Approaching the Cloud: Using Palladio for Scalability, Elasticity, and Efficiency Analyses*†

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

Matthias Becker
Heinz Nixdorf Institute
University of Paderborn
Zukunftsmeile 1
33102 Paderborn, Germany
matthias.becker@upb.de

Abstract: In cloud computing, software architects develop systems for virtually unlimited resources that cloud providers account on a pay-per-use basis. Elasticity management systems provision these resource autonomously to deal with changing workloads. Such changing workloads call for new objective metrics allowing architects to quantify quality properties like scalability, elasticity, and efficiency, e.g., for software design analysis. However, analysis approaches such as Palladio so far did not support these novel metrics, thus rendering such analyzes inefficient.

To tackle this problem, we (1) extended Palladio's simulation approach SimuLizar by additional metrics for scalability, elasticity, and efficiency and (2) integrated the Architectural Template language into Palladio allowing architects to model cloud computing environments efficiently. A novel analysis process guides software architects through these new capabilities. In this paper, we focus on illustrating this new process by analyzing a simple, self-adaptive system.

1 Introduction

In cloud computing, software architects develop applications on top of compute environments being offered by cloud providers. For these applications, the amount of offered resources is virtually unlimited while elasticity management systems provision resources autonomously to deal with changing workloads. Furthermore, providers bill provisioned resources on a per-use basis [AFG+10]. As a consequence of these characteristics, architects want their applications to use as few resources as possible in order to save money while still maintaining the quality requirements of the system. Quality properties that focus directly on these aspects are scalability, elasticity, and efficiency [BLB15, HKR13].

These quality properties need to be quantified for requirements engineering and software design analysis by means of suitable metrics. For instance, cloud consumers and cloud

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

†This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

Proc. SOSP 2014, Nov. 26–28, 2014, Stuttgart, Germany

providers need to negotiate service level objectives (SLOs), i.e., metrics and associated thresholds [EMP13]. Such SLOs have to consider characteristics like changing workloads ("how fast can an application adapt to a higher workload?") and pay-per-use pricing ("how expensive is serving an additional consumer?"). However, architecture-level analysis approaches so far did not support these novel metrics. Therefore, software architects currently cannot efficiently analyze their designs at the architectural level and potentially have to implement and try all reasonable design variants for making informed design decisions. Such an approach leads to high effort and high development costs in the end.

In related work, design-time engineering methods for analyzing performance properties exist. These approaches currently have a limited support for scalability, elasticity, and efficiency analyses. Scalability analyses (e.g., Palladio [BKR09]) are semi-automated, i.e., based on a series of manually conducted and interpreted performance analyses. Elasticity analyses (e.g., Palladio's simulation approach SimuLizar [BBM13]) allow to model and analyze self-adaptations, typically used by cloud computing environments, but have a high modeling effort. Efficiency analyses (e.g., CDOSim [FFH12]) require an implemented SaaS application to determine the most cost-efficient cloud computing environment but are limited to IaaS environments and lack support for early design-time analyses.

To tackle these problems, we (1) extended SimuLizar [BBM13] by additional metrics for scalability, elasticity, and efficiency and (2) integrated the Architectural Template language [Leh13] into Palladio allowing architects to model cloud computing environments efficiently. A novel analysis process guides software architects through these new capabilities.

The contribution of this paper is a tool-based illustration of this new process. For our illustration, we analyze a simple, self-adaptive system.

This paper is structured as follows. We introduce the simple, self-adaptive system we use as a running example in Sec. 2. In Sec. 3, we overview our novel process for analyzing cloud computing systems regarding scalability, elasticity, and efficiency. Afterwards, we illustrate this process and accompanying tools by applying our running example to this process (Sec. 4). We conclude the paper with a summary and an outlook on future work in Sec. 5.

2 Running Example: A Simplified Online Book Shop

As an example scenario, we consider a simplified online book shop. An enterprise assigns a software architect to design this shop, given the following requirements:

R_{fct}**: Functionality** In the shop, customers shall be able to browse and order books.

R_{scale}**: Scalability** The enterprise expects an initial customer arrival rate of 100 customers per minute. It further expects that this rate will grow by 12% in the first year, i.e., increase to 112 customers per minute. In the long run, the shop shall therefore be able to handle this increased load without violating other requirements.

