Architectural Templates: Engineering Scalable SaaS Applications Based on Architectural Styles

Sebastian Lehrig**

Software Engineering Group & Heinz Nixdorf Institute
University of Paderborn, Paderborn, Germany
sebastian.lehrig@uni-paderborn.de

Abstract. Software architects plan, model, and analyze the high-level design of software systems. Today, these systems are often deployed in cloud computing environments as Software-as-a-Service (SaaS) applications. The scalability of these applications is crucially impacted by architects' early design decisions. Architects decide based on their experience and known architectural styles like a 3-tier architecture. In new application domains, however, architects lack the experience to determine whether their designs will result in scalable implementations. This lack leads to the high risk of unsatisfying scalability and expensive re-implementations.
To tackle this problem, we propose initial ideas and concepts for *architectural templates* (ATs), defined as a language to formalize architectural styles on component models. This formalization allows to enrich styles by quality annotations and completions for model-driven quality analyses. As we focus on SaaS applications, we exemplify this idea by enriching ATs by scalability annotations and completions allowing architects to analyze their applications' scalability. To illustrate such an analysis, we introduce and use a toy example. Based on this example, we derive initial working packages describing how we plan to realize and validate ATs.

Keywords: Model-driven, Architectural Templates, Cloud Computing, SaaS, Styles, Component-Based, Scalability, Performance, Engineering

1 Problem

In forward engineering, software architects plan, model, and analyze the high-level design of large software systems, i.e., their architecture. Today, these software systems are often deployed as so-called Software-as-a-Service (SaaS) applications [16] in cloud computing environments because of their advantageous characteristics. Cloud computing is characterized by (a) elasticity, i.e., an access to computing resources that can be provisioned and released on demand [16] and (b) an accounting model in which only actually-demanded resources determine costs (pay-per-use) [16]. These two characteristics induce a dependency between costs and required computing resources. To minimize costs, a typical requirement for SaaS applications is, therefore, that SaaS applications use as little additional computing resources as possible when demand increases. This requirement describes the *scalability* of a system, i.e., its ability "to sustain increasing workloads by making use of additional resources" [11].

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

For software architects, scalability induces the need of guidance toward its achievement. For instance, an architect may require guidance when deciding between a relational and a NoSQL [19] database as both promise different scalability depending on stored data. Today, architects can be guided by (1) architectural styles [18] that inherently foster scalability in cloud computing environments and (2) scalability analyses allowing architects to predict scalability properties.

To illustrate these ideas, we introduce an example scenario in Sec. 1.1. Afterwards, we discuss why neither architectural styles nor scalability analyses sufficiently guide architects in designing scalable SaaS applications in Sec. 1.2.

1.1 Example Scenario

As an example scenario, we consider a simplified book shop that shall be newly designed for running in a cloud computing environment. In this scenario, an enterprise assigns a software architect to design the book shop. The enterprise has the following requirements for this shop:

R1: Functionality In the shop, customers can (1) browse and (2) order books.

R2: Handling of Environmental Changes The enterprise expects that the environment for the book shop 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 response times of the shop shall stay within 3 seconds even if the customer arrival rates in- or decrease by 1,000 customers per hour (at maximum).

R3: Linear Scalability The costs for operating the book shop are only allowed to increase (decrease) by $0.01 per hour when the number of customers using the shop increases (decreases) by 1. *Costs per hour* is a metric to measure the amount of additional resources (cf., the scalability definition in Sec. 1).

Requirements R2 and R3 are typical reasons to operate a system in an elastic cloud computing environment [11], i.e., an environment where application servers automatically provision the required amount of resources to cope with environmental changes. Therefore, the software architect will design the shop as an SaaS application operating in a rented cloud computing environment.

1.2 Guidance for Scalability

To provide a scalable SaaS application (R3), the architect considers using (1) architectural styles and (2) scalability analyses.

The first option (architectural styles) requires that the architect is aware of appropriate cloud-based architectural styles as well as their application and assumptions. Currently, only a few architectural styles for SaaS applications in the cloud exist [8]. One example for such a style is SPOSAD [13].

