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.

1 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], [4], [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 2 briefly revises ProtoCom and its underlying model-2-text transformation and highlights extensions of the improved version. Section 3 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 4. After relating our

- *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*

work to other Palladio Component Model literature and non-PCM works in Section 5, we conclude our paper.

2 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].

2.1 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.

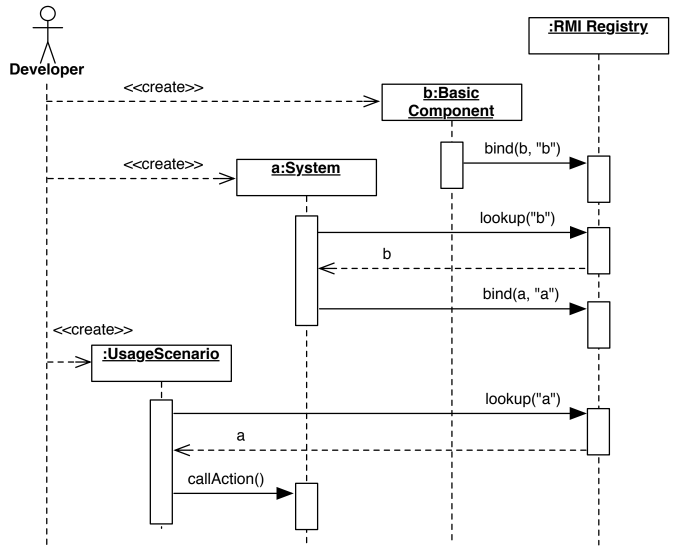

Fig. 1. Sequence Diagram for initialisation and assembly using RMI

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.

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.

2.2 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.

3 Measurement Results on Blade Server

This section describes the measurement on the virtual environment running on Blade servers. Section 3.1 describes the case studies under consideration. Section 3.2 continues with the description of the environment and the process for taking measurements. We discuss issues about the CPU and HDD calibration in Section 3.3. Finally, Section 3.4 briefly presents the results of the measurements.

3.1 Case Studies

We applied ProtoCom in four case studies, each modelling distributed, component-based systems. Fig. 3.1 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 interarrival 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.

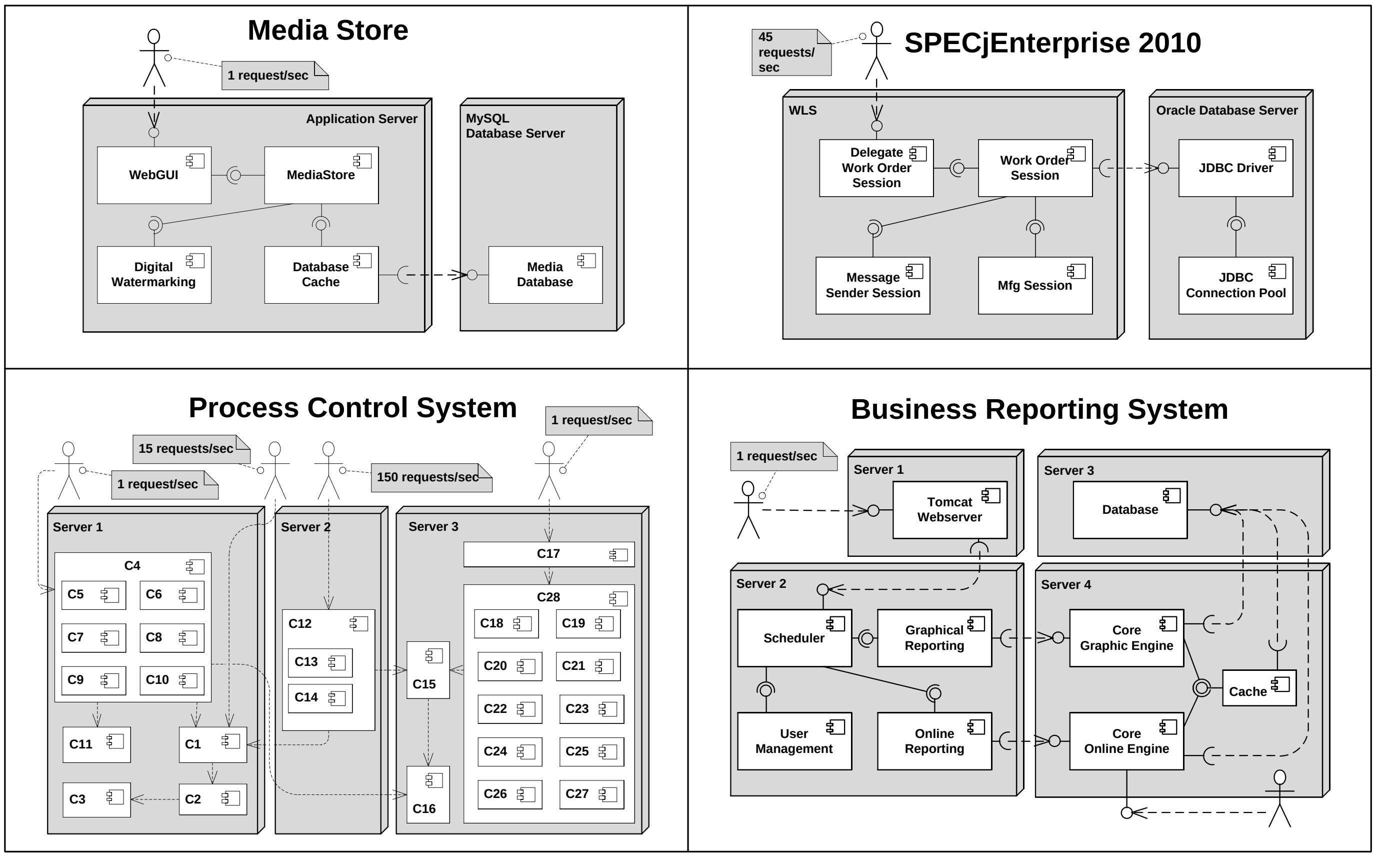

Fig. 2. Case study systems [9]

3.2 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]).