R_{elast}**: Elasticity** The enterprise expects that the context for the book shop repeatedly

changes over time. For example, it expects that books sell better around Christmas while they sell worse around the holiday season in summer. Therefore, the system shall proactively adapt to anticipated changes of the context, i.e., maintain a response time of 3 seconds or less as well as possible. For non-anticipated changes of the context, e.g., peak workloads, the system shall re-establish a response time of 3 seconds or less within 10 minutes once a requirement violation is detected.

R_{eff}**: Efficiency** The costs for operating the book shop shall only increase (decrease) by \$0.01 per hour when the number of customers concurrently using the shop increases (decreases) by 1. In other words, the marginal cost of the enterprise for serving an additional customer shall be \$0.01.

Requirements R_{scale}, R_{elast}, and R_{eff} are typical reasons to operate a system in an elastic cloud computing environment [HKR13], i.e., an environment that autonomously provisions the required amount of resources to cope with contextual changes. Thus, the software architect designs the shop as a 3-layer Software as a Service (SaaS) application operating in a rented Infrastructure as a Service (IaaS) cloud computing environment that provides replicable virtual servers (see Fig. 1). The three layers involve the typical layers of web applications: presentation, application, and data layer.

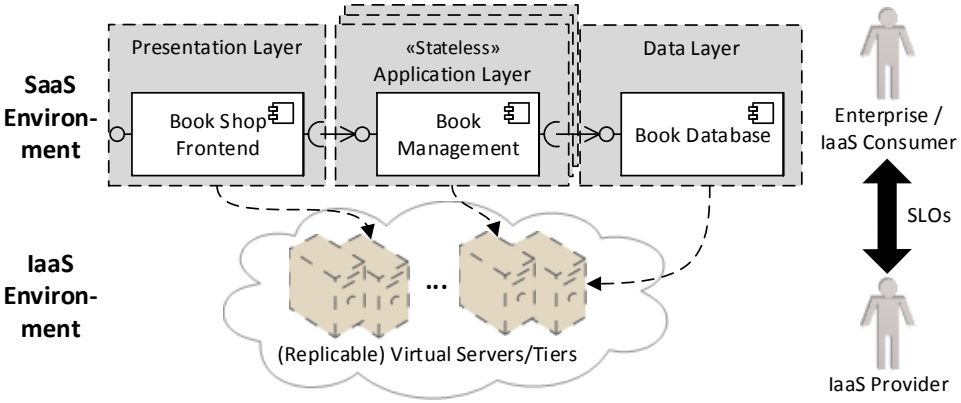

Figure 1: Overview of the simplified online book shop.

The architect investigates possible variants of the 3-layer architectural style. First, the architect considers a 3-layer/3-tier architecture [TMD09]. In a 3-layer/3-tier architecture, the three layers of a 3-layer architecture (presentation, application, and data) are allocated to three different tiers. In an elastic IaaS environment, these tiers are represented by different replicable virtual servers. Second, the architect considers the SPOSAD architectural style [Koz11], a 3-layer variant with an application layer that can safely be replicated to foster scalability. The SaaS middle layer has to be stateless to achieve this safe replication.

Now, the software architect would like to know whether the planned online shop should be realized according to (a) a 3-layer/3-tier architecture, (b) the SPOSAD architectural style, or (c) neither of the two. The architect wants to decide based on whether the scalability (R_{scale}), elasticity (R_{elast}), and efficiency (R_{eff}) requirements will be met by the

finally implemented application. In other words, the architect would like to conduct an architecture-level what-if analysis for the available options to make an informed decision.

However, as current analysis tools do not support metrics for scalability, elasticity, and efficiency in the context of cloud computing. Therefore, the architect cannot efficiently analyze the book shop at the architectural level and, therefore, potentially has to implement and try all considered variants (leading to high effort and high costs).

3 Process

In this section, we propose a novel high-level process for modeling and analyzing cloud computing based systems. Our process supports architects in conducting architecture-level analyses for scalability, elasticity, and efficiency. Moreover, we integrated a set of analysis tools into our process for making such analyzes highly efficient: Palladio [BKR09], Architectural Templates [Leh13], and SimuLizar [BBM13].

