Using Java EE ProtoCom for SAP HANA Cloud*

Christian Klaussner
Heinz Nixdorf Institute
University of Paderborn
Zukunftsmeile 1
33102 Paderborn, Germany
cfk@mail.uni-paderborn.de

Sebastian Lehrig
Software Engineering Chair
Chemnitz University of Technology
Straße der Nationen 62
09107 Chemnitz, Germany
sebastian.lehrig@informatik.tu-chemnitz.de

Abstract: Performance engineers analyze the performance of software architectures before their actual implementation to resolve performance bottlenecks in early development phases. Performance prototyping is such an approach where software architecture models are transformed to runnable performance prototypes that can provide analysis data for a specific target operation platform. This coupling to the operation platform comprises new challenges for performance prototyping in the context of cloud computing because a variety of different cloud platforms exists. Because the choice of platform impacts performance, this variety of platforms induces the need for prototype transformations that either support several platforms directly or that are easily extensible. However, current performance prototyping approaches are tied to only a small set of concrete platforms and lack an investigation of their extensibility for new platforms, thus rendering performance protoyping ineffective for cloud computing.

To cope with this problem, we extended Palladio's performance prototyping approach ProtoCom by an additional target platform, namely the SAP HANA Cloud, and analyzed its extensibility during the extension process. In this tool paper, we focus on illustrating the capabilities of our extension of ProtoCom. For this illustration, we use a simple example system for which we create a ProtoCom performance prototype. We particularly run this prototype in the SAP HANA Cloud, thereby showing that our extension can efficiently be applied within a practical context.

1 Introduction

Performance engineers analyze the performance of software architectures before their actual implementation to resolve performance bottlenecks in early development phases. This early detection and resolving of performance problems reduces fix-it-later costs and promises a high customer satisfaction, due to a high quality of service right from the beginning of system operation. Performance prototyping is such an approach to performance engineering. In performance prototyping, engineers transform software architecture models to runnable performance prototypes. These prototypes can provide analysis data for a specific target operation platform.

The coupling of performance prototypes to the operation platform comprises new challenges in the context of cloud computing. In cloud computing, a variety of different cloud

*The research leading to these results has received funding from the EU Seventh Framework Programme (FP7/2007-2013) under grant no 317704 (CloudScale).

platforms exists. Because the choice of platform impacts performance, this variety of platforms needs to be covered by transformations to performance prototypes [LLK13]. Therefore, prototype transformations either need to support several platforms directly or have to be easily extensible. However, current performance prototyping approaches are tied to only a small set of concrete platforms and lack an investigation of their extensibility for new platforms, thus rendering performance prototyping ineffective for cloud computing.

In related work, initial attempts have been made to support a larger quantity of platforms [Bec08, GL13]. However, such attempts remain on a conceptual level and do not provide transformation implementations that would make performance prototyping more efficient. Other works provide such implementations but are tied to single platforms only. For example, the implementation by Lehrig and Zolynski [LZ11] only has support for Java SE as a target platform. To the best of our knowledge, no related approach investigates the extensibility of performance prototypes.

To cope with this problem, we extended Palladio's Java SE performance prototyping approach ProtoCom [Bec08] by an additional target platform and analyzed its extensibility during the extension process. We decided to enrich ProtoCom by Java EE capabilities such that we could reuse first conceptual ideas of our previous work [GL13]. Moreover, we use the SAP HANA Cloud as concrete target platform because it represents a practically used cloud computing environment with Java EE support.

The contribution of this tool paper is an illustration of the capabilities of our ProtoCom extension. For this illustration, we use a simple example system for which we create a Java EE ProtoCom performance prototype. We particularly run this prototype in the SAP HANA Cloud, thereby showing that our extension can efficiently be applied within a practical context. For our results regarding the extensibility of ProtoCom, we refer to Klaussner's Bachelor's thesis [Kla14] (we identify both, implementation parts that are easily extensible and parts that can be improved regarding extensibility). His thesis particularly gives a complete technical overview of our ProtoCom extension.

