` comprises exactly one `UsageBehaviour` and one `Workload`. As with RD-SEFFs, behaviour is captured by a chain of actions. Usage actions, however, are of the type `AbstractUserAction`.

III. Related Work

This section provides a brief overview of existing simulators for PCM models. This includes in particular the SimuCom simulator, which we used as reference simulator to validate and evaluate our work in sections V and VI, respectively.

SimuCom is a software performance and reliability simulator for PCM models, which is comprised of a collection of Eclipse plug-ins implemented in Java. It is delivered with the PCM and thus can be considered as Palladio's reference simulator. Simulation models in SimuCom are automatically generated from PCM models by an Xpand[1] model-to-text transformation. More precisely, the Xpand template engine generates an executable Java representation of the PCM model. Each component, for instance, is mapped to a Java class whose methods represent the component's services. In this way, a

[1] http://www.eclipse.org/modeling/m2t/

service call can be simulated by calling the corresponding method on an instance of the generated class. As described in the introductory section, major parts of SimuCom adhere to the process interaction simulation paradigm. Namely, processes are used for modelling the behaviour of users and their system requests. The behaviour of simulated resources, by contrast, relies on the event scheduling approach. SimuCom relies on the so-called abstract simulation engine, which generalises simulation concepts usually provided by simulation libraries like Desmo-J[2] or SSJ[3]. For example, these concepts include an implementation for simulated processes, for events and a simulation clock. A mapping from abstract concepts to their library-specific implementations is provided for both engines mentioned before, Desmo-J as well as SSJ. At the time of writing, however, SimuCom's resource layer bypasses the abstract simulation engine since SSJ is used directly. Consequently, SimuCom is limited to SSJ at this time. It also has to be noted that SimuCom makes no use of simulated processes offered by SSJ, but uses a tailor-made process implementation instead.

In contrast to SimuCom, *SLAstic.SIM* [13] is a purely event oriented simulator based on PCM models. Being developed in the scope of the SLAstic approach [14], SLAstic.SIM focuses on runtime reconfiguration of component-based software architectures. For this, SLAstic.SIM is supplied with reconfiguration plans over the course of a simulation run. Each plan defines a sequence of reconfiguration operations to be applied to the current system architecture. Available reconfiguration operations are (de-)replication and migration of component instances as well as (un-)provisioning of resource containers being the target of replication and migration operations. However, not all regular PCM modelling elements are supported in SLAstic.SIM. For example, no stochastic expressions can be used in performance annotations [13].

SimQPN [15] is an event oriented simulator for software performance models described by the queuing Petri net (QPN) formalism. Using the model transformation proposed by Meier et al. [16], PCM models can be automatically transformed to queuing Petri nets accepted by SimQPN. However, some PCM modelling elements can not be accurately modelled by a QPN leading to prediction errors.

IV. EVENTSIM

EventSim is a software architecture simulator for software quality attributes based on the Palladio component model. At the time of writing, EventSim is capable of simulating software performance and provides extension mechanisms for further quality dimensions like reliability, for instance. When provided with a PCM architectural model along with a usage profile, EventSim predicts a number of performance metrics, such as response time and resource utilisation. The prediction is conducted by a discrete-event simulation driven by the event scheduling world-view.

Our simulator is implemented in Java and bundled as an Eclipse/OSGi plug-in. This way, existing Palladio plug-ins can be easily reused. In particular, this applies to the following parts of SimuCom: (i) the resource layer responsible for simulating active resource behaviour, specifically the respective scheduling policies, (ii) the pseudo-random number generator, (iii) the framework supporting the collection of measurements, namely the ProbeSpecification in conjunction with the SensorFramework and (iv) the facility to evaluate stochastic expressions. Furthermore, EventSim requires a simulation library such as Desmo-J or SSJ, for example. For this, the abstract simulation engine introduced in Section III has been factored out from SimuCom and reused in EventSim.

The remainder of this section deals with the actual simulation process and the building blocks involved.

