Applying Architectural Templates for Design-Time Scalability and Elasticity Analyses of SaaS Applications*

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

ABSTRACT

Software architects plan, model, and analyze architectures of large software like Software-as-a-Service (SaaS) applications. The scalability and elasticity of these applications are crucially impacted by architects' early decision for an architectural style. However, whether this decision fostered scalability and elasticity can currently only be tested with the final application deployed in the target cloud computing environment. This process leads to the high risk of unsatisfying scalability/elasticity and expensive re-implementations.

To tackle this problem, we propose an early design-time scalability/elasticity analysis using *architectural templates* (ATs). ATs are a language to formalize architectural styles on component models for the purpose of early quality analyses. This work-in-progress paper provides a first formalization of ATs and investigates their applicability to analyze the scalability and elasticity of SaaS applications at early design-time by using a 3-tier example scenario. Our results indicate that ATs are applicable to such 3-tier scenarios.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Architectural styles*; D.2.8 [**Software Engineering**]: Metrics—*Scalability and Elasticity*

Keywords

Architectural Templates, Cloud Computing, Analysis

1. INTRODUCTION

Software architects plan, model, and analyze architectures of large software like Software-as-a-Service (SaaS) cloud computing applications. Cloud computing is characterized by virtually unlimited resources that cloud providers account on a pay-per-use basis and an elasticity management that autonomically provisions resources to deal with changing workload [18]. To benefit from such an elasticity management, more resources need to help SaaS applications to cope with higher load. This need describes the *scalability* of a SaaS application, i.e., its "ability to sustain increasing workloads by making use of additional resources" [13].

Scalability and elasticity of applications are crucially impacted by architects' early decision for an architectural style. For instance, an architect can decide to design the middle tier of a 3-tier application to be stateless such that middle tier components can scale, i.e., safely be replicated and load-balanced. However, whether this decision indeed fostered scalability and elasticity can currently only be tested with the final application deployed in the target cloud computing environment. Because this process involves the complete implementation of the application, architects have to cope with the high risk of unsatisfying scalability and elasticity as well as of expensive re-implementations, e.g., if scalability issues of the 3-tier application above resides in the data tier.

In related work, authors have begun to investigate analysis approaches that allow software architects to analyze quality properties at design-time. For example, Palladio [4] enables architects to analyze properties like performance based on a component model enriched with performance annotations. However, these approaches lack explicit support for scalability and elasticity [3], e.g., Palladio cannot model and analyze changing workloads. Moreover, no current analysis approach supports architectural styles, thus, making early informed decisions impossible for architects.

To tackle this problem, we propose early design-time analyses using *architectural templates* (ATs) [17]. ATs are a language to formalize architectural styles on component models for the purpose of model-driven quality analyses. For this purpose, ATs allow to enrich styles by quality completions, i.e., transformations to quality analysis or simulation models. Because we focus on SaaS applications, we enrich styles by *scalability/elasticity* completions that allow architects to analyze their applications' scalability/elasticity by reusing architectural knowledge manifested in such completions.

The contribution of this work-in-progress paper is a first formalization of ATs and an investigation of their applicability to analyze SaaS applications using a 3-tier example scenario. Our results indicate that ATs can analyze scalability and limitedly elasticity in such scenarios.

The remainder of this paper is organized as follows. We introduce the example scenario in Sec. 2. Afterwards, we describe and formalize ATs in Sec. 3. We evaluate the applicability of ATs to the example scenario in Sec. 4. Finally, we discuss related work in Sec. 5 and conclude in Sec. 6.

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

HotTopiCS 2014, Dublin, Ireland

2. EXAMPLE SCENARIO

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:

R1: Functionality In the shop, customers shall be able to browse and order books.

R2: Scalability The enterprise expects a customer arrival rate of 100 customers per minute (on average). It further expects that this rate will grow by 10% in the first year, i.e., increase to 110 customers per minute. In the long run, the shop shall therefore be able to handle this increased load without violating other requirements.

R3: Elasticity The enterprise expects that the workload for the online book shop repeatedly changes over time. For example, it expects that books are sold better around Christmas while they are sold worse around the holiday season in summer. Therefore, the system shall proactively adapt to the anticipated workload changes, i.e., maintain a response time of 2 seconds or less as good as possible. For non-anticipated workload changes, e.g., peak workloads, the system shall re-establish a response time of 2 seconds or less within 10 minutes once a requirement violation is detected.

Requirements R2 and R3 are typical reasons to operate a system in an elastic cloud computing environment [13], i.e., an environment where an elasticity management autonomously provisions the required amount of resources to cope with workload changes. Examples for such elastic cloud computing environments are different Platform-as-a-Service (PaaS) environments such as AWS Elastic Beanstalk [1], Google App Engine [11], Windows Azure [20], and mOSAIC [21]. Because of their elasticity management, the architect decides to operate the online shop in one of these PaaS environments.

Web applications are commonly designed as 3-tier applications [23]. Therefore, the architect investigates possible variants of the 3-tier architectural style. The architect considers a classical 3-tier architecture [23] as well as the SPOSAD architectural style [16], a 3-tier variant with stateless middle tier to foster scalability.

Figure 1 illustrates the early design-time model of the shop as planned by the architect. In a 3-tier architecture, the three layers of a 3-layer architecture (presentation, middle, and data) are allocated to three different tiers. In an elastic PaaS environment, these tiers are represented by different replicable virtual servers. The middle tier (layer) will be stateless if the architect follows SPOSAD.

