The Journal of Systems and Software 91 (2014) 48–62

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Efficient customization of multi-tenant Software-as-a-Service applications with service lines

Stefan Walraven[a,*], Dimitri Van Landuyt[a], Eddy Truyen[a], Koen Handekyn[b], Wouter Joosen[a]

[a] iMinds-DistriNet, KU Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium
[b] UnifiedPost, Avenue Reine Astrid 92A, 1310 La Hulpe, Belgium

ARTICLE INFO

Article history:
Received 15 November 2012
Received in revised form 9 January 2014
Accepted 13 January 2014
Available online 22 January 2014

Keywords:
Multi-tenancy
SaaS
Variability

ABSTRACT

Application-level multi-tenancy is an architectural approach for Software-as-a-Service (SaaS) applications which enables high operational cost efficiency by sharing one application instance among multiple customer organizations (the so-called tenants). However, the focus on increased resource sharing typically results in a one-size-fits-all approach. In principle, the shared application instance satisfies only the requirements common to all tenants, without supporting potentially different and varying requirements of these tenants. As a consequence, multi-tenant SaaS applications are inherently limited in terms of flexibility and variability.

This paper presents an integrated service engineering method, called *service line engineering*, that supports co-existing tenant-specific configurations and that facilitates the development and management of customizable, multi-tenant SaaS applications, without compromising scalability. Specifically, the method spans the design, implementation, configuration, composition, operations and maintenance of a SaaS application that bundles all variations that are based on a common core.

We validate this work by illustrating the benefits of our method in the development of a real-world SaaS offering for document processing. We explicitly show that the effort to configure and compose an application variant for each individual tenant is significantly reduced, though at the expense of a higher initial development effort.

1. Introduction

Software as a Service (SaaS) has become a common software delivery model. It is a form of cloud computing that involves offering software services in an on-line and on-demand fashion (with the Internet as the main delivery mechanism). One of the key enablers in cloud computing to achieve economies of scale is *multi-tenancy* (Chong and Carraro, 2006; Guo et al., 2007): the sharing of resources among a large group of customer organizations, called *tenants*. This architectural concept can be applied at various levels of the software stack: at the infrastructure level (i.e. virtualization), at the OS and middleware level, and even at the application level. Each approach makes a different trade-off between (i) maximizing scalability and operational cost benefits (including hardware and software resource usage as well as maintenance effort), and (ii) maximizing flexibility to meet the potentially different and varying tenant-specific requirements (Walraven et al., 2011).

This paper focuses on application-level multi-tenancy. Application-level multi-tenancy achieves the highest degree of resource sharing between tenants (Chong and Carraro, 2006; Walraven et al., 2011). End users from different tenants are simultaneously served by a single application instance on top of shared infrastructure. However, when compared to infrastructure-level and middleware-level multi-tenancy, the inherent limitations in variability form a crucial disadvantage. The high degree of resource sharing typically results in a one-size-fits-all approach: the multi-tenant application only satisfies the requirements that are common to all tenants. Support for different and varying requirements of the different tenants is lacking.

To shorten the time-to-market, the initial development and release cycles of a SaaS application are typically focused on the

* Corresponding author. Tel.: +32 16327640.
E-mail addresses: stefan.walraven@cs.kuleuven.be (S. Walraven), dimitri.vanlanduyt@cs.kuleuven.be (D. Van Landuyt), eddy.truyen@cs.kuleuven.be (E. Truyen), koen.handekyn@unifiedpost.com (K. Handekyn), wouter.joosen@cs.kuleuven.be (W. Joosen).

0164-1212/$ – see front matter © 2014 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2014.01.021

needs of the first customer organizations (tenants). As the SaaS offering becomes more successful, an increasing amount of variations (ranging from minimal to substantial) is implemented to service new tenants and an increasing amount of tenant-specific configurations therefore have to co-exist at run time. This easily leads to an explosion of relatively small variations in the implementation as well as in the different configurations. This real-world scenario is experienced in many specific business cases; we have identified a lack in *methodical support for the development and customization of multi-tenant applications*. This lack of support can be characterized by two essential challenges:

First, SaaS providers need to be able to *manage and reuse the different configurations and software variations in an efficient way*, without compromising scalability; e.g. by avoiding additional overhead when provisioning new tenants.

Secondly, part of realizing the scalability benefits of SaaS is achieved by *self-service*: shifting some of the configuration efforts to the tenant side, e.g. by allowing the tenant to manage his tenant-specific requirements and by automating the run-time configuration process. Therefore, tenants require *additional support to manage the configuration in a tenant-driven customization approach*.