A. Simulation Overview

We begin with a high-level overview of the simulation process as shown in Fig. 1. The *user* entity simulates the behaviour of system users, each of which issues a sequence of calls to the simulated system (`EntryLevelSystemCalls`). For each system call, the simulation spawns a *request*, whose task is to simulate the system behaviour resulting from the service call. When the system behaviour contains control flow forks, a dedicated request is spawned for each fork (`ForkedBehaviour`). While simulating the system behaviour, requests demand shared *resources*, which are either active or passive. Active resources are, for instance, processors or storage devices. Examples of passive resources are semaphores or database connection pools.

Resources are limited in capacity, and multiple requests may be active at the same time. As a result, requests compete for scarce resources causing resource contention. In consequence, if a requested resource is busy with a competitor, a request might have to wait for its turn, leading to waiting times.

The presence of multiple requests in the simulated system, each issuing demands to shared resources, leads to the overall system behaviour that the simulation is to imitate. Observing the involved entities over the course of a simulation run yields the simulation results. These are, for instance, resource utilisation over simulation time as well as response times of system calls and usage scenarios.

The simulation in EventSim is mainly driven by the three entities presented above, an interpreter employed by the entities, and events which continuously trigger the interpretation process over the course of the simulation. Below, we cover each of these parts in more detail.

B. Entities

As mentioned before, entities represent simulated system users (user entity), their system requests (request entity) and resources available to requests, such as processors and storage devices (resource entity). Additionally, a request may also represent a fork of the simulated control flow.

It is the responsibility of each entity to simulate its own behaviour. The behaviour of a user is described by a `UsageScenario` and the behaviour of a request is modelled by one or more `ResourceDemandingSEFFs`. Both

[2]http://desmoj.sourceforge.net/

[3]http://www.iro.umontreal.ca/~simardr/ssj/

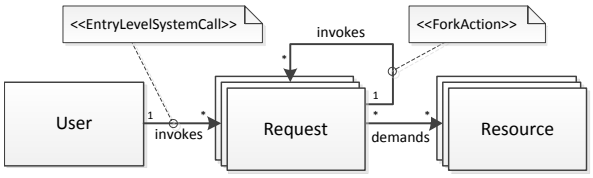

Fig. 1. Overview of entities and their interrelation

behaviour representations are similar in that they are modelled by a chain of actions, which is why entities simulate their behaviour by passing along the corresponding action chain while simulating each action encountered on the way towards the end of the chain. As this traversal procedure of users and requests is quite similar, both entities employ a common behaviour interpreter.

C. Behaviour Interpreter

The behaviour interpreter simulates the behaviour of users and their requests in an interpretive way. Starting at a supplied action, the interpreter passes along the chain of actions arising from the predecessor-successor relationship between actions. For each action encountered, the interpreter performs the action-specific simulation logic.

The interpreter's traversal procedure can be seen in Fig. 2. The traversal starts with the behaviour interpreter being passed an action to begin with. Depending on the action's type, the interpreter loads the *traversal strategy* registered for this type of action. On behalf of the interpreter, the strategy then executes the simulation logic for that action. This way, the simulation logic of actions is not concentrated in the interpreter and can be easily replaced, as described below in Section IV-F.

The interpreter receives instructions from the respective traversal strategies on how to proceed the traversal, i.e. which action is next. This information is encapsulated by a *traversal instruction*. Being unaware of an action's simulation logic, the interpreter can not have this knowledge. Consider a `Branch` action, for instance, which encapsulates two control flow alternatives, each with a transition probability of 0.5. The choice for one of the control flow alternatives is in the responsibility of the corresponding traversal strategy, which bases its decision on a pseudo-random number. Thus, the selected control flow alternative can not be known to the traversal procedure in advance.

The *traversal state* (not depicted) captures the interpretation progress along with the state of the entity being simulated. The interpretation progress essentially includes the action that has just been simulated (the previous action), the action that is being processed at the moment (the current action) and a reference to the simulated component instance[4], if the interpreter simulates a system request. Taking into account the hierarchical nesting of actions, the interpretation progress is maintained for each level of traversal hierarchy by means of a stack.