Architectural Templates and SimuLizar are extensions to the Palladio tool suite. Architectural Templates (ATs) are a mean to express architectural blueprint that architects can efficiently use and refined for a broad range of software systems. ATs can be specified and reused for various architectural styles, like "3-layer/3-tier" and "SPOSAD", and can be analyzed with SimuLizar. SimuLizar provides modeling capabilities for self-adaptive systems, i.e., different views for a system and its reconfiguration, as well as a simulation-based scalability-, elasticity-, and efficiency-analysis.

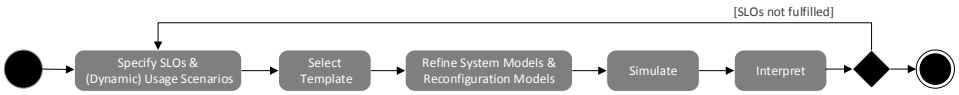

Figure 2: High-level process for applying Palladio in cloud computing scenarios.

Figure 2 illustrates our high-level process. The process starts with specifying service-level objectives (SLOs) and (dynamic) usage scenarios that both formalize the requirements of the system to be developed (e.g., the ones of our example scenario in Sec. 2). Next, the software architect has to select an appropriate Architectural Template. The Architectural Template has then to be further refined by application-specific parts, i.e, interfaces, components, and service effect specifications have to model the target systems structure and behavior. Reconfiguration rules, defined by the Architectural Template, can optionally be refined as well. To validate whether the SLOs for the target system are met, the system can be simulated with SimuLizar. The result of the simulation is a set of measurements for different metrics as defined in [BLB15]. The software architect has now to interpret these measurements and check whether the modeled target system fulfills the SLOs. If not, the process continues with the first step to incrementally refine the SLOs and the system until SLOs can be met.

4 Application

In this section, we apply the book shop scenario of Sec. 2 to our novel process as described in Sec. 3. This application serves as a first proof-of-concept evaluation. We describe each process step in a separate subsection.

4.1 Specify SLOs & (Dynamic) Usage Scenarios

Based on the requirements of the book shop scenario, we derive and specify SLOs and (dynamic) usage scenarios.

For the specification of SLOs, we use a dedicated SLO language of our tool suite. Our language allows to organize SLO specifications in repositories as shown in Fig. 3. For example, we derived a "3 Seconds Response Time SLO" based on R_{elast} where the requirement is checked against a threshold of three seconds. Our SLO of Fig. 3 makes this threshold explicit as can be seen within the properties view. The simulation can later-on make use of such information.

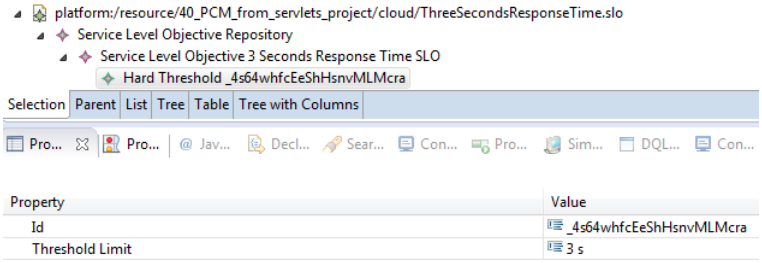

Figure 3: Example SLO specification as derived from R_{elast}.

For the specification of dynamic usage scenarios, we integrated LIMBO [vKHK14], a tool for load intensity modeling, into our tool suite. Before our integration, usage scenarios had to be specified in a static manner, i.e., could not vary over time. Therefore, scenarios like we described for the book shop scenario could not be realized, e.g., to vary load around Christmas. Because LIMBO allows to model such time-dependent changes in workloads, we extended our simulation such that LIMBO's dynamic changes in workloads can be applied on Palladio's usage scenarios.

For example, we modeled a change of arrival rates over one year for the book shop scenario as shown in Fig. 4. The arrival rates generally increase from 100 users per second to 112 users per second as that is the expected trend for one year. We additionally modeled a peak around Christmas as well as lower arrival rates around the holiday season in summer. Finally, we added some noise to reflect a realistic use of web applications. Our extended simulation later-on follows this specification for the arrival rate.