SPOSAD suggests a 3-tier system with stateless middle tier, but leaves open the decision for a concrete database paradigm on the data tier [13]. Therefore, the architect may design the book shop as shown in Fig. 1: he assigns each of the three SPOSAD tiers (presentation, middle, data) to a component (Book Shop Frontend, Book Management, Book Database). The tiers of SPOSAD can be seen as SPOSAD's roles, i.e., as set of constraints for associated components.

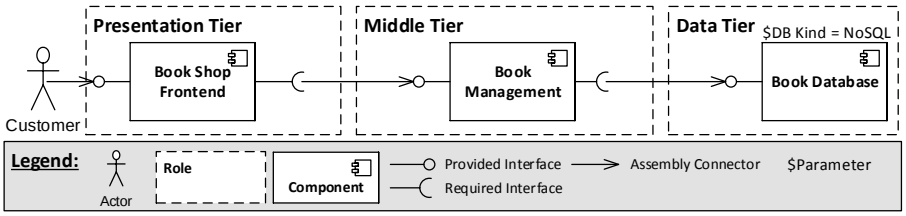

Fig. 1. Model of the designed book shop scenario according to SPOSAD.

For example, the data tier may only allow connections from the middle tier. As SPOSAD does not constrain the data tier further, the architect is unsure whether to design the system with a relational or a NoSQL database that both promise a different scalability. Because of this variability point, he needs further guidance; SPOSAD alone is not enough. For achieving particular scalability requirements, there is, hence, the need to refine SPOSAD. Also the few other SaaS styles lack detailed suggestions for selecting an appropriate database paradigm.

The second option (scalability analyses) would allow the architect to model both database alternatives and to compare their scalability (what-if analysis). Scalability analyses require simulation or analytical models allowing architects to predict an SaaS applications' scalability in cloud computing environments. However, according to Becker et al. [2], current analysis models lack (1) support for comprehensive design-time analyses based on architectural models, (2) realistic case studies in cloud computing environments, and (3) explicit support for scalability because of their focus on performance analysis. These lacks hinder the architect to use and trust existing approaches in order to analyze the scalability of the modeled book shop. For instance, these approaches are unable to analyze whether a NoSQL or a relational database suits the shops' scalability requirements best. Hence, these approaches need to be extended and improved.

The lack of guidance (regarding styles and scalability analyses) for the architect leads to the high risk of realizing a cost-inefficient SaaS application and of an expensive re-implementation. For example, it may be expensive to refactor the book shop with an established but non-scalable relational database implementation to an implementation using a NoSQL database. This risk becomes even more severe in case the enterprise discovers scalability issues when high costs for hosting their application have already incurred, e.g., during system operation.

2 Related Work

Related work tackling the engineering of scalable SaaS applications can be classified into two areas: (1) architectural styles for scalability and (2) performance engineering serving as a basis for scalability engineering.

A generally rich set of literature provides, classifies, and surveys sets of *architectural styles* (e.g., [5], [20]). However, these styles lack an explicit consideration of cloud computing environments as well as a focus on scalability.

In the context of cloud computing, typical styles are REST [9] for HTTP and SPIAR [17] for AJAX. Also Erl et al. [8] describe a set of cloud computing styles that foster scalability (e.g., load balanced virtual server instances). These styles have in common that they target *the infrastructure* in which SaaS

applications run. Third party cloud computing providers typically provide this infrastructure by offering (1) a deployment of SaaS applications in application servers (Platform-as-a-Service; PaaS) or (2) access to (virtual) nodes where users can operate their SaaS application (Infrastructure-as-a-Service; IaaS). Therefore, these third party providers can utilize the presented architectural styles. However, as these styles lack a focus on implementing SaaS applications, they only implicitly help architects who engineer the SaaS layer of these applications. In contrast, we will focus directly and explicitly on architectural styles for SaaS applications, starting with investigating the few architectural styles that do cover aspects of SaaS applications. Two examples for these styles are SPOSAD [13] and SOCCA [21]. For instance, SPOSAD describes a 3-tier variation that promotes a stateless middle tier for scalability [13]. As they are stateless, components of the middle tier can then safely be replicated (scaled-out) and load-balanced.