[4]A deployment component instance (see [17, pp. 5-7], for example), though not explicitly modelled in the PCM, is created at simulation runtime for each deployed `AssemblyContext` (i.e., for each `AllocationContext`).

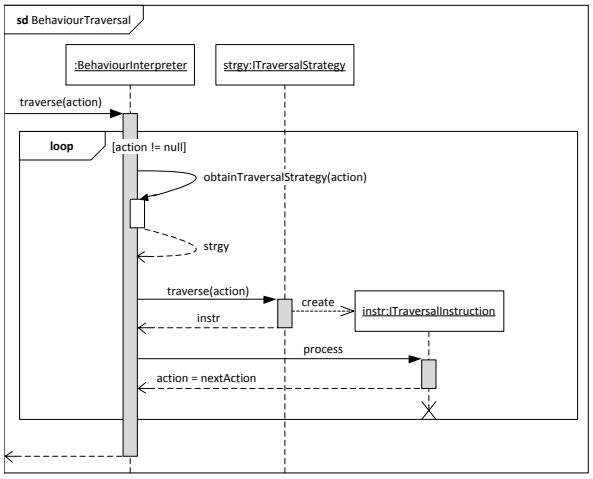

Fig. 2. Interpretive approach for simulating usage and component behaviour

The actions traversed by the interpreter are interdependent in that one action may characterise a variable that is read by some subsequent action. Specifically, a user may issue a system call with a number of arguments that are used by the called system service. Variables are characterised by stochastic expressions, which is why a sample is drawn for each user or request upon first access. These samples form the major part of the traversal state of a user or request. Following the approach adopted by SimuCom, samples (or more precisely, evaluated variable characterisations) are organised as a stack of frames, where each stack frame represents a variable scope and holds the evaluated variable characterisations that are valid in the respective scope (cf. [4]).

Apart from information stored in the traversal state, the interpreter is stateless, which is utilised to resume an interrupted interpretation process by passing the corresponding traversal state to an existing or newly created interpreter instance. The interpretation must be interrupted whenever the simulation time has to be advanced as described below in Section IV-D.

D. Events

Events are scheduled by the various entities to trigger the simulation of usage or component behaviour at certain points in simulation time. For this, entities create events, each of which encapsulates a call to the behaviour interpreter. Events are handed over to the event scheduler along with the intended occurrence time. The scheduler ensures the timely execution of events once the associated simulation time is reached.

Events are instantaneous occurrences, and no simulation time may pass within the execution of an event. In EventSim, this specifically applies to the behaviour interpreter since the interpretation procedure is executed in the scope of an event. For this reason, the interpretation has to be interrupted whenever an advance in simulation time is to be realised. Before doing so, the interpreter or, more specifically, the current traversal strategy creates and schedules a resumption event.

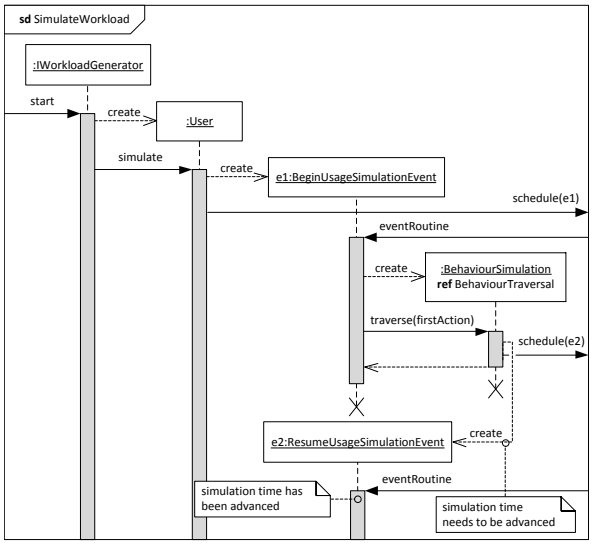

Fig. 3. Simulating a workload (up to the first simulated time advance)

E. Interaction between Simulation Building Blocks

The interaction of entities, events and the behaviour interpreter is shown in Fig. 3 for the start of a simulation run. For brevity, the illustration omits requests and resource accesses as well as associated events.