This paper is structured as follows. We introduce our example system in Sec. 2. We use this system throughout our paper as running example. In Sec. 3, we describe the fundamentals of our work (performance prototyping with ProtoCom, Java EE, and the SAP HANA Cloud). Afterwards, we describe our Java EE extension to ProtoCom in Sec. 4. This extension allows us to use ProtoCom within SAP HANA Cloud in Sec. 5. In Sec. 6, we discuss related work in the area of performance prototyping. We close this paper by giving concluding remarks and an outlook on future work in Sec. 7.

2 Running Example: The Alice&Bob System

As running example, we use the *Alice&Bob* system as illustrated in Fig. 1. This system consists of two Java EE servers: JavaEE-Server-A allocates the Alice component that provides the interface IAlice with the operation callBob() and JavaEE-Server-B allocates the Bob component that provides the interface IBob with the operation sayHello(). The IAlice interface is provided to a user who can invoke its operation through a client-side technol-

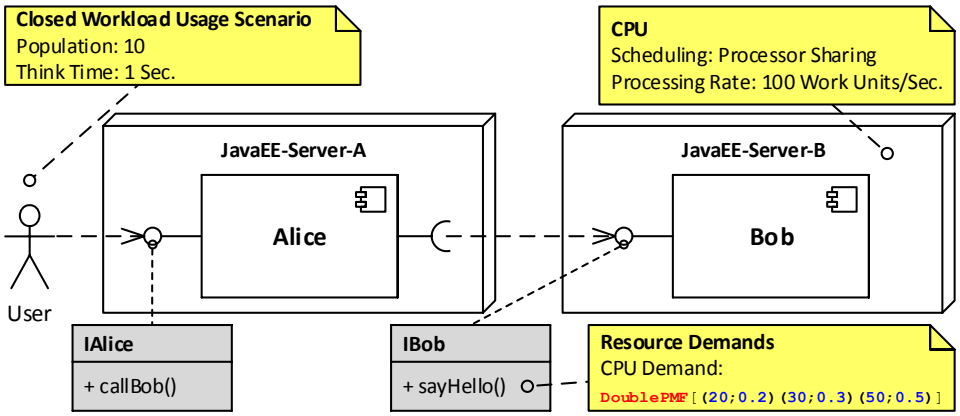

Figure 1: Alice&Bob System

ogy like a browser. This invocation can be received by a component on the server side – in our case by the Alice component implementing the IAlice interface.

Furthermore, we annotated performance-relevant information to the *Alice&Bob* system using yellow sticky notes. These sticky notes state that (1) the user population is 10 with a think time of 1 second within a closed workload, (2) the CPU of JavaEE-Server-B follows a processor sharing (round-robin) scheduling strategy while processing with 100 work units per seconds, and (3) calls to sayHello() cause a CPU demand as specified by the given probability mass function. The latter specifies that 20 CPU work units are demanded in 20% of the cases, 30 CPU work units are demanded in 30% of the cases, and 50 CPU work units are demanded in 50% of the cases.

3 Fundamentals

In this section, we describe the fundamentals needed to understand our extension to the performance prototyping approach ProtoCom. We accordingly describe the state-of-the art in performance prototyping with ProtoCom in Sec. 3.1. Afterwards, we describe the target platform for our extension – Java EE and the SAP HANA Cloud – in Sec. 3.2.

3.1 Performance Prototyping with ProtoCom

Software engineers and architects aim to analyze quality attributes of a software system, e.g., performance, early in the design process in order to avoid costs for subsequent adjustments. Performance prototyping is an approach that facilitates such analyses by simulating the performance of a software system in a realistic execution environment and under different load levels.

Palladio supports engineers in efficiently constructing such performance prototypes. For this support, the Palladio Component Model (PCM) provides a component-based Architecture Description Language (ADL). The PCM allows engineers to create a formalized model of the components and performance-relevant properties of a software architecture, similar to the model of our Alice&Bob system illustrated in Fig. 1.