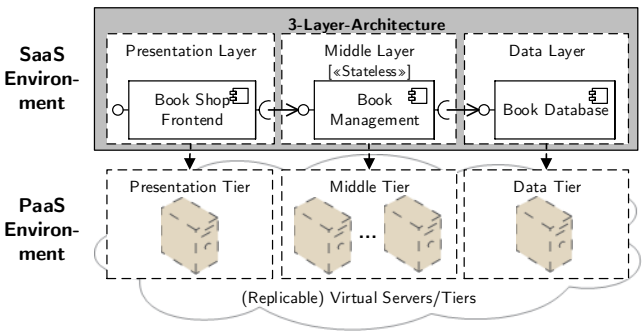

Figure 1: Overview of the online book shop.

3. ARCHITECTURAL TEMPLATES

The software architect of the scenario in Sec. 2 would like to know whether the planned online shop should be realized according to (a) a classical 3-tier architecture, (b) the SPOSAD architectural style, or (c) neither of the two. The architect wants to decide based on whether the scalability (**R2**) and elasticity (**R3**) requirements will be met by the finally implemented application. In other words, the architect would like to conduct a what-if analysis for the available options to make an informed decision.

The purpose of the *architectural template* (AT) language is to help the architect in this concern: the AT language is a language to enable analyses of early design-time models, thus, allowing the architect to conduct the desired what-if analysis for the online book shop [17]. The goal of the AT language is that such analyses require as little information as possible because architects should not have to thoroughly implement each option to make informed decisions (no "try & pray").

In our previous work on the AT language, we have informally defined the AT language as *a language to formalize architectural styles on component models for the purpose of model-driven quality analyses* [17]. However, our previous work lacks a formal definition of the AT language, e.g., based on a metamodel. As of today, this lack makes it impossible for software architects to apply our AT language. Therefore, the goal of this section is to derive a first version of a formal AT language definition. Only afterwards, we can evaluate the applicability of the language.

According to our informal definition, we apply the AT language in the context of model-driven analyses and component models, thus, are interested in language-based models and metamodels. Therefore, we can apply the metamodeling concepts introduced by Kühne [14] to formally define the AT language. Accordingly, a language $\mathcal{L}$ can equivalently be defined by the metamodel $\mathcal{MM}$ or by the intension ι of the language, i.e., $\mathcal{MM}(\mathcal{L}) \sim \iota(\mathcal{L})$ [14]. The intension of a language is the sum of its attributes [7], e.g., "represents architectural style" $\wedge$ "has name" $\wedge \ldots$. Therefore, we have two options to approach the definition of the AT language $\mathcal{L}_{\mathcal{AT}}$: (a) collect all required attributes of $\mathcal{L}_{\mathcal{AT}}$, i.e., derive $\iota(\mathcal{L}_{\mathcal{AT}})$, or (b) specify the metamodel of $\mathcal{L}_{\mathcal{AT}}$, i.e., define $\mathcal{MM}(\mathcal{L}_{\mathcal{AT}})$ directly. Note that architects eventually need $\mathcal{MM}(\mathcal{L}_{\mathcal{AT}})$ to apply $\mathcal{L}_{\mathcal{AT}}$ in a model-driven way, for instance, graphical editors require a metamodel.

Our idea is to begin with option (a) by deriving required attributes based on our example scenario in Sec. 2. For this derivation, we apply the envisioned process of "AT application" [17], describing how software architects use the AT language, to this scenario.[1] After we have collected the attributes of $\mathcal{L}_{\mathcal{AT}}$, we end up with a first version of $\iota(\mathcal{L}_{\mathcal{AT}})$. Based on this intension, we can subsequently derive a first version of $\mathcal{MM}(\mathcal{L}_{\mathcal{AT}})$. The crux of the matter is that we can, by this approach, assure that the metamodel is correct by construction, i.e., that we have $\mathcal{MM}(\mathcal{L}_{\mathcal{AT}}) \sim \iota(\mathcal{L}_{\mathcal{AT}})$.

We guide through the envisioned process of AT application in Sec. 3.1. Afterwards, we can derive the intension and first formalisms of the AT language in Sec. 3.2. We define the metamodel in Sec. 3.3, allowing us in Sec. 4 to evaluate the applicability of ATs based on the book shop scenario.

[1]This approach is similar to eliciting requirements based on scenarios.

3.1 Applying Architectural Templates

The software architect of the scenario in Sec. 2 applies ATs to design the online book shop and to analyze the shops' scalability and elasticity. Figure 2 illustrates the process steps for this AT application [17].

At the beginning, the architect selects an AT from a repository of ATs, assuming an "AT" is a selectable entity representing an architectural style, specified in the AT language (step 1). For example, the architect selects a 3-tier AT.

Afterwards, the architect assigns all roles the AT requires to the elements of the architecture, assuming "roles" refer to the different parts of the architectural style (step 2). This assignment of roles initiates the transition from architectural style to high-level system components of the architecture. As illustrated in Fig. 1, the architect assigns presentation, application, and data layer roles to book shop components as well as connects these components appropriately. The architect also assigns tier roles to presentation, middle, and data tier resources of the architecture and specifies the allocation of components to such tiers. Being inherent to the 3-tier style, the AT should assure that none of the styles' constraints are violated, e.g., an AT editor should forbid connecting presentation and data layer components.