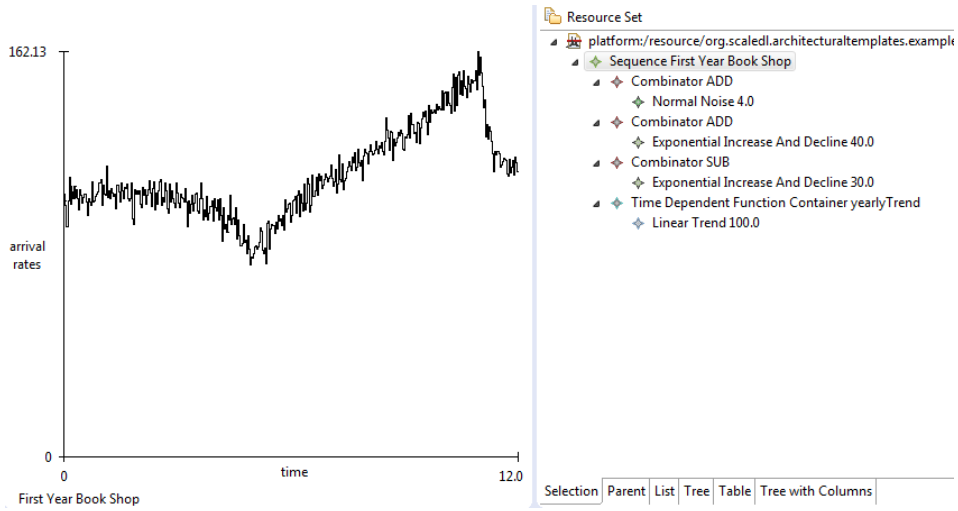

Figure 4: Example LIMBO specification as derived from R_{scale} and R_{elast}.

4.2 Select Template

The template-based design process starts with selecting an Architectural Template (AT) from a repository of available ATs. We created such a repository for the CloudScale project[1].

The architect of the book shop scenario directly models the system in two variants, as a 3-layer/3-tier architecture and as a SPOSAD architecture. For each variant, the architect selects an appropriate AT from the repository. As an example, the AT specification for SPOSAD is illustrated in Fig. 5. This AT specifies the different roles of SPOSAD (SPOSAD itself; presentation, middle, data layers) and a completion. The latter is a transformation able to weave application-independent SPOSAD information into the model of the book shop. For example, the replication logic in the form of adaptation rules is application-independent and can, thus, be included in the completion of the AT. ATs particularly provide the means to parametrize such completions, e.g., by the concrete condition when replication shall be triggered.

Having decided to choose a particular AT for system design, architects can apply that AT to their system. In Fig. 6, we illustrate such an application for the SPOSAD AT. The figure shows the PCM Profiles view; a special view for listing applications of Palladio's profile mechanism. Each AT role can be applied via this mechanisms, thus, allowing the architect to assign the different AT roles to the corresponding entities. Accordingly, he has top assign the SPOSAD role to the system itself and each layer role to the corresponding component assembly within the system. Creating and refining such component assemblies is the task of the next process step.

[1]The repository is available at `github.com/CloudScale-Project`; the process to engineer ATs is described by Lehrig [Leh13].

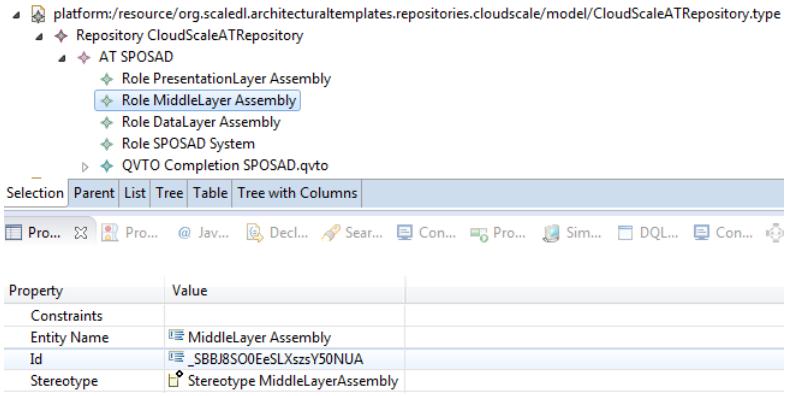

Figure 5: Example AT for SPOSAD. The AT includes a role for the system and three roles for assigning the layers of a 3-layer architecture to component assemblies. The SPOSAD completion weaves application-independent SPOSAD information into the model.