After such a PCM model is created by performance engineers, it serves as input for the code generator ProtoCom, which transforms it into a runnable performance prototype for the desired target technology. Currently, ProtoCom supports the generation of performance prototypes for three target technologies: Java SE with RMI, Java EE with EJBs (Enterprise Java Beans), and Java EE with Servlets for the SAP HANA Cloud, which we introduce in this paper.

Our implementation transforms components from the model to Java EE Servlets. These Servlets can communicate via HTTP (cf. Sec. 4), while all previous implementations were based on RMI communication for distributing components. When running a generated performance prototype, an external HTTP load generator, e.g., JMeter, is used to simulate users interacting with the system according to the usage scenario specified in the model. Meanwhile, the performance prototype takes several mesasurements, e.g., response times, which allow a subsequent examination of the software architecture's performance. We give a more detailed description of this workflow in Sec. 5.

3.2 Java EE and SAP HANA Cloud

The Java Enterprise Edition (Java EE) is a set of specifications and APIs for Java that facilitate the development of enterprise software. This type of software is usually run on one or more servers (as illustrated in Fig. 1) and makes use of web technologies. There exist several implementations of the Java EE specifications, e.g., GlassFish and WildFly (formerly JBoss). Additionally, some projects implement only a subset of the specifications. For example, Apache Tomcat[1] implements Servlets, which are Java classes that can respond to requests. A common use case for Servlets is the HTTP request-response model for typical websites.

The SAP HANA Cloud is a Platform-as-a-Service (PaaS) that provides a Java EE environment and a cloud infrastructure for running enterprise applications. It is based on Apache Tomcat and includes additional services for applications, e.g., a document service that can be used to store unstructured or semi-structured data.

4 Java EE ProtoCom

Adding Java EE for the SAP HANA Cloud as target technology for ProtoCom required the implementation of new transformations and a new framework (the ProtoCom "runtime").

[1] `http://tomcat.apache.org`

When generating code from PCM instances, the entities and concepts of the source model have to be mapped to constructs of the target language. Although both the PCM and Java have similar constructs, e.g., interfaces and components/classes, they differ in expressiveness. Therefore, a one-to-one mapping is sometimes impossible. Tab. 1 lists the PCM entities and concepts regarded in our extension together with their respective Java mapping. Especially the provided and required roles of components need specific patterns to be transformed correctly [Kla14].

PCM Entity/Concept	**Java**
Interface	Interface
Component	Component class
Provided role	Port class
Required role	Context class
System	System class
Assembly context	Component class instance
Call action	RPC over HTTP
Control flow	Control flow
Resource environment	Resource environment class
Allocation	Allocation class
Usage scenario	[External]

Table 1: Mapping of PCM entities and concepts to Java, cf. [GL13, Kla14]

The addititions we made can be grouped intro three categories: inter-component communication, user interface, and load generator.

4.1 Inter-Component Communication

For the communication between components across resource container boundaries (i.e., ExternalCallAction entities in PCM), we use a custom, lightweight RPC protocol based on JSON and HTTP. Compared with other RPC protocols like SOAP, our custom protocol is easier to process in intermediary components and tools involved in the execution of the performance prototype. For example, when using Apache JMeter for the simulation of usage scenarios, data has to be exchanged between JMeter and the performance prototype. Thanks to JMeter's scripting capabilities (including JavaScript) the data received from the performance prototype can be processed without any manual parsing as would be the case with SOAP and XML.

Conceptually, our RPC method works similar to Java RMI. Components register themselves with a unique name at a central registry that is accessible via a Servlet. Other components can then contact this registry and obtain references to other named components. After a connection is established, method calls to remote components are initiated by sending the method name, parameter types, and arguments (serialized to JSON) to the target component, as shown in List. 1.