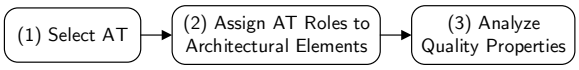

Figure 2: Process steps of AT application for software architects (from [17]).

Because the resulting architecture is based on an AT, the architect can automatically run quality analyses such as scalability and elasticity analyses afterwards (step 3). Internally, the AT has to include a quality completion for this purpose, i.e., a transformation to a quality analysis/simulation model (we describe the process how to come to such a transformation in [17]). For the book shop, 3-tier AT's completion obtains analysis results that accurately reflect the scalability and the elasticity of the shop. Therefore, the 3-tier AT allows the architect to analyze his scalability/elasticity requirements *at design time*.

Note that we evaluate AT application in Sec. 4 based on the book shop scenario, including the specification of a scalability and elasticity completion and the conduction of a scalability and elasticity analysis.

3.2 Formalizing Architectural Templates

For deriving the intension and first formalisms of the AT language, we analyze the AT application scenario in Sec. 3.1 for minimally required attributes enabling AT application. Table 1 summarizes our analysis results.

The selection of ATs (step 1) induces the need of a repository of ATs to select from ("AT collectable in repository"), along with unique identifiers ("AT has ID") and names ("AT has name") for ATs. The selection criteria are based on the architectural style represented by the AT ("AT represents architectural style") and supported quality analyses. For example, architects may want to select a 3-tier AT because this AT represents the 3-tier architectural style (along with its promised quality properties).

Moreover, the selection of an AT from a repository requires the notion of AT types and AT instances. For example, because a selection should not alter the entity in the repository, instantiation can be applied. Therefore, we distinguish between AT types (or short ATs), i.e., the entities that reside in the repository, and AT instances that result from selecting an AT (type). This notion of types and instances is indeed suitable because ATs *classify* the AT instances that can be created based on them (classification is needed for instance-of relationships, cf. [14]). For instance, the architect of the shop could use a different set of components for the assignment in step 2 and the resulting architecture would still conform to the selected AT. Therefore, AT instances are classified by referred ATs ("AT instances are classified by ATs").

Following the terminology of Kühne [14], ATs are type models for AT instances, i.e., $AT_{instance} \lhd_t AT$, and AT instances are prescriptive token models of the planned software, i.e., $software \lhd_i AT_{instance}$. The instance-of relation between ATs and AT instances describes an ontological instantiation because both reside on the same linguistic level, i.e., $AT_{instance} \lhd_t^o AT$. However, a change to a higher linguistic level is needed in order to *specify* such ATs and AT instances. This specification is enabled by our AT language ("AT language has ATs" and "AT language has AT instances"). Accordingly, a specified AT is a linguistic instance of the AT model element of the AT language, i.e., $AT \lhd_t^l AT_{lang}(AT)$. Analogously, a specified AT instance is a linguistic instance of the AT instance model element of the AT language, i.e., $AT_{instance} \lhd_t^l AT_{lang}(AT\ instance)$. Furthermore, the latter two model elements of the AT language require an association describing the ontological instance-of relation, i.e., $AT_{lang}(AT\ instance)$ has a type $AT_{lang}(AT)$.

In Fig. 3, we illustrate the discussed instance-of relationships based on the 3-tier AT. At the upper-right of the figure, we show these relationships along the two linguistic levels L_0 and L_1 and the two ontological levels O_0 and O_1. Furthermore, we use Kühne's notion of a meaning μ that "assigns meaning to a model (element)" [14]. Our AT language defines the meaning for elements of linguistic level L_1 using $\iota(\mathcal{L}_{\mathcal{AT}})$. The meaning of the 3-tier AT (O_1 on L_0) is based on the referred architectural style concept using $\iota(\textit{3-tier})$. This concept defines the meaning of the book shop model (O_0 on L_0) via its extension $\epsilon(\textit{3-tier})$, i.e., all elements falling under the 3-tier concept [14]. Also our AT language spans an extension $\epsilon(\mathcal{L}_{\mathcal{AT}})$, defining all valid models specified by our language, e.g., our book shop model.

Given these fundamental formalisms of our AT language, we can now inspect the assignment of AT roles to architectural elements (step 2). This assignment requires ATs to specify the available roles ("AT has roles"). Furthermore, ATs can define constraints based on such roles ("AT has constraints" and "constraints relate to roles"). For the assignment of roles to components, AT instances have to maintain an appropriate reference ("AT instance has references between components and roles"). The same need for references holds for resources of the architecture that, for instance, represent the tiers of the 3-tier architecture ("AT instance has references between resources and roles").

Finally, the quality property analysis (step 3) requires that ATs specify a quality completion ("AT has quality completion"), e.g., for scalability and elasticity. For such a completion, ATs should reference a model transformation that transforms to an appropriate quality analysis or simulation model.