The simulation starts with the `IWorkloadGenerator` creating an initial user population according to the workload specified in the `UsageScenario`. Depending on the workload type, either a `ClosedWorkloadGenerator` or an `OpenWorkloadGenerator` is used. Then, each generated user is asked to begin simulating its behaviour. For this, each user creates an event that triggers the start of the usage behaviour simulation. This event is then scheduled to occur at simulation time t = 0, i.e. at once. Next, the event scheduler invokes the event routine since the event's occurrence time matches the current simulation time, which is still at zero. As stated earlier, the event makes use of the behaviour interpreter, which is why the event routine first retrieves an interpreter instance, which is then used to begin simulating the user's behaviour.

As motivated in Section IV-D, all events have to finish processing before an advance in simulation time may occur. Therefore, if the simulation of a specific action requires a time advance, the interpretation needs to be interrupted to allow the enclosing event to finish its processing. Before doing so, the interpreter schedules a resumption event to occur at the intended simulation time. This event is created with the current traversal state being passed as an argument to allow for a seamless continuation of the traversal when the event's occurrence time is reached.

F. Extensibility

The approach of having a particular traversal strategy for each type of control flow action, as presented in Section IV-C, along with a repeated strategy lookup for each action being simulated, allows for a flexible extension of EventSim. For example, introducing an additional control flow action in the PCM merely requires a suitable traversal strategy to be implemented and announced to the behaviour interpreter. The announced mapping between an action type and its respective traversal strategy can be changed even at simulation runtime, thus allowing for exchanging parts of the simulation logic at certain points in simulation time. If the simulation logic is to be extended while preserving existing functionality at the same time, the decorator design pattern could be used to enrich existing traversal strategies.

G. Limitations

At the time of writing, the simulation of software quality attributes in EventSim is limited to performance. Predicting reliability, as additionally offered by SimuCom, could be realised by decorating the existing performance-centred traversal strategies. Furthermore, EventSim does not yet support the whole range of PCM modelling elements. Namely, composite components and subsystems have yet to be implemented, as well as the simulation of throughput and latency of network links between resource containers. Traversal strategies are not yet implemented for `CollectionIteratorActions` and synchronous `Fork` actions. Asynchronous `Forks`, however, are already available.

V. VALIDATION

The purpose of the validation is to show that EventSim yields correct simulation results. We assume that the results provided by SimuCom are valid, which allows us to judge the correctness of EventSim by comparing the simulation results to those of SimuCom. Several case studies underpin this assumption (e.g., [4], [18]). The results with regard to a specific PCM model are considered consistent if i) both simulators yield exactly the same results, or if ii) the results differ, but the difference is not statistically significant.

The first case is true, if the sum of differences between the simulation results yielded by SimuCom and EventSim is zero. Simulation results are sequences of values ordered in time, thus allowing for a pairwise comparison between the simulators. In the latter case, we utilise the two-sample Kolmogorov-Smirnov test (ks-test) in order to determine whether the two simulation results underlie the same probability distribution.

As a prerequisite, both simulators are required to work in a deterministic way. Otherwise, it would not be clear whether observed differences are due to differences in the simulator implementations or whether they arise due to the indeterministic behaviour of one of the simulators. We therefore require the results of subsequent simulation runs to be equivalent, provided that the same simulator has been used and the input to the simulator does not vary between the runs. For this purpose, we initialised the pseudo-random number generator with a fixed seed, thus resulting in a deterministic sequence of numbers from which the respective simulator draws its pseudo-random numbers.

A. Experiments

Three variants of the MediaStore PCM model [19] were simulated with both simulators, resulting in a total of six result sets. The experiments described in the following differ in workload intensity and the scheduling policies used by simulated resources.