The formalParameters array consists of the type names of the formal parameters and is used – together with the name – to find the correct method to invoke at the destination, whereas the actualParameters array specifies the actual type names of the serialized arguments. These type names are required during deserialization in order to recreate the serialized objects. For example, the argument in List. 1 is deserialized as StackContext object and passed to the callBob0 method which expects a StackContext argument. In this case, the serialized argument consists of an empty JSON object, instructing the deserializer to create a new StackContext object to be passed to the method.

```
{
  "name":"callBob0",
  "formalTypes":["de.uka.ipd.sdq.simucomframework.variables.StackContext"],
  "actualTypes":["de.uka.ipd.sdq.simucomframework.variables.StackContext"],
  "arguments":[{}]
}
```

Listing 1: RPC protocol representation of a call to Alice's callBob method

4.2 User Interface

Since the Java EE performance prototypes run on the SAP HANA Cloud, they are inaccessible through a console. Therefore, we developed an HTML user interface that can be accessed through a web browser. Fig. 2 shows a screenshot of the user interface used to operate the performance prototype generated from the *Alice&Bob* system introduced in Sec. 2. It allows performance engineers to specify the location of the central component registry and to start particular modules, i.e., startable entities of the performance prototype like resource containers (that start all allocated components) and systems. Additionally, it provides downloads for the transformed usage scenarios (JMeter files) and analysis results.

4.3 Load Generator

In order to generate load on the performance prototype, we provide an interface for external load generators, e.g., Apache JMeter. The ProtoCom transformations automatically generate a JMeter test plan for each usage scenario in the model. ExternalCallActions are realized by sending HTTP requests to the respective components according to the RPC protocol described in Sec. 4.1.

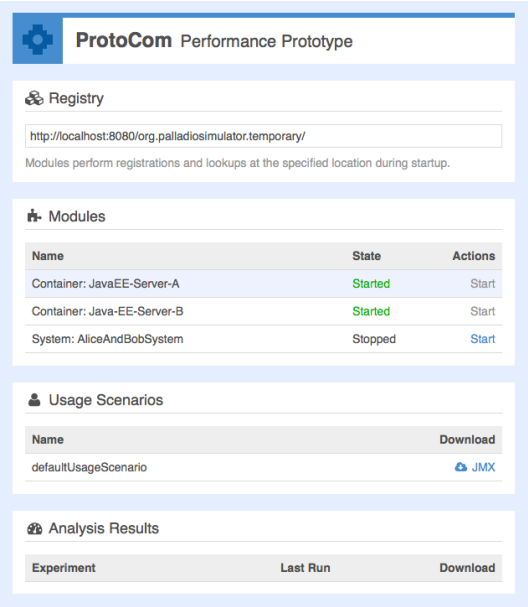

Figure 2: The user interface of the performance prototype

5 Using Java EE ProtoCom for SAP HANA Cloud

This section describes how ProtoCom is used in the SAP HANA Cloud. First, it explains how the abstract resource demands from the PCM are translated to actual demands on the hardware of the cloud platform. Afterwards, it shows the workflow of transforming and executing performance prototypes.

5.1 Hardware Calibration

Before running a performance prototype, the abstract resource demands for CPU and HDD specified in the SEFFs of the PCM components have to be translated to demands for the hardware that the performance prototype is running on. For that purpose, ProtoCom provides a set of calibration strategies, e.g., Fibonacci, which simulates CPU intensive tasks with minimized RAM access [LZ11]. The calibration computes several iterations – in this case the n^{th} Fibonacci numbers – and measures the time required for these computations. The results are stored in a calibration table for later lookup, as shown in Tab. 2. When a demand from the model has to be translated to a real hardware demand, the calibration table is used to find the iteration count n for the given demand. For example, a CPU demand of 0.036 seconds would be achieved by computing the 40^{th} Fibonacci number.

[2]Results taken from `http://www.cs.utsa.edu/~wagner/CS3343/recursion/fibrecurs2.html`

n	Time
10	0.032 Sec.
20	0.033 Sec.
30	0.035 Sec.
40	0.036 Sec.
50	0.038 Sec.
...	...

Table 2: A calibration table for the Fibonacci strategy[2]

Lehrig and Zolynski [LZ11] validate such calibration tables by creating a model with a processing resource that has a processing rate of 1000 units and a closed workload usage scenario in which a single user repeatedly invokes a task that consumes 1000 units. Ideally, the response time of this usage scenario should consume a time of 1 second. The results for the Large Chunks HDD calibration show that the mean response time of the usage scenario is indeed around 1 second, with deviations resulting from external factors, e.g., process scheduling behavior controlled by the operating system.