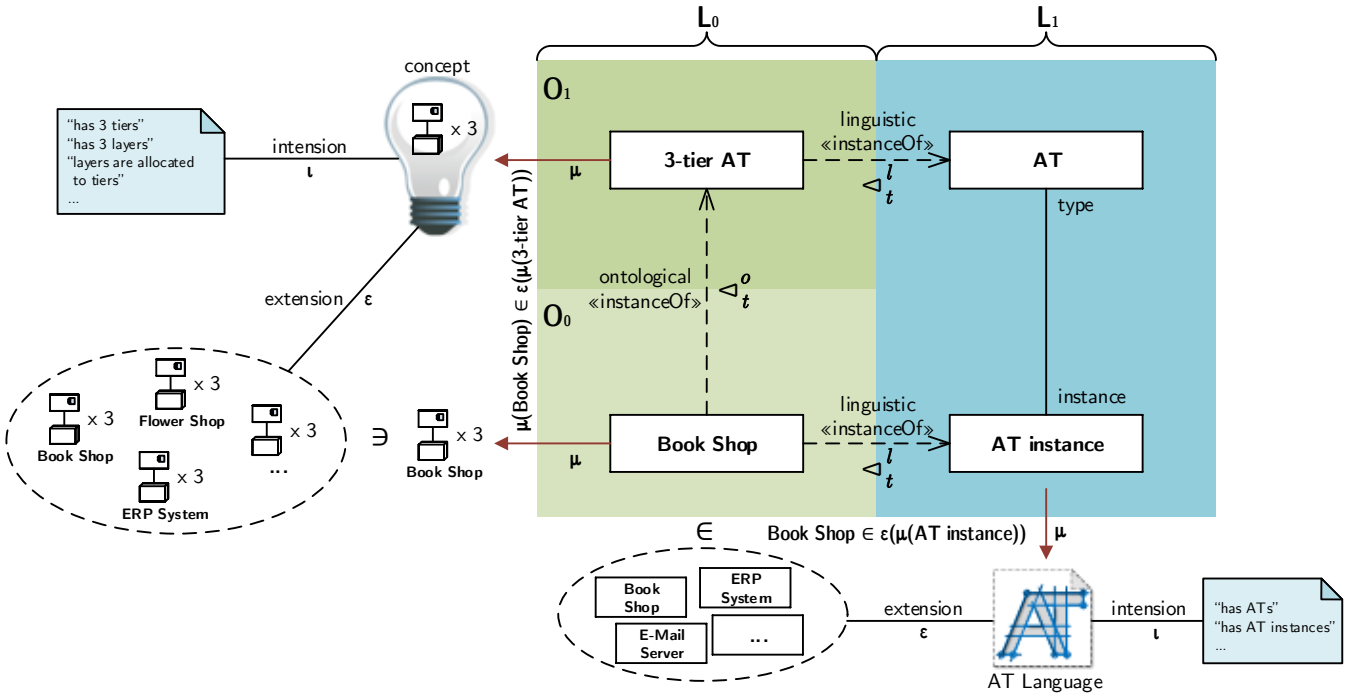

Figure 3: Instance-of relationships of the book shop example (based on [14]).

Table 1: Intension of the AT language ($\iota(\mathcal{L}_{\mathcal{AT}})$)

Element	Attributes
AT language	"has ATs", "has AT instances"
AT	"collectable in repository", "has ID", "has name", "represents architectural style", "has roles", "has constraints", "constraints relate to roles", "has quality completion"
AT instance	"classified by ATs", "has references between components and roles", "has references between resources and roles"

3.3 Metamodeling Architectural Templates

Based on the formalization of the AT language in Sec. 3.2, we construct an initial metamodel for the AT language in this section. For this construction, we assure that the metamodel adheres to the intension of the AT language $\iota(\mathcal{L}_{\mathcal{AT}})$ given in Tab. 1. Because this intension only covers the essential aspects of our AT language for evaluating the applicability of ATs, we exclude aspects for reusability and extendability in our metamodel. For example, we provide no means for extending a 3-layer AT in order to define a 3-tier AT. We leave the extension of our initial metamodel with such additional aspects as future work.

Moreover, our AT metamodel extends the Palladio Component Model (PCM) [4], a component-model for performance analysis. By extending the PCM, we can reuse several elements of the PCM: (1) AT instances reuse PCM's component instances (so-called "AssemblyContexts") for the reference between components and roles, (2) AT instances reuse PCM's hardware resource descriptions (so-called "ResourceContainers") for the reference between resources and roles, (3) constraints of ATs can directly be specified by reusing PCM's contracts for provided operations (so-called "ProvidedComponentTypes") and for provided plus required operations (so-called "CompleteComponentTypes"). Another advantage of extending the PCM is that it can be used for performance analysis. This performance analysis can also be used as a basis for analyzing scalability and efficiency as we see in the evaluation of the applicability of ATs in Sec. 4.

Following the intension of the AT language, its metamodel consists of two sub metamodels: a metamodel for ATs (i.e., AT types) and a metamodel for AT instances. We illustrate both metamodels in separate figures, the AT metamodel in Fig. 4 and the AT instance metamodel in Fig. 5.

The central metaclass of the AT metamodel (Fig. 4) is the AT metaclass, representing ATs. According to the intension of ATs, this metamodel includes the metaclasses Repository, Role, Constraint, and Completion. These metaclasses are associated to the AT metaclass as described by the intension of ATs: ATs have a Completion, Repository allows to collect a set of ATs because of its containment association, and analogously, ATs include a set of Roles and Constraints. Furthermore, all of these metaclasses inherit from PCM's Entity, thus, giving each metaclass a name and a unique identifier attribute. Constraints have to relate to one or more Roles, thereby, defining their semantics.