1) No resource contention (E1): In this experiment, the number of users passing through the simulated system concurrently is set to one. As a result, no resource contention between different users can occur. Furthermore, no control flow forks are present, which is why there is also no resource contention between requests of the same user.

2) Resource contention (E2): In this experiment, the number of concurrent system users is increased to ten. In contrast to *E1*, these users compete for limited resources. In consequence, waiting times arise when a user demands a resource that is busy with another user. For the scheduling of resource demands, resources use either the FCFS (first-come, first-served) scheduling policy or PS (processor sharing).

3) Avoiding processor sharing (E3): In this experiment, the PS scheduling policy has been replaced with FCFS so that each resource uses FCFS for scheduling. Apart from that, this experiment is equal to *E2*.

B. Validation Results

In experiment *E1*, the sum of differences is zero for all performance metrics resulting from the simulation runs. We can therefore argue that not only the various performance metrics predicted by SimuCom and EventSim underlie the same probability distribution, but both simulators even produce exactly the same sequence of results when provided with the PCM model corresponding to *E1*. Preliminary experiments suggest that this result can be generalised to all PCM models without resource contention, i.e. to PCM models with a single-user closed workload and at the same time an absence of control flow forks.

The absence of resource contention is not a realistic scenario for most software systems, which is why experiment *E2* aims at comparing the simulators in the presence of resource contention. In contrast to the previous experiment, the sum of differences is not applicable since the sample size differs; but, even when forcing an equal sample size by truncating the larger sample, the sum of differences is greater than zero for each of the performance metrics. Applying the ks-test, the null hypothesis H_0 could not—with a single exception—be rejected at an 0.95 confidence level, where H_0 states that the predictions of both simulators follow the same probability distribution. For the utilisation of the application server's CPU, H_0 was rejected. A visual inspection, however, suggests that both simulators predict nearly the same utilisation for the application server's CPU. We therefore conclude that the probability distributions that underlie the predictions of SimuCom and EventSim do not differ significantly with respect to experiment *E2*.

Nevertheless, provided that a fixed seed is used for the pseudo-random number generator, the simulators should yield exactly the same results—-not only results that follow the same probability distribution. Further investigations indicate that the PS scheduling policy implementation (which is identical for both simulators) causes the indeterministic behaviour. For this reason, in experiment *E3* PS has been replaced with FCFS. Despite of the resource contention, both simulators produce exactly the same results now (i.e., the sum of differences is zero). Therefore, our simulator can be considered semantically equivalent to SimuCom with regard to the PCM models used in the validation. That is, the predictions yielded by SimuCom and EventSim are indistinguishable as long as none of the simulated resources use the PS scheduling policy. The indeterminism, however, is not an issue with PS in general, but with the current PS implementation shared between SimuCom and EventSim.

VI. Evaluation

In this section, we compare EventSim to its process-oriented counterpart SimuCom in terms of performance and scalability. In addition, we identify and examine scalability limits of the two simulators. Some aspects of the simulation, like the evaluation of stochastic expressions and storage of measurements, were excluded from the comparison as both simulators use the same implementation.

A. Experimental Setting

For the performance evaluation, we measured the growth in simulation time when increasing the complexity of the simulated PCM model, where complexity refers to the number of performance-relevant modelling elements used. For this purpose, numerous experiments were conducted, each consisting of an artificial PCM model along with a simulation configuration that determines, amongst other things, the stopping criteria for the respective simulation run. An experiment automation tool has been developed to generate experiments according to a configuration model. In a second step, the experiment automation simulated the generated models in both simulators while observing simulation runtime and resource utilisation.

Each experiment covers one performance-relevant modelling element (hereafter: *factor*) in isolation at a specific factor level ranging from 1 to 1,000 with a step width of 100. That is, the corresponding PCM models contain between 1 and 1,000 modelling elements of the same type. The usage scenario has a closed workload with a population of 1 and is simulated 1,000 times before the simulation stops. Experiments are repeated 30 times to reduce the measurement error, thus resulting in a total of 300 experiments per factor.