The calibration of the hardware resources is a time-consuming task and has to be performed once for each hardware configuration. Regarding the SAP HANA Cloud, all calibration strategies of ProtoCom work for both local installations (development and testing) and the actual cloud platform.

5.2 Prototyping Workflow for SAP HANA Cloud

This section describes the typical workflow of performance prototyping with ProtoCom for SAP HANA Cloud. The activity diagram in Fig. 3 shows all steps involved.

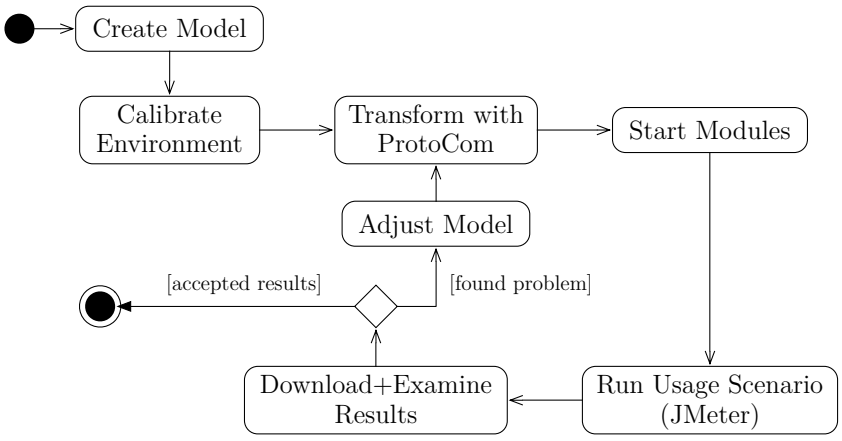

Figure 3: Performance prototyping workflow with ProtoCom for SAP HANA Cloud

Performance engineers start with creating a model of the software architecture to examine. Afterwards, they calibrate the environment that the performance prototype will run on. This can either be the environment of the cloud infrastructure or a local installation of the SAP HANA Cloud runtime. Subseqently, the model is transformed to a performance prototype and deployed on the target platform. At this point, performance engineers interact with the user interface illustrated in Fig. 2. They start the modules, i.e., the resource containers and the system, and download the generated JMeter test plan. Loading this test plan into JMeter allows them to configure and run the measurements.

Finally, the results of the analysis are provided as a download in the user interface. Performance engineers examine these results in the Palladio-Bench and decide if the model needs further improvements, e.g., caused by a performance bottleneck, or if the results are satisfying. In this case, software engineers can start with the implementation of the architecture.

Following this workflow for our Alice&Bob system shows that the performance prototype generated from the model in Fig. 1 can be deployed in the SAP HANA Cloud to take performance measurements. Evaluating the results of the prototype and comparing them to the results of ProtoCom for Java SE with RMI is part of our future work.

6 Related Work

Besides works on ProtoCom, there are only a few approaches on performance prototyping. Becker et al. [BDH08] describe these approaches and their limitations compared to ProtoCom in their related work. In this section, we therefore only focus on related work on ProtoCom directly.

In his PhD thesis, Becker [Bec08] provides the groundwork for ProtoCom and for transforming PCM instances to performance prototypes using Java EE EJBs (Enterprise Java Beans). However, the performance prototypes generated by these transformations suffer from several usability issues. For example, the generated code requires manual adjustments after the transformations. Furthermore, the deployment process of the generated performance prototypes proved to be inefficient.