As an extension to the intension of the AT language, we propose several Constraint types: ProvidedInterfaceConstraint constraints a role to be a component that provides a given set of operations (via ProvidesComponentType); CompleteInterfaceConstraint constraints a role to be a component that provides a given set of operations and that can only require a given sets of operations at maximum (via CompleteComponentType); AllocationConstraint constraints the allocation between a component and resource roles; OCLConstraint con-

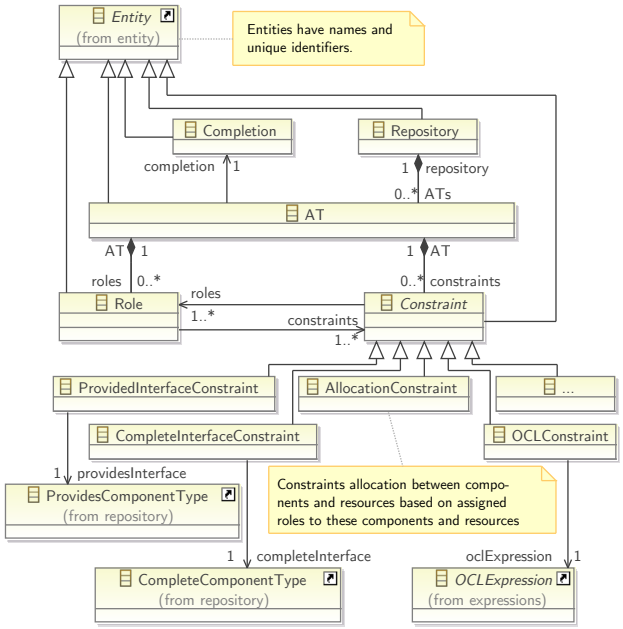

Figure 4: AT (type) metamodel.

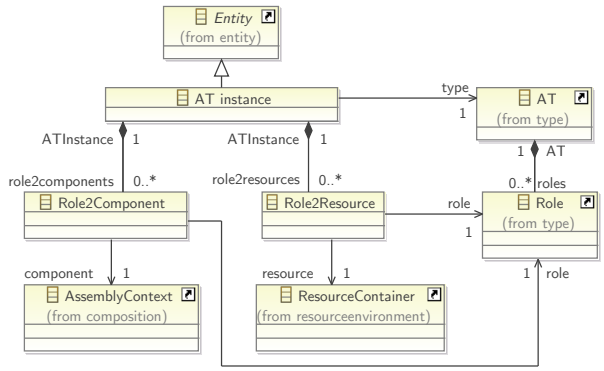

Figure 5: AT instance metamodel.

straints roles based on OCL expressions; and other types (not shown) like ComponentRoleConstraint and ResourceRoleConstraint that restrict the assignability of roles to components or resources. In future work, we expect to define more of such commonly applicable constraint types, thus, providing specialized constraints with more focused semantics than the general-purpose OCLConstraint.

The central metaclass of the AT instance metamodel is AT instance (Fig. 5). Because AT instance inherits from Entity, AT instances have a name and a unique identifier. Furthermore, AT instance has exactly one AT defining its type (also see Fig. 3). The latter also specifies the set of roles that are part of the AT. This specification enables AT instances to contain references between roles and components (Role2Component) as well as between roles and resources (Role2Resource). Components and resources are represented by PCM's AssemblyContexts and ResourceContainers, respectively.

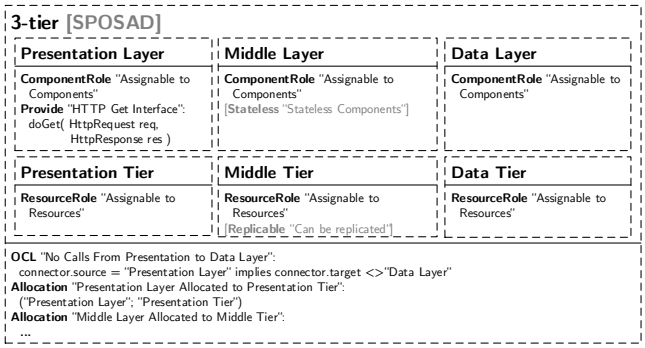

Figure 6: Roles and constraints of 3-tier and SPOSAD ATs. The SPOSAD AT extends the 3-tier AT by additional constraints on the middle layer and middle tier (marked in gray).

4. EVALUATION OF APPLICABILITY

To evaluate the applicability of ATs, we first use the AT language defined in Sec. 3 to specify a 3-tier AT and a SPOSAD AT (Sec. 4.1). Subsequently, we apply these ATs to the book shop scenario of Sec. 2 by specifying a book shop AT instance for each of these types, i.e., we specify a 3-tier book shop and a SPOSAD book shop model using AT instances (Sec. 4.2). Finally, we use these instances to conduct scalability and elasticity analyses (Sec. 4.3). We conclude that ATs are applicable for scalability analyses but have limits for elasticity analyses (Sec. 4.4).

4.1 3-Tier and SPOSAD AT

To specify a 3-tier and a SPOSAD AT, we first formalize roles and constraints given by the 3-tier and the SPOSAD architectural styles, respectively. Fig. 6 illustrates our resulting formalization for both ATs. We illustrate roles by dashed rectangles. Role names are at the top of the rectangle and role constraints at the bottom of the rectangle.