Both simulators were executed on top of Eclipse Galileo (v. 3.5) and SSJ 2.1.3 running in a Java 1.6 HotSpot™ 64-bit server virtual machine (VM). The *-xms* and *-xmx* VM arguments were set to 512M and 1024M, respectively. The default stack size has been used, which is 1024 KB for the 64-bit VM and the used operating system. The simulation was executed on Windows 7 running on an Intel® Core™2 Quad Q8300 clocked at 2.5 GHz per core.

The performance factors were identified in a preliminary step and a selection was made on the factors to be used

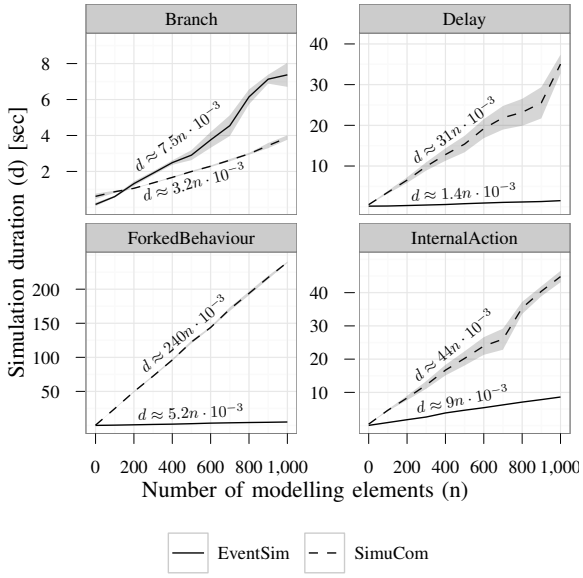

Fig. 4. Rise in simulation duration when increasing the number of modelling elements

in the simulator comparison. The selection process and the associated assumptions are beyond the scope of this paper, but can be found in [20]. The outcome are four types of modelling elements, each describing simulated control flow: `Branches`, `Delays`, `ForkedBehaviours` and `InternalActions`.

B. Results

The experiment results in terms of simulation duration can be seen in Fig. 4. Both simulators scale linearly when the complexity of the simulated system rises, but the rate at which the simulation runtime increases differs to a great amount.

The greatest difference between the two simulators can be observed when the number of `ForkedBehaviours` rises. Adding a further `ForkedBehaviour` to the simulation model slows down SimuCom by around 240 ms compared to a slow-down of just 5 ms observed with EventSim. In SimuCom, each `ForkedBehaviour` is simulated by a dedicated thread whereas a single thread is sufficient for EventSim. Due to the experimental setting, the `ForkedBehaviour` does not encapsulate actions apart from a `Start` and a `Stop` action directly connected to each other. Hence, the increase in simulation time is not due to the encapsulated action chain and we attribute the performance drawback observed with SimuCom to the effort required to create and release threads.

Simulating a `Delay` in EventSim is around 22 times faster compared to SimuCom; `InternalActions` are approximately 5 times faster with EventSim. Both cases are similar in that the simulator is concerned with advancing the simulation time. While `Delays` advance the simulation time by a fixed amount, the time advance caused by `InternalActions` is usually influenced by the utilisation of the requested resource. Each time advance in SimuCom causes a performance overhead due to the synchronisation between concurrent users or their requests, respectively, and the resulting context switches. By contrast, being simulated in the same thread, simulated users and their request need not be synchronised in EventSim and no context switches result from simulating `Delays` and `InternalActions`. This explains the comparatively high performance of EventSim with respect to time-advancing actions.

When simulating a `Branch`, however, SimuCom outperforms EventSim by a factor of two. In SimuCom, branches of the simulated control flow are mapped to branches of the actual control flow of the simulator. Each branch transition is represented by an if-block, where the probability of entering a block is equal to the probability of the respective branch transition. EventSim, by contrast, loads and executes a suitable traversal strategy whenever it encounters a `Branch`. The implementation within this strategy is virtually equivalent to the implementation in SimuCom. Therefore, we attribute the inferior performance in EventSim to the computational effort to delegate the control to the suitable traversal strategy, as well as the effort to construct and return the corresponding traversal instruction.