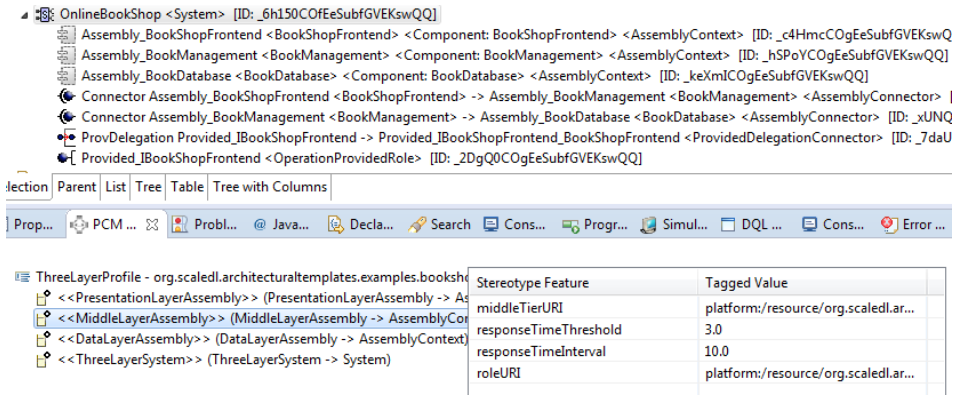

Figure 6: Example application of the SPOSAD AT. Each AT role is assigned to corresponding PCM entities using its profile mechanisms. The middle layer assembly profile allows for specifying response time thresholds and intervals for triggering self-adaptations. In the example, response time is averaged in intervals of 10 seconds and checked against a 3 seconds threshold.

4.3 Refine System Models & Reconfiguration Models

Now that the software architect has configured two system models to make use of ATs (3-layer/3-tier AT and SPOSAD AT), the architect has to assign all AT roles to application-dependent component assemblies. For the presentation and data layer, the architect can assemble the same repository components for both design variants. The two variants, 3-layer/3-tier AT and SPOSAD AT, do not constrain such components differently. For the application layer, the architect has to assemble a stateless variant of the book management component to the system if following the SPOSAD AT. Here, the architect has to follow the constraints of the SPOSAD architectural style. For the case of the 3-layer/3-tier AT, no such constraints limit the design of the book management component. Both variants, stateless and statefull, are allowed here.

Fig. 7 illustrates the system model as designed by the architect. The system model is similar for both variants; only the encapsulating component of the book management assembly may be different as explained above. Having these assemblies available, the architect can assign the remaining AT roles to these. This assignment is shown in Fig. 6 (note the three profile applications for each layer).

For the application layer of the SPOSAD AT, the architect has an additional option to parametrize the adaptation rules used for replication. As illustrated in Fig. 6 (bottom, right), the application layer profile provides tagged values for specifying replication threshold and observation intervals.