Accordingly, our 3-tier AT consists of three presentation layer roles and three tier roles (presentation, middle, and data, respectively). Appropriate constraints ensure that layer roles can only be assigned to components (ComponentRoleConstraint), tier roles can only be assigned to resources (ResourceRoleConstraint), and layers can only be allocated to the corresponding tiers (AllocationConstraint). Furthermore, we require at least an HTTP interface on the presentation layer (ProvideConstraint) and that presentation layer components cannot directly communicate to data layer components (OCLConstraint).

In Fig. 6, we also illustrate roles and constraints of the SPOSAD AT. Because SPOSAD is a 3-tier variation, we illustrate the SPOSAD AT by extending the 3-tier AT illustration (marked in gray). The only differences to the 3-tier AT are the different name of the AT ("SPOSAD" instead of "3-tier") and two additional constraints, a StatelessConstraint for the middle layer and a ReplicableConstraint for the middle tier role. The StatelessConstraint ensures that only stateless components are allowed to play the middle layer role. In conjunction with the ReplicableConstraint, one of SPOSAD's main characteristics is ensured, i.e., that middle layer components have to be stateless such that they can safely be duplicated and load-balanced using replicated tiers.

Besides the formalization of roles and constraints, we also have to specify a completion for each of the two ATs. To specify such a completion, we have to describe how to map to an analysis model that can serve for scalability and elasticity analyses. Such analyses shall allow us to check the scalability (**R2**) and elasticity (**R3**) requirements we stated in Sec. 2. Our idea to realize such a completion is to map to usual PCM models. Then, we can conduct a Palladio analysis based on these models and use Palladio's analysis results to calculate the scalability and elasticity metrics formulated in **R2** and **R3** if the mapping is semantic-preserving.

For the 3-tier AT, such a completion is the identity mapping to a usual PCM model. All constraints of the 3-tier AT are already fulfilled if we assume that the AT is applied on a usual PCM model and that, during PCM model specification, the constraints are assured by construction. For instance, an AT editor extension for PCM models could assure such a behavior.

For the SPOSAD AT, a completion additionally has to consider the StatelessConstraint and the ReplicableConstraint. These constraints cannot be assured by construction because neither stateless components nor replication are default PCM features. Therefore, our idea is to make use of the SimuLizar [2] PCM extension that allows for specifying self-adaptation rules, e.g., for replication.

Because of the constraints ReplicableConstraint and StatelessConstraint, our completion can safely create adaptation actions (using SimuLizar models) that (1) dynamically replicate middle layer components to additional tiers and (2) connect these components to a load balancer that is allocated on a separate tier between presentation and middle tier. We can safely model the load balancer with zero performance impact on a separate tier if we assume that the load balancer itself has a negligible performance impact. However, without further assumptions, the resulting model is incomplete because a complete self-adaptation rule consists of adaptation actions and adaptation constraints that trigger these actions. We do not specify such adaptation constraints because such a constraint is application-specific and can, thus, not be specified in an AT. Therefore, we require that the ReplicableConstraint additionally constraints AT instances to specify adaptation constraints that can be used by our completion to map to a complete PCM (extended by SimuLizar) model. Then, this model preserves the semantics of the SPOSAD AT and is analyzable by Palladio.

4.2 3-Tier and SPOSAD Book Shop Models

In our scenario, the application of 3-tier AT and SPOSAD AT results in a 3-tier model and a SPOSAD model of the book shop. For this application, we can follow the process described in Sec. 3.1.

We start by selecting the 3-tier AT first, thus, creating a 3-tier AT instance. Having the 3-tier AT selected, we assign all of its roles to an initial, component-based design of our architecture using a PCM model. We illustrate this assignment in Fig. 7 using a diagram that combines several view types (view types for component structure, resources, and customer scenarios). The online shop model consists of the three inter-connected components and resources as already discussed before. The allocation of components to resources can be derived based on the selected AT (layers are always allocated to corresponding tiers). Furthermore, we specify several scalability- and elasticity-relevant information like the speed of resources, components' resource demand, and the workloads on the system in the form of customer scenarios (all of these information are denoted in yellow UML notes). For resource speed and demands, we use Amazon's EC2 Compute Unit (ECU) units that describe CPU integer processing power relative to hardware (ECUs are based on hardware benchmarks). For customer scenarios, we specify the changing workload information described in the book shop scenario in Sec. 2. For instance, we expect an initial customer arrival rate of 100 customers per minute and a long-term increase of this rate by 10%. We exemplified non-anticipated peak workloads that increase the arrival rate by 30% using an occurrence probability of 0.1%. Based on this model, we analyze it for scalability and elasticity properties in Sec. 4.3.

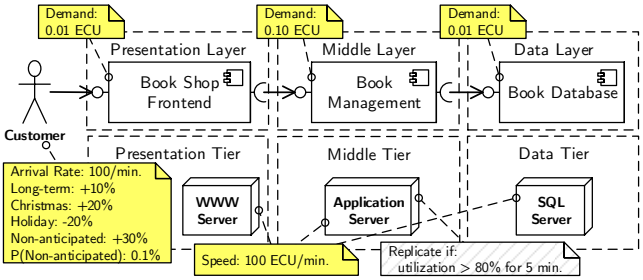

Figure 7: Assignment of roles to components and resources using AT instances. The SPOSAD-based model extends the 3-tier-based model by specifying an adaptation constraint for the middle tier resource (UML note with striped background).