Building on this work, Lehrig and Zolynski [LZ11] present an improved version of ProtoCom that aims to resolve these shortcomings. They extend the transformations and the framework of ProtoCom such that the generated performance prototypes target the Java SE technology and use Java RMI for inter-process communication. However, the transformations in this version of ProtoCom proved to be slow and inextensible due to the use of Xpand[3] templates as a means for model-to-text transformations. Therefore, Lehrig and Zolynski provide a reimplemented version of ProtoCom ("ProtoCom 3") [Kar] that replaces the Xpand templates with templates for the programming language Xtend 2[4]. This version of ProtoCom served us as a good basis for analyzing the extensibility of ProtoCom. As Klaussner describes in his Bachelor's thesis [Kla14], the new transformation parts in-

[3] `http://www.eclipse.org/modeling/m2t/?project=xpand`
[4] `http://www.eclipse.org/xtend`

deed provide a good extensibility. However, some parts of ProtoCom's generic framework still provide room for improvement regarding extensibility [Kla14].

Since the release of Becker's PhD thesis, a new version of Java EE EJBs was published, providing features that simplify the transformations. Furthermore, there is a need for performance prototype transformations targeting multiple platforms [LLK13]. These developments led to the addition of new Java EE EJB transformations by Giacinto and Lehrig [GL13]. However, these transformations are provided only on a conceptual level and also lack an implementation. In our work, we provide such a Java EE implementation and use it within the SAP HANA Cloud, thus making our approach more efficient.

7 Conclusions

In this tool paper, we introduce a novel ProtoCom version that is capable of automatically generating Java EE performance prototypes that can directly operate within the SAP HANA Cloud (out-of-the-box). In this context, we give an overview of novel Java EE features in ProtoCom and describe how generated performance prototypes are used within the SAP HANA Cloud.

Performance engineers can now efficiently generate performance prototypes based on Java EE. Engineers achieve best efficiency within the SAP HANA Cloud because our ProtoCom version is capable of using dedicated SAP HANA Cloud features such as its document service (e.g., to store calibration tables). However, engineers can now also analyze other Java EE platforms (Glassfish, Tomcat, etc.) more efficiently because our ProtoCom version introduces features shared among all of such platforms (e.g., RPC over HTTP communication). Here, we profit from the standardization process behind Java EE. Furthermore, we showed that engineers can easily extend ProtoCom if new requirements arise [Kla14].

In future work, we want to conduct several case studies within the SAP HANA Cloud using our novel ProtoCom version. We plan to reuse the case studies from our earlier work with ProtoCom in virtualized environments [LZ11]. This reuse will allow us to compare our previous results with new results gained within a less-controlled environment (the SAP HANA Cloud), eventually leading to assessing the predictability of cloud computing environments.

References

[BDH08] Steffen Becker, Tobias Dencker, and Jens Happe. Model-Driven Generation of Performance Prototypes. In Samuel Kounev, Ian Gorton, and Kai Sachs, editors, *Performance Evaluation: Metrics, Models and Benchmarks*, volume 5119 of *Lecture Notes in Computer Science*, pages 79–98. Springer Berlin Heidelberg, 2008.

[Bec08] Steffen Becker. *Coupled Model Transformations for QoS Enabled Component-Based Software Design*. PhD thesis, University of Oldenburg, Germany, January 2008.

[GL13] Daria Giacinto and Sebastian Lehrig. Towards Integrating Java EE into ProtoCom. In *KPDAYS*, pages 69–78, 2013.

[Kar] Karlsruhe Institute of Technology. ProtoCom - SDQ Wiki. `http://sdqweb.ipd.kit.edu/wiki/ProtoCom`. Retrieved: 07/12/2014.

[Kla14] Christian Klaussner. Extensible Performance Prototype Transformations for Multiple Platforms. Bachelor thesis, Software Engineering Group, University of Paderborn, Software Engineering Group, Paderborn, Germany, July 2014.

[LLK13] Michael Langhammer, Sebastian Lehrig, and Max E. Kramer. Reuse and Configuration for Code Generating Architectural Refinement Transformations. In *VAO '13*. ACM, 2013.

[LZ11] Sebastian Lehrig and Thomas Zolynski. Performance Prototyping with ProtoCom in a Virtualised Environment: A Case Study. In *Proceedings to Palladio Days 2011, 17-18 November 2011, FZI Forschungszentrum Informatik, Karlsruhe, Germany*, November 2011.