The second area, *performance engineering*, offers several approaches recently classified and surveyed by Koziolek [12]. These approaches allow for analyzing the performance (response time, throughput, utilization) of component-based systems as, for example, the PCM [3]. However, they lack support for cloud computing characteristics, e.g., an elastic provisioning of computing resources.

Becker et al. [2] survey model-driven performance engineering approaches that support elasticity via self-adaptation, e.g., the SimuLizar [1] approach that extends the PCM. They conclude that these approaches are still limited. For example, only two approaches target design-time models and realistic validations by case studies are missing. However, software architects require design-time approaches, and appropriate case studies are the means to systematically find architectural styles for scalable SaaS applications. Especially the latter aspect, i.e., using model-driven performance engineering techniques to conduct scalability analyses, lacks investigation. Therefore, we plan to extend existing performance analysis approaches like SimuLizar and PCM to enable model-driven scalability analyses of SaaS applications. In particular, we want to conduct case studies to identify appropriate architectural styles for scalable SaaS applications.

3 Proposed Solution

To cope with the lack of guidance for software architects (cf., Sec. 1), we propose and introduce *architectural templates* (ATs). We define ATs as a language to formalize architectural styles on component models. This formalization allows to enrich styles by quality annotations and completions. Quality annotations characterize a concrete quality property of interest. Quality completions utilize these annotations to derive quality models analyzable by quality analysis tools. As we focus on SaaS applications, we enrich ATs by *scalability* annotations and completions allowing architects to analyze their applications' scalability.

Technically, we plan to apply ATs on systems modeled in the PCM [3]. For this application, we will extend the PCM metamodel (i.e., PCM's model for specifying component-based architectures) by ATs. For scalability analyses, we will extend the SimuLizar [1] approach to take information of ATs into account.

Fig. 2. Process steps of AT application for software architects.

Software architects can use our ATs to design systems as well as to analyze their scalability. AT engineers, on the other hand, enrich our ATs by annotations and completions. We guide through these processes of AT application (Sec. 3.1) and AT creation (Sec. 3.2) by using the book shop scenario of Sec. 1.

3.1 Applying a SPOSAD Architectural Template

Software architects apply ATs to design SaaS applications and to analyze their scalability. Fig. 2 illustrates the process steps for applying ATs. In step 1, architects select an AT from a repository of ATs. For example, the architect of the book shop scenario selects a SPOSAD AT. Afterwards, the architect assigns all roles the AT requires to the components of his architecture (step 2).

As illustrated in Fig. 1, he may assign SPOSAD's presentation, application, and data tier roles to book shop components as well as connects these components appropriately. Firstly, the AT formalism assures that no constraints are violated, e.g., it forbids connecting presentation and data tier components. Secondly, the AT formalism allows architects to specify whether a data tier component corresponds to a relational or a NoSQL database (scalability annotation).

As this design is based on an AT, the architect can automatically run scalability analyses afterwards by using the analysis tool we will provide (step 3). The key idea is that the AT includes a scalability completion for this purpose. For the book shop, SPOSAD AT's scalability model ensures that the architect obtains results that accurately reflect the scalability of the selected database kind. Therefore, the SPOSAD AT allows the architect to analyze which kind of database better fits his scalability requirements *at design time*.

3.2 Creating the Architectural Template for SPOSAD

To create ATs, AT engineers apply the process illustrated in Fig. 3. In the first step, they formalize AT roles (e.g., SPOSAD's data tier) based on architectural styles (e.g., known from architecture handbooks).

Besides formalizing architectural styles, AT engineers enrich the AT with scalability annotations and completions to enable an automated scalability analysis. For this automation, AT engineers first have to identify scalability-relevant parameters and quantify them (step 2). For example, the kind of database (relational vs. NoSQL database) on the data tier could be a scalability-relevant parameter because NoSQL databases are often designed for scaling horizontally.

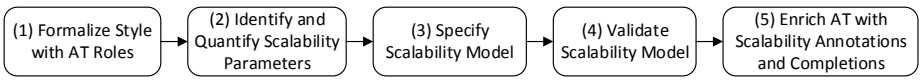

Fig. 3. Process steps of AT creation for AT engineers.