C. Scalability Limitations

Various limitations in scalability were observed with SimuCom, which is why we compared not only the performance of the simulators, but also their scalability boundaries. The results can be seen in Table I. On the test system described before, we were able to simulate PCM models containing up to approximately 820 `ForkedBehaviours` before SimuCom aborted with a *StackOverflowError*. Likewise, the number of `InternalActions` in SimuCom is limited to around 940 elements, and `Delays` cause a stack overflow when reaching the upper limit of around 1,560 elements. In each of these cases, the overflow is caused by the Xpand template engine. Whether the high stack consumption is caused by erroneous code-generation templates or by the Xpand engine itself has not been assessed.

The number of `Branches` in SimuCom is limited to around 1,250 elements. When this limit is reached, and SimuCom tries to compile the generated Java class files, the Java compiler throws an exception indicating that the maximum size of 64 KB per method has been exceeded. Again, this issue is associated with the code-generation facility in SimuCom.

The workload population in SimuCom is bound by an upper limit of 90,000 simulated users circulating through the system concurrently. Reaching this limit causes an `OutOfMemoryError`. On first glance, this amount of simulated users seems to be more than enough for most application scenarios. But, when the usage or system complexity rises, each simulated user has a higher stack space consumption decreasing the observed limit. Even worse, using a 32-bit JVM usually imposes a 2 GB memory limit, which decreases the user limit further. The observed limit in concurrent system users can be directly attributed to the way in which the process interaction simulation is implemented in SimuCom and, more generally, in the majority of process oriented simulations in Java.

With EventSim, no scalability limitations were observed. Each factor was increased to a number of 100,000 model

Factor	SimuCom	EventSim
Number of `ForkedBehaviours`	$< 820^a$	$> 100,000$
Number of `InternalActions`	$< 940^a$	$> 100,000$
Number of `Delays`	$< 1,560^a$	$> 100,000$
Number of `Branches`	$< 1,250^c$	$> 100,000$
Workload Population	$< 90,000^b$	$> 100,000$

[a] raised a `StackOverflowError`
[b] raised an `OutOfMemoryError`
[c] exceeded the 64 KB method size limit

TABLE I
SCALABILITY LIMITS OF SIMUCOM COMPARED TO EVENTSIM

elements (or simulated users in the case of the workload population) and simulated successfully.

Surprisingly, the limits observed with SimuCom are in contradiction to the experiments shown in Fig. 4, where each factor has been varied in a range between 1 and 1,000 elements without running into scalability issues. This can be explained by the memory footprint of the experiment automation tool and its evolution over time; apparently, the results shown in Fig. 4 have been gathered using a more memory-efficient release of our automation tool. Therefore, the results have to be considered in relative terms.

VII. SUMMARY AND CONCLUSION

In this paper, we presented a novel software performance simulator called EventSim, which accepts architecture-level abstractions of software systems modelled by Palladio component models for predicting performance metrics. While Palladio's reference simulator SimuCom is mainly driven by processes, EventSim relies entirely on events leading to a substantial performance gain compared to SimuCom.

Experiments have shown a decrease in simulation duration by up to 97% when simulating concurrent behaviour induced by control flow forks. By contrast, however, the experiments also revealed a weakness of EventSim with the simulation of control flow branches since their simulation is around twice as fast using SimuCom. Nevertheless, branches in SimuCom usually have a small influence on the overall simulation runtime since they are merely responsible for a small fraction of the simulation duration compared to forks, for instance.

Hence, using EventSim pays off particularly in the presence of a high degree of simulated concurrency, which either occurs at high workloads (leading to many concurrent users) or when the simulated control flow contains relatively many forks (leading to many concurrent requests). In view of the performance and scalability benefits presented in this paper, we plan to extend the capabilities of EventSim to support PCM modelling elements that have not been covered so far. Furthermore, support for multiple replicated experiments in parallel is planned for future versions allowing for a higher degree of utilisation of multi core processors.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable and thorough comments. We appreciate their effort and tried to be just as careful in revising this paper.