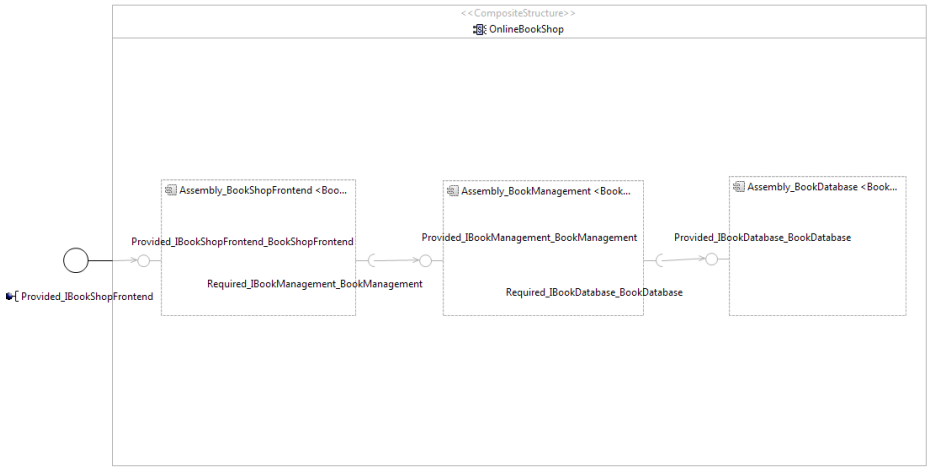

Figure 7: The system model of the book shop scenario

4.4 Simulate

For our simulation, we reimplemented and improved the Experiment Automation framework, initially developed by Merkle [Mer11]. Experiment Automation is now able to start several SimuLizar [BBM13] simulations, each time varying model parameters in order to measure our novel metrics. For instance, we measure user capacity (prerequisite for our scalability range metric, cf. [BLB15]) by varying the number of users within the system over several simulation runs. For details about Experiment Automation, we refer to our developer guide[2].

We configured an appropriate Experiment Automation model for the book shop scenario. Our model references SimuLizar as simulation tool, due to its self-adaptation capabilities.

4.5 Interpret

In this section, we exemplify some first new metric measurements that are now supported. Implementing the full range of our proposed metrics (and even more) is part of our future work.

In Fig. 8, we illustrate a result for our user capacity metric (scalability metric). For the book shop implemented as 3-layer/3-tier variant, we observe a user capacity of 30. Therefore, if more than 30 users reside within the system, SLOs (and corresponding requirements) are violated.

The architect of the book shop investigates the root causes for this insufficient user capacity; a higher number was expected. By investigating the CPU utilization of the application

[2]Quality Analysis Lab (QuAL): Software Design Description and Developer Guide - Version 0.2: `https://svnserver.informatik.kit.edu/i43/svn/code/QualityAnalysisLab/Documentation/trunk/org.palladiosimulator.qual.docs/QualityAnalysisLab.pdf` (User: anonymous; Password: anonymous; Visited on 31/10/2014).

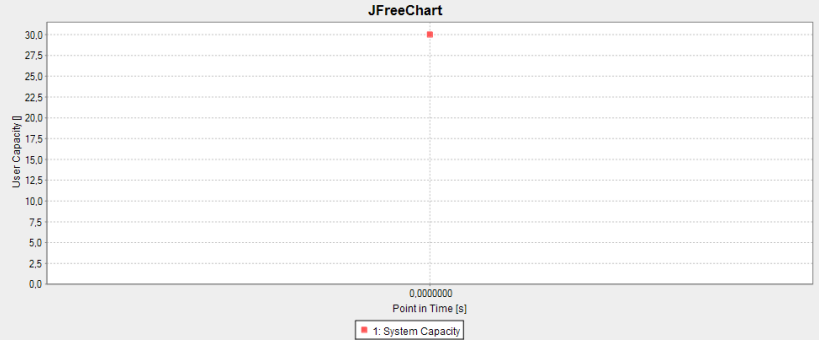

Figure 8: User capacity is 30 for the book shop scenario implemented as 3-layer/3-tier variant

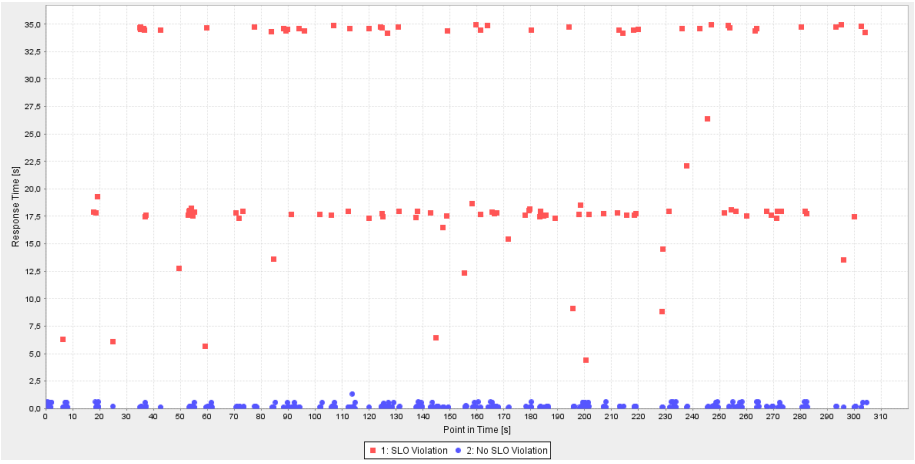

Figure 9: SLO violations over time for the book shop scenario implemented as SPOSAD variant