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.

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

3.3 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.

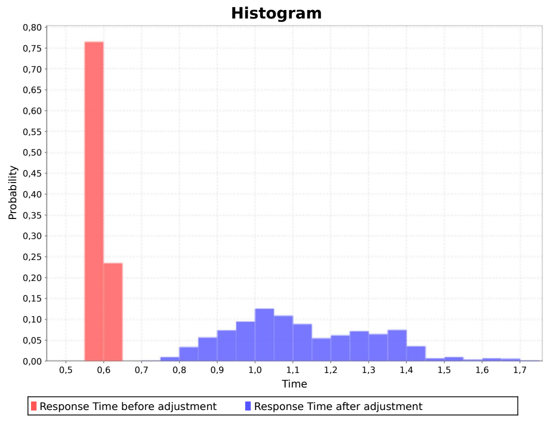

Fig. 3. Results for the simple model for testing HDD calibration

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 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.

3.4 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

4 Discussion

In this section we discuss our process and results with respect to usability, the techniques we used for taking

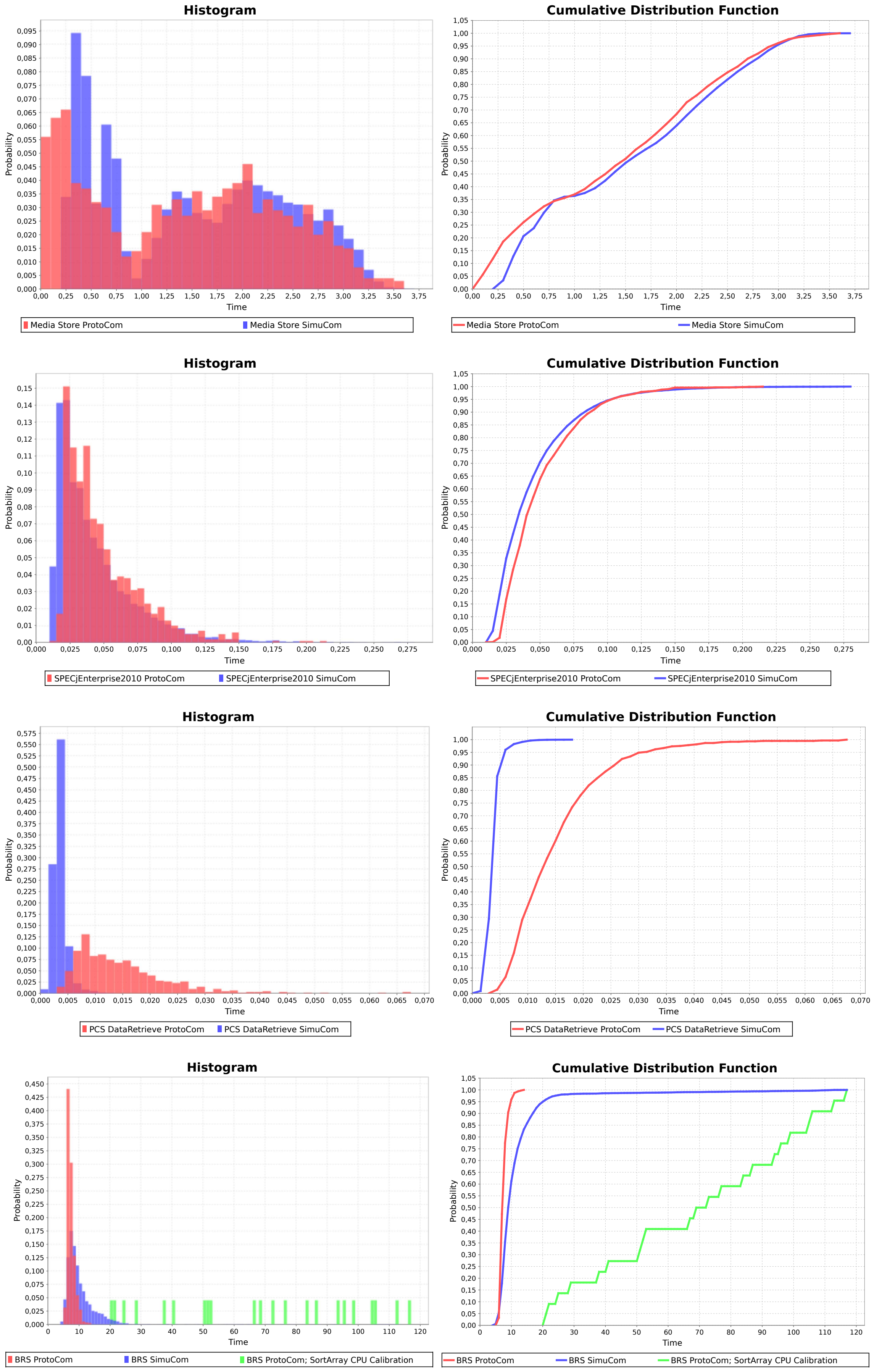

Fig. 4. Results for the case studies

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 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 3.4 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.

5 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.

6 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 Multicore 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.