Policy-Driven Middleware for Multi-Tenant SaaS Services Configuration

Khadija Aouzal, SEEDS Team, STRS Lab, INPT, Rabat, Morocco
Hatim Hafidi, SEEDS Team, STRS Lab, INPT, Rabat, Morocco & IMS Team, ADMIR Lab, ENSIAS, Rabat, Morocco
Mohamed Dahchour, SEEDS Team, STRS Lab, INPT, Rabat, Morocco

ABSTRACT

The multi-tenancy architecture allows software-as-a-service applications to serve multiple tenants with a single instance. This is beneficial as it leverages economies of scale. However, it does not cope with the specificities of each tenant and their variability; notably, the variability induced in the required quality levels that differ from a tenant to another. Hence, sharing one single instance hampers the fulfillment of these quality levels for all the tenants and leads to service level agreement violations. In this context, this article proposes a policy-driven middleware that configures the service according to the non-functional requirements of the tenants. The adopted approach combines software product lines engineering and model driven engineering principles. It spans the quality attributes lifecycle, from documenting them to annotating the service components with them as policies, and it enables dynamic configuration according to service level agreements terms of the tenants.

KEYWORDS

MDE, Multi-Tenancy, Non-Functional Variability, Policy, SaaS, SLA, SPLE

INTRODUCTION

Software as a Service (SaaS) applications are generally built based on the multi-tenancy architecture (Kabbedijk et al., 2015). This architecture style allows serving multiple tenants with a single instance of the application. Hence, it enables, among others, high utilization of resources, easy maintenance and cost efficiency. However, it limits the support of service variability among tenants, since the same instance of the SaaS service is shared between them and the particularities of each tenant’s needs are not considered. To cope with this, customization and adaptation techniques must be enforced to get SaaS services tailored to tenants' needs and their specific contexts, while maintaining multi-tenancy.

Note that not only has the SaaS service to be adapted to functional requirements of the different tenants, but it has also to meet non-functional requirements in their various levels. In fact, the required service quality level may differ from a tenant to another. For example, a tenant can just require minimal security level as it needs solely to be authenticated and has no other security concerns; while another may require a high security level and advanced and supplementary security mechanisms such as access control and encryption. There are also situations when the same tenant changes its desired quality level through time; for instance, when there is an increase of service requests in a critical period, the tenant may want to decrease the response time and to increase the uptime value in order to smoothly perform his requests. Therefore, the application needs to be adapted dynamically in order to meet every tenant’s quality requirements as if he is the only one to consume the service.

However, configuring the service utterly for each tenant, at a time, is a heavy and tedious task for developers and architects. Thus, commonalities and variations among tenants regarding quality
attributes should be first captured. This is one of the principles of Software Product Lines Engineering (SPLE) that enables high reusability in shorter time, at lower costs and with higher quality (Pohl et al., 2005). The SPL paradigm allows creating product families or lines sharing commonalities and differing in some aspects. These aspects are generally represented as variation points, in most of the variability modeling languages, e.g. Feature Model (FM) (Batory, 2005), Orthogonal Variability Modeling (OVM) (Pohl et al., 2005), Extended Feature Model (Benavides et al., 2005), etc. In order to derive the final product tailored to specific needs, configuration and adaptation mechanisms are performed at the level of variation points while considering the dependencies between all the different artifacts composing the final product and the product line as well. SPL involves two complementary phases: Domain Engineering and Application Engineering. Domain Engineering allows the management of a product line or family as a whole and not as individual products. It follows “for reuse” development strategy, starting by domain analysis to identify the commonalities and variabilities among the requirements, and aiming at defining the architecture of a product line, its reusable artifacts and the relationships and dependencies between them. Application Engineering, on the other hand, follows “by reuse” development strategy. In this phase the final products are derived by configuring the artifacts defined in the Domain Engineering phase to meet specific requirements. It allows also conformity checking to validate that the final derived product is conformed to the specific needs it intends to respond to.

The proposed approach uses the SPL principles to build SaaS services tailored to tenant-specific Service Level Agreements (SLAs). In a previous work (Aouzal et al., 2018), the authors defined the Domain Engineering phase to build SLA families. For that purpose, they combined Model Driven Engineering (MDE) and SPL to model Non-Functional Requirements as features composing the FM. Based on domain analysis, the NFRs of the tenants, their commonalities, variation points and variants are modeled according to the proposed NFVariability Metamodel. Model-to-model transformations are then performed to generate the corresponding Generic SLA in conformance with the proposed VariableSLA metamodel. The Generic SLA is a document that contains the terms and the Service Level Objectives (SLOs) of all the contracting tenants, i.e. it encompasses mandatory terms and all the variants for the optional terms.

In this paper, the authors will continue on this SPL-MDE synergy while using EFM to model the non-functional properties attributed with their corresponding tenants. They propose a middleware responsible for intercepting tenants’ requests and mediating them to the corresponding components of the application, configured with adequate quality attributes. To achieve that, the middleware generates core and tenant-specific policies from the Generic SLA. These policies constitute the reference for the Configurator component to adapt the application according to the defined policies. Those configurations are based on annotations in the form of key-value pairs. This annotation-based approach is used for its simplicity and flexibility (Nosál & Porubán, 2014). As contributions of this paper: the authors present the architecture of the proposed middleware and its different components; they define a Policy metamodel, in order to model policies retrieved from the Generic SLA as annotations, along with model-to-model transformations from VariableSLA metamodel to Policy metamodel; and they propose a Configurator metamodel that models and describes how the Configurator component of the middleware configures the SaaS service. The proposition in its all phases is illustrated by a case-study.

The rest of the paper is structured as follows. Section 2 presents the motivating scenario that illustrates the problem. An overview of the proposed approach is described in Section 3, and the middleware architecture is presented in Section 4. Section 5 presents the model-to-model transformations performed to generate tenant-specific policies. The policy-driven service configuration is depicted in Section 6. Section 7 discusses and evaluates the approach. Section 8 presents related work. Finally, Section 9 concludes the paper and presents future work.
Related Content

E-Recruiting System Development and Architecture
In Lee (2007). Utilizing and Managing Commerce and Services Online (pp. 234-257).
www.igi-global.com/chapter/recruiting-system-development-architecture/30698?camid=4v1a

Mining and Analysis of Periodic Patterns in Weighted Directed Dynamic Network
Genetic Algorithm to Solve Multi-Period, Multi-Product, Bi-Echelon Supply Chain Network Design Problem
[www.igi-global.com/chapter/genetic-algorithm-solve-multi-period/42665?camid=4v1a](www.igi-global.com/chapter/genetic-algorithm-solve-multi-period/42665?camid=4v1a)

The 4th Industrial Revolution: Business Models and Evidence From the Field
[www.igi-global.com/article/the-4th-industrial-revolution/228962?camid=4v1a](www.igi-global.com/article/the-4th-industrial-revolution/228962?camid=4v1a)