We proceed analogously for the SPOSAD AT. However, we have to provide additional information when assigning the role of middle tier to the application server. Because the SPOSAD AT describes a replicable middle tier, its AT instances request for adaptation conditions, which are only available at instantiation time. Therefore, we specify the condition to replicate whenever the utilization of a middle tier CPU is greater than 80% for consecutive 5 minutes (UML note with striped background in Fig. 7).

4.3 Analysis of the Book Shop Models

For analyzing the 3-tier and the SPOSAD book shop models of Sec. 4.2, we first apply the completions described in Sec. 4.1 to each model and, subsequently, run a normal Palladio/SimuLizar analysis on the completions' output models. In this section, we describe these output models and argue why we can only conduct a scalability analysis and no elasticity analysis without extending Palladio.

For component structure and resources, we simply apply the identity mapping of our models to the corresponding Palladio models for components and resources. We derive Palladio's allocation model from the AT specifications that state that layers are allocated to corresponding tiers. We can also directly map the adaptation condition and adaptation rule specified in SPOSAD AT and SPOSAD book shop model to corresponding SimuLizar models.

However, we cannot directly map the customer scenario specification to Palladio/SimuLizar because this specification alters the workload on the system depending on passed

time, e.g., Christmas characterizes a concrete point in time where workload increases. This dependency of the workload to time is currently impossible to model and analyze with Palladio/SimuLizar; only static workload distributions can be modeled and analyzed. This lack does not prevent us from analyzing scalability but from analyzing elasticity.

For scalability, we run several independent Palladio/SimuLizar analyses. We start with an analysis with an arrival rate of 100 customers per minute. Then, we consecutively increase this rate until we reach the highest possible arrival rate of ~ 172 customers per minute (Christmas after 1 year plus a non-anticipated peak: $100 \cdot 1.1 \cdot 1.2 \cdot 1.3 \approx 172$). For every analysis result, we investigate the violation of requirements in the steady state, e.g., whether the response time limit of 2 seconds is exceeded. We can neglect elasticity for these investigations because we investigate the steady-state and no transient adaptation phases. If every analysis result confirms that no other requirements are violated, we can conclude that our scalability requirement is met.

For both of our models, our analysis yields that the scalability requirement is violated as soon as the arrival rate increases beyond 100 customers per minute because "WWW Server" as well as "SQL Server" become bottleneck resources. Therefore, we would suggest the architect to follow neither of the two designs in the current form. Instead, the architect should, e.g., increase the processing speed of the two bottleneck servers to 200 ECU per minute, which would result in a scalable system (confirmable by re-executing the scalability analysis).

For elasticity, Palladio needs to be extended to support time-dependent, dynamic customer scenarios. Without such an extension, transient phases resulting from sudden peaks or periodic workload changes cannot be modeled and analyzed. However, these transient phases have to be investigated for elasticity analyses because elasticity properties are related to such phases only. For example, a possible elasticity metric is the number of requirement violations during such a transient phase. Because of this lack in Palladio, our future work will target a suitable Palladio extension.

4.4 Assessing the Applicability of ATs

Because we successfully conducted a scalability analysis in Sec. 4.3, we conclude that ATs are applicable for scalability analyses of the two variants of 3-tier architectures (classical 3-tier and SPOSAD). Because we investigated two variants, our results indicate that ATs are generalizable to 3-tier-based architectures in general. However, we plan to extend our results by more 3-tier-based examples to strengthen these initial insights. In this context, we plan to enrich our initial ATs by multi-tenancy features as well as special technologies on the data layer such as (replicable) NoSQL databases and a MapReduce programming model.

Because we were unable to conduct an elasticity analysis, we conclude that our approach to map ATs to Palladio/SimuLizar is currently inapplicable for such analyses. However, we identified Palladio's lack of time-dependent, dynamic customer scenarios as the reason for this result. We expect that ATs are applicable for elasticity analyses once we provide a suitable Palladio extension for such purposes.

In summary, our initial results indicate that ATs are applicable for variants of 3-tier architectures. These results are currently limited and more investigations have to be provided to strengthen these.

5. RELATED WORK

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

A generally rich set of literature provides, classifies, and surveys sets of *architectural styles* (e.g., [6], [22]). However, these styles lack an explicit consideration of cloud computing environments as well as a focus on scalability and elasticity.

In the context of cloud computing, typically applied styles are REST [10] for HTTP and SPIAR [19] for AJAX. Also Erl et al. [9] describe a set of cloud computing styles that foster scalability and elasticity (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 (PaaS providers) or (2) access to (virtual) nodes where users can operate their SaaS application (IaaS providers). Therefore, these third party providers can utilize the presented architectural styles. However, because 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 focus directly and explicitly on architectural styles for SaaS applications by investigating the few architectural styles that do cover aspects of SaaS applications and analyzing system designs based on such styles using our AT language. Two examples for these styles are SPOSAD [16] and SOCCA [24]. As we have seen, SPOSAD describes a 3-tier variation that promotes a stateless middle tier for scalability and elasticity [16]. Because 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 [15]. These approaches allow for analyzing the performance (response time, throughput, utilization) of component-based systems as, for example, Palladio [4]. However, they lack support for cloud computing characteristics, e.g., an elastic provisioning of computing resources. Because of these characteristics, traditional performance metrics are insufficient.