To confirm and quantify the influence on the scalability of this parameter empirically, an AT engineer (1) implements a series of automated test-drivers (focusing on the database parameter) that systematically collect the necessary data based on scalability metrics and (2) runs these test-drivers in a cloud computing environment. The AT engineer can subsequently check and quantify the influence of the "kind of database" parameter by means of a regression analysis.

In case AT engineers successfully identified a scalability parameter, they specify a scalability model for this parameter (step 3). The AT engineer of the book shop has, e.g., to specify a scalability model for NoSQL databases. Because NoSQL databases can scale horizontally, an accurate scalability model may model this NoSQL database as a "simple database" component, attached to a "load balancer" component. As soon as load exceeds a certain threshold, an adaptation rule assures that another "simple database" is spawned and added to load balancing. To determine concrete thresholds of the load balancer as well as processing times of the "simple database", the AT engineer integrates the results of the regression analysis. This model can, finally, be analyzed by ordinary analysis tools supporting components annotated with processing times and adaptation rules, e.g., the PCM with SimuLizar extended by scalability metrics.

Next, AT engineers validate the scalability model (step 4). For this validation, they use a set of case studies, like the book shop scenario, that include identified scalability parameters. For each case study, AT engineers (1) measure its scalability in the considered cloud computing environment and (2) predict its scalability based on the scalability model. In case the predictions accurately reflect the scalability of the measurements, the AT engineers successfully validated the model. Otherwise, they have to iterate the process by refining the automated test-drivers or by identifying and integrating further scalability parameters.

In the final step of Fig. 3, AT engineers enrich the AT with scalability annotations and completions. The specification of a "NoSQL database" for the "kind of database" parameter corresponds to a scalability annotation. The transformation to a scalability model corresponds to a scalability completion. Therefore, analysis tools can analyze the scalability of a model specified by ATs.

We based the process of Fig. 3 on established processes in performance engineering [10]. The basic idea is that concepts for performance engineering (e.g., performance model specification) apply similarly on scalability as well.

4 Preliminary Work

In [4], we described an overall process for applying ATs (there, termed "patterns") to design scalable SaaS applications. In this context, we also showed how to reverse engineer PCM models from existing systems [7].

Furthermore, we extended the PCM to run automatically measurements based on Java SE and RMI [15]. We also conducted several measurements in a virtualized environment and showed PCM's suitability for these environments. In [14], we show how to support automatic measurements for different target platforms than Java SE. This additional support is especially useful when the targeted platform is a cloud computing platform, e.g., a PaaS environment.

5 Expected Contributions

Our main contribution will be the AT language and AT processes helping software architects to design scalable SaaS applications. For the concrete realization, we plan to contribute (1) an AT metamodel extending the PCM, (2) evaluated processes for creating and applying ATs, and (3) an initial repository of evaluated ATs for designing scalable SaaS applications. We will base our evaluations on (industry) case studies, which are our final contribution.

6 Plan for Evaluation and Validation

We plan to evaluate expected benefits of ATs based on case studies. As case studies, we will use (1) the simple book shop scenario of Sec. 1.1, (2) the TPC-W benchmark [6] describing a more complex book shop, and (3) industry case studies based on our on-going work in the CloudScale [4] project. Additionally, we will validate benefits of ATs by controlled experiments with student groups.

7 Current Status

Currently, we are in the planning phase of our work. The ideas presented in this paper reflect the current status of our plans. Therefore, we introduce a set of five work packages (WPs) describing how we want to progress in actually realizing ATs. We plan to provide first results in every WP within one year. Afterwards, we plan to refine and extend them in a time frame over two more years.

WP1: Architectural Templates The goal of this WP is to formalize ATs by extending the PCM metamodel as well as to provide an initial set of example ATs. A minimal requirement is that at least SPOSAD is specified using ATs.

WP2: Scalability Formalization This WP targets to provide a quantifiable scalability formalization that architects can use to compare different designs of systems. Therefore, we want to clarify two main questions: (1) "Which scalability definition fits to the needs of cloud computing (e.g., regarding influence of elasticity)?" and (2) "Can a particular scalability definition be formalized in order to quantify scalability (e.g., as a metric)?". To answer these questions, we plan to conduct a systematic literature review.