REFERENCES

[1] C. U. Smith and L. G. Williams, *Performance solutions : a practical guide to creating responsive, scalable software*, 1st ed. Boston, Mass.: Addison-Wesley, 2002.
[2] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: a survey," *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, May 2004.
[3] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Performance Evaluation*, vol. 67, no. 8, pp. 634–658, 2010, special Issue on Software and Performance.
[4] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, 2009.
[5] S. Becker, "Coupled Model Transformations for QoS Enabled Component-Based Software Design," Ph.D. dissertation, University of Oldenburg, Germany, Mar. 2008.
[6] J. Banks, Ed., *Discrete-event system simulation*, 5th ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2010.
[7] L. Stadler, T. Würthinger, and C. Wimmer, "Efficient coroutines for the Java platform," in *Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java*, ser. PPPJ '10. New York, NY, USA: ACM, 2010, pp. 20–28.
[8] P. Jacobs and A. Verbraeck, "Single-threaded specification of process-interaction formalism in Java," in *Simulation Conference, 2004. Proceedings of the 2004 Winter*, vol. 2, dec. 2004, pp. 1548–1555.
[9] P. L'Ecuyer and E. Buist, "Simulation in Java with SSJ," in *Proceedings of the 37th conference on Winter simulation*, ser. WSC '05. Winter Simulation Conference, 2005, pp. 611–620.
[10] R. Reussner, S. Becker, E. Burger, J. Happe, M. Hauck, A. Koziolek, H. Koziolek, K. Krogmann, and M. Kuperberg, "The Palladio Component Model," Karlsruhe, Tech. Rep., 2011. [Online]. Available: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000022503
[11] 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 (CBSE'06)*, vol. 4063, 2006.
[12] D. Hamlet, D. Mason, and D. Woit, *Component-Based Software Development: Case Studies*, ser. Series on Component-Based Software Development. World Scientific Publishing Company, March 2004, vol. 1, ch. Properties of Software Systems Synthesized from Components.
[13] R. von Massow, "Performance Simulation of Runtime Reconfigurable Software Architectures," Master's thesis, University of Oldenburg, 2010.
[14] A. van Hoorn, M. Rohr, A. Gul, and W. Hasselbring, "An adaptation framework enabling resource-efficient operation of software systems," in *Proceedings of the Warm Up Workshop for ACM/IEEE ICSE 2010*, ser. WUP '09. New York, NY, USA: ACM, 2009, pp. 41–44.
[15] S. Kounev and A. Buchmann, "SimQPN – a tool and methodology for analyzing queueing Petri net models by means of simulation," *Performance Evaluation*, vol. 63, no. 4-5, pp. 364–394, May 2006.
[16] P. Meier, S. Kounev, and H. Koziolek, "Automated Transformation of Palladio Component Models to Queueing Petri Nets," in *19th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2011.
[17] J. Cheesman and J. Daniels, *UML components : a simple process for specifying component-based software*. Boston, Mass.: Addison-Wesley, 2001.
[18] S. Becker, H. Koziolek, and R. H. Reussner, "Model-based Performance Prediction with the Palladio Component Model," in *WOSP '07: Proceedings of the 6th International Workshop on Software and performance*. New York, NY, USA: ACM, February 5–8 2007.
[19] H. Koziolek, S. Becker, and J. Happe, "Predicting the Performance of Component-based Software Architectures with different Usage Profiles," in *Proc. 3rd International Conference on the Quality of Software Architectures (QoSA'07)*, vol. 4880, July 2007, pp. 145–163.
[20] P. Merkle, "Comparing process- and event-oriented software performance simulation," Master's thesis, Karlsruhe Institute of Technology, 2011. [Online]. Available: http://sdqweb.ipd.kit.edu/publications/pdfs/merkle2011a.pdf