Becker et al. [3] survey model-driven performance engineering approaches that support an elasticity management via self-adaptation, e.g., the SimuLizar [2] approach that extends Palladio. They conclude that these approaches are still limited because only two approaches target design-time models and because 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 and elastic SaaS applications. Especially the latter aspect lacks investigation, i.e., using performance engineering techniques to conduct scalability and elasticity analyses. Therefore, we extend existing performance analysis approaches like Palladio and SimuLizar to support scalability and elasticity analyses. For example, we propose to extend Palladio by time-dependent, dynamic customer scenarios. Furthermore, we investigate ATs for design-time scalability and elasticity analyses of SaaS applications based on architectural styles. In particular, to cope with the lack of realistic validations, we want to conduct case studies to identify appropriate architectural styles for scalable and elastic SaaS applications.

6. CONCLUSION

In this work-in-progress paper, we propose a formalization of the Architectural Template (AT) language in terms of a novel metamodel. We use this language to formalize 3-tier and SPOSAD architectural styles. By applying the formalized styles to design and analyze an example online book shop, we provide evidence that our AT language is applicable for variants of 3-tier architectures (such as SPOSAD). Moreover, our evaluation shows that scalability analyses can be conducted by utilizing existing performance engineering approaches. However, for elasticity analyses, it is currently impossible to apply the AT language. We identify the missing support for dynamic customer scenarios in performance engineering approaches as main reason for this lack.

Our findings help software architects in making architectural decisions when designing scalable and elastic SaaS applications. The AT language helps these architects to make such decisions explicit, analyzable, and reusable. Therefore, software architects benefit from a lower risk of implementing SaaS applications with unsatisfying scalability and elasticity properties. Moreover, we point to deficiencies in performance engineering approaches, thus, motivating scientists to investigate possible improvements to these approaches.

The AT language is still work-in-progress, thus, requiring several future work directions. First, the AT language can be extended by further role constraints and extension mechanisms. For instance, extending the 3-tier AT to specify the SPOSAD AT can be enabled. Second, performance engineering approaches can be extended by dynamic customer scenarios to support elasticity analyses. Third, tool support can be provided to automate scalability and elasticity analyses based on the AT language. Fourth, our initial evaluation results should be strengthened by additional evaluations.

7. REFERENCES

[1] Amazon.com Inc. AWS elastic beanstalk. http://aws.amazon.com/elasticbeanstalk/, Last visited: 30 Jan 2014.

[2] M. Becker, S. Becker, and J. Meyer. SimuLizar: design-time modelling and performance analysis of self-adaptive systems. In *Proceedings of Software Engineering 2013 (SE2013), Aachen*, 2013.

[3] M. Becker, M. Luckey, and S. Becker. Model-driven performance engineering of self-adaptive systems: a survey. In *QoSA '12*, pages 117–122, New York, 2012. ACM.

[4] S. Becker, H. Koziolek, and R. Reussner. The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1), Jan. 2009.

[5] G. Brataas, E. Stav, S. Lehrig, S. Becker, G. Kopcak, and D. Huljenic. CloudScale: Scalability Management for Cloud Systems. In *4th Int. Conf. on Performance Engineering*. ACM, Apr. 2013.

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, and M. Stal. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Wiley, volume 1 edition, Aug. 1996.

[7] R. Carnap. *Meaning and Necessity: A Study in Semantics and Modal Logic*. Midway reprints. University of Chicago Press, 1988.

[8] T. P. Council. 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.

[9] T. Erl, R. Puttini, and Z. Mahmood. *Cloud Computing: Concepts, Technology and Design*. Prentice Hall PTR, 2013.

[10] R. Fielding and R. Taylor. Principled design of the modern web architecture, 2000.

[11] Google Inc. Google app engine. http://developers.google.com/appengine/, Last visited: 30 Jan 2014.

[12] J. Happe. *Predicting Software Performance in Symmetric Multi-core and Multiprocessor Environments*. PhD thesis, University of Oldenburg, Germany, 2008.

[13] N. R. Herbst, S. Kounev, and R. 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.

[14] T. Kühne. Matters of (meta-)Modeling. *Software and System Modeling*, 5(4):369–385, 2006.

[15] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Perform. Eval.*, 67(8):634–658, Aug. 2010.

[16] H. 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.

[17] S. 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.

[18] P. Mell and T. Grance. The NIST definition of cloud computing. *NIST Special Publication*, 145(6):7, 2011.

[19] A. Mesbah and A. van Deursen. A component- and push-based architectural style for ajax applications. *Journal of Systems and Software*, 81(12):2194–2209, 2008.

[20] Microsoft Corporation. Microsoft windows azure. http://www.windowsazure.com/, Last visited: 30 Jan 2014.

[21] mOSAIC EU Project. mOSAIC: open-source API and platform for multiple clouds. http://www.mosaic-cloud.eu/, Last visited: 30 Jan 2014.

[22] M. Shaw and P. C. Clements. A field guide to boxology: Preliminary classification of architectural styles for software systems. In *Proceedings of the 21st International Computer Software and Applications Conference*, pages 6–13. IEEE, 1997.

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

[24] W.-T. Tsai, X. Sun, and J. Balasooriya. Service-oriented cloud computing architecture. In *ITNG'10*, pages 684–689, Washington, DC, USA, 2010. IEEE.