tier, the architect figures out that the application layer component overloads the CPU. Because the demands to the CPU are realistically modeled, the architect decides to apply the SPOSAD variant instead (SPOSAD ensures a scalable application layer).

The architect figures out that the capacity of the new system is around 500, which seems to be fine. However, by investigating the measurements for our "SLO violations over time" elasticity metric (Fig. 9), the architect observes that there are too many SLO violations throughout the year.

The software architect now suspects that the database is the new bottle neck resource. The architect therefore remodels the system with a faster HDD on the data tier and reexecutes the simulation. Now, no SLOs are violated anymore. Eventually, the architect therefore suggests to implement the book shop following the SPOSAD architectural style and with a fast HDD for the data tier. The architect additionally provides the 1-year operation costs for this variant based on dedicated cost metrics that our simulation now supports.

5 Conclusions

In this paper, we show how we integrated novel metrics for quality properties of cloud computing systems into Palladio, accompanied by a novel process and a more efficient modeling language. Following our process, we where able to efficiently analyze a first, simple example regarding scalability, elasticity, and efficiency.

Our new process and tool suite helps software architects in efficiently analyzing applications that shall operate within cloud computing environments. For scientists, our tools are especially interesting because their capabilities for scalability, elasticity, and efficiency analysis open a plethora of new possibilities for engineering systems.

In future work, we plan to further improve the usability of our tools because several editors

are in early-stage development. Afterwards, we want to evaluate these tools and our process with more extensive examples. We also plan to optimize our analysis by building up on already gained measurements and only analyzing changed/additional parts of system models (scalability, elasticity, and efficiency analysis composition).

References

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[BBM13] Matthias Becker, Steffen Becker, and Joachim Meyer. SimuLizar: Design-Time Modelling and Performance Analysis of Self-Adaptive Systems. In *Proceedings of Software Engineering 2013 (SE2013), Aachen*, 2013.

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1), January 2009.

[BLB15] Matthias Becker, Sebastian Lehrig, and Steffen Becker. Systematically Deriving Quality Metrics for Cloud Computing Systems. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE '15, New York, NY, USA, 2015. ACM. Accepted for publication.

[EMP13] Thomas Erl, Zaigham Mahmood, and Ricardo Puttini. *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall, 2013.

[FFH12] Florian Fittkau, Sören Frey, and Wilhelm Hasselbring. CDOSim: Simulating Cloud Deployment Options for Software Migration Support. In *MESOCA '12*, page 3746, 2012.

[HKR13] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity: What it is, and What it is Not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013), San Jose, CA, June 24–28*, 2013.

[Koz11] Heiko Koziolek. The SPOSAD Architectural Style for Multi-tenant Software Applications. In *Proc. 9th Working IEEE/IFIP Conf. on Software Architecture*, pages 320–327. IEEE, July 2011.

[Leh13] Sebastian Lehrig. Architectural Templates: Engineering Scalable SaaS Applications Based on Architectural Styles. In *Proceedings of the MODELS 2013 Doctoral Symposium co-located with the 16th International ACM/IEEE Conference on Model Driven Engineering Languages and Systems (MODELS 2013)*, volume 1071, pages 48–55, Miami, USA, 2013. CEUR-WS.org.

[Mer11] Philipp Merkle. Comparing Process- and Event-oriented Software Performance Simulation. Master's thesis, Karlsruhe Institute of Technology (KIT), Germany, 2011.

[TMD09] R.N. Taylor, N. Medvidovic, and E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

[vKHK14] Jóakim Gunnarson von Kistowski, Nikolas Roman Herbst, and Samuel Kounev. LIMBO: A Tool For Modeling Variable Load Intensities. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*, ICPE '14, pages 225–226, New York, NY, USA, March 2014. ACM.