WP3: Domain Mapping In this WP, we want to select and characterize the application domain we focus on. Possible candidates are PaaS environments like SAP HANA Cloud, mOSAIC, and Google App Engine. We will develop scalability measurement concepts for selected domains based on the PCM.

WP4: Tool Integration This WP targets integrating developed concepts of ATs to the PCM. Firstly, we integrate general tool support for ATs. Secondly, we enrich the PCM by dedicated support for scalability measurements (currently, the PCM focuses on performance only). Thirdly, we extend the PCM to provide tool support for selected application domains of WP3.

WP5: Evaluation In this WP, we evaluate ATs as described in Sec. 6.

References

1. Becker, M., Becker, S., Meyer, J.: SimuLizar: design-time modelling and performance analysis of self-adaptive systems. In: Proceedings of Software Engineering 2013 (SE2013), Aachen (2013)
2. Becker, M., Luckey, M., Becker, S.: Model-driven performance engineering of self-adaptive systems: a survey. In: QoSA '12. pp. 117–122. ACM, New York (2012)
3. Becker, S., Koziolek, H., Reussner, R.: The palladio component model for model-driven performance prediction. Journal of Systems and Software 82(1) (Jan 2009)
4. Brataas, G., Stav, E., Lehrig, S., Becker, S., Kopcak, G., Huljenic, D.: CloudScale: Scalability Management for Cloud Systems. In: 4th Int. Conf. on Performance Engineering. ACM (Apr 2013)
5. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Stal, M.: Pattern-Oriented Software Architecture Volume 1: A System of Patterns. Wiley, volume 1 edn. (Aug 1996)
6. Council, T.P.: TPC-W benchmark (web commerce) specification version 1.8. `http://www.tpc.org/tpcw/spec/tpcw_V1.8.pdf` (Feb 2002), last visited: 12 Sep 2013
7. von Detten, M., Lehrig, S.: Reengineering of component-based software systems in the presence of design deficiencies – an overview. In: WSR'13 (May 2013)
8. Erl, T., Puttini, R., Mahmood, Z.: Cloud Computing: Concepts, Technology and Design. Prentice Hall PTR (2013)
9. Fielding, R., Taylor, R.: Principled design of the modern web architecture (2000)
10. Happe, J.: Predicting Software Performance in Symmetric Multi-core and Multiprocessor Environments. Ph.D. thesis, University of Oldenburg, Germany (2008)
11. Herbst, N.R., Kounev, S., Reussner, R.: 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)
12. Koziolek, H.: Performance evaluation of component-based software systems: A survey. Perform. Eval. 67(8), 634–658 (Aug 2010)
13. Koziolek, H.: The SPOSAD architectural style for multi-tenant software applications. In: Proc. 9th Working IEEE/IFIP Conf. on Software Architecture. pp. 320–327. IEEE (Jul 2011)
14. Langhammer, M., Lehrig, S., Kramer, M.E.: Reuse and configuration for code generating architectural refinement transformations. In: VAO '13. ACM (2013)
15. Lehrig, S., Zolynski, T.: 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 (Nov 2011)
16. Mell, P., Grance, T.: The NIST definition of cloud computing. NIST Special Publication 145(6), 7 (2011)
17. Mesbah, A., van Deursen, A.: A component- and push-based architectural style for ajax applications. Journal of Systems and Software 81(12), 2194–2209 (2008)
18. Reussner, R.H., Hasselbring, W.: Handbuch der Software-Architektur. dPunkt.verlag Heidelberg, 2 edn. (Dec 2008)
19. Sadalage, P., Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison Wesley Professional (2012)
20. Shaw, M., Clements, P.C.: A field guide to boxology: Preliminary classification of architectural styles for software systems. In: Proceedings of the 21st International Computer Software and Applications Conference. pp. 6–13. IEEE (1997)
21. Tsai, W.T., Sun, X., Balasooriya, J.: Service-oriented cloud computing architecture. In: ITNG'10. pp. 684–689. IEEE, Washington, DC, USA (2010)