In the state of the art, some work has been performed to combine the benefits of software product line engineering (SPLE) (Clements and Northrop, 2001; Pohl et al., 2005) with those of multi-tenancy to facilitate the customization of SaaS applications tailored to the tenant-specific needs (Mietzner and Leymann, 2008; Mietzner et al., 2009; Walraven et al., 2011; Schroeter et al., 2012a). We define a *service line* as a specific concept that leverages on the notion of software product lines by offering a shared application instance that still is dynamically customizable to different tenant-specific requirements. However, none of the current service engineering approaches offer a complete customization process for multi-tenant applications. Moreover, customization is often limited to specific technologies or to specific types of applications.

Our solution is a feature-oriented method that is highly integrated, in the sense that the feature-level variability that is introduced in the early development stages is consistently and explicitly supported in each of the subsequent development stages, also in the run-time environment. This is a key difference with respect to traditional SPLE. Instead of delivering a dedicated, separate application product for each tenant (cf. the application engineering phase in SPLE), the entire service line (including all variations) is instantiated and deployed only once and simultaneously shared by all tenants. Specific software variants are activated at run time within one single SaaS application instance.

The main contribution of this paper is an integrated service line engineering method that focuses on addressing variability up front without compromising the scalability of SaaS applications. This method starts with the initial development phases (requirements engineering, architectural design and implementation) of the service line, but also focuses on the deployment, run-time configuration and composition, the operations and maintenance. The method is generic in the sense that each stage is open for existing work in the state of the art to be leveraged upon, yet it imposes some specific constraints (e.g. composability and traceability of features) and requires some enablers (e.g. multi-tenancy, feature-level versioning, tenant-level configuration interfaces) for service line variability.

We have validated this method in the development of an industry-level SaaS application in the domain of online B2B document processing. To this end, we closely collaborated with an industrial SaaS provider in the context of a collaborative research project (CUSTOMSS, 2011). Our evaluation focuses on illustrating (i) the efficiency benefits with respect to addressing the management complexity of many co-existing tenant-specific configurations, and (ii) the trade-off between the *early* effort required to design and implement the initial service line, and the *late* effort required to configure and compose application variants, to provision new tenants as well as to update and maintain the service line as a whole.

The structure of this paper is as follows. Section 2 motivates this work and elaborates on the problem statement. Section 3 articulates the concept of a service line and describes the service line engineering method. In Section 4, this method is applied on a SaaS application in the domain of document processing. We evaluate and discuss our work in terms of the efficiency benefits associated with service lines in this specific SaaS application in Section 5. In Section 6, related work is discussed and Section 7 concludes the paper.

2. Problem elaboration

The motivation for this paper is based on our extensive insight into the current state of practice of a number of industrial SaaS providers (which was obtained in the context of several applied research projects, e.g. CUSTOMSS, 2011). In this section, we first present our characterization of this current state of practice of developing, operating, and maintaining multi-tenant SaaS applications. Then, we list the main challenges that are addressed in this paper.

2.1. State of the practice

We present a number of development and management activities that occur in the lifecycle of a multi-tenant SaaS application. Specifically, we focus on the customization and management challenges that SaaS providers face to efficiently offer and manage their offerings.

The following stakeholders are involved in these scenarios. The *SaaS architect*, *SaaS developer* and *SaaS operator* are employees of the SaaS provider. In addition, each tenant should assign a *tenant administrator* role to the person responsible for managing the SaaS application on behalf of the tenant organization (e.g. for user and configuration management). In theory, this role can be assigned to an internal user of the tenant or to the SaaS provider, or even to value-added resellers who are a business channel between tenants and the SaaS provider. In practice, the tenant administrator is often an employee of the SaaS provider, as detailed technical knowledge of the SaaS application is required.

Scenario #1: Initial development of the SaaS application. This scenario entails the initial design and development of a multi-tenant SaaS application by the SaaS architect and the SaaS developers. The SaaS operator is responsible for deploying and managing the (running) implementation of this SaaS application.

To shorten the initial time-to-market, the first development cycles of a SaaS application are typically focused strongly on the needs of the first customer organizations (tenants). As a result, the SaaS application typically supports limited variability beyond the scope of the initial tenants.

Scenario #2: Provision a new tenant. In this scenario, a new tenant wants to use the SaaS application and customize it to its requirements. We assume that the SaaS application already covers the requirements of the new tenant, e.g. because these requirements are very similar to those of already provisioned tenants. (Scenario #4 covers the case where new requirements have to be supported.)

Based on his (technical) knowledge about the application, the tenant administrator has to manually translate the tenant's requirements into a software configuration for the multi-tenant SaaS application. Obviously when certain requirements of the new tenant are similar to those of other tenants, parts of the existing configuration can be reused. In practice, this is done in a weakly controlled and error-prone manner (e.g. by copy-pasting